Solving and Verifying the Boolean Pythagorean Triples Problem via Cube-and-Conquer

Marijn J.H. Heule

THE UNIVERSITY OF TEXAS AT AUSTIN

Joint work with
Oliver Kullmann and Victor W. Marek

Introduction

Schur's Theorem [Schur 1917]

Can the set of natural numbers $\{1, 2, \ldots\}$ be partitioned into k parts with no part containing a, b, and c such that $a+b=c$? Otherwise, what is the smallest finite counter-example?

Schur's Theorem [Schur 1917]

Can the set of natural numbers $\{1, 2, \dots\}$ be partitioned into k parts with no part containing a, b, and c such that $a+b=c$? Otherwise, what is the smallest finite counter-example?

$$\begin{array}{ccccccccc} \{1\} & & \{1\} & & \{1,4\} & & \{1,4\} & & \\ & \rightarrow & & \rightarrow & & \rightarrow & & \rightarrow & \times \\ \{\} & & \{2\} & & \{2\} & & \{2,3\} & & \\ & & & & & & & & 1+4=5 \\ \text{init} & & 1+1=2 & & 2+2=4 & & 1+3=4 & & \\ & & & & & & & & 2+3=5 \end{array}$$

Schur's Theorem [Schur 1917]

Can the set of natural numbers $\{1, 2, \ldots\}$ be partitioned into k parts with no part containing a, b, and c such that $a+b=c$? Otherwise, what is the smallest finite counter-example?

$$\begin{array}{ccccccccc} \{1\} & & \{1\} & & \{1,4\} & & \{1,4\} & & \\ & \to & & \to & & \to & & \to & \times \\ \{\} & & \{2\} & & \{2\} & & \{2,3\} & & \end{array}$$

$$\begin{array}{ccccc} \text{init} & 1+1=2 & 2+2=4 & 1+3=4 & \begin{array}{c} 1+4=5 \\ 2+3=5 \end{array} \end{array}$$

Theorem (Schur's Theorem)

For each $k > 0$, there exists a number $S(k)$, known as Schur number k, such that $[1, S(k)]$ can be partitioned into k parts with no part containing a, b, and c such that $a + b = c$, while this is impossible for $[1, S(k)+1]$.

$S(1) = 1, S(2) = 4, S(3) = 13, S(4) = 44$ [Baumert 1965], $160 \leq S(5) \leq 315$ [Exoo 1994, Fredricksen 1979].

Schur's Theorem on Squares

Can the set of squares $\{1^2, 2^2, 3^2, \dots\}$ be partitioned into k parts with no part containing a, b, and c such that $a + b = c$?

Schur's Theorem on Squares

Can the set of squares $\{1^2, 2^2, 3^2, \dots\}$ be partitioned into k parts with no part containing a, b, and c such that $a + b = c$?

The case $k = 2$ is already very difficult to determine:

$\{1^2, 2^2, 3^2, 4^2, 6^2, 7^2, 8^2, 9^2, 11^2, 12^2, 13^2, 14^2, 16^2, 17^2, 18^2, 19^2, \dots\}$
$\{5^2, 10^2, 15^2, 20^2, \dots\}$

Schur's Theorem on Squares

Can the set of squares $\{1^2, 2^2, 3^2, \dots\}$ be partitioned into k parts with no part containing a, b, and c such that $a + b = c$?

The case $k = 2$ is already very difficult to determine:

$\{1^2, 2^2, 3^2, 4^2, 6^2, 7^2, 8^2, 9^2, 11^2, 12^2, 13^2, 14^2, 16^2, 17^2, 18^2, 19^2, \dots\}$
$\{5^2, 10^2, 15^2, 20^2, \dots\}$

Partitioning the first thousand squares is easy (even manually).

A computer program can partition the first several thousands squares ($\{1^2, \dots, 7664^2\}$) [Cooper and Overstreet 2015].

Can the infinite set of squares be partitioned into two parts?

Pythagorean Triples Problem [Erdős and Graham 1980]

The Boolean Pythagorean Triples problem is a reformulation of Schur's theorem on squares restricted to two parts:

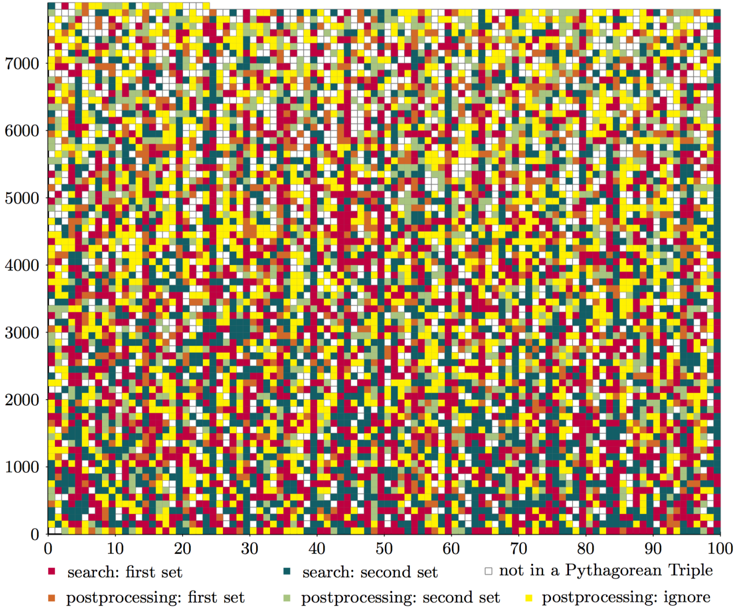
Can the set of natural numbers $\{1, 2, 3, \ldots\}$ be partitioned into two parts such that no part contains a Pythagorean triple ($a, b, c \in \mathbb{N}$ with $a^2 + b^2 = c^2$)?

A partition into two parts is encoded using Boolean variables x_i with $i \in \{1, 2, 3, \ldots\}$ such that $x_i = 1$ $(= 0)$ means that i occurs in the first (second) part. For each Pythagorean triple (a, b, c) two clauses are added: $(x_a \vee x_b \vee x_c) \wedge (\bar{x}_a \vee \bar{x}_b \vee \bar{x}_c)$.

Theorem (Main result via parallel SAT solving)

$[1, 7824]$ *can be partitioned into two parts, such that no part contains a Pythagorean triple. This is impossible for* $[1, 7825]$.

An Extreme Solution (a valid partition of $[1, 7824]$)



Main Contribution

We present a framework that combines, for the first time, all pieces to produce verifiable SAT results for very hard problems.

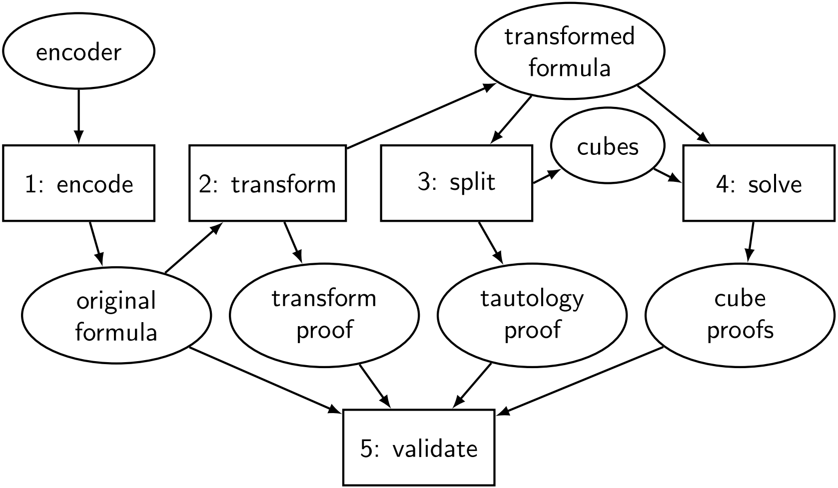
The status quo of using combinatorial solvers and years of computation is arguably intolerable for mathematicians:

- Kouril and Paul [2008] computed the sixth van der Waerden number ($W(2,6) = 1132$) using dedicated hardware without producing a proof.
- McKay's and Radziszowski's big result [1995] in Ramsey Theory ($R(4,5) = 25$) still cannot be reproduced.

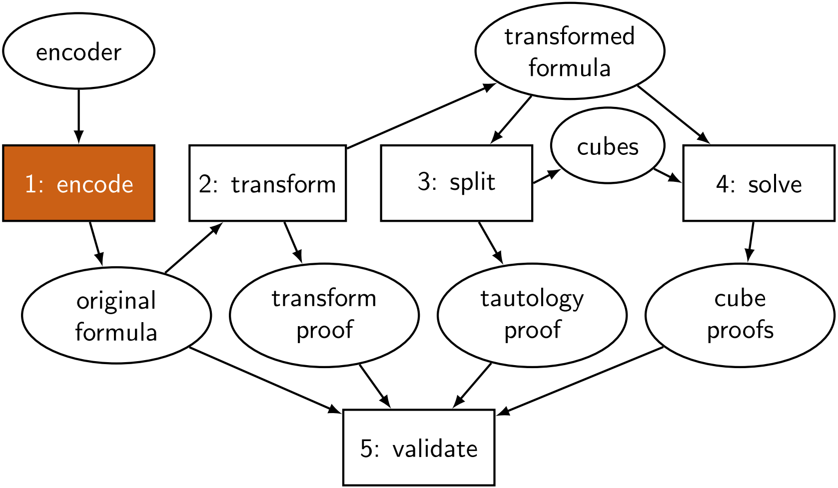
We demonstrate our framework on the Pythagorean triples problem, potentially the hardest problem solved with SAT yet.

Solving Framework for Hard-Combinatorial Problems

Overview of Solving Framework



Overview of Solving Framework: Phase 1



Phase 1: Encode

Input: encoder program

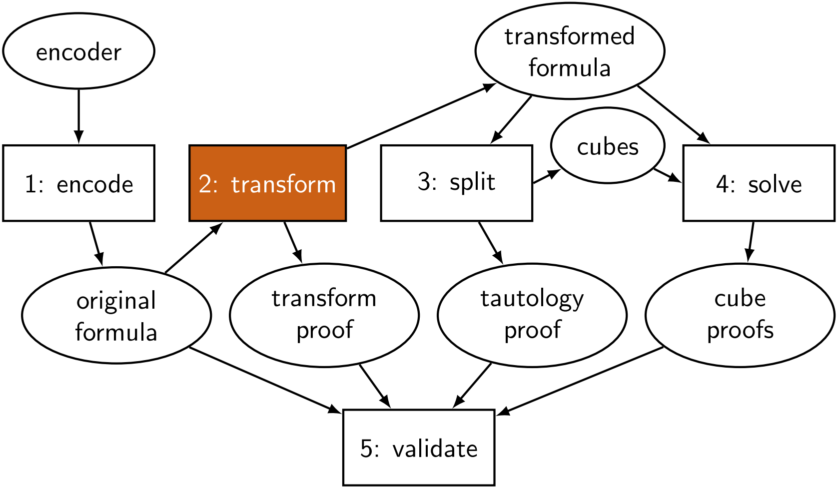
Output: the "original" CNF formula

Goal: make the translation to SAT as simple as possible

```
for (int a = 1; a <= n; a++)
  for (int b = a; b <= n; b++) {
     int c = sqrt (a*a + b*b);
     if ((c <= n) && ((a*a + b*b) == (c*c))) {
       addClause ( a,  b,  c);
       addClause (-a, -b, -c); } }
```

F_{7824} has 6492 (occurring) variables and 18930 clauses, and
F_{7825} has 6494 (occurring) variables and 18944 clauses.

Overview of Solving Framework: Phase 2



Phase 2: Transform

Input: original CNF formula

Output: transformed CNF formula and a transformation proof

Goal: optimize the formula regarding the later (solving) phases

We applied two transformations:

- Pythagorean Triple Elimination removes Pythagorean Triples that contain an element that does not occur in any other Pythagorean Triple, e.g. $3^2 + 4^2 = 5^2$. (till fixpoint)
- Symmetry breaking places the number most frequently occurring in Pythagorean triples (2520) in the first part.

All transformation (pre-processing) techniques can be expressed using RAT steps [Järvisalo, Heule, and Biere 2012].

Phase 2: Blocked Clauses [Kullmann'99]

Definition (Blocking literal)

A literal l in a clause C of a CNF F blocks C w.r.t. F if for every clause $D \in F_{\bar{l}}$, the resolvent $(C \setminus \{l\}) \cup (D \setminus \{\bar{l}\})$ obtained from resolving C and D on l is a tautology.

With respect to a fixed CNF and its clauses we have:

Definition (Blocked clause)

A clause is blocked if it contains a literal that blocks it.

Phase 2: Blocked Clauses [Kullmann'99]

Definition (Blocking literal)

A literal l in a clause C of a CNF F blocks C w.r.t. F if for every clause $D \in F_{\bar{l}}$, the resolvent $(C \setminus \{l\}) \cup (D \setminus \{\bar{l}\})$ obtained from resolving C and D on l is a tautology.

With respect to a fixed CNF and its clauses we have:

Definition (Blocked clause)

A clause is blocked if it contains a literal that blocks it.

Example (Blocking literals and blocked clauses)

Consider the formula $(a \vee b) \wedge (a \vee \bar{b} \vee \bar{c}) \wedge (\bar{a} \vee c)$.
First clause is not blocked.
Second clause is blocked by both a *and* $\bar{c}$.
Third clause is blocked by c.

Phase 2: Blocked Clauses [Kullmann'99]

Definition (Blocking literal)

A literal l in a clause C of a CNF F blocks C w.r.t. F if for every clause $D \in F_{\bar{l}}$, the resolvent $(C \setminus \{l\}) \cup (D \setminus \{\bar{l}\})$ obtained from resolving C and D on l is a tautology.

With respect to a fixed CNF and its clauses we have:

Definition (Blocked clause)

A clause is blocked if it contains a literal that blocks it.

Example (Blocking literals and blocked clauses)

Consider the formula $(a \vee b) \wedge (a \vee \bar{b} \vee \bar{c}) \wedge (\bar{a} \vee c)$.
First clause is not blocked.
Second clause is blocked by both a *and* $\bar{c}$.
Third clause is blocked by c.

Proposition

Removal of an arbitrary blocked clause preserves satisfiability.

Phase 2: Blocked Clause Elimination (BCE)

Definition (BCE)

While a clause C in a formula F is blocked, remove C from F.

Example (BCE)

Consider $(a \vee b) \wedge (a \vee \bar{b} \vee \bar{c}) \wedge (\bar{a} \vee c)$*.*
After removing either $(a \vee \bar{b} \vee \bar{c})$ *or* $(\bar{a} \vee c)$*, the clause* $(a \vee b)$ *becomes blocked (no clause with either* $\bar{b}$ *or* $\bar{a}$*).*
An extreme case in which BCE removes all clauses!

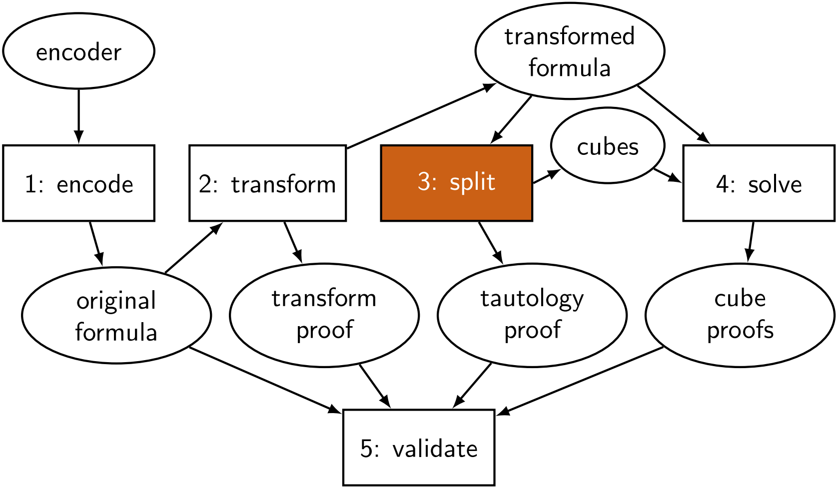
Example (Pythagorean Triples)

The clauses $(x_3 \vee x_4 \vee x_5)$ *and* $(\bar{x}_3 \vee \bar{x}_4 \vee \bar{x}_5)$ *are blocked in* F_{7824} *and* F_{7825} *(actually in any* F_n*).*
BCE (F_{7824}) *has* 3740 *variables and* 14652 *clauses, and*
BCE (F_{7825}) *has* 3745 *variables and* 14672 *clauses.*

BCE can simulate many high-level reasoning techniques.

[Järvisalo, Biere, and Heule 2010]

Overview of Solving Framework: Phase 3

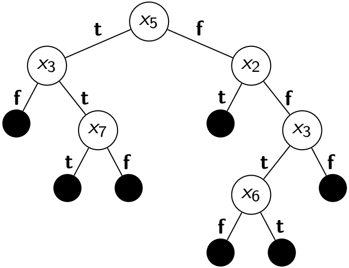


Phase 3: Split

Input: transformed formula

Output: cubes and tautology proof

Goal: partition input in as many subproblems such that total wallclock time is minimal



Two layers of splitting F_{7824}:

- The top level split partitions the transformed formula into exactly a million subproblems;
- Each subproblem is partitioned into tens of thousands of subsubproblems. Total time: 35,000 CPU hours

```
a  5 -3  0
a  5  3  7  0
a  5  3 -7  0
a -5  2  0
a -5 -2  3 -6  0
a -5 -2  3  6  0
a -5 -2 -3  0
```

Cube-and-Conquer [Heule, Kullmann, Wieringa, and Biere 2011]

There exists two main SAT solving paradigms:

- Look-ahead aims to construct a small binary search-tree using (expensive) global heuristics.
- Conflict-driven clause-learning (CDCL) aims to find a short refutation using (cheap) local heuristics.

Cube-and-Conquer [Heule, Kullmann, Wieringa, and Biere 2011]

There exists two main SAT solving paradigms:

- Look-ahead aims to construct a small binary search-tree using (expensive) global heuristics.
- Conflict-driven clause-learning (CDCL) aims to find a short refutation using (cheap) local heuristics.

Combining look-ahead and CDCL, called cube-and-conquer, does not work out of the box. Crucial details are:

- Partition a given formula into many (millions) of subproblems. When just a few subproblems are created, say only 32, the performance could actually decrease.
- Use heuristics to create equally hard subproblems, i.e., not simply using the depth of the search-tree.

Cube-and-conquer solves many hard-combinatorial problems significantly faster than both pure CDCL and pure look-ahead.

Details regarding the splitting heuristics

Splitting based on look-aheads:

- Count the number of assigned variables (large formulas)
- Count the number of clause reductions (medium formulas)
- Sum of the weighted new binary clauses (3-SAT)

Details regarding the splitting heuristics

Splitting based on look-aheads:

- Count the number of assigned variables (large formulas)
- Count the number of clause reductions (medium formulas)
- Sum of the weighted new binary clauses (3-SAT)

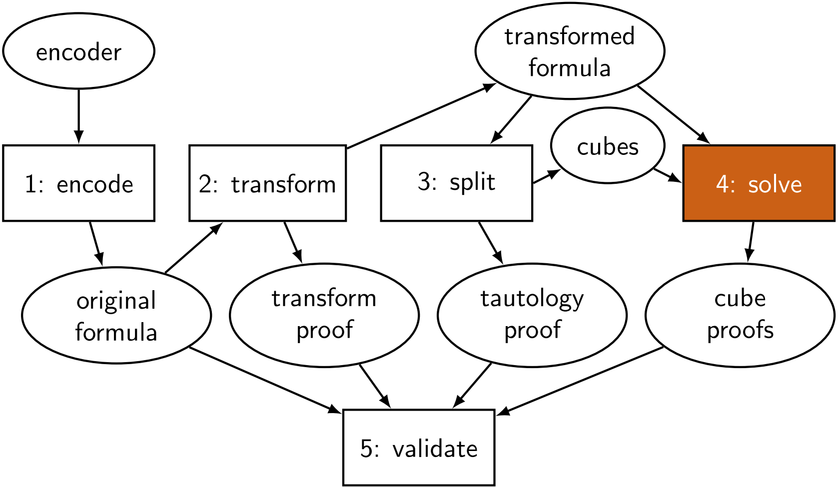
Equations (init, average, and update) for 3-SAT heuristics h_4:

$$h_0(x) = h_0(\bar{x}) = 1 \qquad \mu_i = \frac{1}{2n} \sum_{x \in \mathrm{var}(F)} \left(h_i(x) + h_i(\bar{x})\right)$$

$$h_{i+1}(x) = \max(\alpha, \min(\beta, \sum_{(x \vee y \vee z) \in F} \left(\frac{h_i(\bar{y})}{\mu_i} \cdot \frac{h_i(\bar{z})}{\mu_i}\right) + \gamma \sum_{(x \vee y) \in F} \frac{h_i(\bar{y})}{\mu_i})).$$

Rnd: $\alpha = 0.01, \beta = 25, \gamma = 3.3$ Ptn: $\alpha = 8, \beta = 550, \gamma = 25$

Overview of Solving Framework: Phase 4



Phase 4: Solve

Input: transformed formula and cubes

Output: cube proofs (or satisfying assignment)

Goal: solve (with proof logging) all cubes as fast as possible

Let φ_i be the i^{th} cube with $i \in [1, 1000000]$.

We first solved all $F_{7824} \wedge \varphi_i$, total runtime was 13,000 CPU hours (less than a day on the cluster). One cube is satisfiable.

The backbone of a formula is the set of literals that are assigned to true in all solutions. The backbone of F_{7824} after symmetry breaking consists of 2304 literals, including

- x_{5180} and x_{5865}, while $5180^2 + 5865^2 = 7825^2$
- $\bar{x}_{625}$ and $\bar{x}_{7800}$, while $625^2 + 7800^2 = 7825^2$

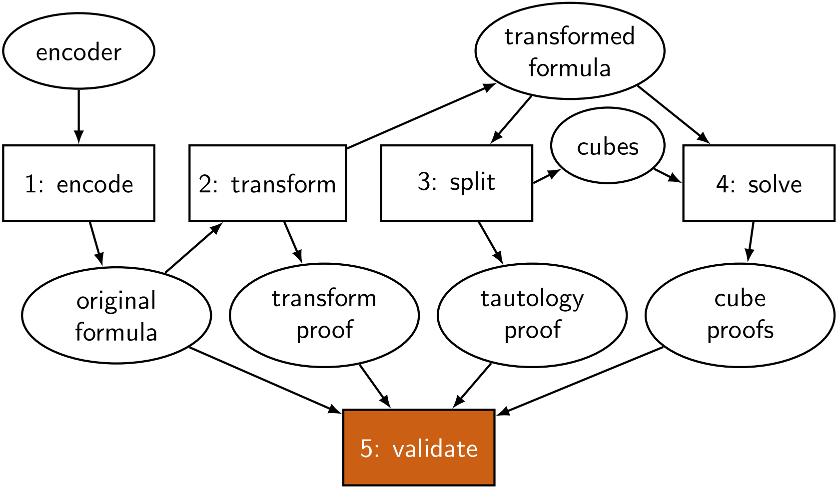
Phase 4: Short history of CDCL improvements

Conflict-driven clause learning (CDCL) has been the dominant SAT solving paradigm and improved significantly in 20 years.

- Invented by Marques-Silva and Sakallah [1997];
- Dedicated data-structure and variable selection heuristics made CDCL really competitive [MoskewiczMZZM 2001];
- Efficient implementation [Een and Sörensson 2003];
- New value selection and rapid restarts make CDCL "local search for UNSAT" [Pipatsrisawat and Darwiche 2007];
- An alternative restart implementation makes ultra rapid restarts optimal [van der Tak, Ramos, and Heule 2011].

Crucial for our framework: reusing the heuristics and learnt clauses while solving similar (sub)problems, known as Incremental SAT Solving [Een and Sörensson 2003]

Overview of Solving Framework: Phase 5



Phase 5: Motivation for validating unsatisfiability proofs

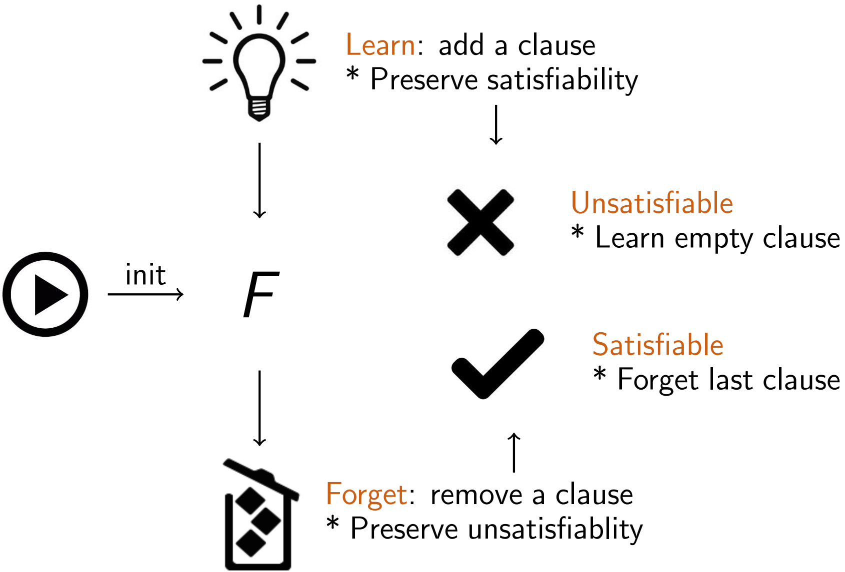
Satisfiability solvers are used in amazing ways...

- Hardware and software verification (Intel and Microsoft)
- Hard-Combinatorial problems:
  - van der Waerden numbers [Dransfield, Marek, and Truszczynski, 2004; Kouril and Paul, 2008]
  - Gardens of Eden in Conway's Game of Life [Hartman, Heule, Kwekkeboom, and Noels, 2013]
  - Erdős Discrepancy Problem [Konev and Lisitsa, 2014]

..., but satisfiability solvers have errors and only return yes/no.

- Documented bugs in SAT, SMT, and QBF solvers [Brummayer and Biere, 2009; Brummayer et al., 2010]
- Implementation errors often imply conceptual errors
- Mathematical results require a stronger justification than a simple yes/no by a solver. UNSAT must be checkable.

Clausal Proof System [Järvisalo, Heule, and Biere 2012]



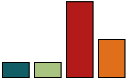
Ideal Properties of a Proof System for SAT Solvers

Easy to Emit

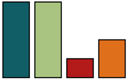
Compact

Checked Efficiently

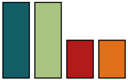
Expressive



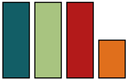
Resolution Proofs
Zhang and Malik, 2003
Van Gelder, 2008; Biere, 2008



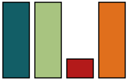
Clausal Proofs
Goldberg and Novikov, 2003
Van Gelder, 2008



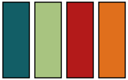
Clausal proofs + clause deletion
Heule, Hunt, Jr., and Wetzler [STVR 2014]



Optimized clausal proof checker
Heule, Hunt, Jr., and Wetzler [FMCAD 2013]



Clausal RAT proofs
Heule, Hunt, Jr., and Wetzler [CADE 2013]



DRAT proofs (RAT + deletion)
Wetzler, Heule, and Hunt, Jr. [SAT 2014]

Ideal Properties of a Proof System for SAT Solvers

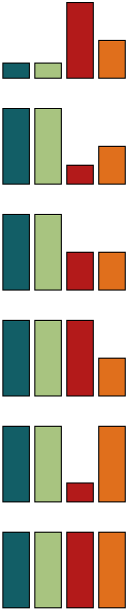
Easy to Emit

Compact

Checked Efficiently

Expressive

Verified



Resolution Proofs
Zhang and Malik, 2003
Van Gelder, 2008; Biere, 2008

Clausal Proofs
Goldberg and Novikov, 2003
Van Gelder, 2008

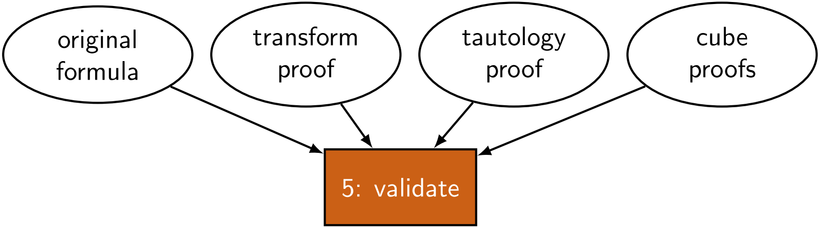
Clausal proofs + clause deletion
Heule, Hunt, Jr., and Wetzler [STVR 2014]

Optimized clausal proof checker
Heule, Hunt, Jr., and Wetzler [FMCAD 2013]

Clausal RAT proofs
Heule, Hunt, Jr., and Wetzler [CADE 2013]

DRAT proofs (RAT + deletion)
Wetzler, Heule, and Hunt, Jr. [SAT 2014]

Phase 5: Validate Pythagorean Triples Proofs.



We check the proofs with the DRAT-trim checker, which has been used to validate the UNSAT results of the international SAT Competitions since 2013.

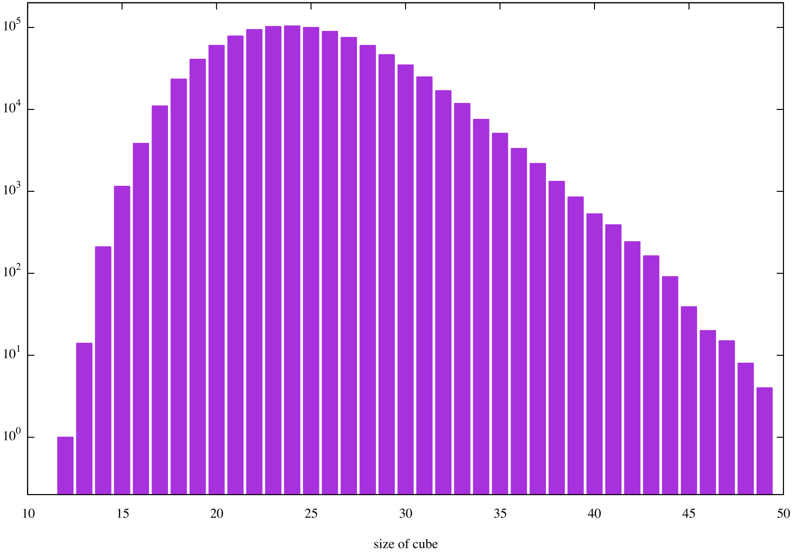
Recently it was shown how to validate DRAT proofs in parallel [Heule and Biere 2015].

The size of the merged proof is almost 200 terabyte and has been validated in 16,000 CPU hours.

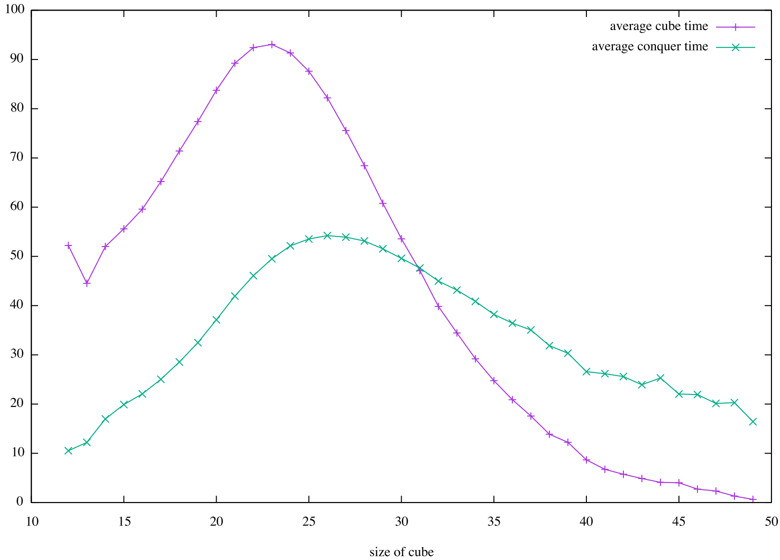
Results

on the million subproblems

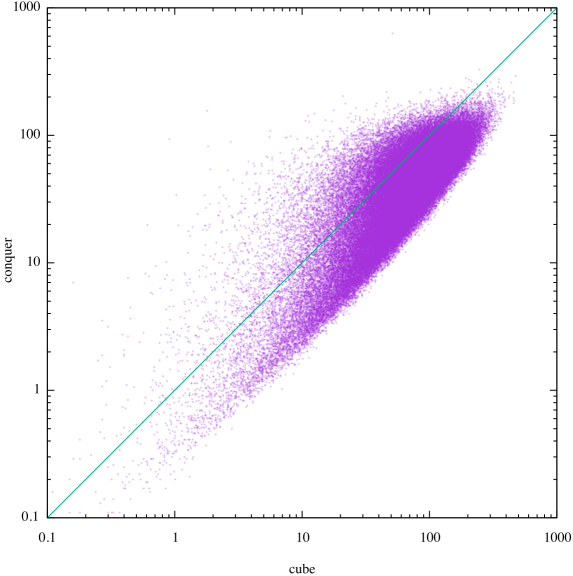
Histogram of Frequency of Cube Sizes



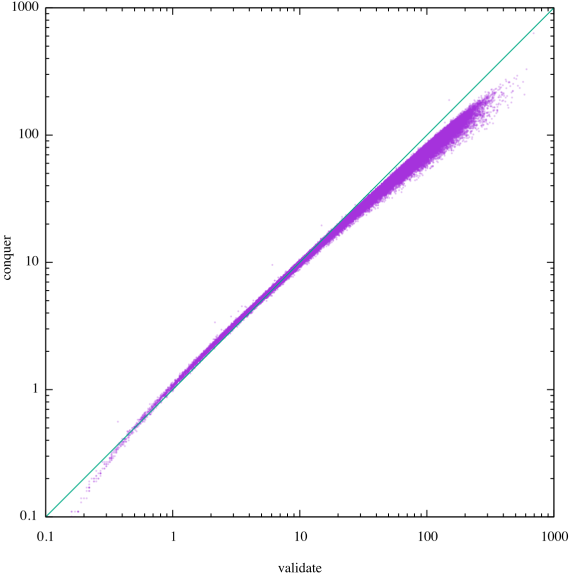
Runtime of Cube and Conquer Averaged per Cube Size



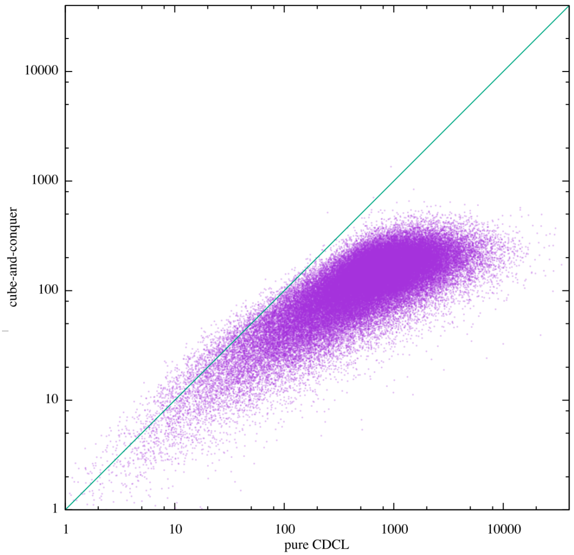
Scatterplot: Cube versus Conquer Runtimes



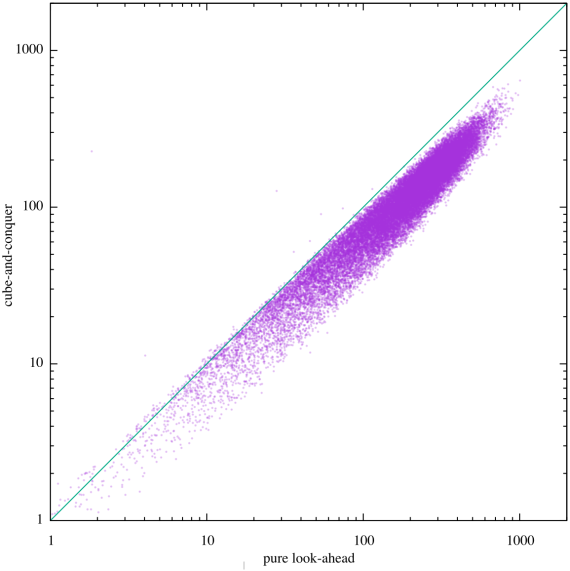
Scatterplot: Validation versus Conquer Runtimes



Scatterplot: CDCL versus Cube-and-Conquer Runtimes

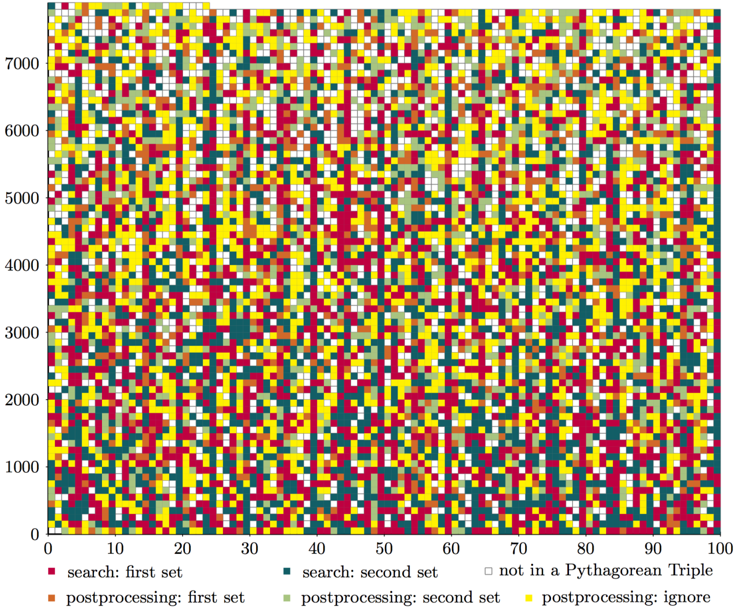


Scatterplot: Look-ahead versus Cube-and-Conquer Runtimes



Conclusions and Future Work

An Extreme Solution (a valid partition of $[1, 7824]$)



Conclusions

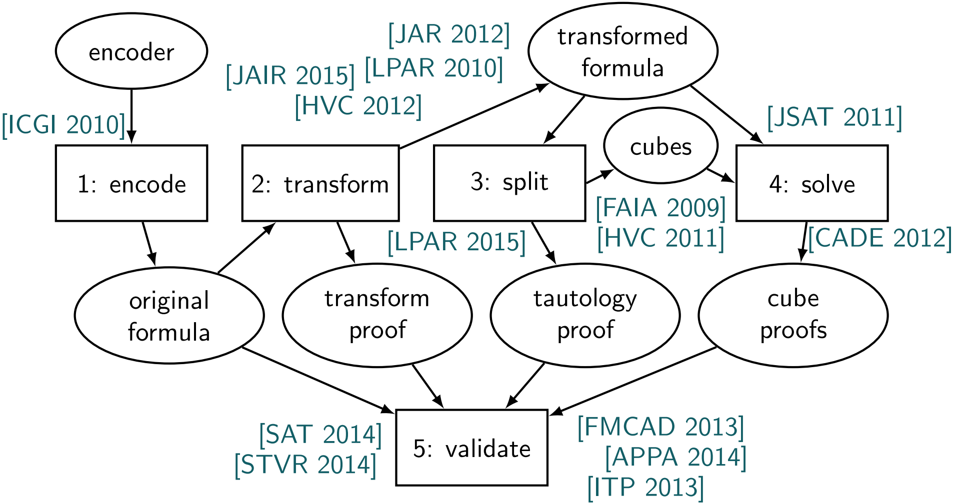
Theorem (Main result)

$[1, 7824]$ *can be partitioned into two parts, such that no part contains a Pythagorean triple. This is impossible for* $[1, 7825]$.

We solved and verified the theorem via SAT solving:

- Cube-and-conquer facilitated massive parallel solving.
- A new look-ahead heuristic was developed to substantially reduce the search space.
- The proof is huge (200 terabyte), but can be compressed to 68 gigabyte (13,000 CPU hours to decompress) and be validated in 16,000 CPU hours.

Heule's Contributions to Solving Framework



Future Directions

Apply our solving framework to other challenges in Ramsey Theory and elsewhere:

- Existing results for which no proof was produced, for example W(2,6) $= 1132$ [Kouril and Paul 2008].
- Century-old open problems appear solvable now, e.g. $S(5)$.

Look-ahead heuristics are crucial and we had to develop dedicated heuristics to solve the Pythagorean triples problem.

- Develop powerful heuristics that work out of the box.
- Alternatively, add heuristic-tuning techniques to the tool chain [Hoos 2012].

Develop a mechanically-verified, fast clausal proof checker.

Future Directions

Apply our solving framework to other challenges in Ramsey Theory and elsewhere:

- Existing results for which no proof was produced, for example W(2,6) = 1132 [Kouril and Paul 2008].
- Century-old open problems appear solvable now, e.g. $S(5)$.

Look-ahead heuristics are crucial and we had to develop dedicated heuristics to solve the Pythagorean triples problem.

- Develop powerful heuristics that work out of the box.
- Alternatively, add heuristic-tuning techniques to the tool chain [Hoos 2012].

Develop a mechanically-verified, fast clausal proof checker.

Thanks!