Artificial Intelligence Research and Development
A. Cortés et al. (Eds.)

doi:10.3233/FAIA220310

Yet Another (Fake) Proof of P=NP

Carlos ANSÓTEGUI [a] and Jordi LEVY [b,1]
[a] *Universitat de Lleida, Spain.*
[b] *IIIA, CSIC, Spain.*

Abstract. Obviously, we do not prove $\mathbf{P} = \mathbf{NP}$ in this article. In fact, the title only refers to the first part, where the proof that we present contains an error that, to make reading more attractive, is only revealed in the second part.

In the second part, we describe how the reduction of SAT to Max2XOR and the proof system presented in the first part –although they do not solve one of the Millennium Prize Problems– may trigger new complementary ways of solving the SAT problem.

Keywords. Proof systems, SAT, MaxSAT, MaxCUT

1. Introduction

We start this fake proof by introducing the Cook and Reckhow program [1], as a way to resolve the **P** versus **NP** problem. The program is based on the following observation. The complexity class **P** is closed under complement, therefore, if $\mathbf{P} = \mathbf{NP}$, then **NP** is also closed under complement. In other words,

$$\mathbf{NP} \neq \mathbf{CoNP} \text{ implies } \mathbf{P} \neq \mathbf{NP} \tag{1}$$

The class **P** does not need any introduction. The class **NP** is the set of decision problems that can be solved in polynomial time by a non-deterministic Turing Machine. Equivalently, **NP** can be defined as the class of decision problems P for which $x \in P$ has *certifications* or *proofs* verifiable in polynomial time by a deterministic Turing machine. The classical **NP**-hard problem is SAT [2,3], defined as the set of propositional formulas in conjunctive normal form that are satisfiable. In this case, a certification that a given formula is in SAT can be simply a truth assignment to the variables that satisfies all the clauses. Given a formula and a truth assignment, we can verify that the assignment certifies the satisfiability of the formula using a deterministic Turing Machine in polynomial time.

The class **CoNP** is the complement of **NP**. It can be defined as the class of decision problems P for which $x \notin P$ has certifications verifiable in polynomial time by a deterministic Turing machine. The classical **CoNP**-hard problem is the complement of SAT, i.e. TAUT.

In order to prove that $\mathbf{NP} = \mathbf{CoNP}$, since TAUT is **CoNP**-hard, we only need to prove that $x \in$ TAUT has certifications verifiable in polynomial time. Here, it makes

[1]Corresponding Author: Jordi Levy, IIIA, CSIC, Campus de la UAB, 08193 Bellaterra, Spain; E-mail: levy@iiia.csic.es.

sense to call these certifications *proofs*. These proofs are defined in a very general way, as sequences of bits $s \in \{0,1\}^+$. A *proof system PS* is defined as a polynomial-time algorithm that, for any formula f, we have $f \in TAUT$ if, and only if, there exists a proof s such that the algorithm PS accepts (f,s). We say that the proof system is *p-bounded* if, in addition, we can ensure the existence of a polynomially-bounded proof. In this case, PS accepts (f,s) in polynomial time on $|f|$. Therefore,

$$\text{there is a p-bounded proof system for TAUT if, and only if, } \mathbf{NP} = \mathbf{CoNP}. \qquad (2)$$

It is important to remark that we do not care about how difficult is to find the proof. We only have to ensure that there exists a *short* (i.e. p-bounded) proof. A completely different question is the practical use of these proof systems, in a SAT solver, for instance. Thus, we observe that most SAT solvers are based on the *resolution* proof system, although we know that it is one of the weakest proof systems, in the sense that it has long proofs for tautologies that have short proofs in other proof systems.

2. Reducing SAT to Max2XOR

XOR clauses are similar to SAT clauses, using exclusive OR instead of the traditional OR. They can be written in the form of parity constraints or clauses

$$x_1 \oplus \ldots \oplus x_n = k,$$

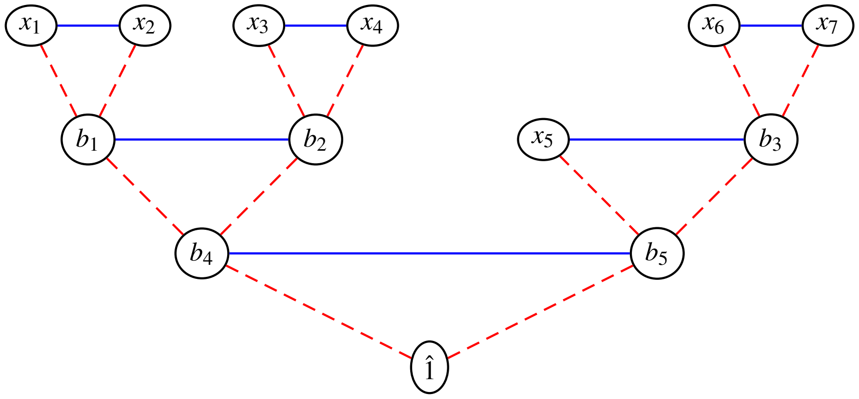
where x_i's are Boolean variables (that may take value 0 or 1), $\oplus$ is the sum-modulo-two operator, and k is either the constant 0 or 1. The particular case where $n = 0$ and $k = 1$ is an unsatisfiable clause, similar to the empty clause in SAT, that we will represent as $\Box$. The case where $n = 0$ and $k = 0$ is always satisfied (a tautological clause) and is removed from the formula. Repeated variables are also removed from clauses, thus $x \oplus x \oplus A = k$ is simplified to $A = k$.

Here we will consider parity constraints with at most two variables. The satisfiability of a set of these constraints, called 2XOR –or even the more general XOR problem– is in **P**.

Max2XOR is the optimization version of this problem, where we try to find an assignment that maximizes the number of satisfied constraints. The decision version of this problem, where given a set of parity constraints and an integer C, we decide if there is an assignment that satisfies at least C constraints, is **NP**-hard. This can be proved by reducing SAT to Max2XOR using the gadget we describe in [4]. Bellow, we introduce this reduction through an example.[2]

Example 1. The clause $x_1 \vee \cdots \vee x_7$ can be reduced to the following Max2XOR problem, where a dashed red line between x and y represents the clause $x \oplus y = 0$, or equivalently the equal-variable constraint $x = y$, and a solid blue line between x and y represents the clause $x \oplus y = 1$, or equivalently the non-equal-variable constraint $x \neq y$. The special node labeled as $\hat{1}$ is used to represent unary clauses as edges: $x \oplus \hat{1} = 1$ is equivalent to $x = 0$, and $x \oplus \hat{1} = 0$ is equivalent to $x = 1$. The variables $b_1, \ldots, b_5$ are fresh variables not occurring elsewhere.

[2]We refer to [4] for a formal definition and proof of correctness. In this original publication, XOR clauses are weighted, and all weights are multiples of $1/2$. Here we avoid the use of weights.



Any similar set of XOR clauses, representing this kind of tree with the variables $x_1,\dots,x_7$ in the leaves, would also work.

If the original clause contains negated literals, we proceed in the same way, transforming the resulting XOR clauses by removing negations, using: $\neg x \oplus A = k$ is equivalent to $x \oplus A = 1-k$.

Proposition 2 ([4]). *The Translation of a SAT clause C with $n \geq 2$ literals results into a multiset $T(C)$ of $3(n-1)$ XOR clauses.*

For any SAT clause C of size $n \geq 2$: any assignment I satisfying C, can be extended (assigning the auxiliary variables b's) to an assignment satisfying $2(n-1)$ XOR clauses of $T(C)$, and any assignment I falsifying C can only be extended to satisfy at most $2(n-2)$ of the XOR clauses.

Any SAT problem P with clause sizes $C_i \geq 2$, for $i = 1,\dots,m$, can be reduced to a Max2XOR problem $T(P)$ with $3\sum_{i=1}^m (C_i - 1)$ XOR clauses, such that
P is satisfiable if, and only if, the maximum number of satisfiable clauses in $T(P)$ is at least $2\sum_{i=1}^m (C_i - 1)$; or equivalently,
P is unsatisfiable if, and only if, the minimum number of unsatisfied clauses in $T(P)$ is at least $2 + \sum_{i=1}^m (C_i - 1)$.

The previous proposition allows us to reduce SAT to (the decision version of) Max2XOR. In the next section we tackle the problem of defining a proof system for Max2XOR or, in general, for MaxXOR.

MaxCUT is the problem of, given a graph, finding a partition of the vertices into two sets such that the number of edges connecting two nodes of distinct partitions is maximized. It is easy to see that MaxCUT is the same as maximizing the satisfaction of constraints of the form $x \oplus y = 1$. We can reduce Max2XOR to MaxCUT as follows: We introduce a new variable, called $\hat{1}$, and replace every constraint $x = 0$ by $x \oplus \hat{1} = 1$, and $x = 1$ by $x \oplus \hat{1} = 0$. Then, we replace every constraint of the form $x \oplus y = 0$ by two constraints $x \oplus b = 1$ and $b \oplus y = 1$, where b is a fresh variable. In this way, we get a set of constraints of the form $x \oplus y = 1$ that, interpreted as edges, can be solved as a MaxCUT problem.

3. A Polynomial Proof System for MaxXOR

The equivalent to the resolution rule for XOR constraints, called *XOR resolution rule* or *Gaussian elimination*, is

$$\frac{\begin{array}{c} x \oplus A = k_1 \\ x \oplus B = k_2 \end{array}}{A \oplus B = k_1 \oplus k_2} \tag{3}$$

When we apply this rule, we must remove repeated variables (we assume that there is at least one of such repeated variables, called x in the previous scheme). Thus, these are particular instances of the rule scheme:

$$\frac{\begin{array}{c} x \oplus y \oplus z = 1 \\ x \oplus y \oplus t = 1 \end{array}}{z \oplus t = 0} \qquad\qquad \frac{\begin{array}{c} x \oplus y = 1 \\ x \oplus y = 0 \end{array}}{\Box}$$

where, in addition to x, we also remove the repeated variable y. Based on this rule we can prove that deciding satisfiability of XOR constraints is in **P**.

In order to extend this rule to MaxXOR or Max2XOR, we can take inspiration from the extension of resolution to MaxSAT resolution [5,6]. The MaxSAT resolution rule is:

$$\frac{\begin{array}{c} x \vee A \\ \neg x \vee B \end{array}}{\begin{array}{c} A \vee B \\ x \vee A \vee \neg B \\ \neg x \vee B \vee \neg A \end{array}}$$

where the rule deals with multisets of clauses, *replacing* premises by conclusions, and where, if $B = b_1 \vee \cdots \vee b_n$, the meta-clause $x \vee A \vee \neg B$ stands for the set $\{x \vee A \vee \neg b_1, x \vee A \vee b_1 \vee \neg b_2, \ldots, x \vee A \vee b_1 \vee \cdots \vee b_{n-1} \vee \neg b_n\}$. This rule is sound and complete for MaxSAT, in the sense that, m is the minimum number of unsatisfiable clauses in a multiset Γ if, and only if, there exists a derivation $\Gamma \vdash \{\Box, \overset{m}{\ldots}, \Box\} \cup \Delta$, where Δ is a multiset of satisfiable clauses and $\Box$ stands for the empty clause. The proof of the soundness of this rule is based on the fact that, for any assignment, the number of falsified premises is equal to the number of falsified conclusions. The proof of completeness is more complicated. Basically, we prove that we can always construct a regular refutation in the following way. We consider a fixed list of variables $x_1, \ldots, x_n$. We only resolve the variable x_1 until all occurrences of this variable have the same sign, or all possible pairs are of the form $x_1 \vee A$ and $\neg x_1 \vee B$ with a variable y satisfying $y \in A$ and $\neg y \in B$. In this situation, the new clause $A \vee B$ is a tautology. Then, we proceed to resolve x_2, and so on. In the end, we only have empty clauses and satisfiable clauses.

In the case of MaxXOR, simply consider the same XOR-resolution rule (3), but applied *replacing* premises by the conclusion. First, we have the following observation:

Lemma 3. *For any assignment, in the XOR-resolution rule, the number of falsified premises is always equal to or greater than the number of falsified conclusions.*

The number of falsified clauses only decreases when the assignment falsifies both premises.

Proof. An analysis of all the cases allows us to verify that, for any assignment, there are only three possibilities: 1) the assignment satisfies both premises and the conclusion, 2) the assignment satisfies one of the premises and falsifies the other premise and the conclusion, or 3) the assignment falsifies both premises and satisfies the conclusion. $\square$

This fact is enough to prove the soundness of a proof system based on this rule:

Lemma 4 (Soundness)**.** *The XOR-resolution rule is sound for MaxXOR, i.e. if there exists a refutation of the form* $\Gamma \vdash \{\square, \overset{m}{\ldots}, \square\} \cup \Delta$*, then the minimal number of unsatisfiable constraints in* Γ *is at least* m*.*

Proof. For any assignment I, by Lemma 3, the number of clauses falsified by I in Γ is bigger or equal to the number of clauses falsified by I in $\{\square, \overset{m}{\ldots}, \square\} \cup \Delta$, that is at least m because empty clauses are always falsified. Therefore, m is a minimum of the number of clauses falsified in Γ for any assignment. $\square$

Completeness is more complicated. As for the MaxSAT resolution rule, to construct the proof $\Gamma \vdash \{\square, \overset{m}{\ldots}, \square\} \cup \Delta$, we can fix an ordering of the variables $x_1, \ldots, x_n$. Then, we apply XOR-resolution to remove all pairs of occurrences of x_1, except those that generate tautologies.[3] When finished, we continue with x_2, and so on until we get the empty clauses. However, in this process, we have to ensure that, at least for one assignment, the number of unsatisfied clauses is preserved. Therefore, using this *guiding* assignment, we will avoid applications of the XOR-resolution when the two premises are falsified and the conclusion is satisfied. As we have seen in the proof of Lemma 3, this is the only situation where the number of unsatisfied clauses is not preserved.

Given an assignment I, we write $\Gamma \vdash_I \Delta$ when we can derive Δ from Γ using the MaxXOR inference rule (i.e. replacing premises by the conclusion) and, in all steps, the assignment satisfies at least one of the premises. This ensures that the number of clauses falsified by I in Γ is equal to the number of clauses falsified by I in Δ.

The following lemma states that, for any variable x, we can always choose an assignment –in fact, any of the optimal assignments will work– that allows us to derive $\Gamma \vdash_I \Delta$ where Δ only contains one kind of clause for x. Basically, this allows us to forget about x and, *eventually*, proceeding in the same way with the rest of the variables, removing all of them, obtaining at the end a set of empty clauses. Since the set of unsatisfied clauses is preserved, the cardinality of this multiset of empty clauses would be equal to the number of clauses falsified by I in the original problem, i.e. equal to the minimum number of unsatisfiable clauses.

Lemma 5 (Iteration)**.** *For any XOR formula* Γ *and any variable* x*, there exists an assignment* I *and a derivation* $\Gamma \vdash_I \Delta$*, where all clauses in* Δ *that contain* x *are equal. In other words,* $\Gamma \vdash_I C_x \cup \Delta'$*, where* Δ' *does not contain* x *and* $C_x = \{x \oplus A = k, \ldots, x \oplus A = k\}$*.*

Proof. Let I be any optimal assignment that maximizes the number of satisfied clauses. Let Γ_x be the subset of clauses that contain x. Since I is optimal, the number of clauses of Γ_x satisfied by I is bigger than the number of clauses of Γ_x falsified by I. If this were not true, we could consider another assignment I' defined as $I'(y) = I(y)$, if $x \neq y$, and $I'(x) = 1 - I(x)$ that satisfies more clauses than I. Then, we can partition $\Gamma_x = \bigcup_{i=1}^{r} A_i \cup$

[3]Notice that the only way to obtain a tautology is to resolve a clause with another identical clause, i.e. if we resolve $x \oplus A = k$ and $x \oplus A = k$. Therefore, we only stop when all clauses containing x_1 are equal.

$\bigcup_{j=1}^{s} B_j \cup C$, where A_i contain a couple of distinct clauses both satisfied by I, B_j contain a pair of clauses, one satisfied by I and the other falsified by I, and all clauses in C are equal. We can apply $\vdash_I$ to every pair in A_i's and in B_j's, leaving C as the only clauses that contain x. □

Notice that, since there is only one kind of clause containing x, these clauses $x \oplus A = k$ in C_x are always satisfiable by taking $I(x) = k - I(A)$.

Finally, we can prove that any XOR-resolution proof is polynomially bounded:

Lemma 6. *For any Γ, the length of any proof $\Gamma \vdash \Delta$ is linearly bounded on $|\Gamma|$.*

Proof. It is trivial, since any XOR-resolution step removes one clause from the multiset. □

4. Where is the Bug?

The attentive reader can probably skip this section, or probably wants to think about the MaxXOR proof system a little bit more before we reveal the solution..., no? Then,...

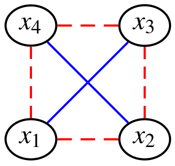
In fact, to the best of our knowledge, everything we have said so far is true! However, it is said in a way that might lead the reader to conclude that this proves $\mathbf{P} = \mathbf{NP}$. Actually, there are two *pitfalls*.

The first one is that, even if we were able to find a p-bounded proof system for TAUT, we cannot conclude $\mathbf{P} = \mathbf{NP}$. Notice that statement (2) is a double implication, but statement (1) only works in one direction. This means that we could prove $\mathbf{NP} = \mathbf{CoNP}$, and make all the arithmetic hierarchy collapse, but we could still have $\mathbf{P} \subsetneq \mathbf{NP} = \mathbf{CoNP}$. This possibility is not usually considered when discussing the $\mathbf{P}$ versus $\mathbf{NP}$ problem, but it is perfectly plausible. It imply that we could no longer try to use the Cook and Reckhow's approach to solve the $\mathbf{P}$ versus $\mathbf{NP}$ problem.

The second pitfall makes the proof system that we have defined for MaxXOR incomplete. This can be better seen through the following example.

Example 7. Consider the following MaxXOR problem:

$$\begin{array}{ll} x_1 \oplus x_2 = 0 & x_1 \oplus x_3 = 1 \\ x_2 \oplus x_3 = 0 & x_2 \oplus x_4 = 1 \\ x_3 \oplus x_4 = 0 & \\ x_4 \oplus x_1 = 0 & \end{array}$$

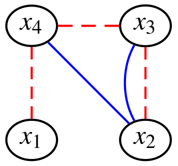


where, on the right, we represent the 2XOR clauses with red dashed lines to represent equal-variable constraints, and with blue lines to represent non-equal-variable constraints.

The minimum number of unsatisfiable clauses is 2 and there are several optimal assignments. One of them is $I(x_1) = I(x_2) = I(x_3) = I(x_4) = 1$. Using this optimal assignment, the four equal-variable constraints are satisfied, and the two non-equal-variable constraints are falsified. Assume that we want to start by removing variable x_1. As shown in the proof of Lemma 5, among the clauses that contain x_1, there are more satisfied than falsified by I, and we can arrange them in the partition $B_1 = \{x_1 \oplus x_2 = 0, x_1 \oplus x_3 = 1\}$

and $C = \{x_4 \oplus x_1 = 0\}$. Applying one XOR resolution step, we replace B_1 by $\{x_2 \oplus x_3 = 1\}$,leaving C *frozen*:

$$\begin{array}{ll} x_2 \oplus x_3 = 1 & \\ x_2 \oplus x_3 = 0 & \qquad x_2 \oplus x_4 = 1 \\ x_3 \oplus x_4 = 0 & \\ x_4 \oplus x_1 = 0 & \end{array}$$



The problem is that I is no longer optimal for this new MaxXOR problem. This implies, for instance, that for some variables, the set of clauses containing it may contain more falsified clauses than satisfied. In our example, there are two clauses involving x_2 that are falsified by I, whereas only one is satisfied by I. Moreover, if we try to remove x_3 or x_4, in both cases, we reach a situation where we cannot continue the application of XOR resolution steps. In fact, whatever we do, our proof system only can obtain one empty clause from our original MaxXOR instance.

The previous example shows that, although Lemma 5 is true and allows us to remove one variable, the optimal assignment is not optimal for the resulting MaxXOR problem. Therefore, we can not use the same lemma to iteratively remove another variable. In the previous section we have added an "*eventually*" to avoid the introduction of false statements and used the name "*iteration*" for Lemma 5 that is misleading, since it only applies to the first variable.

Taking into consideration the reduction of Max2XOR to MaxCUT, or from SAT to MaxCUT, it is easy to see that the existence of polynomial certifications or proofs for the MaxCUT problem would have important consequences and would also prove $\mathbf{NP} = \mathbf{CoNP}$.

5. Three Pigeons and Two Holes

The Pigeon-hole principle, noted PHP_n^{n+1}, is a family of tautologies that state that we can not place $n+1$ pigeons in n holes with at most one pigeon in each hole. In the form of SAT clauses, it is encoded as:

$$\begin{array}{ll} x_i^1 \vee \ldots \vee x_i^n & \qquad \text{for } i = 1, \ldots, n+1 \\ \neg x_i^k \vee \neg x_j^k & \qquad \text{for } 1 \leq i < j \leq n+1 \text{ and } k = 1, \ldots, n \end{array}$$

where x_i^j means that pigeon i is placed in hole j. This principle was used by Haken [7] to prove that some tautologies require super-polynomial proofs in the resolution proof system.

In this section, we show how the pigeon-hole principle PHP_2^3, with 3 pigeons and 2 holes, may be proved with the XOR-resolution proof system. We use the SAT to Max2XOR reduction described in Section 2. The clauses $x_i^1 \vee x_i^2$, for $i = 1,2,3$, generate the constraints $\{x_i^1 = 1,\ x_i^2 = 1,\ x_i^1 \oplus x_i^2 = 1\}$. The clauses $\neg x_i^j \vee \neg x_{i'}^j$, for $i, i' = 1,2,3$ and $i < i'$, and $j = 1,2$, generate the constraints $\{x_i^j = 0,\ x_{i'}^j = 0,\ x_i^j \oplus x_{i'}^j = 1\}$. All these constraints are represented in Figure 1. From them, only resolving the constraints with the same colors, we get 11 copies of $\square$. The Max2XOR problem comes from the translation of 9 binary clauses. Therefore, according to Proposition 2, we had to get $2 + 9(2-1) = 11$

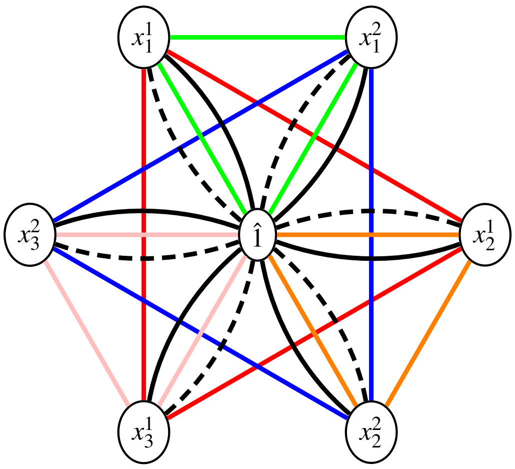


Figure 1. Graphical representation of the proof of PHP_2^3. Solid lines between two nodes x and y represent $x \oplus y = 1$, and between x and $\hat{1}$ represents $x = 0$. Similarly, dashed lines represent $x \oplus y = 0$ or $x = 1$. Every pair of black lines with the same origin and target generates an empty clause (there are 6 pairs). For the rest 5 colors, each set of 3 lines of the same color that forms a triangle generates another empty clause.

copies of $\square$ to prove the unsatisfiability of the original formula. This concludes the proof of PHP_2^3.

6. Makes It Sense to Reduce SAT to Max2XOR?

A priory, the reduction of SAT to Max2XOR has some pros and cons. As for cons, since we are trying to maximize the number of satisfied clauses, instead of satisfying all of them, it is more difficult to make inferences. For instance, it is difficult to apply learning techniques, which have been so successful in modern SAT solvers. In the case of MaxSAT, we can use MaxSAT resolution (instead of classical resolution) to make inferences. In the case of MaxXOR, it is difficult to obtain a similar complete proof system. As an advantage, we can mention that the transformation produces *small* clauses. Anybody familiarized with SAT techniques would appreciate the generation of these small clauses, because they reduce drastically the number of possible satisfying assignments. In [8], we show how the reduction of SAT to MaxSAT and the use of MaxSAT resolution on the resulting formula, allows us to obtain polynomial proofs for the pigeon-hole principle that, as we mentioned, has super-polynomial proofs in resolution.

It is preferable to reduce SAT to Max2SAT, as in [8], or to Max2XOR, as in [4]? Again, a priory, there are advantages in both cases. In the case of Max2SAT, we have a complete proof system, that we do not have in the case of Max2XOR. However, in the case of Max2SAT, we have to consider that, even if all clauses are unary or binary, as a result of the application of the MaxSAT resolution rule, we can get bigger clauses, that have to be reduced again. In the case of MaxXOR, we have the advantage that the decision problem XOR is in **P**, whereas the decision problem for MaxSAT, i.e. SAT, is **NP**-complete. This means that, once we have decided which clauses we want to satisfy, it is easy to check if they are satisfiable.

The classical scheme of a SAT solver may be seen as a search engine that tries to find a satisfying assignment working together with an inference engine that tries to construct a proof of unsatisfiability. The great advantage of resolution as a proof system is that the construction of the proof is similar to the search of the assignment. Therefore, both engines work closely interleaved.

On the other side, so-called local search SAT solvers, only try to find a satisfying assignment, and can work forever if such an assignment does not exist. If we do not have a complete proof system for Max2XOR, it seems that we can only undertake to implement local-search solvers.

The best approximated algorithm for MaxCUT is based on a relaxation to Semidefinite Programming (SDP) [9]. In practice, although SDP is polynomial, it is still inefficient. One possibility is to reduce the number of dimensions used to represent the vectors associated with Boolean variables. It is known that, for a sufficient rank ($\sqrt{2n}$ instead of n), the solution of the approximating problem is still unique and the so-called *mixing method* converges to it [10]. The mixing method has been adapted to Max2SAT [11], and it can be adapted to the Max2XOR problem. We have started to explore this possibility, and the results obtained with this approximated algorithm are promising.

Once we have an approximated algorithm, we can apply several techniques on top of it. One possibility is to implement a *decimation* algorithm. It consists of, once we have a kind of probability for each variable to have a certain truth value, provided by the approximated algorithm, we can fix this value for one or a fraction of the variables for which this probability is higher, and we call again to the approximated algorithm. Another possibility is to use a branch-and-bound algorithm. In the case of Max2XOR, the approximated algorithm also returns a kind of probability for each clause to be satisfied. Therefore, in parallel, we can fix the values of variables and also force the satisfaction of clauses with higher probabilities.

Finally, we know that real-world SAT instances are highly modular [12,13]. The reduction defined in [4] allows us to group variables in any way. Therefore, we can analyze the modular structure of the original SAT instance, and get a Max2XOR problem where only variables closely related occur together in most of the clauses. This would contribute to increasing the modularity and thus make it easier to find a solution.

7. Conclusions and Further Work

We have used a fake proof of $\mathbf{P} = \mathbf{NP}$ as an excuse to introduce (we hope that in a funny way) a reduction of SAT to Max2XOR. This has also served us to comment on the difficulties of defining a complete proof system for Max2XOR. To finish, we have discussed the possibilities of implementing a local-search algorithm for Max2XOR based on the mixing method. The preliminary results using these ideas are promising. The modular structure of the original SAT instance can be used in the reduction to obtain an even more modular Max2XOR problem.

References

[1] Cook SA, Reckhow RA. The Relative Efficiency of Propositional Proof Systems. J Symb Log. 1979;44(1):36-50. DOI: 10.2307/2273702.

[2] Cook SA. The Complexity of Theorem-Proving Procedures. In: Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, STOC'71. ACM; 1971. p. 151-8. DOI: 10.1145/800157.805047.

[3] Levin LA. Universal Sequential Search Problems. Problems of Information Transmission. 1973;9(3).

[4] Ansótegui C, Levy J. Reducing SAT to Max2XOR. ArXiv. 2022;2204.01774. DOI: 10.48550/ARXIV.2204.01774.

[5] Bonet ML, Levy J, Manyà F. A Complete Calculus for Max-SAT. In: Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing, SAT'06. vol. 4121 of Lecture Notes in Computer Science. Springer; 2006. p. 240-51. DOI: 10.1007/11814948_24.

[6] Bonet ML, Levy J, Manyà F. Resolution for Max-SAT. Artif Intell. 2007;171(8-9):606-18. DOI: 10.1016/j.artint.2007.03.001.

[7] Haken A. The intractability of resolution. Theoretical Computer Science. 1985;39:297-308. Third Conference on Foundations of Software Technology and Theoretical Computer Science. DOI: 10.1016/0304-3975(85)90144-6.

[8] Ansótegui C, Levy J. Reducing SAT to Max2SAT. In: Proceedings of the 30th International Joint Conference on Artificial Intelligence, IJCAI'21; 2021. p. 1367-73. DOI: 10.24963/ijcai.2021/189.

[9] Goemans MX, Williamson DP. Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming. J ACM. 1995 nov;42(6):11151145. DOI: 10.1145/227683.227684.

[10] Wang PW, Chang WC, Kolter JZ. The Mixing method: low-rank coordinate descent for semidefinite programming with diagonal constraints. ArXiv. 2018;1706.00476.

[11] Wang P, Kolter JZ. Low-Rank Semidefinite Programming for the MAX2SAT Problem. In: Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence, AAAI'19. AAAI Press; 2019. p. 1641-9. DOI: 10.1609/aaai.v33i01.33011641.

[12] Ansótegui C, Giráldez-Cru J, Levy J. The Community Structure of SAT Formulas. In: Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing, SAT'12. vol. 7317 of Lecture Notes in Computer Science. Springer; 2012. p. 410-23. DOI: 10.1007/978-3-642-31612-8_31.

[13] Ansótegui C, Bonet ML, Giráldez-Cru J, Levy J, Simon L. Community Structure in Industrial SAT Instances. J Artif Intell Res. 2019;66:443-72. DOI: 10.1613/jair.1.11741.