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

Frank Vega

September 11, 2023

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

Frank Vega
NataSquad, 10 rue de la Paix 75002 Paris, France

Abstract

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 2. In addition, we show its optimization variant is equivalent to solving a problem of quadratic polynomial optimization without the restriction that the variables must be necessarily integers. This means that this optimization problem can be solved over the domains of real numbers with at most quadratic exponent and so, we expect these pre-conditions can turn this problem to be feasibly solved.

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

In 1936, Turing developed his theoretical computational model [15]. The deterministic and nondeterministic Turing machines have become in two of the most important definitions related to this theoretical model for computation [15]. A deterministic Turing machine has only one next action for each step defined in its program or transition function [15]. A nondeterministic Turing machine could contain more than one action defined for each step of its program, where this one is no longer a function, but a relation [15].

Let Σ be a finite alphabet with at least two elements, and let Σ^* be the set of finite strings over Σ [1]. A Turing machine M has an associated input alphabet Σ [1]. For each string w in Σ^* there is a computation associated with M on input w [1]. We say that M accepts w if this computation terminates in the accepting state, that is $M(w) = \text{“}yes\text{”}$ [1]. Note that, M fails to accept w either if this computation ends in the rejecting state, that is $M(w) = \text{“}no\text{”}$, or if the computation fails to terminate, or the computation ends in the halting state with some output, that is $M(w) = y$ (when M outputs the string y on the input w) [1].

Another relevant advance in the last century has been the definition of a complexity class. A language over an alphabet is any set of strings made up of symbols from that alphabet [5]. A complexity class is a set of problems, which are represented as a language, grouped by measures such as the running time, memory, etc [5]. The language accepted by a Turing machine M, denoted $L(M)$, has an associated alphabet Σ and is defined by:

$$L(M) = \{w \in \Sigma^* : M(w) = \text{“}yes\text{”}\}.$$

Moreover, $L(M)$ is decided by M, when $w \notin L(M)$ if and only if $M(w) = \text{“}no\text{”}$ [5]. We denote by $t_M(w)$ the number of steps in the computation of M on input w [1]. For $n \in \mathbb{N}$ we denote by $T_M(n)$ the worst case run time of M; that is:

$$T_M(n) = max\{t_M(w) : w \in \Sigma^n\}$$

where Σ^n is the set of all strings over Σ of length n [1]. We say that M runs in polynomial time if there is a constant k such that for all n, $T_M(n) \leq n^k + k$ [1]. In other words, this means the language $L(M)$ can be decided by the Turing machine M in polynomial time. Therefore, P is the complexity class of languages that can be decided by deterministic Turing machines in polynomial time [5]. A verifier for a language L_1 is a deterministic Turing machine M, where:

$$L_1 = \{w : M(w, u) = \text{“}yes\text{” } \textit{for some string } u\}.$$

We measure the time of a verifier only in terms of the length of w, so a polynomial time verifier runs in polynomial time in the length of w [1]. A verifier uses additional information, represented by the string u, to verify that a string w is a member of L_1. This information is called certificate. NP is the complexity class of languages defined by polynomial time verifiers [11].

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 $x \in \{0,1\}^*$:

$$x \in L_1 \textit{ if and only if } f(x) \in L_2.$$

An important complexity class is *NP–complete* [8]. 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* [5]. Moreover, if $L_1 \in NP$, then $L_1 \in$ *NP–complete* [5]. A principal *NP–complete* problem is SAT [8]. 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), $\rightharpoondown$(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 [8]. 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 [5]. 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 [5]. A Boolean formula is in 3-conjunctive normal form or $3CNF$, if each clause has exactly three distinct literals [5]. For example, the Boolean formula:

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

is in $3CNF$. The first of its three clauses is $(x_1 \vee \rightharpoondown x_1 \vee \rightharpoondown x_2)$, which contains the three literals x_1, $\rightharpoondown x_1$, and $\rightharpoondown x_2$.

P-Selective Sets is the class of decision problems for which there's a polynomial-time algorithm with the following property. Whenever it's given two instances, a "*yes*" and a "*no*" instance, the algorithm can always decide which is the "*yes*" instance. Defined in [14], where it was also shown that if NP is contained in P-Selective Sets then $P = NP$. P-Selective Sets have been studied in relation to the complexity class NP and the satisfiability problem (SAT). Using all this knowledge as background, then we may be able to prove our main results.

2 Issues and Motivation

We show the *NP–completeness* in the problem of deciding whether a homogeneous Diophantine equations of degree 2 has a solution residues of modulo 2. The whole reduction algorithm runs in polynomial time since we can reduce SAT to *NAE–3SAT* in a feasible way: This is a trivial and well-known polynomial time reduction [13]. We could transform this algorithm to a quadratic polynomial optimization problem that is algorithmically practical solving P-Selective Sets on SAT instances when both formulas have approximately the same number of variables and clauses [9]. The whole algorithm is based on the problem of quadratic polynomial optimization which is feasible when we do not restrict the variables to be integers [3].

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. A polynomial time algorithm for P-Selective Sets on SAT instances implies that $P = NP$ [14].

It is good to take into account that the Israel Journal of Mathematics and the 13th International Conference on Algorithms and Complexity (CIAC 2023) were agreed that our original reduction was correct, but they principally focused that the author did not mention any practical application and so, the paper would have solely an educational purpose which is not a sufficiently merit to be published. For that reason, this approach on P-Selective Sets has an important role in the goals of this manuscript.

3 Summary of the Main Results

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 [8]. *NAE–3SAT* remains *NP–complete* when all clauses are monotone (meaning that variables are never negated), by Schaefer's dichotomy theorem [13]. 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 [10], [12]. 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 [6]. 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 [6]. 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 another 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.*

Finally, we deduce our main goal.

▶ **Theorem 5.** *P-Selective Sets on monotone NAE–3SAT instances could be solved in polynomial time when the pair of formulas have approximately the same number of variables and clauses.*

4 Main Results

4.1 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 for some truth assignment if and only if d_i has exactly one unsatisfied clause for the same assignment. 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 duplicated 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. ◀

4.2 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 uniquely 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 for some truth assignment if and only if $P(x_a, x_b) = 1$ for the equivalent and translated assignment (otherwise $P(x_a, x_b) = 0$). Finally, we obtain a polynomial

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

iterating for each clause in φ which is exactly a Homogeneous Diophantine Polynomial of degree two. Indeed, K satisfied clauses in φ for a truth assignment correspond to K distinct small pieces of polynomials $P(x_i, x_j)$ equal to 1 inside of the Homogeneous Diophantine Polynomial of degree two after its corresponding evaluation on x_i, x_j. In this way, we create 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 $u_1, u_2, \ldots, u_n$ over $\{0, 1\}^n$ is equal to K in polynomial time. ◀

4.3 Proof of Theorem 5

Proof. We claim that the P-Selective Sets on monotone *NAE–3SAT* instances could be solved in polynomial time when the pair of formulas have approximately the same number of variables and clauses [9]. This is because of we can reduce the instances from *NAE–3SAT* to HDE into a parsimonious way [11]. We assume that the problem of quadratic polynomial optimization could be feasible when we do not restrict the variables to be integers [3]. 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 domains according to our described and explained reduction in Theorem 4. ◀

5 Explanation of their Significance

No one has been able to find a polynomial time algorithm for any of more than 300 important known *NP–complete* problems [8]. A proof of $P = NP$ will have stunning practical consequences, because it possibly leads to efficient methods for solving some of the important problems in computer science [4]. The consequences, both positive and negative, arise since various *NP–complete* problems are fundamental in many fields [7].

We should seriously take into account these positive and negative consequences, since a polynomial time algorithm for P-Selective Sets on SAT instances implies that $P = NP$ [14]. Certainly, a polynomial time algorithm for P-Selective Sets on SAT instances could be transformed into a polynomial time algorithm for SAT [9]. Indeed, if there is any *NP–complete* language in P, then every NP can be solved in polynomial time [4].

Cryptography, for example, relies on certain problems being difficult. A constructive and efficient solution to an *NP–complete* problem such as SAT will break most existing cryptosystems including: Public-key cryptography, symmetric ciphers and one-way functions used in cryptographic hashing. These would need to be modified or replaced by information-theoretically secure solutions not inherently based on *P–NP* equivalence.

There are positive consequences that will follow from rendering tractable many currently mathematically intractable problems. For instance, many problems in operations research are *NP–complete*, such as some types of integer programming and the traveling salesman problem [7]. Efficient solutions to these problems have enormous implications for logistics [7]. Many other important problems, such as some problems in protein structure prediction, are also *NP–complete*, so this will spur considerable advances in biology [2].

Since all the *NP–complete* optimization problems become easy, everything will be much more efficient [7]. Transportation of all forms will be scheduled optimally to move people and goods around quicker and cheaper [7]. Manufacturers can improve their production to increase speed and create less waste [7]. Learning becomes easy by using the principle of Occam's razor: We simply find the smallest program consistent with the data [7]. Near perfect vision recognition, language comprehension and translation and all other learning tasks become trivial [7]. We will also have much better predictions of weather and earthquakes and other natural phenomenon [7].

But such changes may pale in significance compared to the revolution an efficient method for solving *NP–complete* problems will cause in mathematics itself [4]. Research mathematicians spend their careers trying to prove theorems, and some proofs have taken decades or even centuries to be discovered after problems have been stated [4]. For instance, Fermat's Last Theorem took over three centuries to be proved [4]. A method that guarantees to find proofs for theorems, should one exist of a "reasonable" size, would essentially end this struggle [4].

References

1 Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach.* Cambridge University Press, USA, 2009.

2 Bonnie Berger and Tom Leighton. Protein folding in the hydrophobic-hydrophilic (HP) model is Np-complete. *Journal of computational biology: a journal of computational molecular cell biology*, 5(1):27–40, 1998. `doi:10.1145/279069.279080`.

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

4 Stephen Arthur Cook. The P versus NP Problem. `https://www.claymath.org/wp-content/uploads/2022/06/pvsnp.pdf`, June 2022. Clay Mathematics Institute. Accessed 9 September 2023.

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

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

7 Lance Fortnow. The status of the P versus NP problem. *Communications of the ACM*, 52(9):78–86, 2009. `doi:10.1145/1562164.1562186`.

8 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.

9 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`.

10 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`.

11 Christos Harilaos Papadimitriou. *Computational complexity.* Addison-Wesley, USA, 1994.

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

13 Thomas J Schaefer. The complexity of satisfiability problems. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 216–226, 1978. `doi:10.1145/800133.804350`.

14 Alan L Selman. P-selective sets, tally languages, and the behavior of polynomial time reducibilities on NP. *Mathematical Systems Theory*, 13:55–65, 1979. `doi:10.1007/BF01744288`.

15 Michael Sipser. *Introduction to the Theory of Computation*, volume 2. Thomson Course Technology Boston, USA, 2006.