Happy Ending: An Empty Hexagon in Every Set of 30 Points

Marijn J. H. Heule[1,2(✉)] and Manfred Scheucher[3]

[1] Carnegie Mellon University, Pittsburgh, USA
marijn@cmu.edu
[2] Amazon Scholar, Seattle, USA
[3] Institute of Mathematics, Technische Universität Berlin, Berlin, Germany
scheucher@math.tu-berlin.de

Abstract. Satisfiability solving has been used to tackle a range of long-standing open math problems in recent years. We add another success by solving a geometry problem that originated a century ago. In the 1930s, Esther Klein's exploration of unavoidable shapes in planar point sets in general position showed that every set of five points includes four points in convex position. For a long time, it was open if an empty hexagon, i.e., six points in convex position without a point inside, can be avoided. In 2006, Gerken and Nicolás independently proved that the answer is no. We establish the exact bound: Every 30-point set in the plane in general position contains an empty hexagon. Our key contributions include an effective, compact encoding and a search-space partitioning strategy enabling linear-time speedups even when using thousands of cores.

Keywords: Erdős–Szekeres problem · empty hexagon theorem · planar point set · cube-and-conquer · proof of unsatisfiability

1 Introduction

In 1932, Esther Klein showed that every set of five points in the plane *in general position* (i.e., no three points on a common line) has a subset of four points in convex position. Shortly after, Erdős and Szekeres [8] generalized this result by showing that, for every integer k, there exists a smallest integer $g(k)$ such that every set of $g(k)$ points in the plane in general position contains a k-*gon* (i.e., a subset of k points that form the vertices of a convex polygon). As the research led to the marriage of Szekeres and Klein, Erdős named it the *happy ending problem*. Erdős and Szekeres constructed witnesses of $g(k) > 2^{k-2}$ [9], which they conjectured to be maximal. The best upper bound is $g(k) \leq 2^{k+o(k)}$ [20,30].

Determining the value $g(5) = 9$ requires a more involved case distinction compared to $g(4) = 5$ [23]. It took until 2006 to determine that $g(6) = 17$ via an exhaustive computer search by Szekeres and Peters [31] using 1500 CPU hours. Marić [25] and Scheucher [28] independently verified $g(6) = 17$ using satisfiability (SAT) solving in a few CPU hours. This was later reduced to 10 CPU minutes [29]. The approach presented in this paper computes it in 8.53 CPU seconds, showing the effectiveness of SAT compared to the original method.

B. Finkbeiner and L. Kovács (Eds.): TACAS 2024, LNCS 14570, pp. 61–80, 2024.
https://doi.org/10.1007/978-3-031-57246-3_5

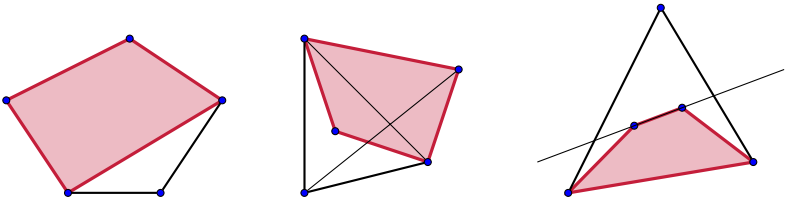


Fig. 1. An illustration for the proof of $h(4) = 5$: The three possibilities of how five points can be placed. Each possibility implies a 4-hole.

Erdős also asked whether every sufficiently large point set contains a *k-hole*: a k-gon without a point inside. We denote by $h(k)$ the smallest integer—if it exists—such that every set of $h(k)$ points in general position in the plane contains a k-hole. Both $h(3) = 3$ and $h(4) = 5$ are easy to compute (see Fig. 1 for an illustration) and coincide with the original setting. Yet the answer can differ a lot, as Horton [21] constructed arbitrarily large point sets without 7-holes.

While Harborth [14] showed in 1978 that $h(5) = 10$, the existence of 6-holes remained open until the late 2000s, when Gerken [12][4] and Nicolás [26] independently proved that $h(6)$ is finite. Gerken proved that every 9-gon yields a 6-hole, thereby showing that $h(6) \leq g(9) \leq 1717$ [33]. The best-known lower bound $h(6) \geq 30$ is witnessed by a set of 29 points without 6-holes which was found by Overmars [27] using a local search approach.

We close the gap between the upper and lower bound and ultimately answer Erdős' question by proving that every set of 30 points yields a 6-hole.

Theorem 1. $h(6) = 30$.

Our result is actually stronger and shows that the bounds for 6-holes in point sets coincide with the bounds for 6-holes in *counterclockwise systems* [24]. This represents another success of solving long-standing open problems in mathematics using SAT, similar to results on Schur number five [16] and Keller's conjecture [4].

We also investigate the combination of 6-holes and 7-gons and show

Theorem 2. *Every set of 24 points in the plane in general position contains a* 6*-hole or a* 7*-gon.*

We achieve these results through the following contributions:

- We develop a compact and effective SAT encoding for k-gon and k-hole problems that uses $O(n^4)$ clauses, while existing encodings use $O(n^k)$ clauses.
- We construct a partitioning of k-gon and k-hole problems that allows us to solve them with linear-time speedups even when using thousands of cores.
- We present a novel method of validating SAT-solving results that checks the proof while solving the problem using substantially less overhead.
- We verify most of the presented results using clausal proof checking.

[4] Gerken's groundbreaking work was awarded the Richard-Rado prize by the German Mathematical Society in 2008.

2 Preliminaries

The SAT problem. The satisfiability problem (SAT) asks whether a Boolean formula can be satisfied by some assignment of truth values to its variables. The Handbook of Satisfiability [2] provides an overview. We consider formulas in *conjunctive normal form* (CNF), which is the default input of SAT solvers. As such, a formula Γ is a conjunction (logical "AND") of *clauses*. A clause is a disjunction (logical "OR") of literals, where a literal is a Boolean variable or its negation. We sometimes write (sets of) clauses using other logical connectives.

If a formula Γ is found to be satisfiable, modern SAT solvers commonly output a truth assignment of the variables. Additionally, if a formula turns out to be unsatisfiable, sequential SAT solvers produce an independently-checkable proof that there exists no assignment that satisfies the formula.

Verification. The most commonly-used proofs for SAT problems are expressed in the DRAT clausal proof system [15]. A DRAT proof of unsatisfiability is a list of clause addition and clause deletion steps. Formally, a clausal proof is a list of pairs $\langle s_1, C_1\rangle, \ldots, \langle s_m, C_m\rangle$, where for each $i \in \{1, \ldots, m\}$, $s_i \in \{\mathsf{a}, \mathsf{d}\}$ and C_i is a clause. If $s_i = \mathsf{a}$, the pair is called an *addition*, and if $s_i = \mathsf{d}$, it is called a *deletion*. For a given input formula Γ_0, a clausal proof gives rise to a set of *accumulated formulas* Γ_i ($i \in \{1, \ldots, m\}$) as follows:

$$\Gamma_i = \begin{cases} \Gamma_{i-1} \cup \{C_i\} & \text{if } \mathsf{s}_i = \mathsf{a} \\ \Gamma_{i-1} \setminus \{C_i\} & \text{if } \mathsf{s}_i = \mathsf{d} \end{cases}$$

Each clause addition must preserve satisfiability, which is usually guaranteed by requiring the added clauses to fulfill some efficiently decidable syntactic criterion. Deletions help to speed up proof checking by keeping the accumulated formula small. A valid proof of unsatisfiability must add the empty clause.

Cube And Conquer. The cube-and-conquer approach [18] aims to *split* a SAT instance Γ into multiple instances $\Gamma_1, \ldots, \Gamma_m$ in such a way that Γ is satisfiable if and only if at least one of the instances Γ_i is satisfiable, thus allowing work on the different instances Γ_i in parallel. A *cube* is a conjunction of literals. Let $\psi = (c_1 \vee \cdots \vee c_m)$ be a disjunction of cubes. When ψ is a tautology, we have

$$\Gamma \iff \Gamma \wedge \psi \iff \bigvee_{i=1}^{m} (\Gamma \wedge c_i) \iff \bigvee_{i=1}^{m} \Gamma_i,$$

where the different $\Gamma_i \coloneqq (\Gamma \wedge c_i)$ are the instances resulting from the split.

Intuitively, each cube c_i represents a *case*, i.e., an assumption about a satisfying assignment to Γ, and soundness comes from ψ being a tautology, which means that the split into cases is exhaustive. If the split is well designed, then each Γ_i is a particular case that is substantially easier to solve than Γ, and thus solving them all in parallel can give significant speed-ups, especially considering the sequential nature of CDCL at the core of most solvers.

However, the quality of the split (ψ) has an enormous impact on the effectiveness of the approach. A key challenge is figuring out a high-quality split.

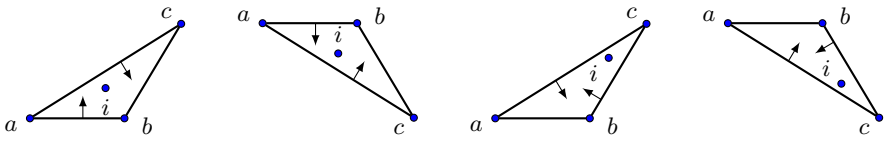


Fig. 2. The four ways a point p_i can be inside triangle $\{p_a, p_b, p_c\}$ based on whether $i < b$ (left two images) and whether p_c is above the line $p_a p_b$ (first and third image).

3 Trusted Encoding

To obtain an upper-bound result using a SAT-based approach, we need to show that every set of n points contains a k-hole. We will do this by constructing a formula based on n points that asks whether a k-hole can be avoided. If this formula is unsatisfiable, then we obtain the bound $h(k) \leq n$. Instead of reasoning directly whether an empty k-gon can be avoided, we ask whether every k points contain at least one triangle with a point inside. The latter implies the former.

We only need to know for each triple of points whether it is empty. Throughout the paper, we assume that points are sorted with strictly increasing x-coordinates. This gives us only four options for a point p_i to be inside the triangle formed by points p_a, p_b, p_c, see Fig. 2. For example, the left image shows that p_i is inside if $a < i < b$, p_c and p_i are above the line $\overline{p_a p_b}$, and p_i is below the line $\overline{p_a p_c}$. So we need some machinery to express that points are above or below certain lines. That is what the encoding will provide. For readability, we sometimes identify points by their indices, that is, we refer to p_a by its index a.

We first present what we call the *trusted encoding* to determine whether a 6-hole can be avoided. The encoding needs to be trusted in the sense that we do not provide a mechanically verified proof of its correctness. Building upon existing work [28], our primary focus is on 6-holes, which constitute our main result. The encoding of 6-gons and 7-gons is similar and more simple. During an initial study, the estimated runtime for showing $h(6) \leq 30$ using this encoding and off-the-shelf partitioning was roughly 1000 CPU years. The optimizations in Sections 4 and 5 reduce the computational costs to about 2 CPU years.

3.1 Orientation Variables

We formulate the problem in such a way that all reasoning is based solely on the relative positions of points. Thus, we do not encode coordinates but only orientations of point triples. For a point set $S = \{p_1, \ldots, p_n\}$ with $p_i = (x_i, y_i)$, the triple (p_a, p_b, p_c) with $a < b < c$ is *positively oriented* (resp. *negatively oriented*) if p_c lies above (resp. below) the line $\overline{p_a p_b}$ through p_a and p_b. The notion of positive orientation corresponds to Knuth's *counterclockwise relation* [24]. Fig. 3 illustrates a positively-oriented triple (p_a, p_b, p_c) and a negatively-oriented triple (p_a, p_b, p_d).

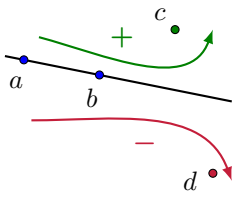


Fig. 3. An illustration of triple orientations.

To search for point sets without k-gons and k-holes, we introduce a Boolean *orientation variable* $\mathsf{o}_{a,b,c}$ for each triple (p_a, p_b, p_c) with $a < b < c$. Intuitively, $\mathsf{o}_{a,b,c}$ is supposed to be true if the triple is positively oriented. Since we assume general position, no three points lie on a common line, so $\mathsf{o}_{a,b,c}$ being false means that the triple is negatively oriented.

3.2 Containment Variables, 3-Hole Variables, and Constraints

Using orientation variables, we can now express what it means for a triangle to be empty. We define *containment variables* $\mathsf{c}_{i;a,b,c}$ to encode whether point p_i lies inside the triangle spanned by $\{p_a, p_b, p_c\}$. Since the points have increasing x-coordinates, containment is only possible if $a < i < c$. We use two kinds of definitions, depending on whether i is smaller or larger than b (see Fig. 2). The first definition is for the case $a < i < b$. Note that if $\mathsf{o}_{a,b,c}$ is true, we only need to know whether i is above the line $\overline{p_a p_b}$ and below the line $\overline{p_a p_c}$. Earlier work [28] used an extended definition that included the redundant variable $\mathsf{o}_{i,b,c}$. Avoiding this variable makes the definition more compact (six instead of eight clauses) and the resulting formula is easier to solve.

$$\mathsf{c}_{i;a,b,c} \leftrightarrow \Big(\big(\mathsf{o}_{a,b,c} \rightarrow (\overline{\mathsf{o}_{a,i,b}} \wedge \mathsf{o}_{a,i,c})\big) \wedge \big(\overline{\mathsf{o}_{a,b,c}} \rightarrow (\mathsf{o}_{a,i,b} \wedge \overline{\mathsf{o}_{a,i,c}})\big) \Big) \tag{1}$$

The second definition is for $b < i < c$, which avoids using the variable $\mathsf{o}_{a,b,i}$:

$$\mathsf{c}_{i;a,b,c} \leftrightarrow \Big(\big(\mathsf{o}_{a,b,c} \rightarrow (\mathsf{o}_{a,i,c} \wedge \overline{\mathsf{o}_{b,i,c}})\big) \wedge \big(\overline{\mathsf{o}_{a,b,c}} \rightarrow (\overline{\mathsf{o}_{a,i,c}} \wedge \mathsf{o}_{b,i,c})\big) \Big) \tag{2}$$

Each definition translates into six clauses (without using Tseitin variables).

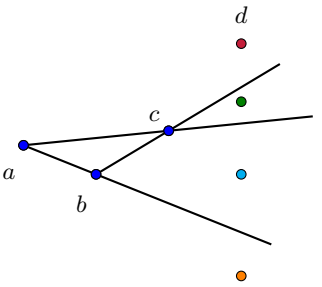
Additionally, we introduce definitions $\mathsf{h}_{a,b,c}$ of 3-*hole variables* that express whether the triangle spanned by $\{p_a, p_b, p_c\}$ is a 3-hole. The triangle $\{p_a, p_b, p_c\}$ forms a 3-hole if and only if no point p_i lies in its interior. A point p_i can only be an inner point if it lies in the vertical strip between p_a and p_c and if it is distinct from p_b. Since the points are sorted, the index i of an interior point p_i must therefore fulfill $a < i < c$ and $i \neq b$. Logically, the definition is as follows:

$$\mathsf{h}_{a,b,c} \leftrightarrow \bigwedge_{\substack{a<i<c \\ i \neq b}} \overline{\mathsf{c}_{i;a,b,c}}. \tag{3}$$

Finally, we encode the "forbid k-hole" constraint as follows: For each subset $X \subseteq S$ of size k, at least one of the triangles formed by three points in X must not be a 3-hole. So for $k = 6$, each clause consists of $\binom{k}{3} = 20$ literals.

$$\bigwedge_{\substack{X \subseteq S \\ |X|=k}} \Big(\bigvee_{\substack{a,b,c \in X \\ a<b<c}} \overline{\mathsf{h}_{a,b,c}} \Big) \tag{4}$$

In Section 4, we will optimize the encoding. Most optimizations aim to improve the encoding of the constraint (4).



$o_{a,b,c}$	$o_{a,b,d}$	$o_{a,c,d}$	$o_{b,c,d}$
+	+	+	+
+	+	+	−
+	+	−	−
+	−	−	−
−	−	−	−
−	−	−	+
−	−	+	+
−	+	+	+

Fig. 4. All possibilities to place four points, when points are sorted from left to right.

3.3 Forbidding Non-Realizable Patterns

Only a small fraction of all assignments to the $\binom{n}{3}$ orientation variables, $2^{\Theta(n \log n)}$, actually describe point sets [3]. However, we can reduce the search space from $2^{\Theta(n^3)}$ to $2^{\Theta(n^2)}$ by forbidding non-realizable patterns [24]. Consider four points p_a, p_b, p_c, p_d in a sorted point set with $a < b < c < d$. The leftmost three points determine three lines $\overline{p_a p_b}$, $\overline{p_a p_c}$, $\overline{p_b p_c}$, which partition the open half-plane $\{(x, y) \in \mathbb{R}^2 : x > x_c\}$ into four regions (see Fig. 4). After placing p_a, p_b, p_c, observe that all realizable positions of point p_d obey the following implications: $o_{a,b,c} \wedge o_{a,c,d} \Rightarrow o_{a,b,d}$ and $o_{a,b,c} \wedge o_{b,c,d} \Rightarrow o_{a,c,d}$. Similarly for the negations, $\overline{o_{a,b,c}} \wedge \overline{o_{a,c,d}} \Rightarrow \overline{o_{a,b,d}}$ and $\overline{o_{a,b,c}} \wedge \overline{o_{b,c,d}} \Rightarrow \overline{o_{a,c,d}}$. These implications are equivalent to the following clauses (grouping positive and negative):

$$(\overline{o_{a,b,c}} \vee \overline{o_{a,c,d}} \vee o_{a,b,d}) \wedge (o_{a,b,c} \vee o_{a,c,d} \vee \overline{o_{a,b,d}}) \tag{5}$$

$$(\overline{o_{a,b,c}} \vee \overline{o_{b,c,d}} \vee o_{a,c,d}) \wedge (o_{a,b,c} \vee o_{b,c,d} \vee \overline{o_{a,c,d}}) \tag{6}$$

Forbidding these non-realizable assignments was also used for $g(6) \leq 17$ [31]. Some call the restriction *signotope axioms* [10]. The counterclockwise system axioms [24] achieve the same effect, but require $\Theta(n^5)$ clauses instead of $\Theta(n^4)$.

3.4 Initial Symmetry Breaking

To further reduce the search space, we ensure that p_1 lies on the boundary of the convex hull (i.e., it is an extremal point) and that $p_2, \ldots, p_n$ appear around p_1 in counterclockwise order, thus providing us the unit clauses $(o_{1,a,b})$ for $1 < a < b$. Without loss of generality, we can label points to satisfy the above, because the labeling doesn't affect gons and holes. However, we also want points to be sorted from left to right. One can satisfy both orderings at the same time using the lemma below. We attach a proof in the extended version [19].

Lemma 1 ([28, Lemma 1]). *Let $S = \{p_1, \ldots, p_n\}$ be a point set in the plane in general position such that p_1 is extremal and $p_2, \ldots, p_n$ appear (clockwise or counterclockwise) around p_1. Then there exists a point set $\tilde{S} = \{\tilde{p}_1, \ldots, \tilde{p}_n\}$ with the same triple orientations (in particular, $\tilde{p}_1$ is extremal and $\tilde{p}_2, \ldots, \tilde{p}_n$ appear around $\tilde{p}_1$) such that the points $\tilde{p}_1, \ldots, \tilde{p}_n$ have increasing x-coordinates.*

4 Optimizing the Encoding

An ideal SAT encoding has the following three properties:

1) it is compact to reduce the cost of unit propagation (and cache misses);
2) it detects conflicts as early as possible (i.e., is domain consistent [11]); and
3) it contains variables that can generalize conflicts effectively.

The trusted encoding lacks these properties because it has $O(n^6)$ clauses, cannot quickly detect holes, and has no variables that can generalize conflicts. In this section, we show how to modify the trusted encoding to obtain all three properties. All the modifications are expressible in a proof to ensure correctness.

4.1 Toward Domain Consistency

The effectiveness of an encoding depends on how quickly the solver can determine a conflict. Given an assignment, we want to derive as much as possible via unit propagation. This is known as *domain consistency* [11]. The trusted encoding does not have this property. We modify the encoding below to boost propagation.

We borrow from the method by Szekeres and Peters that a k-gon can be detected by looking at assignments to $k-2$ orientation variables [31]. For example, if $\mathsf{o}_{a,b,c}$, $\mathsf{o}_{b,c,d}$, $\mathsf{o}_{c,d,e}$, and $\mathsf{o}_{d,e,f}$ with $a<b<c<d<e<f$ are assigned to the same truth value, then this implies that the points form a 6-gon. An illustration of this assignment is shown in Fig. 5 (left). We combine this with our observation below that only a specific triangle has to be empty to infer a 6-hole somewhere.

Consider a scenario involving six points, a, b, c, d, e, and f, that are arranged from left to right. In this scenario, the orientation variables $\mathsf{o}_{a,b,c}$, $\mathsf{o}_{b,c,d}$, $\mathsf{o}_{c,d,e}$, and $\mathsf{o}_{d,e,f}$ are all set to false, while the 3-hole variable $\mathsf{h}_{a,c,e}$ is set to true. As mentioned above, this implies that the points form a 6-gon. Together with 3-hole variable $\mathsf{h}_{a,c,e}$ being set to true, we can deduce the existence of a 6-hole: The 6-gon is either a 6-hole or it contains a 6-hole. The reasoning will be explained in the next paragraph. Note that in the trusted encoding of this scenario, only one out of the twenty literals in the corresponding 'forbid 6-hole' clause is false. This suggests that the solver is still quite far from detecting a conflict.

A crucial insight underpinning our efficient encoding is the understanding that the truth of the variable $\mathsf{h}_{a,c,e}$ alone is sufficient to infer the existence of a 6-hole. Consider the following rationale: If the triangle $\{a, b, c\}$ contains any points, then there must be at least one point inside the triangle that is closer to the line $\overline{ac}$ than point b is. Let's denote the nearest point as i. The proximity of i to the line $\overline{ac}$ guarantees that the triangle $\{a, i, c\}$ is empty. We can substitute b with i to create a smaller but similarly shaped hexagon. This logic extends to other triangles as well; specifically, the truth values of $\mathsf{h}_{c,d,e}$ and $\mathsf{h}_{a,e,f}$ are not necessary to infer the presence of a 6-hole.

Our insight emerged when we noticed that the SAT solver eliminated 3-hole literals from previous encodings. This elimination occurred primarily when only a few points existed between the leftmost and rightmost points of a triangle. On

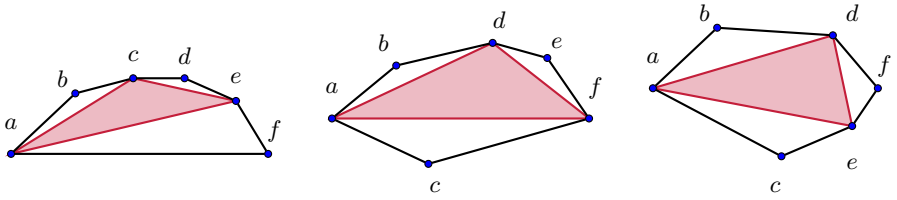


Fig. 5. Three types of 6-gons: left, all points are on one side of line $\overline{af}$ (2 cases); middle, three points are on one side and one point is on the other side of line $\overline{af}$ (8 cases); and right, two points are on either side of line $\overline{af}$ (6 cases). If the marked triangle is empty, we can conclude that there exists a 6-hole.

the other hand, the solver struggles significantly to identify the redundancy of these 3-hole literals when the leftmost and rightmost points of a triangle were far apart. Therefore, to enhance the encoding's effectiveness, we chose to omit these 3-hole literals (instead of letting the solver figure it out).

Blocking the existence of a 6-hole within the 6-gon described above can be achieved with the following clause (which simply negates the assignment):

$$\mathsf{o}_{a,b,c} \vee \mathsf{o}_{b,c,d} \vee \mathsf{o}_{c,d,e} \vee \mathsf{o}_{d,e,f} \vee \overline{\mathsf{h}_{a,c,e}} \tag{7}$$

For each set of six points, 16 different configurations can result in a 6-hole. These configurations depend on which points are positioned left or right the line connecting the leftmost and rightmost points among the six. The three types of such configurations are illustrated in Fig. 5, while the remaining configurations are symmetrical to these. It is important to note that this adds $16 \times \binom{n}{6}$ clauses to the formula, significantly increasing its size.

We can reduce the number of clauses by about a 30% by strategically selecting which triangle within a 6-gon is checked to be empty (i.e., which 3-hole literal will be used). The two options are the triangle that includes the leftmost point (as depicted in Fig. 5) and the triangle with the second-leftmost point. If the leftmost point is p_1, we opt for the second-leftmost point; otherwise, we choose the leftmost point. After propagating the unit clauses $\mathsf{o}_{1,a,b}$, the clauses that describe configurations with three points below the line $\overline{af}$ become subsumed by the clause for the configuration with four points below the line $\overline{1f}$.

4.2 An $O(n^4)$ Encoding

This section is rather technical. It introduces auxiliary variables to reduce our encoding to $O(n^4)$ clauses. The process is known as structured bounded variable addition (SBVA) [13], which in each step adds a new auxiliary variable to encode a subset of the formula more compactly. SBVA heuristically selects the auxiliary variables. Instead, we select them manually because it is more effective, the new variables have meaning, and SBVA is extremely slow on this problem. Eliminating the auxiliary variables results in the encoding of Section 4.1.

The first type of these variables, $\mathsf{u}^4_{a,c,d}$, represents the presence of a 4-gon $\{a,b,c,d\}$ such that points a,b,c,d appear in this order from left to right and b and c are above the line $\overline{ad}$. Furthermore, the variables $\mathsf{u}^5_{a,d,e}$ indicate the existence of a 5-gon $\{a,b,c,d,e\}$ with the property that the points a,b,c,d,e appear in this order from left to right, the points b, c, and d are above the line $\overline{ae}$, and the triangle $\{a,c,e\}$ is empty. This configuration implies the existence of a 5-hole within $\{a,b,c,d,e\}$ using similar reasoning as described in Section 4.1. The clauses enforcing these properties are outlined below.

$$\mathsf{u}^4_{a,c,d} \vee \mathsf{o}_{a,b,c} \vee \mathsf{o}_{b,c,d} \qquad \text{with } a<b<c<d \tag{8}$$

$$\mathsf{u}^5_{a,d,e} \vee \overline{\mathsf{u}^4_{a,c,d}} \vee \mathsf{o}_{c,d,e} \vee \overline{\mathsf{h}_{a,c,e}} \qquad \text{with } a<c<d<e \tag{9}$$

In the following we distinguish five types of 6-holes by the number of its points that lie above/below the line connecting its leftmost and rightmost points. Fig. 5 shows the three configurations with four, three, and two points above the line, respectively. The two cases with three and four points below the line are symmetric but will be handled in a different and more efficient manner below.

To block all 6-holes with configurations having three or four points above the line connecting the leftmost and rightmost points, we utilize the variables $\mathsf{u}^5_{a,d,e}$. Specifically, a configuration with three points above occurs if there is a point b situated between a and e, lying below the line $\overline{ae}$. Also, the configuration with four points above arises when a point f, located to the right of e, falls below the line $\overline{de}$. The associated clauses for these configurations are detailed below. The omission of 3-hole literals is justified by our knowledge that a 3-hole exists among a, c, and e for some point c positioned above the line $\overline{ae}$.

$$\overline{\mathsf{u}^5_{a,d,e}} \vee \overline{\mathsf{o}_{a,b,e}} \qquad \text{with } a<d<e, a<b<e \tag{10}$$

$$\overline{\mathsf{u}^5_{a,d,e}} \vee \mathsf{o}_{d,e,f} \qquad \text{with } a<d<e<f \tag{11}$$

To block the third type of a 6-hole, we need to introduce variables $\mathsf{v}^4_{a,c,d}$ which, similar as $\mathsf{u}^4_{a,c,d}$, indicate the presence of a 4-gon $\{a,b,c,d\}$ with the property that the points a,b,c,d appear in this order from left to right and b and c are *below* the line $\overline{ad}$. The clauses that encode these variables are:

$$\mathsf{v}^4_{a,c,d} \vee \overline{\mathsf{o}_{a,b,c}} \vee \overline{\mathsf{o}_{b,c,d}} \qquad \text{with } a<b<c<d \tag{12}$$

Using the variables $\mathsf{u}^4_{a,c,d}$ and $\mathsf{v}^4_{a,c',d}$ we are now ready to block the configuration of the third type of a 6-hole where two points lie above and two points lie below the line connecting the leftmost and rightmost points; see Fig. 5 (right). Recall that $\mathsf{u}^4_{a,c,d}$ denotes a 4-gon situated above the line $\overline{ad}$, with c being the second-rightmost point. Also, $\mathsf{v}^4_{a,c',d}$ denotes a 4-gon below the line $\overline{ad}$, with c' as the second-rightmost point. A 6-hole exists if both $\mathsf{u}^4_{a,c,d}$ and $\mathsf{v}^4_{a,c',d}$ are true for some points a and d when there are no points within the triangle formed by a, c, and c'. Or, in clauses:

$$\overline{\mathsf{u}^4_{a,c,d}} \vee \overline{\mathsf{v}^4_{a,c',d}} \vee \overline{\mathsf{h}_{a,c,c'}} \qquad \text{with } a<c<c'<d \tag{13}$$

$$\overline{\mathsf{u}^4_{a,c,d}} \vee \overline{\mathsf{v}^4_{a,c',d}} \vee \overline{\mathsf{h}_{a,c',c}} \qquad \text{with } a<c'<c<d \tag{14}$$

The remaining configurations to consider involve those with three or four points below the line joining the leftmost and rightmost points. As we discussed at the end of Section 4.1, these configurations can be encoded more compactly. We only need to block the existence of 5-holes $\{a, b, c, d, e\}$ with the property that the points $1, a, b, c, d, e$ appear in this order from left to right and the points b, c, and d are below the line $\overline{ae}$. The reasoning is as follows: if such a 5-hole exists, it can be expanded into a 6-hole by the closest point to line $\overline{ab}$ within the triangle $\{1, a, b\}$ (which is point 1 if the triangle is empty). Additionally, by blocking these specific 5-holes, we simultaneously block all 6-holes with three or four points below the line between the leftmost and rightmost points. Following the earlier cases, we only require a single 3-hole literal which ensures that the triangle $\{a, c, e\}$ is empty. The clauses to block these 5-holes are as follows:

$$\overline{\mathsf{v}^4_{a,c,d}} \vee \overline{\mathsf{o}_{c,d,e}} \vee \overline{\mathsf{h}_{a,c,e}} \qquad \text{with } 1 < a < c < d < e \tag{15}$$

This encoding uses $O(n^4)$ clauses, while it has the same propagation power as having all the $16 \times \binom{n}{6}$ clauses in the domain-consistent encoding of Section 4.1. In general, the trusted encoding for k-holes uses $O(n^k)$ clauses, while the optimized encoding when generalized to k-holes has only $O(kn^4)$ clauses, or $O(n^4)$ for every fixed k. An encoding of size $O(n^4)$ for k-gons is analogous: simply remove the 3-hole literals from the clauses.

4.3 Minor Optimizations

We can make the encoding even more compact by removing a large fraction of the clauses from the trusted encoding. Note that constraints to forbid 6-holes contain only negative 3-hole literals. That means that only half of the constraints to define the 3-hole variables are actually required. This in turn shows that only half of the inside variable definitions are required. So, instead of (1), (2), and (3), it suffices to use the following:

$$\mathsf{c}_{i;a,b,c} \rightarrow \Big(\big(\mathsf{o}_{a,b,c} \rightarrow (\overline{\mathsf{o}_{a,i,b}} \wedge \mathsf{o}_{a,i,c})\big) \wedge \big(\overline{\mathsf{o}_{a,b,c}} \rightarrow (\mathsf{o}_{a,i,b} \wedge \overline{\mathsf{o}_{a,i,c}})\big) \Big) \tag{16}$$

$$\mathsf{c}_{i;a,b,c} \rightarrow \Big(\big(\mathsf{o}_{a,b,c} \rightarrow (\mathsf{o}_{a,i,c} \wedge \overline{\mathsf{o}_{b,i,c}})\big) \wedge \big(\overline{\mathsf{o}_{a,b,c}} \rightarrow (\overline{\mathsf{o}_{a,i,c}} \wedge \mathsf{o}_{b,i,c})\big) \Big) \tag{17}$$

$$\mathsf{h}_{a,b,c} \leftarrow \bigwedge_{\substack{a<i<c \\ i \neq b}} \overline{\mathsf{c}_{i;a,b,c}}. \tag{18}$$

It is worth noting that the SAT preprocessing technique blocked-clause elimination (BCE) can automatically remove the omitted clauses [22]. However, for means of efficiency, BCE is turned off by default in top-tier solvers, including the solver CaDiCaL, which we used for the proof. During initial experiments, we observed that omitting these clauses slightly improves the performance.

Finally, the variables $\mathsf{u}^4_{a,c,d}$ and $\mathsf{v}^4_{a,c,d}$ can be used to more compactly encode the clauses (6). We can replace them with the following clauses:

$$(\overline{\mathsf{u}^4_{a,c,d}} \vee \overline{\mathsf{o}_{a,c,d}}) \wedge (\overline{\mathsf{v}^4_{a,c,d}} \vee \mathsf{o}_{a,c,d}) \qquad \text{with } a < c < d \tag{19}$$

4.4 Breaking the Reflection Symmetry

Holes are invariant to reflectional symmetry: If we mirror a point set S, then the counterclockwise order around the extremal point p_1 (which is $p_2, \ldots, p_n$) is reversed (to $p_n, \ldots, p_2$). By relabeling points to preserve the counterclockwise order, we preserve $\mathsf{o}_{1,a,b} = \mathit{true}$ for $a < b$, while the original orientation variables $\mathsf{o}_{a,b,c}$ with $2 \leq a < b < c \leq n$ are mapped to $\mathsf{o}_{n-c+2,n-b+2,n-a+2}$. A similar mapping applies to the containment and 3-hole variables. The trusted encoding maps almost onto itself, except for the missing reflection clauses of (5) and (6). As a fix for verification, we add each reflected clause using one resolution step.

Since only a tiny fraction of triple orientations map to themselves (so-called *involutions*), breaking the reflectional symmetry reduces the search space by a factor of almost 2. We partially break this symmetry by constraining the variables $\mathsf{o}_{a,a+1,a+2}$ with $2 \leq a \leq n-2$. We used the symmetry-breaking predicate below, because it is compatible with our cube generation, described in Section 5.

$$\mathsf{o}_{\lceil \frac{n}{2} \rceil - 1, \lceil \frac{n}{2} \rceil, \lceil \frac{n}{2} \rceil + 1}, \ldots, \mathsf{o}_{2,3,4} \preccurlyeq \mathsf{o}_{\lfloor \frac{n}{2} \rfloor + 1, \lfloor \frac{n}{2} \rfloor + 2, \lfloor \frac{n}{2} \rfloor + 3}, \ldots, \mathsf{o}_{n-2,n-1,n} \tag{20}$$

One symmetry that remains is the choice of the first point. Any point on the convex hull could be picked for this purpose, and breaking it can potentially reduce the search space by at least a factor of 3. However, breaking this symmetry effectively is complicated and we therefore left it on the table.

5 Problem Partitioning

The formula to determine that $h(6) \leq 30$ requires CPU years to solve. To compute this in reasonable time, the problem needs to be partitioned into many small subproblems that can be solved in parallel. Although there exist tools to do the partitioning automatically [18], we observed that this partitioning was ineffective. As a consequence, we focused on manual partitioning.

During our initial experiments, we determined which orientation variables were suitable for splitting. We used the formula for $g(6) \leq 17$ for this purpose because its runtime is large enough to make meaningful observations and small enough to explore many options. It turned out that the variables $\mathsf{o}_{a,a+1,a+2}$ were the most effective choice for splitting the problem. Assigning one of these $\mathsf{o}_{a,a+1,a+2}$ variables to true/false roughly halves the search space and reduces the runtime by a factor of roughly 2.

A problem with n points has $n-3$ free variables of the form $\mathsf{o}_{a,a+1,a+2}$, as the variable $\mathsf{o}_{1,2,3}$ is already fixed by the symmetry breaking. One cannot generate 2^{n-3} equally easy subproblems, because $(\overline{\mathsf{o}_{a,a+1,a+2}} \vee \overline{\mathsf{o}_{a+1,a+2,a+3}} \vee \overline{\mathsf{o}_{a+2,a+3,a+4}})$ and $(\mathsf{o}_{a,a+1,a+2} \vee \mathsf{o}_{a+1,a+2,a+3} \vee \mathsf{o}_{a+2,a+3,a+4} \vee \mathsf{o}_{a+3,a+4,a+5})$ follow directly from the optimized formula after unit propagation. Thus, assigning three consecutive $\mathsf{o}_{a,a+1,a+2}$ variables to true results directly in a falsified clause, as it would create a 6-hole among the points p_1, p_a, …, p_{a+4}. The same holds for four consecutive $\mathsf{o}_{a,a+1,a+2}$ variables assigned to false, which would create a 6-hole among the

points $p_a, \ldots, p_{a+5}$. The asymmetry is due to fixing the variables $\mathsf{o}_{1,a,b}$ to true. If we assigned them to false, then the opposite would happen.

We observed that limiting the partition to variables involving the middle points reduces the total runtime. We will demonstrate such experiments in Section 6.2. So, to obtain suitable cubes, we considered all assignments of the sequence $\mathsf{o}_{a,a+1,a+2}$, $\mathsf{o}_{a+1,a+2,a+3}$, $\ldots$, $\mathsf{o}_{a+\ell-1,a+\ell,a+\ell+1}$ for a suitable constant ℓ and $a = \frac{n+\ell}{2} - 1$ such that the above properties are fulfilled, that is, no three consecutive entries are true and no four consecutive entries are false. In the following we refer to ℓ as the *length* of the cube-space. In our experiments, we observed that picking $\ell < n - 3$ reduces the overall computational costs. Specifically, for the $h(6) \leq 30$ experiments, we use length $\ell = 21$.

Our initial experiments showed that the runtime of cubes grows exponentially with the number of occurrences of the alternating pattern $\mathsf{o}_{b,b+1,b+2} = +$, $\mathsf{o}_{b+1,b+2,b+3} = -$, $\mathsf{o}_{b+2,b+3,b+4} = +$. As a consequence, the hardest cube for $h(6) \leq 30$ would still require days of computing time, thereby limiting parallelism. To deal with this issue, we further partition cubes that contain this pattern. For each occurrence of the alternating pattern in a cube, we split the cube into two cubes: one that extends it with $\mathsf{o}_{b,b+2,b+4}$ and one that extends it with $\overline{\mathsf{o}_{b,b+2,b+4}}$. Note that we do this for each occurrence. So a cube containing m of these patterns is split into 2^m cubes. This reduced the computational costs of the hardest cubes to less than an hour.

6 Evaluation

For the experiments, we use the solver CaDiCaL (version 1.9.3) [1], which is currently the only top-tier solver that can produce LRAT proofs directly. The efficient, verified checker cakeLPR [32] validated the proofs. We run CaDiCaL with command-line options: `--sat --reducetarget=10 --forcephase --phase=0`. The first option reduces the number of restarts. This is typically more useful for satisfiable formulas (as the name suggests), but in this case it is also helpful for unsatisfiable formulas. The second option turns off the aggressive clause deletion strategy. The last two options turn on negative branching, a MiniSAT heuristic [7]. Experiments were run on a specialized, internal Amazon Web Services solver framework that provides cloud-level scaling. The framework used `m6i.xlarge` instances, which have two physical cores and 16 GB of memory.

6.1 Impact of the Encoding

To illustrate the impact of the encoding on the performance, we show some statistics on various encodings of the $h(6) \leq 30$ formula. We restricted this experiment to solving a single randomly-picked subproblem. For other subproblems, the results were similar. We experimented with the following five encodings:

- T: the trusted encoding presented in Section 3
- O_1: T with (4) replaced by the domain-consistent encoding (7) of Section 4.1

- O_2: O_1 with (7) replaced by the $O(n^4)$ encoding (8) - (15) of Section 4.2
- O_3: O_2 with the minor optimizations that replace (1), (2), (3), and (6) by (17), (18), (18), and (19), respectively, see Section 4.3
- O_4: O_3 extended with the symmetry-breaking predicate from Section 4.4

Table 1 summarizes the results. The domain-consistent encoding can be solved more efficiently than the trusted encoding while having over five times as many clauses. The reason for the faster performance becomes clear when looking at the number of conflicts and propagations. The domain-consistent encoding requires just over a fifth as many conflicts and propagations to determine unsatisfiability. The auxiliary variables that enable the $O(n^4)$ encoding reduce the size by almost an order of magnitude. The resulting formula can be solved three times as fast, while using a similar number of conflicts and propagations. The minor optimizations reduce the size by roughly a third and further improve the runtime. Finally, the addition of the symmetry-breaking predicate doesn't impact the performance. Its main purpose is to halve the number of cubes.

We also solved the optimized encoding (O_3) of the formula $g(6) \leq 17$, which takes 41.99 seconds using 623 540 conflicts. Adding the symmetry-breaking predicate (O_4) reduces the runtime to 17.39 seconds using 316 785 conflicts. So the symmetry-breaking predicate reduces the number of conflicts by roughly a factor of 2 (as expected) while the runtime is reduced even more. The latter is due to the slowdown caused by maintaining more conflict clauses while solving the formula without the symmetry-breaking predicate.

6.2 Impact of the Partitioning

All known point sets witnessing the lower bound $h(6) \geq 30$ contain a 7-gon. To obtain a possibly easier problem to test and compare heuristics, we studied how many points are required to guarantee the existence of a 6-hole or a 7-gon. It turned out that the answer is at most 24 (Theorem 2). Computing this is still hard but substantially easier compared to our main result. During our experiments, we observed that increasing the number of cubes can increase the total runtime. We therefore explored which parameters produce the lowest total runtime. The experimental results are shown in Table 2 for various values for the parameter ℓ. Incrementing ℓ by 2 increases the number of cubes roughly by a factor of 3. The optimal total runtime is achieved for $\ell = 15$, which is a 62%

Table 1. Comparison of the different encodings.

formula	#variables	#clauses	#conflicts	#propagations	time (s)
T	62 930	1 171 942	1 082 569	1 338 662 627	243.07
O_1	62 930	5 823 078	228 838	282 774 472	136.20
O_2	75 110	667 005	211 272	343 388 591	45.49
O_3	75 110	436 047	234 755	340 387 692	39.46
O_4	75 110	444 238	234 587	342 904 580	39.41

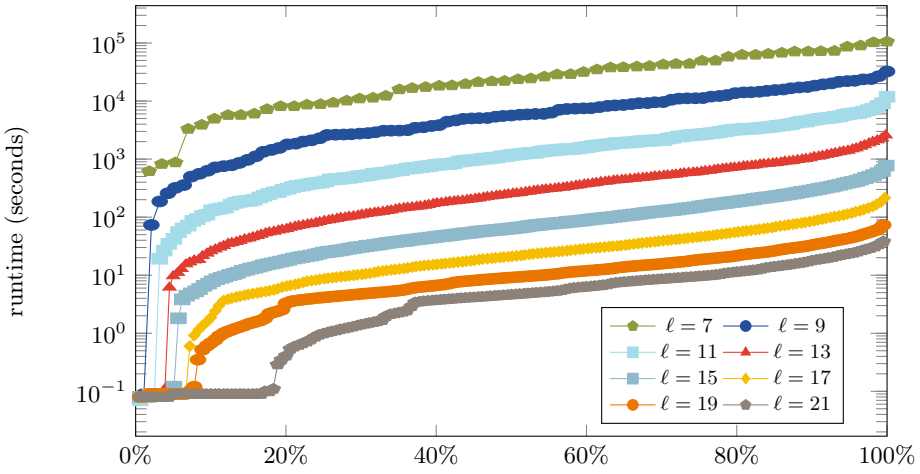


Fig. 6. Runtime to solve the subproblems of Theorem 2 for various splitting parameters.

reduction compared to full partitioning ($\ell = 21$). Note that the solving time for the hardest cube (the max column) increases substantially when using fewer cubes. This in turn reduces the effectiveness of parallelism. The runtime without partitioning is expected to be about 1000 CPU hours, so partitioning achieves super-linear speedups and more than a factor of 4 speedup for $\ell = 15$. Fig. 6 shows plots of cumulatively solved cubes, with similar curves for all settings.

We also evaluated the off-the-shelf tool March for partitioning. This tool was used to prove Schur Number Five [16]. We used option `-d 13` to cut off partitioning at depth 13 to create 8192 cubes. That partition turned out to be very poor: at least 18 cubes took over 100 000 seconds. The expected total costs are about 10 000 CPU hours, so 10 times the estimated partition-free runtime.

A partitioning can also guide the search to solve the formula $g(6) \leq 17$. The partitioning of this formula using $\ell = 12$ results in 1108 cubes. If we add these cubes to the formula with the symmetry-predicate (O_4) in the iCNF format [34], then CaDiCaL can solve it in 8.53 seconds using 205 153 conflicts.

Table 2. Runtime comparison for different values of partitioning parameter ℓ

ℓ	#cubes	average time (s)	max time (s)	total time (h)
21	312 418	6.99	66.86	606.55
19	89 384	13.61	123.70	337.96
17	25 663	34.29	293.10	244.50
15	7393	112.61	949.50	231.27
13	2149	431.26	3 347.59	257.44
11	629	1 847.46	11 844.05	322.79
9	188	7 745.14	32 329.05	404.47
7	57	32 905.90	105 937.76	521.01

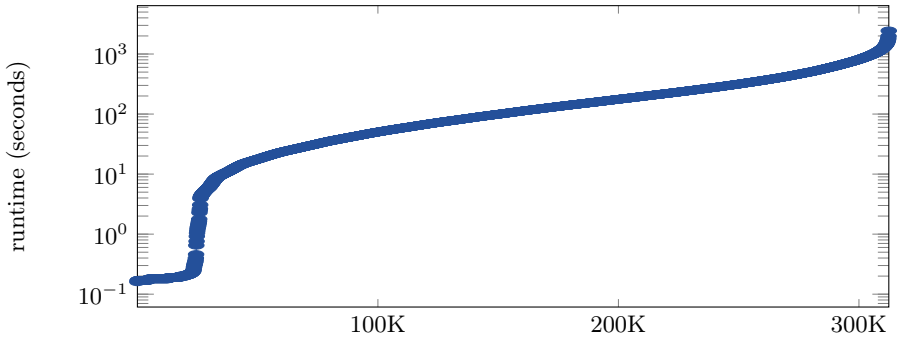


Fig. 7. Reported process time to solve the subproblems of $h(6) \leq 30$ with proof logging while running a formally-verified checker to validate the solver's output.

6.3 Theorem 1

To show that the optimized encoding for $h(6) \leq 30$ is unsatisfiable, we partitioned the problem with the splitting algorithm described in Section 5 with parameter $\ell = 21$, which results in 312 418 cubes. We picked this setting based on the experiments shown in Table 2. Fig. 7 shows the runtime of solving the subproblems. The average runtime was just below 200 seconds. All subproblems were solved in less than an hour. Almost 24 000 subproblems could be solved within a second. For these subproblems, the cube resulted directly in a conflict, so the solver didn't have to do any search.

The total runtime is close to 17 300 CPU hours, or slightly less than 2 CPU years. We could achieve practically a linear speedup using 1000 `m6i.xlarge` instances. The timings include producing and validating the proof as described in Section 7.1. The combined size of the proofs is 180 terabytes in the uncompressed LRAT format used by the cakeLPR checker. In past verification efforts of hard math problems, the produced proofs were in the DRAT format. For this problem, the LRAT proofs are roughly 2.3 times as large as the corresponding DRAT proof. We estimate that the DRAT proof would have been 78 terabytes in size, so approximately one third to the proof of the Pythagorean triples problem [17]. For all problems, the checker was able to easily catch up with the solver while running on a different core, thereby finishing as soon as the solver was done.

7 Verification

We applied three verification steps to increase trust in the correctness of our results. In the first step, we check the results produced by the SAT solver. The second step consists of checking the correctness of the optimizations discussed in Section 4. In the third step, we validate that the case split covers all cases.

7.1 Concurrent Solving and Checking

The most commonly used approach to validate SAT-solving results works as follows. First, a SAT solver produces a DRAT proof. This proof is checked and trimmed using an unverified efficient tool that produces a LRAT proof. The difference between a DRAT proof and a LRAT proof is that the latter contains hints. The LRAT proof is then validated by a formally-verified checker, which uses the hints to obtain efficient performance.

Recently, the SAT solver CaDiCaL added support for producing LRAT proofs directly (since version 1.7.0). This allows us to produce the proof and validate it concurrently. To the best of our knowledge, we are the first to take advantage of this possibility. CaDiCaL sends its proof to a pipe and the verified checker cakeLPR reads it from the pipe. This tool chain works remarkably well and adds little overhead while avoiding storing large files.

7.2 Reencoding Proof

We validated the four optimizations presented in Section 4. Only the trusted encoding has the reflection symmetry, as each of the optimizations don't preserve this symmetry. Each of the clauses in the symmetry-breaking predicate have the substitution redundancy (SR) property [5] with respect to the trusted encoding. However, there doesn't exist a SR checker. Instead, we transformed the SR check into a sequence of DRAT addition and deletion steps. This is feasible for small point sets (up to 10 points), but is too expensive for the full problem. It may therefore be more practical to verify this optimization in a theorem prover.

Transforming the trusted encoding into the domain-consistent one is challenging to validate because the solver cannot easily infer the existence of a 6-hole using only the clauses (7). Since we are replacing (4) by (7) and clause deletion trivially preserves satisfiability, we only need to check whether each of the clauses (7) is entailed by the trusted encoding. This can be achieved by constructing a formula that asks whether there exists an assignment that satisfies the trusted encoding, but falsifies at least one of the clauses (7). We validated that this formula is unsatisfiable for $n \leq 12$ (around 300 seconds).[5] The formula becomes challenging to solve for larger n. However, the validation for small n provides substantial evidence of the correctness of the encoding and the implementation.

Checking the correctness of the other two optimizations is easier. Observe that one can obtain the domain-consistent encoding from the $O(n^4)$ encoding by applying Davis-Putnam resolution [6] on the auxiliary variables. This can be expressed using DRAT steps. The DRAT derivation from the domain-consistent encoding to the $O(n^4)$ encoding applies all these steps in reverse order. The minor optimizations mostly delete clauses, which is trivially correct for proofs of unsatisfiability. The clauses (19) have the RAT property on the auxiliary variables and their redundancy can easily be checked using a DRAT checker.

[5] We implemented an entailment tool, see https://github.com/marijnheule/entailment

7.3 Tautology Proof

The final validation step consists of checking whether the partition of the problem covers the entire search space. This part has also been called the tautology proof [16], because in most cases it needs to determine whether the disjunction of cubes is a tautology. We take a slightly different approach and validate that the following formula is unsatisfiable: the conjunction of the negated cubes; the symmetry-breaking predicate; and some clauses from the formula.

Recall that we omitted various cubes because they resulted in a conflict with the clauses $(\overline{\mathsf{o}_{a,a+1,a+2}} \vee \overline{\mathsf{o}_{a+1,a+2,a+3}} \vee \overline{\mathsf{o}_{a+2,a+3,a+4}})$ with $a \in \{2,\ldots,n-4\}$ and $(\mathsf{o}_{a,a+1,a+2} \vee \mathsf{o}_{a+1,a+2,a+3} \vee \mathsf{o}_{a+2,a+3,a+4} \vee \mathsf{o}_{a+3,a+4,a+5})$ with $a \in \{2,\ldots,n-5\}$. We checked with DRATtrim that these clauses are implied by the optimized formulas, which takes 0.3 CPU seconds. We combined them with the negated cubes and the symmetry-breaking predicate, which results in an unsatisfiable formula that can be solved by CaDiCaL in 12 CPU seconds.

8 Conclusion

We closed the final case regarding k-holes in the plane by showing $h(6) = 30$. This is another example that SAT-solving techniques can effectively solve a range of long-standing open problems in mathematics. Other successes include the Pythagorean triples problem [17], Schur number five [16], and Keller's conjecture [4]. Also, we recomputed $g(6) = 17$ many orders of magnitude faster compared to the original computation by Szekeres and Peters [31] even when taking into account the difference in hardware. So, SAT techniques overwhelmingly outperformed a dedicated approach on this geometry problem. Key contributions include an effective, compact encoding and a partitioning strategy enabling linear-time speedups even when using thousands of cores. We also presented a new concurrent proof-checking procedure to significantly decrease validation costs.

Although the tools are fully automatic, some aspects of our solution require the ingenuity of the user. In particular, we had to develop encoding optimizations and a search-space partitioning strategy to take full advantage of the power of the tools. Constructing the domain-consistent encoding automatically appears challenging. Most other optimizations can be achieved automatically, for example via structured bounded variable elimination [13]. However, the resulting formula cannot be solved as efficiently as the presented one. Substantial research into generating effective partitionings is required to enable non-experts to solve such hard problems. Although we validated most steps, formally verifying the trusted encoding or even the domain-consistent encoding would further increase trust in the correctness of our result.

Acknowledgements Heule is partially supported by NSF grant CCF-2108521. Scheucher was supported by the DFG grant SCHE 2214/1-1. We thank Donald Knuth, Benjamin Kiesl-Reiter, John Mackey, and the reviewers for their valuable feedback. The authors met for the first time during Dagstuhl meeting 23261 "SAT Encodings and Beyond", which kicked off the research published in this paper.

References

1. Biere, A., Fazekas, K., Fleury, M., Heisinger, M.: CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In: Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions. Department of Computer Science Report Series B, vol. B-2020-1, pp. 51–53. University of Helsinki (2020), http://hdl.handle.net/10138/318754
2. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 336. IOS Press, second edn. (2021), https://www.iospress.com/catalog/books/handbook-of-satisfiability-2
3. Björner, A., Las Vergnas, M., White, N., Sturmfels, B., Ziegler, G.M.: Oriented Matroids, Encyclopedia of Mathematics and its Applications, vol. 46. Cambridge University Press, 2 edn. (1999). https://doi.org/10/bhb4rn
4. Brakensiek, J., Heule, M.J.H., Mackey, J., Narváez, D.E.: The resolution of keller's conjecture. Journal of Automated Reasoning **66**(3), 277–300 (2022). https://doi.org/10.1007/S10817-022-09623-5
5. Buss, S., Thapen, N.: DRAT and propagation redundancy proofs without new variables. Logical Methods in Computer Science **17**(2) (2021). https://doi.org/10/mbdx
6. Davis, M., Putnam, H.: A computing procedure for quantification theory. Journal of the ACM **7**(3), 201–215 (1960). https://doi.org/10/bw9h55
7. Eén, N., Sörensson, N.: An extensible sat-solver. In: Theory and Applications of Satisfiability Testing. pp. 502–518. Springer (2004)
8. Erdős, P., Szekeres, G.: A combinatorial problem in geometry. Compositio Mathematica **2**, 463–470 (1935), http://www.renyi.hu/~p_erdos/1935-01.pdf
9. Erdős, P., Szekeres, G.: On some extremum problems in elementary geometry. Annales Universitatis Scientiarium Budapestinensis de Rolando Eötvös Nominatae, Sectio Mathematica **3–4**, 53–63 (1960), https://www.renyi.hu/~p_erdos/1960-09.pdf
10. Felsner, S., Weil, H.: Sweeps, arrangements and signotopes. Discrete Applied Mathematics **109**(1), 67–94 (2001). https://doi.org/10/dc4tb4
11. Gent, I.P.: Arc consistency in SAT. In: European Conference on Artificial Intelligence (ECAI 2002). FAIA, vol. 77, pp. 121–125. IOS Press (2002), https://frontiersinai.com/ecai/ecai2002/pdf/p0121.pdf
12. Gerken, T.: Empty Convex Hexagons in Planar Point Sets. Discrete & Computational Geometry **39**(1), 239–272 (2008). https://doi.org/10/c4kn3s
13. Haberlandt, A., Green, H., Heule, M.J.H.: Effective Auxiliary Variables via Structured Reencoding. In: International Conference on Theory and Applications of Satisfiability Testing (SAT 2023). Leibniz International Proceedings in Informatics (LIPIcs), vol. 271, pp. 11:1–11:19. Dagstuhl, Dagstuhl, Germany (2023). https://doi.org/10.4230/LIPIcs.SAT.2023.11
14. Harborth, H.: Konvexe Fünfecke in ebenen Punktmengen. Elemente der Mathematik **33**, 116–118 (1978), http://www.digizeitschriften.de/dms/img/?PID=GDZPPN002079801
15. Heule, M.J.H.: The DRAT format and DRAT-trim checker (2016), arXiv:1610.06229
16. Heule, M.J.H.: Schur number five. In: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence. AAAI'18, AAAI Press (2018)
17. Heule, M.J.H., Kullmann, O., Marek, V.W.: Solving and verifying the Boolean Pythagorean triples problem via cube-and-conquer. In: Theory and Applications

of Satisfiability Testing (SAT 2016). LNCS, vol. 9710, pp. 228–245. Springer (2016). https://doi.org/10/gkkscn
18. Heule, M.J.H., Kullmann, O., Wieringa, S., Biere, A.: Cube and Conquer: Guiding CDCL SAT Solvers by Lookaheads. In: Hardware and Software: Verification and Testing. pp. 50–65. Springer (2012). https://doi.org/10/f3ss29
19. Heule, M.J.H., Scheucher, M.: Happy Ending: An Empty Hexagon in Every Set of 30 Points (Extended Version) (2024), https://arxiv.org/abs/2403.00737
20. Holmsen, A.F., Mojarrad, H.N., Pach, J., Tardos, G.: Two extensions of the Erdős–Szekeres problem. Journal of the European Mathematical Society pp. 3981–3995 (2020). https://doi.org/10/gsjw4m
21. Horton, J.: Sets with no empty convex 7-gons. Canadian Mathematical Bulletin **26**, 482–484 (1983). https://doi.org/10/chf6dk
22. Järvisalo, M., Biere, A., Heule, M.J.H.: Blocked clause elimination. In: Tools and Algorithms for the Construction and Analysis of Systems. pp. 129–144. Springer (2010)
23. Kalbfleisch, J., Kalbfleisch, J., Stanton, R.: A combinatorial problem on convex regions. In: Proc. Louisiana Conf. Combinatorics, Graph Theory and Computing, Congressus Numerantium, vol. 1, Baton Rouge, La.: Louisiana State Univ. pp. 180–188 (1970)
24. Knuth, D.E.: Axioms and Hulls, LNCS, vol. 606. Springer (1992). https://doi.org/10/bwfnz9
25. Marić, F.: Fast formal proof of the Erdős–Szekeres conjecture for convex polygons with at most 6 points. Journal of Automated Reasoning **62**, 301–329 (2019). https://doi.org/10/gsjw4r
26. Nicolás, M.C.: The Empty Hexagon Theorem. Discrete & Computational Geometry **38**(2), 389–397 (2007). https://doi.org/10/bw3hnd
27. Overmars, M.: Finding Sets of Points without Empty Convex 6-Gons. Discrete & Computational Geometry **29**(1), 153–158 (2002). https://doi.org/10/cnqmr4
28. Scheucher, M.: Two disjoint 5-holes in point sets. Computational Geometry **91**, 101670 (2020). https://doi.org/10/gsjw2z
29. Scheucher, M.: A SAT Attack on Erdős–Szekeres Numbers in $\mathbb{R}^d$ and the Empty Hexagon Theorem. Computing in Geometry and Topology **2**(1), 2:1–2:13 (2023). https://doi.org/10/gsjw22
30. Suk, A.: On the Erdős–Szekeres convex polygon problem. Journal of the AMS **30**, 1047–1053 (2017). https://doi.org/10/gsjw44
31. Szekeres, G., Peters, L.: Computer solution to the 17-point Erdős–Szekeres problem. Australia and New Zealand Industrial and Applied Mathematics **48**(2), 151–164 (2006). https://doi.org/10/dkb9j3
32. Tan, Y.K., Heule, M.J.H., Myreen, M.O.: Verified propagation redundancy and compositional UNSAT checking in cakeml. International Journal on Software Tools for Technology **25**(2), 167–184 (2023). https://doi.org/10/grw7wm
33. Tóth, G., Valtr, P.: The Erdős–Szekeres theorem: Upper Bounds and Related Results. In: Combinatorial and Computational Geometry. vol. 52, pp. 557–568. MSRI Publications, Cambridge Univ. Press (2005), http://www.ams.org/mathscinet-getitem?mr=2178339
34. Wieringa, S., Niemenmaa, M., Heljanko, K.: Tarmo: A framework for parallelized bounded model checking. In: International Workshop on Parallel and Distributed Methods in verifiCation, PDMC 2009. EPTCS, vol. 14, pp. 62–76 (2009). https://doi.org/10.4204/EPTCS.14.5