EPiC Series in Computing

Volume 50, 2017, Pages 1–5

GCAI 2017. 3rd Global Conference on Artificial Intelligence

Everything's Bigger in Texas "The Largest Math Proof Ever"[*]

Marijn J.H. Heule

Department of Computer Science, The University of Texas at Austin

Progress in satisfiability (SAT) solving has enabled answering long-standing open questions in mathematics completely automatically resulting in clever though potentially gigantic proofs. We illustrate the success of this approach by presenting the solution of the Boolean Pythagorean triples problem [7]. We also produced and validated a proof of the solution, which has been called the "largest math proof ever" [12]. The enormous size of the proof is not important. In fact a shorter proof would have been preferable. However, the size shows that automated tools combined with super computing facilitate solving bigger problems. Moreover, the proof of 200 terabytes can now be validated using highly trusted systems [5, 3, 13], demonstrating that we can check the correctness of proofs no matter their size.

The origin of the Boolean Pythagorean triples problem dates back to the work of Schur in the early 20th century [16]. He wondered whether the positive natural numbers can be colored with k colors such that there is no monochromatic solution of a given equation. Let us consider the case of two colors, called red and blue, and the equation $a + b = c$. If we color 1 with red, we have to color 2 with blue due to $1 + 1 = 2$. This forces us to color 4 with red because of $2 + 2 = 4$. After this, 3 must become blue due to $1 + 3 = 4$. But then, no matter if we color 5 with red or blue, we end up with a monochromatic solution of $1 + 4 = 5$ or $2 + 3 = 5$. Schur's Theorem states that every coloring of the positive numbers with finitely many colors results in monochromatic solution of $a + b = c$ [16]. However, for other equations it is possible to avoid a monochromatic solution. For example $a^3 + b^3 = c^3$, because Fermat's Last Theorem [17] states that this equation has no solution in the positive natural numbers. The most famous equation for which this question has remained open for decades is the Pythagorean equation $a^2 + b^2 = c^2$. Ron Graham offered \$100 for the solution of this question since the 1980s.

We answer this question, known as the Boolean Pythagorean triples problem, by encoding it into propositional logic and applying massive parallel SAT solving on the resulting formula. More concretely, we search for the smallest number n such that every coloring of the numbers 1 to n with red and blue results in a monochromatic solution of $a^2 + b^2 = c^2$. For each number i a Boolean variable v_i is introduced. If v_i is assigned to true (or false), then number i is colored red (or blue). For each solution of $a^2 + b^2 = c^2$, the propositional formula contains a clause stating that at least one of a, b, and c must be colored red ($v_a \vee v_b \vee v_c$) and at least one of a, b, and c must be colored blue ($\overline{v}_a \vee \overline{v}_b \vee \overline{v}_c$). This formula is simplified, by removing redundant Pythagorean triples and symmetry breaking, before solving it.

[*]Thanks to the co-authors of the work summarized here: Oliver Kullmann and Victor Marek.

Our main result is a proof that $n = 7825$, which required 40,000 CPU hours of computation, including verification. There exist many red/blue-colorings of the numbers 1 to 7824 without a monochromatic solution of $a^2 + b^2 = c^2$ as shown in Figure 1.

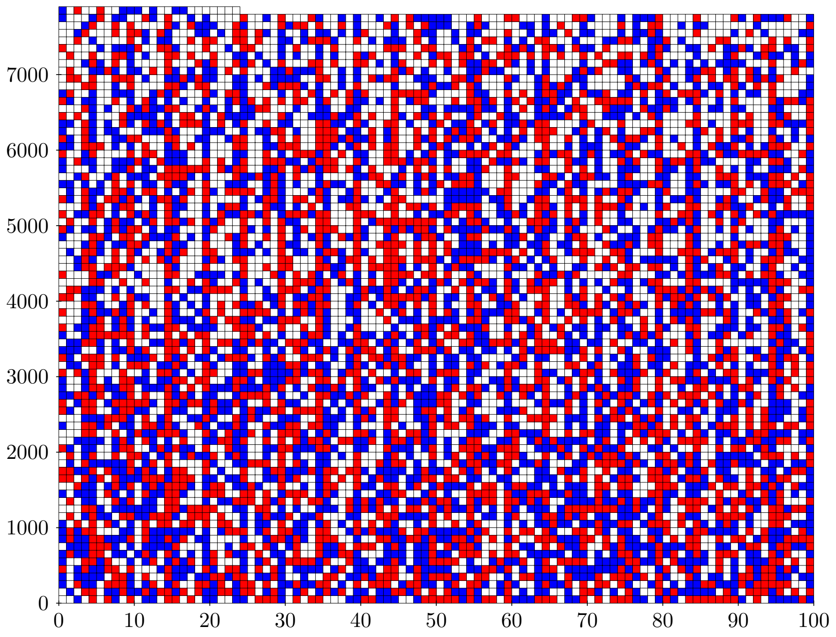

Figure 1: Many red/blue-colorings of the numbers 1 to 7824 without a monochromatic solution of $a^2 + b^2 = c^2$. Numbers in red squares are red, in blue squares are blue, and in white squares can be red or blue. The square in lower left corner represents the number 1 and higher numbers are represented by squares to the right (+1) and up (+100).

Is Mechanized Mathematics Meaningful?

Before describing a mechanized method to answer open questions in mathematics, let us first explore whether this ability is meaningful. Some consider understanding the essence of mathematics [12]. Automatically generated answers and proofs provide mainly facts and few —if any— new insights. However, this point of view ignores many benefits of automatically generated answers. First, the same techniques to answer these questions are absolutely crucial in verification of hardware and software [2, 10]. Advancing these techniques in the context of mathematical problems will result in improvements for industrial applications as well. Second, verifying proofs of unsatisfiability reveals some useful information, such as which parts of the problem are relevant, and therefore does provide some insights [6].

Third, there appears to be a stubborn assumption that somehow, somewhere there exist compact proofs of automatically solved problems, but that nobody has been able to find such a proof yet. This assumption is likely false for many problems. For example, the proofs of small Ramsey numbers rely typically on a large case split [4]. Although our 200 terabytes proof will not be the smallest proof of the Pythagorean triples problem, there may not exist a proof that is shorter than a terabyte. Studying problems such as the Pythagorean triples problem may in fact shed some light on one of the most important questions in computer science: What makes a problem hard?

Last but not least, we need answers to questions that humans cannot deal with. We simply cannot afford having only those answers that mathematicians can provide. Mechanized mathematics is useful even if it only produces facts, such as our answer $n = 7825$, and no understanding. Consider for example two designs of a complex critical system. For one design we can prove —fully automatically— that it is safe, while for the other one we cannot produce such a proof with any available method. One would opt for the first design because we know it is safe even though we don't know why. Increasingly, such scenarios are becoming a daily reality. This making our ability to determine facts like safety very meaningful. By all means, let us not underestimate the value of hard facts.

The Hidden Strength of Cube-and-Conquer

We used the *cube-and-conquer* method [8] to solve the Pythagorean triples problem [7] as it is arguably the most effective method for solving very hard combinatorial problems, such as the Erdős discrepancy problem [11]. Cube-and-conquer is a hybrid parallel SAT solving paradigm that combines *look-ahead* techniques [9] with *conflict-driven clause learning* (CDCL) [14]: Look-ahead techniques are used for splitting a given problem into many millions of subproblems which are then solved with CDCL solvers. CDCL is the most well-known SAT solving paradigm, which allows solving benchmarks from industrial applications with millions of variables and clauses. Since the subproblems are independent, they can be easily solved with CDCL in parallel without requiring communication.

The aim of look-ahead techniques is to find variable assignments that simplify a formula as much as possible. This is achieved with so-called *look-aheads*: A look-ahead on a literal l with respect to a formula F first assigns l to true and then simplifies F to obtain a formula F'. After this, it determines a heuristic value by computing the "difference" between F and F'. A variable v is considered useful for splitting a formula F if the look-aheads on both v and $\overline{v}$ have a high heuristic value. Typically, look-ahead techniques select the variable v for which the product of the heuristic values of v and $\overline{v}$ is the largest. This variable is used to split F into two subformulas: One for which v is true, and one for which v is false.

We used an expensive, but highly effective measurement to compute the difference between F and F'. Notice that F' consists of clauses that occur in F and binary clauses that originate from ternary clauses in F since all unit clauses are simplified away. We denote with $F' \setminus F$ the set of binary clauses. Each clause in $(l_1 \vee l_2) \in F' \setminus F$ is weighted based on the occurrences of the literals $\overline{l}_1$ and $\overline{l}_2$ in F [15]. The heuristic value of looking ahead on v is the weighted sum of all binary clauses in $F' \setminus F$ with F' being the simplified formula after the look-ahead.

Cube-and-conquer is not only useful for partitioning a hard problem into many subproblems that can be solved in parallel, but can also boost performance of solving a problem on a single core. Let N be the number of subproblems. A low value of N indicates that the problem is split into a low number of subproblems, meaning that it is mainly solved with CDCL ($N = 1$ means pure CDCL) while a larger value indicates a more extensive splitting based on look-aheads.

If we experiment with different values for N when trying to solve a problem on a single core, we can observe an interesting pattern: For low values of N, an increase of N leads to an increase of the total runtime—apparently some subproblems are as hard as the original one. If we increase N further, the total runtime starts to decrease and at some point it can even become significantly smaller compared to solving the problem with CDCL alone (again running both on a single core). However, when N becomes really large, the runtime increases again. At this point, splitting starts to dominate the total costs. Figure 2 shows this pattern on a subproblem of Schur number five: What is the largest n such there exists a 5-coloring of 1 to n with a monochromatic solution of $a + b = c$. For this subproblem, the optimal value for N is around 10 000.

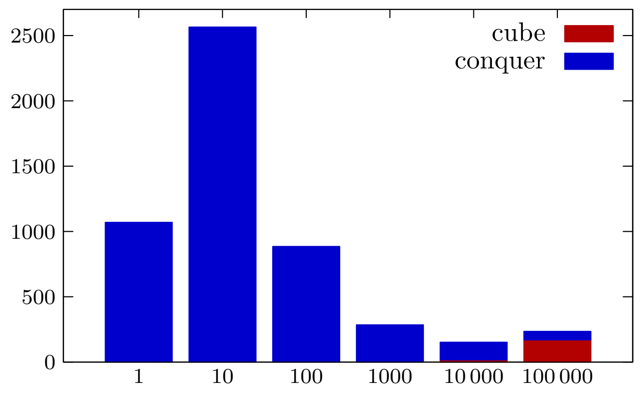

Figure 2: Comparison of the total runtime in seconds (y-axis) for solving a subproblem of Schur number five using different numbers of cubes (x-axis) on a single core (no parallelism).

Proofs are Needed!

If it takes a computer several CPU years to solve a problem, it is only natural to question the correctness of the alleged solution. It is easy to check whether a given coloring has no monochromatic solution of a given equation. Figure 1 offers such red/blue-colorings of the equation $a^2+b^2 = c^2$ with $a, b, c \leq 7824$. However, checking the claim that all red/blue-colorings of the numbers 1 to 7825 result in a monochromatic Pythagorean triple is more complicated. The number of such red/blue-colorings is 2^{7825}, larger than the number of particles in the universe. Our SAT solving approach reduced the number of colorings to be checked to roughly 2^{40}. The question arises whether these 2^{40} colorings cover the entire search space.

If the essence of the solution of the Boolean Pythagorean triples problem would be the number "7825", the numerical information about the point where monochromatic Pythagorean triples are unavoidable, then naturally there would be less pressure on actually having a verifiable proof, and one would just wait for independent re-computations, as is the case currently with all really big computations outside of the SAT-realm — only here we have the ability to extract really big proofs. But in this case, there is currently no independent "mathematical" existence proof. Without our result it wouldn't even be known whether there exists that turning point at all, i.e., whether all positive numbers can be colored with red and blue without resulting in a monochromatic Pythagorean triple. A real proof is needed.

We extracted a proof from solver runs and extended it with a proof of the simplification of the original propositional formula along with a proof that the subproblems cover the entire search space. The size of the combined proof is 200 terabytes in size. However, it has been validated using three formally verified checkers — two of these efforts were by independent groups [5, 3, 13]. Our ability to produce such proofs and certify them using theorem provers provides high confidence in the correctness of our result.

Acknowledgements

The author is supported by the National Science Foundation under grant CCF-1526760 and by AFRL Award FA8750-15-2-0096. The author acknowledges the Texas Advanced Computing Center (TACC) at The University of Texas at Austin for providing HPC resources that have contributed to the research results reported within this paper.

References

[1] Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, February 2009.

[2] Edmund M. Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19(1):7–34, 2001.

[3] Luís Cruz-Filipe and Peter Schneider-Kamp. Formally proving the boolean triples conjecture. In Thomas Eiter and David Sands, editors, *Proceedings of LPAR-21*, volume 46 of *EPiC Series in Computing*, pages 509–522. EasyChair Publications, 2017.

[4] Ronald L. Graham, Bruce L. Rothschild, and Joel H. Spencer. *Ramsey Theory, 2nd Edition*. Wiley, 1990.

[5] Marijn J. H. Heule, Warren A. Hunt Jr, Matt Kaufmann, and Nathan Wetzler. *Efficient, Verified Checking of Propositional Proofs*, pages 269–284. Springer International Publishing, Cham, 2017.

[6] Marijn J. H. Heule, Warren A. Hunt, Jr, and Nathan Wetzler. Trimming while checking clausal proofs. In *Formal Methods in Computer-Aided Design*, pages 181–188. IEEE, 2013.

[7] Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving and verifying the Boolean Pythagorean Triples problem via Cube-and-Conquer. In *Theory and Applications of Satisfiability Testing – SAT 2016*, pages 228–245. Springer, 2016.

[8] Marijn J. H. Heule, Oliver Kullmann, Siert Wieringa, and Armin Biere. Cube and conquer: Guiding CDCL SAT solvers by lookaheads. In *7th International Haifa Verification Conference – HVC 2011*, pages 50–65. Springer, 2012.

[9] Marijn J. H. Heule and Hans van Maaren. *Look-Ahead Based SAT Solvers*, chapter 5, pages 155–184. Volume 185 of Biere et al. [1], February 2009.

[10] Franjo Ivančić, Zijiang Yang, Malay K. Ganai, Aarti Gupta, and Pranav Ashar. Efficient SAT-based bounded model checking for software verification. *Theoretical Computer Science*, 404(3):256–274, 2008.

[11] Boris Konev and Alexei Lisitsa. Computer-aided proof of Erdős discrepancy properties. *Artificial Intelligence*, 224(C):103–118, July 2015.

[12] Evelyn Lamb. Maths proof smashes size record: Supercomputer produces a 200-terabyte proof – but is it really mathematics? *Nature*, 534:17–18, June 2016.

[13] Peter Lammich. Efficient verified (un)sat certificate checking. In *Automated Deduction – CADE 26*, pages 237–254. Springer, 2017.

[14] Joao P. Marques-Silva, Ines Lynce, and Sharad Malik. *Conflict-Driven Clause Learning SAT Solvers*, chapter 4, pages 131–153. Volume 185 of Biere et al. [1], February 2009.

[15] Sid Mijnders, Boris de Wilde, and Marijn J. H. Heule. Symbiosis of search and heuristics for random 3-SAT. In David Mitchell and Eugenia Ternovska, editors, *Third International Workshop on Logic and Search (LaSh 2010)*, 2010.

[16] Issai Schur. Über die Kongruenz $x^m + y^m = z^m \pmod{p}$. *Jahresbericht der Deutschen Mathematikervereinigung*, 25:114–117, 1917.

[17] Andrew Wiles. Modular elliptic curves and fermat's last theorem. *Annals of Mathematics*, 141(3):443–551, 1995.