EasyChair Preprint

№ 9354

SAT is as Hard as Solving Homogeneous Diophantine Equation of Degree Two

Frank Vega

March 7, 2023

SAT is as hard as solving Homogeneous Diophantine Equation of Degree Two

Frank Vega
CopSonic, 1471 Route de Saint-Nauphary 82000 Montauban, France

Abstract

P versus NP is considered as one of the most important open problems in computer science. This consists in knowing the answer of the following question: Is P equal to NP? It was essentially mentioned in 1955 from a letter written by John Nash to the United States National Security Agency. However, a precise statement of the P versus NP problem was introduced independently by Stephen Cook and Leonid Levin. Since that date, all efforts to find a proof for this problem have failed. In mathematics, a Diophantine equation is a polynomial equation, usually involving two or more unknowns, such that the only solutions of interest are the integer ones. A homogeneous Diophantine equation is a Diophantine equation that is defined by a homogeneous polynomial. Solving a homogeneous Diophantine equation is generally a very difficult problem. However, homogeneous Diophantine equations of degree two are considered easier to solve. We prove that this decision problem is actually in NP-complete under the constraints that all solutions contain only positive integers which are actually residues of modulo a single positive integer.

2012 ACM Subject Classification Theory of computation → Complexity classes; Theory of computation → Problems, reductions and completeness

Keywords and phrases complexity classes, boolean formula, completeness, polynomial time

1 Introduction

Let $\{0,1\}^*$ be the infinite set of binary strings, we say that a language $L_1 \subseteq \{0,1\}^*$ is polynomial time reducible to a language $L_2 \subseteq \{0,1\}^*$, written $L_1 \leq_p L_2$, if there is a polynomial time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for all $x \in \{0,1\}^*$:

$x \in L_1$ *if and only if* $f(x) \in L_2$.

An important complexity class is *NP–complete* [5]. If L_1 is a language such that $L' \leq_p L_1$ for some $L' \in$ *NP–complete*, then L_1 is *NP–hard* [3]. Moreover, if $L_1 \in NP$, then $L_1 \in$ *NP–complete* [3]. A principal *NP–complete* problem is SAT [5]. An instance of SAT is a Boolean formula ϕ which is composed of:

1. Boolean variables: $x_1, x_2, \ldots, x_n$;
2. Boolean connectives: Any Boolean function with one or two inputs and one output, such as $\wedge$(AND), $\vee$(OR), $\neg$(NOT), $\Rightarrow$(implication), $\Leftrightarrow$(if and only if);
3. and parentheses.

A truth assignment for a Boolean formula ϕ is a set of values for the variables in ϕ. A satisfying truth assignment is a truth assignment that causes ϕ to be evaluated as true. A Boolean formula with a satisfying truth assignment is satisfiable. The problem SAT asks whether a given Boolean formula is satisfiable [5]. We define a CNF Boolean formula using the following terms:

A literal in a Boolean formula is an occurrence of a variable or its negation [3]. A Boolean formula is in conjunctive normal form, or CNF, if it is expressed as an AND of clauses, each of which is the OR of one or more literals [3]. A Boolean formula is in 3-conjunctive normal

form or $3CNF$, if each clause has exactly three distinct literals [3]. For example, the Boolean formula:

$$(x_1 \vee \neg x_1 \vee \neg x_2) \wedge (x_3 \vee x_2 \vee x_4) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4)$$

is in $3CNF$. The first of its three clauses is $(x_1 \vee \neg x_1 \vee \neg x_2)$, which contains the three literals x_1, $\neg x_1$, and $\neg x_2$. In computational complexity, not-all-equal 3-satisfiability (NAE–$3SAT$) is an *NP–complete* variant of SAT over $3CNF$ Boolean formulas. NAE–$3SAT$ consists in knowing whether a Boolean formula ϕ in $3CNF$ has a truth assignment such that for each clause at least one literal is true and at least one literal is false [5]. NAE–$3SAT$ remains *NP–complete* when all clauses are monotone (meaning that variables are never negated), by Schaefer's dichotomy theorem [11]. We know that the variant of $XOR\ 2SAT$ that uses the logic operator $\oplus$ (XOR) instead of $\vee$ (OR) within the clauses of $2CNF$ Boolean formulas can be decided in polynomial time [7], [9]. Despite of its feasible computation, we announce another problem very similar to this one but in *NP–complete*.

▶ **Definition 1.** ***Monotone Exact XOR 2SAT (EX2SAT)***
INSTANCE: A Boolean formula φ in $2CNF$ with monotone clauses using logic operators $\oplus$ and a positive integer K.
QUESTION: Does φ has a truth assignment such that there are exactly K satisfied clauses?

▶ **Theorem 2.** *$EX2SAT \in$ NP–complete.*

A homogeneous Diophantine equation is a Diophantine equation that is defined by a polynomial whose nonzero terms all have the same degree [4]. The degree of a term is the sum of the exponents of the variables that appear in it, and thus is a non-negative integer [4]. In a general homogeneous Diophantine equations of degree two, we can reject an instance when there is no solution reducing the equation modulo p. We define a decision problem:

▶ **Definition 3.** ***ZERO-ONE Homogeneous Diophantine Equation (HDE)***
INSTANCE: A homogeneous Diophantine equation of degree two

$$P(x_1, x_2, \ldots, x_n) = B$$

with the unknowns $x_1, x_2, \ldots, x_n$ and a positive integer B.
QUESTION: Does $P(x_1, x_2, \ldots, x_n) = B$ has a solution $u_1, u_2, \ldots, u_n$ on $\{0,1\}^n$?

▶ **Theorem 4.** *$HDE \in$ NP–complete.*

▶ **Definition 5.** ***Bounded Homogeneous Diophantine Equation (BHDE)***
INSTANCE: A homogeneous Diophantine equation of degree two

$$P(x_1, x_2, \ldots, x_n) = B$$

with the unknowns $x_1, x_2, \ldots, x_n$ and two positive integers B, M.
QUESTION: Does $P(x_1, x_2, \ldots, x_n) = B$ has a solution $u_1, u_2, \ldots, u_n$ on integers such that $0 \leq u_i < M$ for every $1 \leq i \leq n$?

▶ **Theorem 6.** *$BHDE \in$ NP–complete.*

2 Proof of Theorem 2

Proof. Let's take a Boolean formula ϕ in $3CNF$ with n variables and m clauses when all clauses are monotone. We iterate for each clause $c_i = (a \vee b \vee c)$ and create the conjunctive normal form formula

$$d_i = (a \oplus a_i) \wedge (b \oplus b_i) \wedge (c \oplus c_i) \wedge (a_i \oplus b_i) \wedge (a_i \oplus c_i) \wedge (b_i \oplus c_i)$$

where a_i, b_i, c_i are new variables linked to the clause c_i in ϕ. Note that, the clause c_i has exactly at least one true literal and at least one false literal if and only if d_i has exactly one unsatisfied clause. Finally, we obtain a new formula

$$\varphi = d_1 \wedge d_2 \wedge d_3 \wedge \ldots \wedge d_m$$

where there is not any repeated clause. In this way, we make a polynomial time reduction from ϕ in *NAE–3SAT* to $(\varphi, 5 \cdot m)$ in $EX2SAT$. Certainly, $\phi \in$ *NAE–3SAT* if and only if $(\varphi, 5 \cdot m) \in EX2SAT$, where the new instance $(\varphi, 5 \cdot m)$ is polynomially bounded by the bit-length of ϕ. At the end, we see that $EX2SAT$ is trivially in NP since we could check when there are exactly K satisfied clauses for a single truth assignment in polynomial time. ◀

3 Proof of Theorem 4

Proof. Let's take a Boolean formula φ in $XOR\ 2CNF$ with n variables and m clauses when all clauses are monotone and a positive integer K. We iterate for each clause $c_i = (a \oplus b)$ and create the Homogeneous Diophantine Polynomial of degree two

$$P(x_a, x_b) = x_a^2 - 2 \cdot x_a \cdot x_b + x_b^2$$

where x_a, x_b are variables linked to the positive literals a, b in the Boolean formula φ. When the literals a, b are evaluated in $\{false, true\}$, then we assign the respective values $\{0, 1\}$ to the variables x_a, x_b (1 if it is true and 0 otherwise). Note that, the clause c_i is satisfied if and only if $P(x_a, x_b) = 1$ (otherwise $P(x_a, x_b) = 0$). Finally, we obtain a polynomial

$$P(x_1, x_2, \ldots, x_n) = P(x_a, x_b) + P(x_c, x_d) + \ldots + P(x_e, x_f)$$

that is a Homogeneous Diophantine Polynomial of degree two. Indeed, K satisfied clauses in φ for a truth assignment correspond to K distinct small pieces $P(x_i, x_j)$ of the Homogeneous Diophantine Polynomial of degree two equal to 1 after its evaluation on x_i, x_j. In this way, we make a polynomial time reduction from (φ, K) in $EX2SAT$ to $(P(x_1, x_2, \ldots, x_n), K)$ in HDE. Certainly, $(\varphi, K) \in EX2SAT$ if and only if $(P(x_1, x_2, \ldots, x_n), K) \in HDE$, where the new instance $(P(x_1, x_2, \ldots, x_n), K)$ is polynomially bounded by the bit-length of (φ, K). At the end, we see that HDE is trivially in NP since we could check whether an evaluation of $x_1, x_2, \ldots, x_n$ in the solution $u_1, u_2, \ldots, u_n$ on $\{0, 1\}^n$ is equal to K in polynomial time. ◀

4 Proof of Theorem 6

Proof. This is trivial since we can make a polynomial time reduction from $(P(x_1, x_2, \ldots, x_n), B)$ in HDE to $(P(x_1, x_2, \ldots, x_n), B, 2)$ in $BHDE$ (i.e. using $M = 2$). Due to HDE is in *NP–complete*, then $BHDE$ is in *NP–hard*. Finally, we know that $BHDE$ is in NP. Consequently, $BHDE$ is also in *NP–complete*. ◀

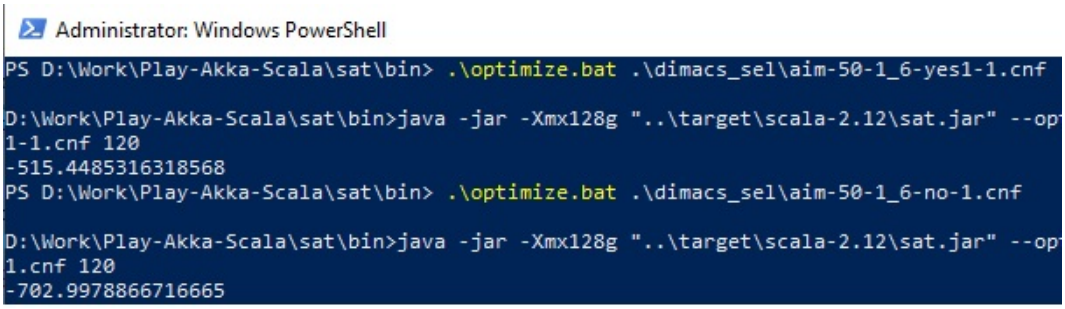

Figure 1 We compare exactly two DIMACS files with the same number of variables (50) and clauses (80). One Boolean formula is SATISFIABLE (a yes instance) and the other one is UNSATISFIABLE (a no instance). Note that, the "yes" instance has a **maximum value** regarding the "no" instance where those values were calculated by two Julia scripts. This is supported on Linux and Windows.

5 Conclusions

We show the *NP–completeness* in the problem of deciding whether a homogeneous Diophantine equations of degree 2 has a solution residues of modulo a single positive integer. The whole reduction algorithm runs in polynomial time since we can reduce SAT to $NAE\text{–}3SAT$ in a feasible way: This is a trivial and well-known polynomial time reduction [11]. We could transform this algorithm to an optimization problem that is algorithmically practical solving P-Selective Sets on SAT instances that works better when both formulas have approximately the same number of variables and clauses [6]. The whole algorithm is based on the problem of quadratic optimization without constraints which is feasible when we do not restrict the variables to be integers [2]. Certainly, the conversion of a clause $c_i = (a \oplus b)$ into a small piece of Homogeneous Diophantine Polynomial of degree two on residues of modulo 2

$$P(x_a, x_b) = x_a^2 - 2 \cdot x_a \cdot x_b + x_b^2 = (x_a - x_b)^2$$

works for integers $x_a, x_b \in \{0, 1\}$ and real values $0 \leq x_a \leq 1$ and $0 \leq x_b \leq 1$ at the same time, since the expression $(x_a - x_b)^2$ is maximized to the optimal value of 1 only on solutions in $\{0, 1\}$ for both cases according to our described and explained reduction.

We implement a software solution to this problem, specifically dealing with the problem $P\text{–}Selective\text{–}SAT$, that runs feasible using the algorithm PRAXIS (PRincipal AXIS): PRAXIS is a gradient-free local optimization via the "principal-axis method" of Richard Brent [2]. We use the Scala programming language for making the whole reduction from SAT to $NAE\text{–}3SAT$ in a simple way [8]. Finally, we generate and run the Julia scripts that approximate using PRAXIS to a very good optimization value for deciding which is one is the satisfiable one between two Boolean formulas with approximately the same number of variables and clauses [1]. The whole project was developed by the author and it is available in GitHub on MIT License [12]. Note that, the performance of the Julia scripts is much better in our application when we re-run for at least a second time (we can obtain a wrong or worsen value from the first time, but a too much better result as long as you increase the number of running times: remember we are using also a local optimization for bringing a faster and feasible response) [10]. We use the DIMACS files as input (`http://www.satcompetition.org/2009/format-benchmarks2009.html`) and test it from the well-known dataset SAT Benchmarks (`https://www.cs.ubc.ca/~hoos/SATLIB/Benchmarks/SAT/DIMACS/AIM/descr.html`). See an example in Figure 1.

References

1 Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017. doi:10.1137/141000671.

2 Richard P Brent. *Algorithms for minimization without derivatives*. Courier Corporation, 2013.

3 Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

4 David A Cox, John Little, and Donal O'shea. *Using algebraic geometry*, volume 185. Springer Science & Business Media, 2006.

5 Michael R Garey and David S Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco: W. H. Freeman and Company, 1 edition, 1979.

6 Edith Hemaspaandra, Ashish V Naik, Mitsunori Ogihara, and Alan L Selman. P-selective sets and reducing search to decision vs self-reducibility. *Journal of Computer and System Sciences*, 53(2):194–209, 1996. doi:10.1006/jcss.1996.0061.

7 Neil D Jones, Y Edmund Lien, and William T Laaser. New problems complete for nondeterministic log space. *Mathematical systems theory*, 10(1):1–17, 1976. doi:10.1007/BF01683259.

8 Martin Odersky, Lex Spoon, and Bill Venners. *Programming in scala*. Artima Inc, 2008.

9 Omer Reingold. Undirected connectivity in log-space. *Journal of the ACM (JACM)*, 55(4):1–24, 2008. doi:10.1145/1391289.1391291.

10 Neven Sajko. Why does my program's performance vary so much from run to run? Can it be fixed? (Julia Community). https://discourse.julialang.org/t/why-does-my-programs-performance-vary-so-much-from-run-to-run-can-it-be-fixed/53678, January 2021. Accessed: 2023-03-05.

11 Thomas J Schaefer. The complexity of satisfiability problems. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 216–226, 1978.

12 Frank Vega. SAT Selective Solver (Github repository). https://github.com/frankvegadelgado/sat, March 2023. commit=0b14401eca6b90ccf0e8d2debca2ccd2a3e02bfa.