QP versus NP

Frank Vega

Joysonic, Belgrade, Serbia
vega.frank@gmail.com

Abstract. Given an instance of *XOR 2SAT* and a positive integer 2^K, the problem exponential exclusive-or 2-satisfiability consists in deciding whether this Boolean formula has a truth assignment with at leat K satisfiable clauses. We prove exponential exclusive-or 2-satisfiability is in QP and *NP-complete*. In this way, we show $QP \subseteq NP$.

Keywords: Complexity Classes · Completeness · Logarithmic Space · Nondeterministic · XOR 2SAT.

1 Introduction

The P versus NP problem is a major unsolved problem in computer science [4]. This is considered by many to be the most important open problem in the field [4]. It is one of the seven Millennium Prize Problems selected by the Clay Mathematics Institute to carry a US$1,000,000 prize for the first correct solution [4]. It was essentially mentioned in 1955 from a letter written by John Nash to the United States National Security Agency [1]. However, the precise statement of the $P = NP$ problem was introduced in 1971 by Stephen Cook in a seminal paper [4].

In 1936, Turing developed his theoretical computational model [14]. The deterministic and nondeterministic Turing machines have become in two of the most important definitions related to this theoretical model for computation [14]. A deterministic Turing machine has only one next action for each step defined in its program or transition function [14]. 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 [14].

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 set of languages decided by deterministic Turing machines within time f is an important complexity class denoted $TIME(f(n))$ [12]. In addition, the complexity class $NTIME(f(n))$ consists in those languages that can be decided within time f by nondeterministic Turing machines [12]. The most important complexity classes are P and NP. The class P is the union of all languages in $TIME(n^k)$ for every possible positive fixed constant k [12]. At the same time,

NP consists in all languages in $NTIME(n^k)$ for every possible positive fixed constant k [12]. Whether $P = NP$ or not is still a controversial and unsolved problem [1].

A Turing machine with input and output has a read-only input tape, a write-only output tape, and a read/write work tape [14]. The work tape may contain $O(f(n))$ symbols which means the Turing machine is computable within space f [14]. In computational complexity theory, $SPACE(f(n))$ is the complexity class containing those decision problems that can be decided within space f by a deterministic Turing machine with input and output [12]. $NSPACE(f(n))$ is the complexity class containing the decision problems that can be decided within space f by a nondeterministic Turing machine with input and output [12]. Another major complexity classes are $SPACE(\log n)$ and $NSPACE(\log n)$. Whether $SPACE(\log n) = NSPACE(\log n)$ is another fundamental question that it is as important as it is unresolved [12].

SAT is easier if the number of literals in a clause is limited to at most 2, in which case the problem is called $2SAT$. This problem can be solved in polynomial time, and in fact is complete for the complexity class $NSPACE(\log n)$ [12]. If additionally all OR operations in literals are changed to XOR operations, the result is called exclusive-or 2-satisfiability, which is a problem complete for the complexity class $SPACE(\log n)$ [2], [13]. The complexity class QP is equal to $TIME(2^{\log^k n})$ for every possible positive fixed constant k. To attack the P versus NP question the concept of *NP-completeness* has been very useful. If any single *NP–complete* problem can be decided in a quasi-polynomial time, then every NP problem has a quasi-polynomial time algorithm. We show a problem related with exclusive-or 2-satisfiability that is in QP and *NP–complete* and thus, $NP \subseteq QP$.

2 Theory

Let Σ be a finite alphabet with at least two elements, and let Σ^* be the set of finite strings over Σ [3]. A Turing machine M has an associated input alphabet Σ [3]. For each string w in Σ^* there is a computation associated with M on input w [3]. We say that M accepts w if this computation terminates in the accepting state, that is $M(w) =$ "*yes*" [3]. Note that M fails to accept w either if this computation ends in the rejecting state, that is $M(w) =$ "*no*", or if the computation fails to terminate [3].

The language accepted by a Turing machine M, denoted $L(M)$, has an associated alphabet Σ and is defined by:

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

We denote by $t_M(w)$ the number of steps in the computation of M on input w [3]. 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 [3]. We say that M runs in polynomial time if there is a constant k such that for all n, $T_M(n) \leq n^k + k$ [3]. In other words, this means the language $L(M)$ can be accepted by the Turing machine M in polynomial time. Therefore, P is the complexity class of languages that can be accepted in polynomial time by deterministic Turing machines [5]. A verifier for a language L is a deterministic Turing machine M, where:

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

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 [3]. A verifier uses additional information, represented by the symbol c, to verify that a string w is a member of L. This information is called certificate. NP is also the complexity class of languages defined by polynomial time verifiers [12].

A function $f : \Sigma^* \to \Sigma^*$ is a polynomial time computable function if some deterministic Turing machine M, on every input w, halts in polynomial time with just $f(w)$ on its tape [14]. 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* [9]. A language $L \subseteq \{0,1\}^*$ is *NP–complete* if

- $L \in NP$, and
- $L' \leq_p L$ for every $L' \in NP$.

If L is a language such that $L' \leq_p L$ for some $L' \in$ *NP–complete*, then L is *NP–hard* [5]. Moreover, if $L \in NP$, then $L \in$ *NP–complete* [5]. A principal *NP–complete* problem is SAT [7]. An instance of SAT is a Boolean formula ϕ which is composed of

- Boolean variables: $x_1, x_2, \ldots, x_n$;
- 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);
- 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 formula with a satisfying truth assignment is a satisfiable formula. The problem SAT asks whether a given Boolean formula is satisfiable [7]. 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:

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

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 $\neg x_2$. Another relevant *NP–complete* language is $3CNF$ satisfiability, or $3SAT$ [5]. In $3SAT$, it is asked whether a given Boolean formula ϕ in $3CNF$ is satisfiable. Many problems have been proved that belong to *NP-complete* by a polynomial time reduction from $3SAT$ [7]. For example, the problem *NOT–ALL–EQUAL 3SAT* defined as follows: Given a Boolean formula ϕ in $3CNF$, is there a truth assignment such that each clause in ϕ has at least one true literal and at least one false literal?

A Boolean formula is in 2-conjunctive normal form, or $2CNF$, if it is in CNF and each clause has exactly two distinct literals. There is a problem called $2SAT$, where we asked whether a given Boolean formula ϕ in $2CNF$ is satisfiable. $2SAT$ is complete for $NSPACE(\log n)$ [12]. Another special case is the class of problems where each clause contains XOR (i.e. exclusive or) rather than (plain) OR operators. This is in P, since an *XOR SAT* formula can also be viewed as a system of linear equations mod 2, and can be solved in cubic time by Gaussian elimination [11]. We denote the XOR function as $\oplus$. The *XOR 2SAT* problem will be equivalent to *XOR SAT*, but the clauses in the formula have exactly two distinct literals. *XOR 2SAT* is complete for $SPACE(\log n)$ [2], [13].

3 Results

We can give a certificate-based definition for $NSPACE(\log^2 n)$ [3]. A certificate-based definition of $NSPACE(\log^2 n)$ assumes that a deterministic Turing machine with input and output has another separated read-only tape [3]. On each step of the machine the machine's head on that tape can either stay in place or move to the right [3]. In particular, it cannot reread any bit to the left of where the head currently is [3]. For that reason this kind of special tape is called "read once" [3].

Definition 1. *A language L is in $NSPACE(\log^2 n)$ if there exists a deterministic Turing machine with input, output and an additional special read-once input tape polynomial $p : \mathbb{N} \to \mathbb{N}$ such that for every $x \in \{0,1\}^*$,*

$$x \in L \Leftrightarrow \exists u \in \{0,1\}^{p(|x|)} \text{ such that } M \text{ accepts } \langle x, u \rangle$$

where by $M(x,u)$ we denote the computation of M where x is placed on its input tape and u is placed on its special read-once tape, and M uses at most $O(\log^2 |x|)$ space on its read/write tape for every input x.

Definition 2. ***EXPONENTIAL EXCLUSIVE-OR 2-SATISFIABILITY***

INSTANCE: A positive integer 2^K and a formula ϕ that is an instance of XOR 2SAT.

QUESTION: Is there a truth assignment in ϕ such that at least K clauses are satisfiable?

We denote this problem as $EXP \oplus 2SAT$.

Theorem 1. $EXP \oplus 2SAT \in NSPACE(\log^2 n)$.

Proof. Given a Boolean formula ϕ that is an instance of *XOR 2SAT* with m clauses, we can enumerate from left to right the clauses in ϕ such that the leftmost clause has the index 1 and the rightmost the number m. Therefore, a combination of K clauses in ϕ corresponds to a subset of size K from the set $\{1, 2, 3, \ldots, m-1, m\}$. This subset of K numbers will be a regular language, because it is finite [10]. Since it is a regular language, then it will be computable by a linear size NC^1 circuit [10]. The best upper bound in general for a set of K numbers from 1 to m will be a circuit with a length in the binary encoding of $O(K \times \log m)$. The encoding of K is a given binary string but in an exponential way, and therefore, the positive integer K complies with $K \leq \log 2^K \leq \log n$ where $n = |\langle 2^K, \phi \rangle|$ and $|\ldots|$ is the bit-length function. Hence, there is a circuit C of size $O(\log^2 n)$ in the element $(2^K, \phi) \in EXP \oplus 2SAT$ which computes every subset of size K from the set $\{1, 2, 3, \ldots, m-1, m\}$ for a specific fixed constant c because of $\log m < \log n$.

There will be a deterministic Turing machine with input and output which receives this circuit C in the special read-once input tape and in the input tape the given Boolean formula ϕ that is an instance of *XOR 2SAT* and the number 2^K. Next, we copy the circuit C to the read/write work tape just reading each bit in the special read-once input tape from left to right until we find the blank symbol. We can copy it to the read/write work tape, because the size of C is $O(\log^2 n)$. Indeed, we can count the size of C with a positive integer d which will have at most $c \times \lceil \log^2 n \rceil + 1$ bit-length, such that when $d > c \times \log^2 n$, then we reject the input and certificate where c is the already mentioned fixed constant. After that, we evaluate in ascending order the numbers in the set $\{1, 2, 3, \ldots, m-1, m\}$ just to verify if there are at least K numbers which leads to an acceptance. This can be done in time $O(\log^2 n)$, because *CIRCUIT VALUE* can be solved in linear time [12]. Consequently, this can be solved in $O(\log^2 n)$ space on the read/write work tape. Besides, we can count the number of different acceptances with a positive integer $d' \leq m$ that will have at most $\lceil \log m \rceil$ bit-length. In this way, the process remains in $O(\log^2 n)$ space.

Finally, if there are at least K acceptances between the numbers 1 and m, then we compute in the read/write work tape the Boolean formula $\psi = c_{i_1} \wedge c_{i_2} \ldots \wedge c_{i_K} \ldots$ such that each number i_j is accepted by C. Since *XOR 2SAT* is complete for $SPACE(\log n)$, then we can decide ψ in a quadratic logarithmic space on the read/write work tape. Notice that, we do not need to copy ψ to the read/write work tape since the membership in ψ of any clause c_{i_j} from the input tape can be done in a quadratic logarithmic space by an evaluation in C

of i_j. In this simulation, we finally accept in case of ψ is satisfiable otherwise we reject the chosen input and certificate. All this process can be done in quadratic deterministic logarithmic space just reading at once the additional special tape. Moreover, the size of the certificate is polynomial due to the size and the depth of C is poly-logarithmic. In conclusion, we show $EXP \oplus 2SAT$ complies with the certificate-based definition of $NSPACE(\log^2 n)$ and thus, $EXP \oplus 2SAT \in NSPACE(\log^2 n)$ [3].

Theorem 2. *$EXP \oplus 2SAT \in NP$–complete.*

Proof. It is trivial that $EXP \oplus 2SAT \in NP$ [12]. Given a Boolean formula ϕ in $3CNF$ with n variables and m clauses, we create the following formulas for each clause $c_i = (x \vee y \vee z)$ in ϕ, where x, y and z are literals,

$$P_i = (x \oplus y) \wedge (y \oplus z) \wedge (x \oplus z).$$

We can notice P_i has exactly two satisfiable clauses if and only if at least one member of $\{x, y, z\}$ is true and at least one member of $\{x, y, z\}$ is false. Hence, we can create the Boolean formula ψ as the conjunction of the P_i formulas for every clause c_i in ϕ, such that $\psi = P_1 \wedge \ldots \wedge P_m$. Finally, we obtain that

$$\phi \in \textit{NOT–ALL–EQUAL 3SAT if and only if } (2^{2\times m}, \psi) \in EXP \oplus 2SAT$$

where $2^{2\times m}$ is a binary string of size $2 \times m + 1$ (we can create this string in polynomial time using the exponentiation by squaring [5]). Consequently, we prove $\textit{NOT–ALL–EQUAL 3SAT} \leq_p EXP \oplus 2SAT$ where $\textit{NOT–ALL–EQUAL 3SAT} \in \textit{NP–complete}$. To sum up, we show $EXP \oplus 2SAT \in \textit{NP–hard}$ and $EXP \oplus 2SAT \in NP$ and thus, $EXP \oplus 2SAT \in \textit{NP–complete}$.

Theorem 3. *$NP \subseteq QP$.*

Proof. If any single *NP–complete* problem can be decided in a quasi-polynomial time, then every NP problem has a quasi-polynomial time algorithm [5]. Every language in the class $NSPACE(\log^2 n)$ is in QP, and therefore, $EXP \oplus 2SAT \in QP$ [12]. Hence, as a consequence of Theorems 1 and 2, we obtain $NP \subseteq QP$.

4 Conclusion

No polynomial time algorithm has yet been discovered for any *NP–complete* problem [6]. The biggest open question in theoretical computer science concerns the relationship between these classes: Is P equal to NP? In 2012, a poll of 151 researchers showed that 126 (83%) believed the answer to be no, 12 (9%) believed the answer is yes, 5 (3%) believed the question may be independent of the currently accepted axioms and therefore impossible to prove or disprove, 8 (5%) said either do not know or do not care or don't want the answer to be yes nor the problem to be resolved [8]. It is fully expected that $P \neq NP$ [12]. Indeed, if $P = NP$ then there are stunning practical consequences [12].

Certainly, P versus NP is one of the greatest open problems in science and a correct solution for this incognita will have a great impact not only for computer science, but for many other fields as well [6]. In this way, we make a breakthrough step forward to reach the final solution for this problem.

References

1. Aaronson, S.: P $\stackrel{?}{=}$ NP. Electronic Colloquium on Computational Complexity, Report No. 4 (2017)
2. Alvarez, C., Greenlaw, R.: A compendium of problems complete for symmetric logarithmic space. Computational Complexity **9**(2), 123–145 (2000)
3. Arora, S., Barak, B.: Computational complexity: a modern approach. Cambridge University Press (2009)
4. Cook, S.A.: The P versus NP Problem (April 2000), available at `http://www.claymath.org/sites/default/files/pvsnp.pdf`
5. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. The MIT Press, 3rd edn. (2009)
6. Fortnow, L.: The status of the P versus NP problem. Communications of the ACM **52**(9), 78–86 (2009)
7. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. San Francisco: W. H. Freeman and Company, 1 edn. (1979)
8. Gasarch, W.I.: Guest column: The second P $\stackrel{?}{=}$ NP poll. ACM SIGACT News **43**(2), 53–77 (2012)
9. Goldreich, O.: P, NP, and NP-Completeness: The basics of computational complexity. Cambridge University Press (2010)
10. Koucky, M.: Circuit complexity of regular languages (April 2012), available at `http://www.cse.iitm.ac.in/~jayalal/teaching/CS6840/2012/project/Regular-Sunil-slides.pdf`
11. Moore, C., Mertens, S.: The Nature of Computation. Oxford University Press (2011)
12. Papadimitriou, C.H.: Computational complexity. Addison-Wesley (1994)
13. Reingold, O.: Undirected connectivity in log-space. Journal of the ACM **55**(4), 1–24 (2008)
14. Sipser, M.: Introduction to the Theory of Computation, vol. 2. Thomson Course Technology Boston (2006)