Solution Space Partitioning for Extremal Set Theory

Jesse Looney* , Jonah McDonald* , Allison Klingler* , Gloria Wu* , Jonad Pulaj† , Haoze Wu*
* Amherst College, Amherst, USA
{jlooney27, jmcdonald27, aklingler27, gwu28, hwu}@amherst.edu
† Davidson College, Davidson, USA
jopulaj@davidson.edu

***Abstract*—We present a method for partitioning the solution space of statements in extremal set theory. Compared with domain-agnostic partitioning methods like look-ahead, we perform case analysis on the strategies by which a candidate solution can be constructed. We demonstrate that our approach can decompose problems in extremal set theory more effectively than look-ahead. Combining this new partitioning strategy with an exact proof-producing MILP solver, we are able to verify larger finite cases of Chvátal's Conjecture—a long-standing open question in extremal combinatorics—compared to previous work.**

***Index Terms*—Extremal Set Theory, Cube and Conquer, Solution Space Partitioning**

I. Introduction

Computational methods have increasingly been used to prove mathematical conjectures, particularly in combinatorics and discrete mathematics [1], [2], [3], [4], [5], [6], [7], [8], [9]. A common paradigm is to encode statements as constraint satisfaction or optimization problems, and use general-purpose solvers to search for counterexamples or certify correctness. For example, it has been demonstrated that carefully engineered integer programming frameworks can produce machine-checkable proofs for instances of conjectures in extremal combinatorics [10].

Statements in extremal set theory often concern families of sets over a ground set $[n]$. Even for modest values of n, the search space becomes prohibitively large: a family of subsets of $[n]$ is a subset of $2^{[n]}$, so the total number of possible families is 2^{2^n}, which grows doubly exponentially in n. As a result, such problems are typically only tractable for small n, with solver performance as the primary bottleneck. Nevertheless, the ability to verify conjectures for larger values of n is valuable: many extremal statements, when false, admit counterexamples at small values of n. Extending the verified range therefore both narrows the domain in which counterexamples can exist and increases confidence in the conjecture.

One promising approach to improving solver scalability is parallelization. The state-of-the-art approach for SAT-based theorem proving is Cube and Conquer [11], which partitions a problem into thousands to millions of independent subproblems using look-ahead heuristics. While effective in many applications, we observe empirically that Cube and Conquer based on look-ahead heuristics does not effectively decompose problems arising from extremal set theory into substantially easier subproblems.

In this paper, we propose a method for partitioning the solution space of statements in extremal set theory at a semantic level. Instead of branching on variables in the underlying encodings, we partition directly at the level of set families. The key idea is to branch on *strategies* by which a counterexample (i.e., a family of sets) could be constructed. We represent a strategy using representative sets that must be included in the family together with banned subfamilies that must be excluded. We formalize this approach as an abstract calculus that supports 1) repeated partitioning of a collection of candidate families into smaller sub-collections, and 2) pruning of sub-collections that cannot satisfy the statement of interest. We prove soundness, completeness, termination, and other fundamental proof-theoretic properties of this calculus.

We implement our proof system in Rust, taking as input a problem encoding, a solver oracle, and a ground set. We focus on Chvátal's Conjecture [12], a long-standing open question in extremal combinatorics, for which the previous best certified computational verification handles ground sets of size 7 [10]. We develop a SAT encoding of Chvátal's Conjecture and demonstrate that our semantic partitioning method decomposes the formula more effectively than look-ahead-based Cube and Conquer. We then apply our partitioning strategy to an existing Integer Linear Programming (ILP) encoding of Chvátal's Conjecture [10] and use the proof-producing MILP solver SCIP [13] to solve the resulting subproblems. In both the SAT and ILP settings, our partitioning significantly reduces problem difficulty. Moreover, with ILP, we generate and verify machine-checkable proof certificates for the partitions, which constitute a proof of Chvátal's Conjecture for a ground set of size 8.

In summary, we make the following contributions:

- a semantic-level partitioning method for extremal set theory;
- a formalization of the method as a calculus, with proofs of soundness, completeness, termination, and related properties;
- an oracle-agnostic Rust implementation of our partitioning strategy;
- an experimental evaluation using both SAT and ILP oracles, demonstrating the effectiveness of our partitioner; and
- a verified proof of Chvátal's Conjecture for $n = 8$.

https://doi.org/

II. Background

a) Extremal Combinatorics: Let n be a positive integer. Define the *ground set* $[n]$ to be the set $\{1, 2, \ldots, n\}$ and denote the powerset of $[n]$ by $2^{[n]}$. A *family* is a collection of sets—an element of $\mathcal{F} = 2^{2^{[n]}}$. A family $d \in \mathcal{F}$ is called a *downset* if it is closed under taking subsets—that is, for any $s \subseteq t \in 2^{[n]}$, if $t \in d$, then $s \in d$. An *intersecting* family is one whose members have pairwise nonempty intersections. A special case of an intersecting family is a *star*, a family for which there is some $e \in [n]$ contained in every member of the family.

b) Chvátal's Conjecture: The conjecture we use as a case study is *Chvátal's Conjecture*. Chvátal's Conjecture states that, in any downset, there are no intersecting subfamilies strictly larger than the largest star. We think of this conjecture in terms of the existence of a counterexample—an intersecting family that generates a downset whose stars are all smaller than the intersecting family.

c) Isomorphism of Families: We say two families f and g are isomorphic, and write $f \cong g$, if there exists a permutation $\pi : [n] \to [n]$ such that $\pi(f) = g$ (where $\pi(f) = \{\pi(s) \mid s \in f\}$ for any family f, and $\pi(s)$ is defined analogously for any set s). Isomorphism is an equivalence relation; we denote the equivalence class of a family f under isomorphism by $[f]$.

d) Cube and Conquer: In [11], Heule et al. take a new approach to SAT solving by combining CDCL and look-ahead solvers that allows for the solving of larger scale problems. Cube and Conquer, their solution, is also natural to parallelize.

Cube and Conquer uses a partitioning heuristic to select a set of n variables, partitioning the problem into 2^n subproblems based on partial assignments. A look-ahead heuristic is used to select the n variables, choosing ones that prune the search space by the greatest amount. The look-ahead solvers choose a variable x and split on it, assigning both true and false values, and use unit propagation to obtain a simplified boolean formula, or a cube.

Cube and Conquer's goal is to generate cubes that simplify the original formula and which can then be solved in parallel as independent cases of the problem. If the difficulty of the original problem is well distributed over the cubes, this approach can result in massive parallel speedup. In the case of Chvátal's Conjecture, however, we found that Cube and Conquer's look-ahead solver produced unbalanced partitions with limited simplification of the original problem. To improve the quality of the partition for problems in extremal set theory, we propose a partitioning method that splits on the inclusion of sets in a family rather than the truth of propositional variables.

III. Related Work

a) Chvátal's Conjecture: Eifler et al. showed that Chvátal's conjecture can be reformulated as an integer program, and were able to prove that the conjecture holds for ground sets of size at most $n = 7$ using an exact, proof-producing variant of the MILP solver SCIP combined with the certificate checker VIPR [10]. Later, an improved version of exact SCIP solved $n = 8$ in about 5.2 days [14]. However, an actual certificate was not generated in this sequental run, and [14] reports that such proof certificate would exceed 1 TB and that this is infeasible for the proof checker VIPR to validate. Plus, generating actual proof would further increase the solver runtime. In this work, we produce a partitioned proof consisting of smaller certificates and verify it with VIPR. Previous mathematical explorations also include [15], which assesses the conjecture using correlation inequalities between Boolean functions on sets. Secondly, [16] proves the conjecture for all downsets where subsets contain at most three elements.

b) Parallel Solving: There are two common paradigms for parallelizing SAT solvers: portfolio solving, where each thread runs the same problem with a different approach, and divide and conquer, where a problem is partitioned into smaller subproblems that are solved in parallel. Cube and Conquer [11] is a standard approach for parallel SAT solving in the case of theorem proving. Other parallel SAT solvers include Paracooba [17] and PaInleSS [18]. While there have been some forays into parallelizing ILP solvers [19], we are not aware of any parallelized exact ILP solvers capable of producing proofs.

c) Symmetry Breaking in SAT and ILP: Symmetry breaking is a method by which to create the subproblems that a divide and conquer parallelization approach requires. [20] introduces symmetry breaking for CDCL-based SAT solving. They utilize a similar method of investigating isomorphisms of a problem that is SAT specific, utilizing a dynamic method to prune the SAT problem as it is being explored. [21] also extends CDCL with symmetry breaking and applies it to the problem of generating graphs. [22] utilizes additional symmetry structures within SAT. [23] prunes by isomorphisms on ILP problems with constraints of the form $Ax \geq b$, utilizing aspects of group theory applied to an LP solver. We focus on solver- and encoding-agnostic symmetry-breaking specialized to extremal set theory.

IV. Methodology

We are interested in proving statements about families of sets. To this end, some definitions are in order. We call ϕ a *predicate on families* if it has the signature $\phi : \mathcal{F} \to \{true, false\}$. For any predicate ϕ on families, we define an *existential predicate* $\phi_\exists$ on collections of families by the following rule: for any collection of families $F \in 2^{\mathcal{F}}$, we have $\phi_\exists(F) = true \iff \exists f \in F. \phi(f)$. We say ϕ is *isomorphism-invariant* if, for any $f, g \in \mathcal{F}$ with $f \cong g$, we have $\phi(f) = \phi(g)$. We say ϕ is *satisfiable* if there exists $g \in \mathcal{F}$ such that $\phi(g) = true$, unsatisfiable otherwise.

Example 1. We can formulate Chvátal's Conjecture as a family predicate: $\phi^{Chv}(g) \iff$ "g is an intersecting family which generates a downset such that the two violate Chvátal's Conjecture". Later, we will check using the following methods that $\phi^{Chv}_\exists(\mathcal{F}) = false$ for $n = 8$ (recall $\mathcal{F} = 2^{2^{[n]}}$), verifying that there are no counterexamples among these families—Chvátal's Conjecture holds for ground sets up to size 8.

Given some family predicate ϕ, we present a method of partitioning $\mathcal{F}$, the collection of all families, into subcollections for which the truth of $\phi_\exists$ (e.g. whether the subcollection

contains counterexamples to Chvátal's Conjecture) can be checked in parallel. We motivate our method by analogy to Cube and Conquer, which we view as partitioning the collection of all total variable assignments into independent subcollections. Each subcollection is represented by a partial assignment, which contains literals specifying the values of certain variables. The partial assignment stands in for every possible total assignment that could be reached by "completing" the partial assignment by including additional literals. The partition is refined by branching on the truth of some variable, turning one partial assignment into two children that each extend it with a new literal.

The following functions partially capture these notions and apply them to the realm of families.

$$\texttt{ext}(f) = \{f \cup \{s\} \mid s \in 2^{[n]} \setminus f\} \tag{1}$$

$$\texttt{comp}(f) = \{g \in \mathcal{F} \mid g \supseteq f\} \tag{2}$$

If we think of a family f as a "partial family", then $\texttt{comp}(f)$ gives all possible "completions" of f, achieved by including additional sets, while $\texttt{ext}(f)$ gives all the incremental "extensions" of f by a single additional set. This pair of functions satisfies the following property:

$$\forall f \in \mathcal{F}. \forall g \in \texttt{ext}(f). \texttt{comp}(g) \subset \texttt{comp}(f) \tag{3}$$

which says that extending f to obtain $g = f \cup \{s\} \in \texttt{ext}(f)$ *strictly* reduces the range of possible completions. Just like we extend partial variable assignments with extra literals to focus on particular cases of total assignment, we can extend partial families with extra sets to focus on particular kinds of families.

For instance, observe that $\texttt{comp}(\emptyset) = \mathcal{F}$ contains intersecting families. However, if we consider the extension $\{\emptyset\} \in \texttt{ext}(\emptyset)$, we find that $\texttt{comp}(\{\emptyset\})$ contains no intersecting families, because any family containing the empty set cannot be intersecting. If we were searching for intersecting families in $\mathcal{F} = \texttt{comp}(\emptyset)$, we could partition into $\texttt{comp}(\{\emptyset\})$ and $\mathcal{F} \setminus \texttt{comp}(\{\emptyset\})$, and then immediately eliminate the first collection without exhaustively checking it. However, the notion of "partial families" does not give us any way to denote the exclusion of a particular set, such as $\emptyset$. For this reason, we will introduce a construct equipped with both a partial family and a collection of banned families, analogous to negative literals in a partial variable assignment.

While we make analogy to Cube and Conquer, the crucial difference is that our method reasons directly at the level of families. This allows us to recognize when two apparently different ways of extending a family are in fact isomorphic, which can greatly reduce duplicated work if we know that ϕ is isomorphism-invariant. In this section, we first describe a general approach to partitioning $\mathcal{F}$ by incrementally extending a partial family. Then we elaborate on an instantiation of this method that employs symmetry-breaking.

A. The Core Proof System

It turns out that property 3 is all we need to state the most general version of our partitioning scheme; we call any pair of functions $\texttt{ext}, \texttt{comp} : \mathcal{F} \to 2^{\mathcal{F}}$ an *extension-completion pair* if they satisfy this property.

Given a predicate ϕ on families and an extension-completion pair $\texttt{ext}$ and $\texttt{comp}$, we define a proof system $\textsc{symbreak}(\phi, \texttt{ext}, \texttt{comp})$ in the following way. First, a *node* is a pair $N = \langle f, B\rangle$ of a current family f and a collection $B \in 2^{\mathcal{F}}$ of banned families. When we ban a family, we really want to ban all its completions. Hence, for any collection $B \in 2^{\mathcal{F}}$ of families, we define $\texttt{comp}(B) = \bigcup_{b \in B} \texttt{comp}(b)$. Every node N represents a single collection of families, called its *concretization*, given by

$$N^{\#} = \texttt{comp}(f) \setminus \texttt{comp}(B). \tag{4}$$

The concretization of a collection $\mathcal{N}$ of nodes is given by $\mathcal{N}^{\#} = \bigcup_{N \in \mathcal{N}} N^{\#}$.

The calculus operates over a *state* consisting of a collection $\mathcal{N}$ of nodes that is updated at each step. There are two inference rules. **Branch** refines the partition by splitting a node into two children based on taking or banning an extension p. **Prune** removes nodes whose concretizations contain no families satisfying ϕ. In either case, we hope to preserve the existence of satisfying families: the $\mathcal{N}$ should contain families satisfying ϕ if and only if the same is true for the final $\mathcal{N}$ after some rule applications.

Branch

$$\frac{N = \langle f, B\rangle \in \mathcal{N} \quad p \in \texttt{ext}(f) \quad \texttt{comp}(p) \not\subseteq \texttt{comp}(B)}{\mathcal{N} \leftarrow (\mathcal{N} \setminus \{N\}) \cup \{\langle p, B\rangle, \langle f, B \cup \{p\}\rangle\}}$$

Prune

$$\frac{N \in \mathcal{N} \quad \phi_{\exists}(N^{\#}) = false}{\mathcal{N} \leftarrow \mathcal{N} \setminus \{N\}}$$

Example 2. Let $\phi(f)$ be the statement that f is an intersecting family, and consider the proof system $\textsc{symbreak}(\phi, \texttt{ext}, \texttt{comp})$ using the definitions of $\texttt{ext}$ and $\texttt{comp}$ given above. Let $N = \langle \emptyset, \emptyset\rangle$. The concretization is $N^{\#} = \mathcal{F}$. If our collection of nodes is $\mathcal{N} = \{N\}$, we can branch on $p = \{\emptyset\}$ to obtain two child nodes $N_1 = \langle \{\emptyset\}, \emptyset\rangle$ and $N_2 = \langle \emptyset, \{\{\emptyset\}\}\rangle$ and an updated $\mathcal{N} = \{N_1, N_2\}$. $N_1^{\#}$ denotes all families containing the empty set, while $N_2^{\#} = \texttt{comp}(\emptyset) \setminus \texttt{comp}(\{\{\emptyset\}\}) = \mathcal{F} \setminus \texttt{comp}(\{\emptyset\})$ denotes all families not containing the empty set. We can easily prove $\neg\phi_{\exists}(N_1^{\#})$ (that is, none of the families in $N_1^{\#}$ are intersecting families, because they contain the empty set), allowing us to prune N_1 and obtain $\mathcal{N} = \{N_2\}$. Note that we have not lost any intersecting families while doing these transformations, a property we will see is guaranteed by the system.

We prove three highly-general results about SYMBREAK. Theorem 1 ensures that the inference rules preserve families satisfying ϕ, so that our search is both sound and complete. Theorem 2 ensures that the "partition" resulting from applications of the inference rules is indeed disjoint. These results follow from induction on Lemmas 1 and 2 proven below (full

proofs for these theorems and any other results not proven in the main text can be found in the online appendix).

Lemma 1. *Let $N_1 = \langle p, B\rangle$ and $N_2 = \langle f, B \cup \{p\}\rangle$ be the child nodes spawned by applying the **Branch** rule to a node $N = \langle f, B\rangle$ in the proof system* SYMBREAK$(\phi, \texttt{ext}, \texttt{comp})$. *Then $N_1^\# \cup N_2^\# = N^\#$.*

Proof. Since $p \in \texttt{ext}(f)$, we have $\texttt{comp}(p) \subset \texttt{comp}(f)$ by Property 3. It follows that $\texttt{comp}(p) \cup (\texttt{comp}(f) \setminus \texttt{comp}(p)) = \texttt{comp}(f)$. Thus

$$\begin{aligned} N_1^\# \cup N_2^\# &= [\texttt{comp}(p) \setminus \texttt{comp}(B)] \\ &\quad \cup [\texttt{comp}(f) \setminus \texttt{comp}(B \cup \{p\})] \\ &= [\texttt{comp}(p) \cup (\texttt{comp}(f) \setminus \texttt{comp}(p))] \setminus \texttt{comp}(B) \\ &= \texttt{comp}(f) \setminus \texttt{comp}(B) = N^\#. \end{aligned}$$

□

Lemma 2. *Let $N_1 = \langle p, B\rangle$ and $N_2 = \langle f, B \cup \{p\}\rangle$ be the child nodes spawned by applying the **Branch** rule to a node $N = \langle f, B\rangle$ in the proof system* SYMBREAK$(\phi, \texttt{ext}, \texttt{comp})$. *Then $N_1^\# \cap N_2^\# = \emptyset$.*

Proof. We have

$$\begin{aligned} N_1^\# \cap N_2^\# &= [\texttt{comp}(p) \setminus \texttt{comp}(B)] \\ &\quad \cap [\texttt{comp}(f) \setminus \texttt{comp}(B \cup \{p\})] \\ &= [\texttt{comp}(p) \cap (\texttt{comp}(f) \setminus \texttt{comp}(p))] \setminus \texttt{comp}(B) \\ &= \emptyset \setminus \texttt{comp}(B) = \emptyset. \end{aligned}$$

□

Theorem 1. *Suppose $\mathcal{N}_0$ is a collection of nodes, and let $\mathcal{N}_k$ be the resulting collection of nodes after $k \in \mathbb{N}$ applications of the **Branch** and **Prune** rules of a proof system* SYMBREAK$(\phi, \texttt{ext}, \texttt{comp})$ *starting from state $\mathcal{N}_0$. If ϕ is a predicate on families, then $\phi_\exists(\mathcal{N}_k^\#) = \phi_\exists(\mathcal{N}_0^\#)$.*

Theorem 2. *Suppose $\mathcal{N}_0$ is a collection of nodes, and let $\mathcal{N}_k$ be the resulting collection of nodes after $k \in \mathbb{N}$ applications of the **Branch** and **Prune** rules of a proof system* SYMBREAK$(\phi, \texttt{ext}, \texttt{comp})$ *starting from state $\mathcal{N}_0$. If the nodes in $\mathcal{N}_0$ have pairwise disjoint concretizations, then the nodes in $\mathcal{N}_k$ have pairwise disjoint concretizations.*

The final major result is Theorem 3, which says that one cannot apply the inference rules of SYMBREAK indefinitely. In particular, any decision procedure that iteratively applies the rules until it no longer can is guaranteed to terminate. The prune rule will inevitably run out of targets unless we can keep adding nodes using the branch rule, so we hope that there is some case in which we no longer branch. The following lemma suggests one such scenario.

Lemma 3. *In any proof system* SYMBREAK$(\phi, \texttt{ext}, \texttt{comp})$, *the **Branch** rule cannot be applied to nodes with empty concretizations.*

Proof. Suppose $N = \langle f, B\rangle$ is a node with $N^\# = \emptyset$. Suppose we wish to branch on $p \in \texttt{ext}(f)$. We have

$$\texttt{comp}(f) \setminus \texttt{comp}(B) = N^\# = \emptyset \tag{5}$$

so $\texttt{comp}(f) \subseteq \texttt{comp}(B)$. We also have $\texttt{comp}(p) \subset \texttt{comp}(f)$ by Property 3. It follows that $\texttt{comp}(p) \subset \texttt{comp}(B)$, so we cannot apply the branch rule for this arbitrary choice of p, because the premise $\texttt{comp}(p) \not\subseteq \texttt{comp}(B)$ is not satisfied. Hence, we cannot apply the **Branch** rule. □

Intuitively, as we apply the **Branch** rule, we more and more precisely specify the range of families represented by the child nodes, shrinking their concretizations until they become empty. Therefore, we want to show that children spawned by the branch rule are smaller, in some sense, than their parent. To this end, we define the *magnitude* of a node $N = \langle f, B\rangle$ to be $\|N\| = |\texttt{comp}(f)||\mathcal{F} \setminus \texttt{comp}(B)|$. The magnitude of a node should not be confused with the cardinality of its concretization, denoted $|N^\#|$. The actual values of nodes' magnitudes are in general only meaningful in comparison to one another. However, we have the important property that a magnitude of zero corresponds to an empty concretization.

Lemma 4. *For any node $N = \langle f, B\rangle$, if $\|N\| = 0$, then $N^\# = \emptyset$.*

Proof. Suppose $\|N\| = 0$. Then at least one of $|\texttt{comp}(f)| = 0$ and $|\mathcal{F} \setminus \texttt{comp}(B)| = 0$ must be true. If the former is true, then

$$N^\# = \texttt{comp}(f) \setminus \texttt{comp}(B) = \emptyset \setminus \texttt{comp}(B) = \emptyset \tag{6}$$

If the latter is true, then we must have $\mathcal{F} \subseteq \texttt{comp}(B)$. But then

$$\texttt{comp}(f) \subseteq \mathcal{F} \subseteq \texttt{comp}(B) \tag{7}$$

so $N^\# = \texttt{comp}(f) \setminus \texttt{comp}(B) = \emptyset$. □

Now we can show that child nodes have strictly smaller magnitudes than their parents. This will immediately allow us to prove that only finite sequences of rule applications are possible in SYMBREAK.

Lemma 5. *Let $N_1 = \langle p, B\rangle$ and $N_2 = \langle f, B \cup \{p\}\rangle$ be the child nodes spawned by applying the **Branch** rule to a node $N = \langle f, B\rangle$ in the proof system* SYMBREAK$(\phi, \texttt{ext}, \texttt{comp})$. *Then $\|N_1\| < \|N\|$ and $\|N_2\| < \|N\|$.*

Proof. By Lemma 3, since we applied the **Branch** rule, we must have $N^\# \neq \emptyset$. Therefore, by Lemma 4, we have $\|N\| \neq 0$, so

$$|\texttt{comp}(f)| > 0, \tag{8}$$

$$|\mathcal{F} \setminus \texttt{comp}(B)| > 0. \tag{9}$$

Since $p \in \texttt{ext}(f)$, we have $\texttt{comp}(p) \subset \texttt{comp}(f)$ by Property 3, so $|\texttt{comp}(p)| < |\texttt{comp}(f)|$. Consequently, using (9), we have

$$\begin{aligned} \|N_1\| &= |\texttt{comp}(p)| \cdot |\mathcal{F} \setminus \texttt{comp}(B)| \\ &< |\texttt{comp}(f)| \cdot |\mathcal{F} \setminus \texttt{comp}(B)| = \|N\|. \end{aligned}$$

Moreover, since we applied the **Branch** rule, we must have that $\texttt{comp}(p) \not\subseteq \texttt{comp}(B)$. Hence, $|\texttt{comp}(B \cup \{p\})| = |\texttt{comp}(B) \cup \texttt{comp}(p)| > |\texttt{comp}(B)|$. Therefore,

$$\begin{aligned}|\mathcal{F} \setminus \texttt{comp}(B \cup \{p\})| &= |\mathcal{F}| - |\texttt{comp}(B \cup \{p\})| \\ &< |\mathcal{F}| - |\texttt{comp}(B)| \\ &= |\mathcal{F} \setminus \texttt{comp}(B)|.\end{aligned}$$

Using (8), we find that

$$\begin{aligned}\|N_2\| &= |\texttt{comp}(f)| \cdot |\mathcal{F} \setminus \texttt{comp}(B \cup \{p\})| \\ &< |\texttt{comp}(f)| \cdot |\mathcal{F} \setminus \texttt{comp}(B)| = \|N\|.\end{aligned}$$

□

Theorem 3. *There are no infinite sequences of rule applications in any proof system* SYMBREAK$(\phi, \texttt{ext}, \texttt{comp})$.

Proof. (Sketch: See the Appendix for a full proof.) Proof by contradiction. Each **Branch** application reduces the number of high magnitude nodes in favor of lower magnitude nodes, while applying **Prune** can never recover lost magnitude. Inevitably, one runs out of valid (positive magnitude) targets to branch on and subsequently out of nodes to prune. □

B. Breaking Symmetry

With the most general results established, we turn towards a specific instantiation of SYMBREAK with an extension-completion pair that will be suitable for breaking symmetry:

$$\texttt{ext}_{\cong}(f) = \{f \cup \{s\} \mid s \in 2^{[u]} \setminus f\} \tag{10}$$

$$\texttt{comp}_{\cong}(f) = \{g \in \mathcal{F} \mid \exists f' \in [f].\, g \supseteq f'\} \tag{11}$$

The definition of $\texttt{ext}_{\cong}$ the same as the $\texttt{ext}$ we defined in an earlier example. In contrast, $\texttt{comp}_{\cong}$ explicitly references isomorphism, defining the completion of a family f to consist of any family that contains f *or a family isomorphic to* f (compare the definition of $\texttt{comp}_{\cong}$ with Equation 2). What this means for the execution of SYMBREAK is that when we ban the completion of a family, we are able to eliminate many families with entirely different compositions that are nonetheless isomorphic (and thus redundant, when the family-predicate ϕ is isomorphism-invariant).

Example 3. Let $N = \langle \emptyset, \emptyset \rangle$. Branching gives $N_1 = \langle \{\{1\}\}, \emptyset \rangle$ and $N_2 = \langle \emptyset, \{\{\{1\}\}\} \rangle$. Note that $[\{\{1\}\}] = \{\{\{i\}\} \mid i \in [n]\}$, so $\texttt{comp}_{\cong}(\{\{1\}\})$ includes all families that contain *any* singleton set. Hence, $N_2^{\#} = \texttt{comp}_{\cong}(\emptyset) \setminus \texttt{comp}_{\cong}(\{\{1\}\})$ denotes all families that do not contain a singleton set. Node N_2 now does not admit branching on, say, $\{\{2\}\}$, since $\texttt{comp}_{\cong}(\{\{2\}\}) = \texttt{comp}_{\cong}(\{\{1\}\})$ (violating the $\texttt{comp}(p) \not\subseteq \texttt{comp}(B)$ premise). Thus, banning $\{\{1\}\}$ also banned $\{\{2\}\}$. This prevents us from duplicating work, as we already consider the isomorphic case in N_1.

In this subsection, we discuss the implications of specifying this extension-completion pair on the properties of SYMBREAK. First, we introduce two quick but important properties of $\texttt{comp}_{\cong}$, and we apply them to show that $\texttt{ext}_{\cong}$ and $\texttt{comp}_{\cong}$ do form an extension-completion pair as we have suggested.

Lemma 6. *For any family* $f \in \mathcal{F}$*, we have* $f \in \texttt{comp}_{\cong}(f)$.

Proof. This follows from $f \in [f]$ and $f \supseteq f$. □

Lemma 7. *Suppose* $f, g \in \mathcal{F}$ *with* $f \in \texttt{comp}_{\cong}(g)$*. Then* $\texttt{comp}_{\cong}(f) \subseteq \texttt{comp}_{\cong}(g)$.

Proof. We know from $f \in \texttt{comp}_{\cong}(g)$ that $f \supseteq g'$ for some $g' \in [g]$ (so there is a permutation π_g that sends g to g'). Suppose $h \in \texttt{comp}_{\cong}(f)$. Then $h \supseteq f'$ for some $f' \in [f]$ (so there is a permutation π_f that sends f to f'). Now we have

$$h \supseteq f' = \pi_f(f) \supseteq \pi_f(g') = \pi_f(\pi_g(g)). \tag{12}$$

Moreover, $\pi_f \circ \pi_g$ is a permutation, so $\pi_f(\pi_g(g)) \in [g]$, and thus $h \in \texttt{comp}_{\cong}(g)$. □

Theorem 4. *The functions* $\texttt{ext}_{\cong}$ *and* $\texttt{comp}_{\cong}$ *are an extension-completion pair.*

Proof. Suppose $f \in \mathcal{F}$ and $g \in \texttt{ext}_{\cong}(f)$. Then $g = f \cup \{s\}$ for some $s \in 2^{[n]} \setminus f$. In particular, $g \supsetneq f$, so $g \in \texttt{comp}_{\cong}(f)$. By Lemma 7, that means $\texttt{comp}_{\cong}(g) \subseteq \texttt{comp}_{\cong}(f)$.

To show that $\texttt{comp}_{\cong}(g) \neq \texttt{comp}_{\cong}(f)$, we will note that $f \in \texttt{comp}_{\cong}(f)$ by Lemma 6, and show that $f \notin \texttt{comp}_{\cong}(g)$. Suppose $f \in \texttt{comp}_{\cong}(g)$. Then $f \supseteq g'$ for some $g' \in [g]$. But

$$|g'| = |g| = |f \cup \{s\}| > |f| \tag{13}$$

because $s \notin f$. Therefore, $|f| \geq |g'| > |f|$, a contradiction. □

Now that we have Theorem 4, we are justified in considering instantiations of SYMBREAK of the form SYMBREAK$(\phi, \texttt{ext}_{\cong}, \texttt{comp}_{\cong})$. It is expected that such instantiations will additionally require that ϕ be isomorphism-invariant, but this constraint is not necessary for the results we develop here.

The main result we aim for is a strengthening of Lemma 5: when breaking symmetry, branching not only decreases the (rather contrived) *magnitude* of a node, but directly decreases the *cardinality* of a node's concretization. All we require before proving this is a small lemma:

Lemma 8. *In a proof system* SYMBREAK$(\phi, \texttt{ext}_{\cong}, \texttt{comp}_{\cong})$*, the* ***Branch*** *rule cannot be applied to nodes* $N = \langle f, B \rangle$ *where* $f \in \texttt{comp}_{\cong}(B)$.

Proof. Suppose $f \in \texttt{comp}_{\cong}(B)$. Then $f \in \texttt{comp}_{\cong}(b)$ for some $b \in B$. By Lemma 7, that means $\texttt{comp}_{\cong}(f) \subseteq \texttt{comp}_{\cong}(b) \subseteq \texttt{comp}_{\cong}(B)$. Hence $N^{\#} = \texttt{comp}_{\cong}(f) \setminus \texttt{comp}_{\cong}(B) = \emptyset$. By Lemma 3, we cannot apply the **Branch** rule to N. □

Theorem 5. *Let* $N_1 = \langle p, B \rangle$ *and* $N_2 = \langle f, B \cup \{p\} \rangle$ *be nodes spawned by applying the* ***Branch*** *rule to a node* $N = \langle f, B \rangle$ *in a proof system* SYMBREAK$(\phi, \texttt{ext}_{\cong}, \texttt{comp}_{\cong})$*. Then* $|N_1^{\#}|, |N_2^{\#}| < |N^{\#}|$.

Proof. Lemmas 1 and 2 together tell us that $|N_1^\#| + |N_2^\#| = |N^\#|$. So we need only show $0 < |N_1^\#| < |N^\#|$. By Lemma 8, we have $f \notin \texttt{comp}_\cong(B)$. We also have $f \in \texttt{comp}_\cong(f)$ by Lemma 6, so $f \in \texttt{comp}_\cong(f) \setminus \texttt{comp}_\cong(B) = N^\#$. But $f \notin \texttt{comp}_\cong(p)$ (or else $\texttt{comp}_\cong(f) \subseteq \texttt{comp}_\cong(p)$ by Lemma 7, violating Property 3), so $f \notin \texttt{comp}_\cong(p) \setminus \texttt{comp}_\cong(B) = N_1^\#$. Hence, $N_1^\# \subset N^\#$, which means $|N_1^\#| < |N^\#|$. Moreover, since we applied the **Branch** rule, we must have $\texttt{comp}_\cong(p) \not\subseteq \texttt{comp}_\cong(B)$, so $N_1^\# = \texttt{comp}_\cong(p) \setminus \texttt{comp}_\cong(B) \neq \emptyset$, which implies $|N_1^\#| > 0$. □

C. Pruning with SAT

Implementing a decision procedure for $\textsc{symbreak}(\phi, \texttt{ext}_\cong, \texttt{comp}_\cong)$ that is able to apply the **Prune** rule requires a decision procedure for determining $\neg\phi_\exists(N^\#)$. Below, we outline how to leverage automated reasoning tools to do this.

In the most general terms, any constrained system of binary variables generates a family predicate. Suppose we have a set $V = \{v_s \mid s \in 2^{[n]}\}$ of binary variables $v_s \in \{true, false\}$. For any family $g \in \mathcal{F}$, we can define a unique assignment $A_g : V \to \{true, false\}$ by the rule $A_g(v_s) = true$ iff $s \in g$. Hence, given any constraint $K = \{A_1, \ldots, A_k\}$ of allowed (e.g. satisfying) assignments, we obtain a family predicate by defining $\phi(g) = true$ iff $A_g \in K$.

Many constraint languages admit interpretation in this way. For example, any propositional formula, together with a mapping of some of its variables to sets, generates a family predicate under the natural constraint that K is the set of assignments to those variables (i.e. partial assignments) that can be extended into satisfying assignments. As another example, consider the ILP problem $P_{red}(n)$ in [10]. Through our lens, V is the set of the y_S variables, and K is the set of assignments to those variables such that the ILP is feasible with optimal objective value zero. This constrained system generates a family predicate, which the authors showed is equisatisfiable to ϕ^{Chv}.

Suppose ϕ is a predicate on families. For any node $N = \langle f, B\rangle$ in the proof system $\textsc{symbreak}(\phi, \texttt{ext}_\cong, \texttt{comp}_\cong)$, we define the *restriction of ϕ corresponding to N* by

$$\phi_N(g) = \phi(g) \wedge f \subseteq g \wedge g \notin \texttt{comp}_\cong(B) \tag{14}$$

That is, ϕ_N is the same predicate as ϕ except that satisfying families must additionally (1) contain f and (2) must not be banned. We thus limit our focus to satisfying families contained in $N^\#$. Even further, we break symmetry by restricting to families that actually contain the representative f itself (not just isomorphic copies), making it feasible to automatically deduce the satisfiability of ϕ_N in practice.

For example, if ψ is a propositional formula containing variables v_s for all $s \in 2^{[n]}$, we can define

$$\psi_N = \psi \wedge \alpha_f \wedge \beta_B \tag{15}$$

$$\alpha_f = \bigwedge_{s \in f} v_s \tag{16}$$

$$\beta_B = \bigwedge_{b \in B} \bigwedge_{b' \in [b]} \bigvee_{s \in b'} \neg v_s \tag{17}$$

If the predicate generated by ψ over the v_s is ϕ, then the predicate generated by ψ_N must be ϕ_N. The α_f constraints enforce that a satisfying assignment represents a superset of f, while the β_B constraints require that it is not in $\texttt{comp}_\cong(B)$ (by ensuring it is not a superset of anything isomorphic to some $b \in B$).

Quite analogously, if ω is an ILP containing variables $v_s \in \{0, 1\}$ for all $s \in 2^{[n]}$, we can add additional constraints on those variables:

$$v_s = 1 \quad s \in f \tag{18}$$

$$\sum_{s \in b'} v_s \leq |b'| - 1 \quad b' \in [b],\, b \in B \tag{19}$$

If ω and the v_s, together with a requirement on the feasibility or optimal value of ω, generate a family predicate ϕ, then the modified problem with the new constraints generates ϕ_N. Analogous encodings could be derived for any other constraint language. Note that the banned family constraints (β_B in the SAT encoding) can in principal result in an exponential blowup of the encoding size. While this is bound to happen at some point, we have not yet observed in practice the subproblem encoding incurring an overhead that outweighs the benefit of solution space reduction.

Now that we know we can automatically deduce the satisfiability of the predicate ϕ_N using a constraint language and solver of our choice, we wish to show that proving ϕ_N unsatisfiable is enough to justify pruning the node N.

Lemma 9. *Suppose B is a collection of families. Suppose $g_1 \cong g_2$ are families. If $g_1 \in \texttt{comp}_\cong(B)$, then $g_2 \in \texttt{comp}_\cong(B)$.*

Proof. Suppose $g_1 \in \texttt{comp}_\cong(B)$. Then g_1 contains some $b' \cong b$ for some $b \in B$. We are guaranteed an isomorphism $\pi(g_1) = g_2$. Therefore, $g_2 = \pi(g_1) \supseteq \pi(b') \cong b$, so $g_2 \in \texttt{comp}_\cong(B)$. □

Theorem 6. *Suppose ϕ is an isomorphism-invariant predicate on families. Suppose $N = \langle f, B\rangle$ is a node in the proof system $\textsc{symbreak}(\phi, \texttt{ext}_\cong, \texttt{comp}_\cong)$. Then $\phi_\exists(N^\#)$ if and only if ϕ_N is satisfiable.*

Proof. ($\implies$) Suppose $\phi_\exists(N^\#)$. Then there is a family $g' \in N^\# = \texttt{comp}_\cong(f) \setminus \texttt{comp}_\cong(B)$ that satisfies $\phi(g')$. Since $g' \in \texttt{comp}_\cong(f)$, we have $g' \supseteq f'$ for some $f' \in [f]$ (so there exists a permutation π_f that sends f' to f). Let $g = \pi_f(g') \in [g']$, so that $g = \pi_f(g') \supseteq \pi_f(f') = f$. Since $g' \notin \texttt{comp}_\cong(B)$, we have $g \notin \texttt{comp}_\cong(B)$ (contrapositive of Lemma 9). Finally, since ϕ is isomorphism-invariant, we must

have $\phi(g) = \phi(g') = true$. Therefore, $\phi_N(g) = true$, so ϕ_N is satisfiable.

($\Longleftarrow$) Suppose there is some $g \in \mathcal{F}$ satisfying $\phi_N(g)$. Then $\phi(g) = true$ and $f \subseteq g$ and $g \notin \texttt{comp}_{\cong}(B)$. Therefore, $g \in N^{\#}$ and $\phi(g) = true$, so $\phi_{\exists}(N^{\#})$. □

For a given node $N = \langle f, B \rangle$, it is possible to restrict ϕ_N even further with *symmetry constraints*. Choose any $p \in \texttt{ext}(f)$, and define

$$\phi_N^p(g) = \phi_N(g) \wedge (g \in \texttt{comp}_{\cong}(p) \implies p \subseteq g) \tag{20}$$

That is, we add the constraint that satisfying families must contain p, or else not contain anything isomorphic to p. The idea is, if we can satisfy ϕ with a family containing $p' \cong p$, we can instead satisfy ϕ (assuming isomorphism-invariance) with an isomorphic family that contains p itself. The added constraint forces us to "look for" solutions containing p "before" considering those containing isomorphic families, limiting the search space further. This constraint could be encoded in SAT as follows:

$$\psi_N^p = \psi_N \wedge \sigma_p \tag{21}$$

$$\sigma_p = \left(\bigwedge_{s \in p \setminus f} v_s \right) \vee \left(\bigwedge_{p' \in [p]} \bigvee_{t \in p'} \neg v_t \right) \tag{22}$$

$$= \bigwedge_{s \in p \setminus f} \bigwedge_{p' \in [p]} \left(v_s \vee \bigvee_{t \in p'} \neg v_t \right) \tag{23}$$

We evaluate the effect of adding these constraints empirically in Section VI-B; they did not improve solving performance in our experiments, so we do not impose them in the main results we report.

Theorem 7. *Suppose ϕ is an isomorphism-invariant predicate on families. Suppose $N = \langle f, B \rangle$ is a node in the proof system* SYMBREAK$(\phi, \texttt{ext}_{\cong}, \texttt{comp}_{\cong})$*, and fix some $p \in \texttt{ext}(f)$. Then $\phi_{\exists}(N^{\#})$ if and only if ϕ_N^p is satisfiable.*

V. Implementation

We implemented a decision procedure for SYMBREAK as a Rust library.[1] Users can implement the `NodeOracle` trait (interface) by providing methods for serializing and solving subproblems given input nodes (e.g. writing a DIMACS file and invoking a SAT solver on that file). The partitioning procedure uses a given `NodeOracle` to determine whether individual nodes can be pruned. The user needs to provide an implementation of the oracle, which determines the family-predicate ϕ being tested. Our prototype also supports a dynamic timeout system for the oracle based on the depth of the node.

Our partitioner, `symbreak`, branches on nodes in which the `NodeOracle` cannot determine the existence of a counterexample. We use `nauty` to check isomorphism of families in order to build the equivalence classes [24]. Nodes are branched on and tested for prunability in parallel, and the partitioner logs its progress as it executes. We implemented a verifier script in Python that inspects the logs and the set of terminal subproblems to ensure that all nodes are accounted for.

To apply **Branch** rule, one needs to choose an extension p to branch on. Our implementation heuristically chooses the extension with the smallest equivalence class. We experimented with alternative heuristics (e.g. the largest equivalence class), but did not find one that was consistently better than the others.

A. Case Study: Chvátal's Conjecture

As a test case for our partitioner, we used it to verify Chvátal's Conjecture for finite ground sets. We developed an implementation of `NodeOracle` that encodes the conjecture as a boolean satisfiability problem using RustSAT [25] and determines prunability with Kissat [26]. We developed a novel SAT encoding of the conjecture based on the ILP encoding in [10]. For each set $s \in 2^{[n]}$, we define propositional variables x_s and y_s. The truth of the former is interpreted as meaning that s is a member of a family X, the latter as meaning that s is a member of a family Y. We impose the following constraints (viewed as clauses) on these variables:

$$\{\neg y_s, \neg y_t\} \quad s, t \in 2^{[n]}, s \cap t = \emptyset \tag{24}$$

$$\{\neg y_t, x_s\} \quad s, t \in 2^{[n]}, s \subseteq t \tag{25}$$

$$\sum_{s \in 2^{[n]}} y_s > \sum_{s \in 2^{[n]}, e \in s} x_s \quad e \in [n] \tag{26}$$

$$\{\neg y_s\} \quad s \in 2^{[n]}, |s| \in \{1, 2\} \tag{27}$$

$$\{x_s\} \quad s \in 2^{[n]}, |s| = 1 \tag{28}$$

$$\{\neg x_s\} \cup \{y_t \mid t \in 2^{[n]}, t \supseteq s\} \quad s \in 2^{[n]}, |s| > 1 \tag{29}$$

$$\{y_t \mid t \in 2^{[n]}, t \cap s = \emptyset\} \cup \{\neg x_s, y_s\} \quad s \in 2^{[n]} \tag{30}$$

Constraints 29 and 30 are justified by the following theorem.

Theorem 8. *If there exists a counterexample to Chvátal's Conjecture, then there exists a counterexample (over the same ground set) in which the downset is minimal and the intersecting family is maximal.*

Proof. Suppose there is a counterexample to Chvátal's Conjecture, so there is a downset X containing an intersecting family larger than every contained star. There may be multiple such intersecting families, but we are free to let Y be a maximal one. Now we can let X' be the minimal downset generated by Y. Since $Y \subseteq X$, and X is a downset, we have $X' \subseteq X$. Clearly, then, the largest stars in X' are no larger than those in X. Thus X' and Y still constitute a counterexample to Chvátal's Conjecture. □

Note that one of the constraints is a cardinality constraint, which we encode using a totalizer [27]. Together, constraints 24–26 assert that Y is a counterexample to Chvátal's Conjecture: Y is an intersecting family which generates a downset X in which all the stars are smaller than Y. These constraints alone would generate (over the variables y_s) a family predicate equivalent to ϕ^{Chv}. The remaining constraints limit the search

[1] Source code available at https://github.com/jesselooney/symbreak.

TABLE I: Interpretation of each constraint in our SAT encoding.

Constraints	Interpretation
24	Y is an intersecting family.
25	X contains the downset generated by Y.
26	Y has greater cardinality than that of any star in X.
27	See constraints (7f) in [10].
28	See constraints (7g) in [10].
29	X is a *minimal* downset containing Y; X contains only sets with supersets in Y.
30	Y is a *maximal* intersecting family in X; any set in $X \setminus Y$ must be disjoint with something in Y.

space further, while ensuring that any counterexample they might exclude would have a counterpart they permit (the interpretation of each constraint is given in Table I). The resulting formula thus generates a predicate ϕ^{SAT} that is equisatisfiable to ϕ^{Chv}. Moreover, ϕ^{SAT} is isomorphism-invariant, since each constraint depends only on the structure of X and Y, not the particular names of the elements they contain. Hence, Theorem 6 justifies using the satisfiability of ϕ_N^{SAT} (implemented as ψ_N) to determine whether a node N can be pruned in the proof system SYMBREAK(ϕ^{SAT}, $\texttt{ext}_{\cong}$, $\texttt{comp}_{\cong}$). We did not add the symmetry constraints σ_p in the experiment since we found that their effect is mixed.

Additionally, we developed a separate implementation of `NodeOracle` that encodes the conjecture as an integer linear program using the PySCIPOpt [28] API for exact rational solving with SCIP [13]. We use the same constraints as in P_{red} of [10], with the exception of (7h) (see below), to specify the base problem. We require that the optimal objective value of the ILP is zero, thus generating a family predicate ϕ^{ILP} equisatisfiable to ϕ^{Chv} (as shown by [10]). Because we left out constraint (7h), the only constraint that cares about the particular names of the elements of X and Y, the generated predicate is isomorphism-invariant. Therefore, Theorem 6 ensures the correctness of pruning by adding the following constraints for a node $N = \langle f, B \rangle$:

$$y_s = 1 \quad s \in f \tag{31}$$

$$\sum_{s \in b'} y_s \leq |b'| - 1 \quad b' \in [b],\, b \in B \tag{32}$$

and checking whether the resulting ILP has optimal value zero. When partitioning with this oracle, we first branch on the extension $\{\{1, 2, 3, 4\}\}$ to partially recover the symmetry breaking that (7h) provides in the serial encoding (taking this family implies (7h)), and fall back to the default branching heuristic described above for all subsequent branches.

VI. Experiments

We evaluated the instantiations of our partitioner described in the previous section, comparing them to state-of-the-art techniques applied to the same problem of solving finite cases of Chvátal's Conjecture. In the SAT world, the dominant paradigm is Cube and Conquer—in particular, look-ahead-based partitioning. We compare our partitioner to the look-ahead solver `march_cu` [29].

For ILP, we instead run our partitioner in solving mode and compare against a state-of-the-art exact solver, SCIP. In doing this, we obtain the first fully verified solution of the case $n = 8$. We detail these experiments in the following subsections. Unless noted otherwise, all computations were run on a computing cluster consisting of Dell PowerEdge R6525 rack servers with 2.6-GHz AMD CPUs.

A. SAT Partitioning: Comparison with Look-Ahead

We compared our partitioner with `march_cu`, the state-of-the-art partitioner used in Cube and Conquer, on a SAT instance of the Chvátal problem for $n = 7$. We ran two configurations of our partitioner, each with a different oracle timeout (1s and 3s), both with 16 threads and a maximum depth of 20. We ran three configurations of `march_cu`, running the default configuration as well forcing it to partition to depth 15 or 20 with the `-d` flag. Table II reports the number of subproblems generated by each partitioner and the time each took to generate. Each subproblem was then solved using Kissat 4.0.4 with the `--unsat` flag with a one-hour time limit. For each partitioner, we report in Table II the total CPU time taken to solve all subproblems that finished within the time limit.

Note that we spent significant effort trying different SAT encodings of Chvátal's Conjecture by tuning the cardinality encodings and adding or removing known encodings of various lemmas that strengthen our assumptions for the conjecture. We also attempted to tune Kissat as in [30] and to use a solver that was more aware of cardinality constraints [31]. None of these methods helped significantly, motivating our current approach.

Figures 1a–1e show the distribution of solve time over the subproblems generated by each partitioner configuration. The distributions share the same right-skew, with more easy subproblems and a few harder ones, though those from our partitioner have a smaller range of solve times. Only the depth 20 configuration of `march_cu` approaches our narrow range, but does so at the cost of nearly a thousand times more instances. The data suggest that `march_cu` is decomposing the original problem less effectively than `symbreak`, carving off many more trivial subproblems in the process. We expect `symbreak`'s advantage is due to its better internal representation of the problem structure and that it is guaranteed to strictly reduce the solution space at each branch (Theorem 5).

B. Solving $n = 8$ with ILP and Dynamic Partitioning

We found that SAT solvers tend to be significantly slower than ILP solvers for verifying the same value of n. Therefore, to tackle Chvátal's Conjecture for larger n, we used an existing ILP encoding and the exact solving mode of SCIP 10.0.3. Instead of pre-determining the depth to which we perform partitioning, we employ a dynamic divide-and-conquer scheme. Each configuration is defined as $\{t, r\}$: the initial per-node

TABLE II: Partition sizes, generation time, and total time to solve resulting instances, for `symbreak` (oracle timeouts 1s and 3s) and `march_cu` (default config and depths 15 and 20) on $n = 7$. Generation wall times are given only for the parallel partitioners.

partitioner	partition size	generation time (s) wall	generation time (s) cpu	total solve time (s) cpu	max solve time (s) cpu
`symbreak` (1s)	102	411.1	6138.9	622.3	34.6
`symbreak` (3s)	46	120.8	2621.7	396.3	47.5
`march_cu`	1570	—	9.0	8650.0*	TO
`march_cu` (d15)	2821	—	21.7	39 127.3	2102.6
`march_cu` (d20)	44 065	—	356.9	56 843.9	276.6

* Does not include four instances that timed out after one hour.

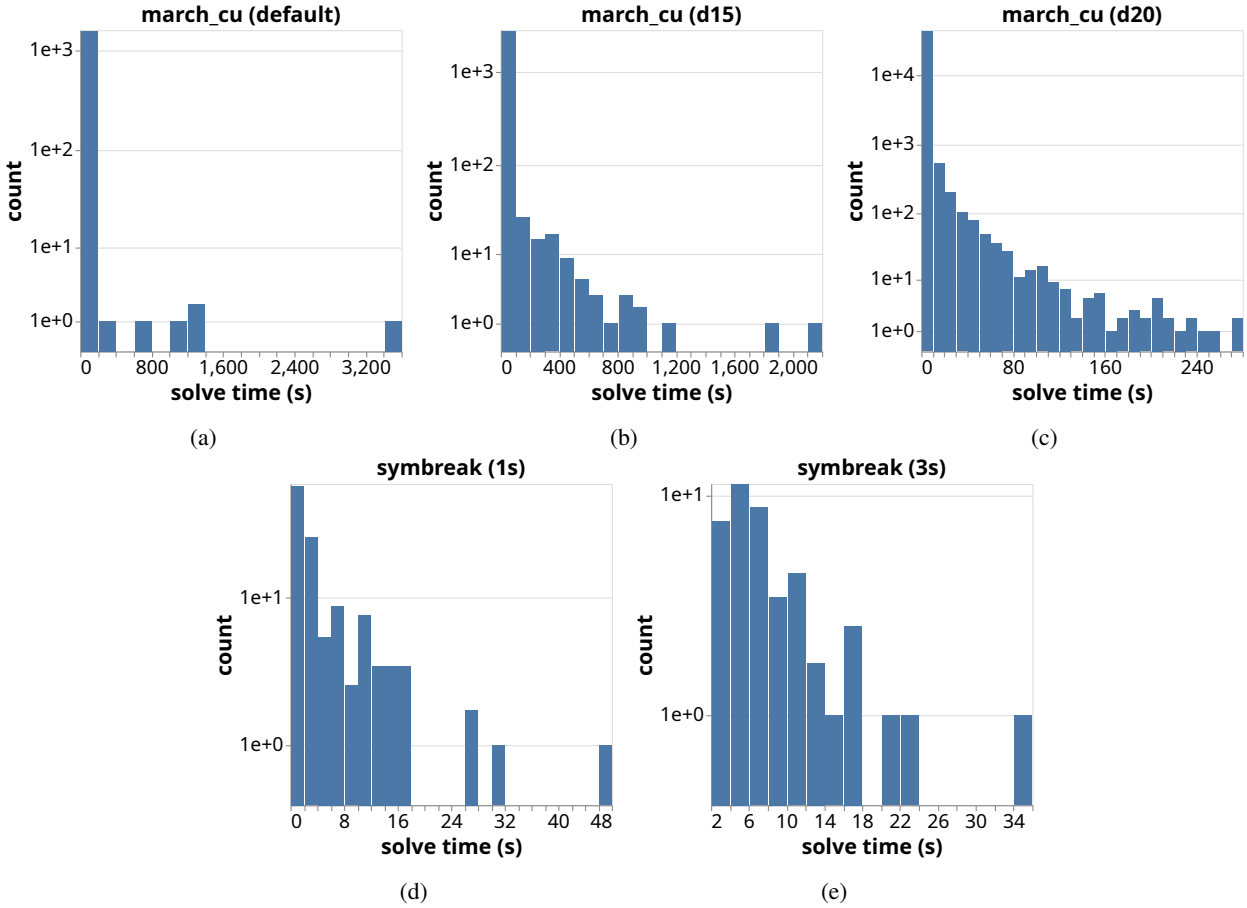

Fig. 1: Distribution of solve times for subproblems of $n = 7$ generated by `march_cu` (a) default config (excluding four timed-out instances), (b) depth 15, (c) depth 20; and SAT-based `symbreak` with oracle timeouts (d) 1s and (e) 3s. The y axis is log-scaled in each case.

oracle timeout of t seconds is used for all nodes up to a switch depth s, after which the timeout is multiplied by r for each subsequent depth. We ran each configuration using 52 cores.

Except where noted otherwise, we ran SCIP with conflict analysis disabled (`conflict/enable = false`) and Gomory cuts disabled (`separating/gomory/freq = -1`). We found that disabling these features is significantly better in both the sequential and the divide-and-conquer setting.

Table III reports, for each configuration with the tuned SCIP settings and switch depth $s = 10$, the wall-clock time, the total time (the sum of all per-node solving and branching times), and the number of leaf subproblems in the final partition. With these settings, serial exact SCIP verifies $n = 7$ in 24 seconds; in the $\{60, 2\}$ and $\{60, 5\}$ configurations, the root problem for $n = 7$ is solved within the initial timeout, so no partitioning takes place and the wall time is simply the root solve time. The root problem takes longer than the serial run because the serial encoding includes constraint (7h) of [10], which asserts that

TABLE III: ILP dynamic splitting run times on 52 cores with tuned SCIP settings. Each configuration $\{t, r\}$ uses an initial per-node timeout of t seconds up to depth 10, after which the timeout is multiplied by r for each subsequent depth. Wall and total times are in H:M:S format; leaves is the number of leaf subproblems in the final partition.

	$n = 7$			$n = 8$		
Config.	Wall	Total	Leaves	Wall	Total	Leaves
Serial	0:00:24	—	—	72:26:02	—	—
{5, 2}	0:01:00	0:01:19	11	2:33:11	27:26:58	197
{5, 5}	0:01:01	0:01:19	11	2:15:59	6:45:03	76
{60, 2}	0:00:56	0:00:56	1	4:11:04	12:40:15	53
{60, 5}	0:00:56	0:00:56	1	1:53:49	5:08:38	33

TABLE IV: ILP dynamic splitting run times on 52 cores with default SCIP settings. The $n = 7$ runs use switch depth $s = 10$ and the $n = 8$ runs use $s = 16$ (the initial timeout t applies up to depth s and is multiplied by r for each subsequent depth). Wall and total times are in H:M:S format; leaves is the number of leaf subproblems in the final partition. The serial $n = 8$ run did not terminate within seven days.

	$n = 7$			$n = 8$		
Config.	Wall	Total	Leaves	Wall	Total	Leaves
Serial	0:05:24	—	—	>7 days	—	—
{5, 7}	0:03:03	0:08:20	35	19:15:05	340:26:21	1679
{10, 6}	0:04:23	0:08:53	26	19:28:20	321:43:16	1214
{20, 5}	0:05:12	0:10:36	21	19:21:27	322:30:35	870
{60, 4}	0:06:03	0:07:12	6	20:24:28	303:47:54	570

the counterexample downset must contain the set $\{1, 2, 3, 4\}$; as discussed above, our divide-and-conquer oracle omits this constraint to keep the encoding isomorphism-invariant.

For $n = 8$, the serial solver took just over three days, whereas every dynamic divide-and-conquer run finished within about four and a half hours; the fastest configuration, $\{60, 5\}$, finished in under two hours, a roughly $38\times$ speedup. Notably, the total time of each configuration is also well below the serial solve time, so the partitioning reduces not only wall-clock time but also the overall work. This suggests that our proposed techniques can indeed effectively decompose the problem and parallelize the solving process. The configuration with a large initial timeout and an aggressive increase rate, $\{60, 5\}$, performed best on $n = 8$, whereas the same initial timeout with a milder rate, $\{60, 2\}$, performed worst; we believe an aggressive rate reduces duplicated work in the case where a child is only slightly simpler than its parent. Figure 2 shows a breakdown of the solver runtime on the leaf subproblems of the $\{5, 2\}$ configuration. The distribution is heavily right-skewed: the median leaf solve time is about 14 seconds, but the hardest leaf required more than an hour. This suggests that a more intelligent dynamic re-splitting scheme could better leverage parallel resources and lead to further performance gain.

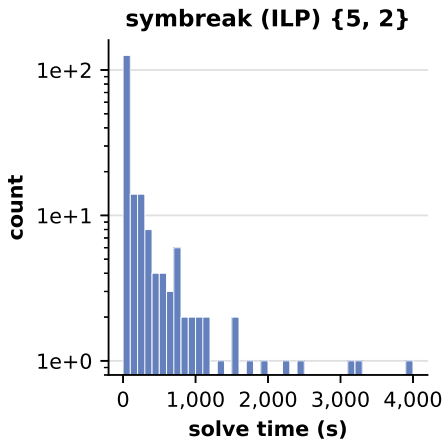

Fig. 2: Distribution of solve times for leaf subproblems in the $\{5, 2\}$ configuration for dynamic splitting with ILP and $n = 8$.

TABLE V: Effect of symmetry-breaking constraints on $n = 8$ with tuned SCIP settings: wall time, total time, and number of leaf subproblems for each configuration of Table III, without and with the ILP analogue of the symmetry constraints σ_p imposed on each subproblem.

	Wall		Total		Leaves	
Config.	base	+sym	base	+sym	base	+sym
{5, 2}	2:33:11	2:42:52	27:26:58	33:03:47	197	209
{5, 5}	2:15:59	2:27:04	6:45:03	8:07:36	76	85
{60, 2}	4:11:04	4:22:47	12:40:15	18:43:41	53	61
{60, 5}	1:53:49	1:47:38	5:08:38	6:04:15	33	35

Table V reports runs of the same configurations on $n = 8$ in which each subproblem additionally imposes the ILP analogue of the symmetry constraints σ_p. Adding these constraints does not boost performance: wall times remain similar (three configurations run slightly slower and one slightly faster), while the total time and the number of leaf subproblems increase in every configuration.

Table IV reports the results of the same experiments run with SCIP's default settings, using switch depth $s = 10$ for $n = 7$ and $s = 16$ for $n = 8$. The contrast with the serial solver is starker in this setting: serial exact SCIP verifies $n = 7$ in 5 minutes and 24 seconds, and on $n = 8$ it timed out after seven days, while every dynamic configuration verified $n = 8$ in under 21 hours. Dynamic partitioning thus renders the problem tractable even without tuning the solver settings.

C. *Proving* $n = 8$

Beyond solving the case $n = 8$, we also produced and checked proof certificates for it. To do so, we re-ran the $\{60, 5\}$ configuration with the tuned SCIP settings on the same cluster, this time with SCIP's proof-production options turned on: as before, each node was pruned if it was solved within its timeout and branched otherwise. The run partitioned

the problem into 38 leaf subproblems, and we successfully extracted a proof certificate in the VIPR format [32] for each of them. Producing certificates roughly doubles the cost of solving: the run took 4:13:42 of wall time and 11:27:21 of total solving and branching time, compared to 1:53:49 and 5:08:38 for the same configuration without proof production. The overhead also slightly alters the shape of the search: the run produced 38 leaf subproblems rather than the 33 of the certificate-free run, as slower oracle calls push a few additional nodes past their timeouts and cause them to be branched. We then ran the official proof checker [32] on those certificates and successfully validated all of them; checking took 1970 seconds of CPU time in total, and the longest proof-checking time for a single certificate was 356 seconds. The total size of the certificates is 14 GB. The proof certificates and the proof-checking logs are publicly available on Zenodo [33]. We have also checked that the leaf nodes form a partition of the original problem, so the validated certificates together constitute a machine-checked proof of Chvátal's Conjecture for $n = 8$. In contrast, prior work solved $n = 8$ with serial exact SCIP without generating a proof certificate [14] and noted that a certificate of such a sequential run would exceed 1 TB, which is infeasible for the proof checker. As far as we know, this is the first time this case of the conjecture has been formally verified.

We note that the certificates cover only the solver's reasoning on the individual subproblems. The partitioning steps themselves, whose correctness is guaranteed by Theorems 1 and 2, are not accompanied by certificates; conceptually, the partitioner carries out part of the deductive work that a monolithic run would have to certify, which also helps explain why our certificates are far smaller than the estimated 1 TB. Unlike the numerical reasoning that VIPR certifies, this part of the reasoning is discrete and involves no floating-point arithmetic; still, producing machine-checkable certificates for the partitioning itself would make the proof verifiable end to end, and is an appealing direction for future work. It might also be desirable to construct a single certificate of the original problem from certificates of sub-problems [34].

VII. Conclusion

We presented a novel partitioning strategy for extremal combinatorics that breaks on isomorphic families of sets. Our method of branching on an extension of a partial family of sets and banning isomorphic equivalents both showed improvements over the look-ahead-based partitioning for SAT and successfully parallelized ILP solving. A dynamic re-splitting scheme that combines our partitioning strategy with an exact MILP solver exactly solved Chvátal's Conjecture for $n = 8$ in a reasonable time, and produced the first verified proof of this case of the statement.

In the future, it would be interesting to test our methodology on other problems in extremal set theory, such as the Union-closed Sets Conjecture [35]. Our method is already general enough to handle these cases, so extensions would merely need to formally describe an appropriate predicate in a language that admits automatic deduction in order to apply our approach. Using our solver-agnosticism, such explorations could also investigate whether ILP or SAT methods are superior on the chosen problems. A related direction is to investigate why ILP is more effective than SAT in the case of Chvátal's Conjecture. Our implementation could also be pushed further by better utilizing available cores with more intelligent dynamic re-splitting. Considering that solver configurations significantly impacted runtime in our experiments, performing parameter tuning [36], [37], [30] may further boost performance. Finally, producing machine-checkable certificates for the partitioning steps themselves would make the resulting proofs verifiable end to end.

Acknowledgment

This work was performed in part using high-performance computing equipment at Amherst College obtained under National Science Foundation Grant No. 2117377. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation. Wu's research is supported by a gift from The VMware University Research Fund.

We are grateful to the anonymous IJCAR and FMCAD reviewers, who together substantially improved the quality of this contribution with their thoughtful and constructive feedback.

References

[1] J. Brakensiek, M. Heule, J. Mackey, and D. Narváez, "The Resolution of Keller's Conjecture," *Journal of Automated Reasoning*, vol. 66, no. 3, pp. 277–300, Aug. 2022. [Online]. Available: https://doi.org/10.1007/s10817-022-09623-5

[2] T. Hales, M. Adams, G. Bauer, T. D. Dang, J. Harrison, L. T. Hoang, C. Kaliszyk, V. Magron, S. McLaughlin, T. T. Niguyen, and et al., "A formal proof of the kepler conjecture," *Forum of Mathematics, Pi*, vol. 5, p. e2, 2017.

[3] M. Heule, "Schur Number Five," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, Apr. 2018. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/12209

[4] M. J. H. Heule, O. Kullmann, and V. W. Marek, "Solving and Verifying the Boolean Pythagorean Triples Problem via Cube-and-Conquer," in *Theory and Applications of Satisfiability Testing – SAT 2016*, N. Creignou and D. Le Berre, Eds. Cham: Springer International Publishing, 2016, pp. 228–245.

[5] F. Kenter and D. Skipper, "Integer-programming bounds on pebbling numbers of cartesian-product graphs," in *Combinatorial Optimization and Applications*, D. Kim, R. N. Uma, and A. Zelikovsky, Eds. Cham: Springer International Publishing, 2018, pp. 681–695.

[6] G. Lancia, E. Pippia, and F. Rinaldi, "Using integer programming to search for counterexamples: A case study," in *Mathematical Optimization Theory and Operations Research*, A. Kononov, M. Khachay, V. A. Kalyagin, and P. Pardalos, Eds. Cham: Springer International Publishing, 2020, pp. 69–84.

[7] O. Parczyk, S. Pokutta, C. Spiegel, and T. Szabó, "Fully computer-assisted proofs in extremal combinatorics," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 10, pp. 12 482–12 490, Jun. 2023. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/26470

[8] J. Pulaj, "Cutting planes for families implying Frankl's conjecture," *Mathematics of Computation*, vol. 89, no. 322, pp. 829–857, Mar. 2020. [Online]. Available: https://www.ams.org/mcom/2020-89-322/S0025-5718-2019-03461-0/

[9] B. Subercaseaux, J. Mackey, M. J. H. Heule, and R. Martins, "Automated Mathematical Discovery and Verification: Minimizing Pentagons in the Plane," in *Intelligent Computer Mathematics*, A. Kohlhase and L. Kovács, Eds. Cham: Springer Nature Switzerland, 2024, pp. 21–41.

[10] L. Eifler, A. Gleixner, and J. Pulaj, "A Safe Computational Framework for Integer Programming Applied to Chvátal's Conjecture," *ACM Transactions on Mathematical Software*, vol. 48, no. 2, pp. 1–12, Jun. 2022. [Online]. Available: https://dl.acm.org/doi/10.1145/3485630

[11] M. J. H. Heule, O. Kullmann, S. Wieringa, and A. Biere, "Cube and Conquer: Guiding CDCL SAT Solvers by Lookaheads," in *Hardware and Software: Verification and Testing*, D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, K. Eder, J. Lourenço, and O. Shehory, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, vol. 7261, pp. 50–65, series Title: Lecture Notes in Computer Science. [Online]. Available: http://link.springer.com/10.1007/978-3-642-34188-5_8

[12] V. Chvátal, "Intersecting families of edges in hypergraphs having the hereditary property," in *Hypergraph Seminar*, C. Berge and D. Ray-Chaudhuri, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1974, pp. 61–66.

[13] C. Hojny, M. Besançon, K. Bestuzheva, S. Borst, J. Dionísio, J. Ehls, L. Eifler, M. Ghannam, A. Gleixner, A. Göß, A. Hoen, J. von Holly-Ponientzietz, R. van der Hulst, D. Kamp, T. Koch, K. Kofler, J. Lentz, M. Lübbecke, S. J. Maher, P. M. Meinhold, G. Mexi, T. Mohr, E. Mühmer, K. K. Patel, M. E. Pfetsch, S. Pokutta, C. R. Groba, F. Serrano, Y. Shinano, M. Turner, S. Vigerske, M. Walter, D. Weninger, and L. Xu, "The scip optimization suite 10.0," 2025. [Online]. Available: https://arxiv.org/abs/2511.18580

[14] L. Eifler, "Algorithms and certificates for exact Mixed Integer Programming," 2025. [Online]. Available: https://depositonce.tu-berlin.de/handle/11303/25121

[15] E. Friedgut, J. Kahn, G. Kalai, and N. Keller, "Chvátal's conjecture and correlation inequalities," 2016. [Online]. Available: https://arxiv.org/abs/1608.08954

[16] E. Czabarka, G. Hurlbert, and V. Kamat, "Chvátal's conjecture for downsets of small rank," 2017. [Online]. Available: https://arxiv.org/abs/1703.00494

[17] M. Heisinger, M. Fleury, and A. Biere, "Distributed cube and conquer with paracooba," in *Theory and Applications of Satisfiability Testing – SAT 2020*, L. Pulina and M. Seidl, Eds. Cham: Springer International Publishing, 2020, pp. 114–122.

[18] L. Le Frioux, S. Baarir, J. Sopena, and F. Kordon, "Painless: A framework for parallel sat solving," in *Theory and Applications of Satisfiability Testing – SAT 2017*, S. Gaspers and T. Walsh, Eds. Cham: Springer International Publishing, 2017, pp. 233–250.

[19] T. Ralphs, Y. Shinano, T. Berthold, and T. Koch, "Parallel solvers for mixed integer linear optimization," in *Handbook of Parallel Constraint Reasoning*, Y. Hamadi and L. Sais, Eds. Cham: Springer International Publishing, 2018, pp. 283–336. [Online]. Available: https://doi.org/10.1007/978-3-319-63516-3_8

[20] H. Metin, S. Baarir, M. Colange, and F. Kordon, "Cdclsym: Introducing effective symmetry breaking in sat solving," in *Tools and Algorithms for the Construction and Analysis of Systems*, D. Beyer and M. Huisman, Eds. Cham: Springer International Publishing, 2018, pp. 99–114.

[21] M. Kirchweger and S. Szeider, "Sat modulo symmetries for graph generation and enumeration," *ACM Trans. Comput. Logic*, vol. 25, no. 3, Jul. 2024. [Online]. Available: https://doi.org/10.1145/3670405

[22] M. Anders, S. Brenner, and G. Rattan, "satsuma: Structure-based symmetry breaking in sat," 2024. [Online]. Available: https://arxiv.org/abs/2406.13557

[23] F. Margot, "Pruning by isomorphism in branch-and-cut," *Mathematical Programming*, vol. 94, pp. 71–90, Dec. 2002. [Online]. Available: https://doi.org/10.1007/s10107-002-0358-2

[24] B. McKay and A. Piperno, "Practical graph isomorphism, ii," *Journal of Symbolic Computation*, vol. 60, pp. 94–112, 2014. [Online]. Available: https://doi.org/10.1016/j.jsc.2013.09.003

[25] C. Jabs, "RustSAT: A library for SAT solving in rust," in *28th International Conference on Theory and Applications of Satisfiability Testing (SAT 2025)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), J. Berg and J. Nordström, Eds., vol. 341. Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 2025, pp. 15:1–15:13.

[26] A. Biere, T. Faller, K. Fazekas, M. Fleury, N. Froleyks, and F. Pollitt, "CaDiCaL, Gimsatul, IsaSAT and Kissat Entering the SAT Competition 2024," in *Proc. of SAT Competition 2024 – Solver, Benchmark and Proof Checker Descriptions*, ser. Department of Computer Science Report Series B, M. Heule, M. Iser, M. Järvisalo, and M. Suda, Eds., vol. B-2024-1. University of Helsinki, 2024, pp. 8–10.

[27] O. Bailleux and Y. Boufkhad, "Efficient cnf encoding of boolean cardinality constraints," in *Principles and Practice of Constraint Programming – CP 2003*, F. Rossi, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 108–122.

[28] S. Maher, M. Miltenberger, J. P. Pedroso, D. Rehfeldt, R. Schwarz, and F. Serrano, "PySCIPOpt: Mathematical programming in python with the SCIP optimization suite," in *Mathematical Software – ICMS 2016*. Springer International Publishing, 2016, pp. 301–307.

[29] M. Heule, "marijnheule/CnC," Jan. 2026, original-date: 2018-05-29T12:17:39Z. [Online]. Available: https://github.com/marijnheule/CnC

[30] H. Wu, C. Barrett, and N. Narodytska, "Cubing for tuning," 2025. [Online]. Available: https://arxiv.org/abs/2504.19039

[31] J. Reeves, M. Heule, and R. Bryant, "From clauses to klauses," 2024. [Online]. Available: https://www.cs.cmu.edu/~jereeves/research/cav24-paper.pdf

[32] K. K. H. Cheung, A. Gleixner, and D. E. Steffy, "Verifying Integer Programming Results," in *Integer Programming and Combinatorial Optimization (IPCO 2017)*, ser. Lecture Notes in Computer Science, vol. 10328. Cham: Springer, 2017, pp. 148–160.

[33] J. Looney, A. Klingler, J. McDonald, G. Wu, J. Pulaj, and H. Wu, "Cube-and-conquer vipr proof of chvátal's conjecture for ground sets of size 8," Jul. 2026. [Online]. Available: https://doi.org/10.5281/zenodo.21603128

[34] A. Nair, S. Chattopadhyay, H. Wu, A. Ozdemir, and C. Barrett, "Proof-stitch: Proof combination for divide-and-conquer sat solvers," in *2022 Formal Methods in Computer-Aided Design (FMCAD)*. IEEE, 2022, pp. 84–88.

[35] H. Bruhn and O. Schaudt, "The Journey of the Union-Closed Sets Conjecture," *Graphs and Combinatorics*, vol. 31, no. 6, pp. 2043–2074, Nov. 2015. [Online]. Available: https://doi.org/10.1007/s00373-014-1515-0

[36] H. Wu, C. Hahn, F. Lonsing, M. Mann, R. Ramanujan, and C. Barrett, "Lightweight online learning for sets of related problems in automated reasoning," in *2023 Formal Methods in Computer-Aided Design (FMCAD)*. IEEE, 2023, pp. 1–11.

[37] A. Wilson, N. Narodytska, C. Barrett, and H. Wu, "Per-instance subproblem generation for strategy selection in smt," in *Formal Methods in Computer Aided Design (FMCAD)*. TU Wien Academic Press, 2025, pp. 94–103.