Moshchenskii, V.A.

P=NP

Counting the number of zero assemblies and satisfiability problem

It is an english translation of the article Moshchenskii, V.A. Counting the number of zero assemblies and satisfiability problem. Publishing Center of the Belarusian State University (Minsk). 2012.pp. 16 (in Russian).

ISBN 978-985-476-988-2

Introduction. It is well known [1, amongst others] that the satisfiability problem is to determine, for each conjunctive normal form (CNF), whether there exists such input that its value is true (one), or it is a constant Boolean function with the value of zero (i.e., its value is zero on any input).

Given CNF with n variables, the number of all different inputs is 2^n. Hence if we prove that this CNF takes the value of zero only on d different inputs, $d<2^n$, then and only then is it satisfiable. This method was used in [2] and we partially use it here in a special case.

At the initial step of computation, each CNF is expressed as a word in alphabet $\{x,\bar{x},0,1,(,)\}$ [1]. For example, CNF K= $\bar{x}_1 v\, x_3 \quad \bar{x}_2 v\, x_3 v\, x_4$ translates to $\bar{x}1\, x11 \quad \bar{x}10\, \bar{x}11x100$. Note that CNF expressed as a word of length m contains no more than m elementary disjunctions (ED).

Lemma 1. Let F and G be two CNFs such that G only contains literals on the variables included in F. Then the number of binary inputs on which CNF $F\wedge G$ takes the value of one, is not greater than the number of such inputs for F.

Proof. If CNF F is not satisfiable, i.e., there is no such input that makes its value one, then it is clear that CNF $F\wedge G$ is also not satisfiable (in this case it does not matter which variables G includes).

Let us now suppose that CNF F takes the values of one on binary inputs $\{a_i\}_{i=1}^{p}$, $p \geq 1$. Since G does not have variables other than those present in F, we can find all its values G(a_i), and hence values of (F$\wedge$G)(a_i) on all these inputs a_i, $i=1, \ldots, p$. If there exists input b not found amongst $\{a_i\}_{i=1}^{p}$ such that G(b)=1, then F(b)=0 and therefore (F$\wedge$G)(b) = 0. This proves the lemma.

Remark 1. The condition imposed in Lemma 1 on G is important. Consider, for example, F=$(x1\bar{x}10)(\bar{x}1x10)$ and G=$(x11x100)$. Then F takes the value of one on 2 inputs, while the number of such inputs for F$\wedge$G is 6.

Main part. For an arbitrary CNF C let us define C1 to be the set (or CNF) of all those of its EDs containing at least one literal without negation. Further, let us define C2 to be the set (or CNF) of all those of C's EDs containing all literals with negation. Analogously, we can define C3 as the set of all those EDs in C that have at least one literal with negation. Finally, C4 is defined as complement of C3, i.e., it is the set of all EDs in C that only have literals without negation.

If at least one of the two pairs (C1, C2), (C3, C4) has an empty component, then satisfiability check for such CNF C can be performed in polynomial time equal to its length (in order to verify this condition it is enough to check whether all EDs have at least one literal with negation and at least one without. Therefore from now on we will assume that for CNF C in question both pairs (C1, C2) and (C3, C4) do not contain empty components. We will also make an assumption that in each ED literals are listed in the increasing order of variable indexes. Thus in each ED the right-most literal has the maximum index.

If an arbitrary ED consists of m different literals, let us call it m-ED.

We will further assume that considered CNFs do not contain 1-EDs and pure inclusions of literals [3, p. 69] (i.e., we do not consider CNFs that include only x_i or only $\bar{x}_i$), because the algorithms for their removal are simple and require linear time.

Remark 2. For a CNF C with the properties just described above, both sets C1 and C3 contain literals on all variables of CNF C, since the set pairs (C1, C2) and (C3, C4) do not have empty components and CNF C does not contain pure inclusion of literals.

For the purposes of manipulation and comparison of EDs below, let us point out that *r*-ED (on *n* variables) of the form

$x_{i_1}^{a_1} \vee x_{i_2}^{a_2} \vee \ldots \vee x_{i_r}^{a_r}$, $1 \le i_1 < i_2 < \ldots < i_r \le n$, at the initial step is expressed in alphabet $x, \bar{x}, 0, 1, ,$ as word $x^{a_1} d\ i_2\ x^{a_2} d\ i_2\ \ldots x^{a_r} d\ i_r$, where $d\ i_j$ is the binary representation of the number i_j. The length of $d\ i_j$ is $\lceil \log_2 i_j + 1 \rceil$, thus the length of this r-ED's notation is not greater than $2 + r \lceil log_2\ n + 1 \rceil + 1$. Below we will assume logarithm base to be 2 and omit it.

All of the algorithms described are deterministic so we will not specifically state it in each case.

THEOREM. The satisfiability problem has an efficient deterministic solution.

Proof. Given some CNF K on *n* variables expressed in alphabet $x, \bar{x}, 0, 1, ,$ by a word of length *m*. Then its number of EDs is not greater than *m* (as a reminder, this CNF does not contain 1-EDs or pure inclusion of literals). Each ED in K is *r*-ED, $r \ge 2$ (if each ED is a 2-ED, then we have 2-CNF, for which the satisfiability problem has an efficient solution [1]). Hence in every ED of this CNF there is at most *n*-2 literals missing – on those variables literals on which are not present in this ED.

If inequality holds

$$8 \lfloor \log m \rfloor \ge n - 2, \qquad (1)$$

then by introducing missing variables we will achieve CNF such that every ED in it contains literals from all n variables. This manipulation is based upon equivalence $D = (D \vee x) \cdot (D \vee \bar{x})$, where D is an ED and x a variable from the set of variables literals on which are not included in D.

Let's show that this conversion is efficient. First, let's build ED

$$x1\, x10\, xd\ 3\ \ldots xd\ i\ \ldots xd\ n\ , \qquad (2)$$

where *d (i)* is binary representation of number *i*, $1 \le i \le n$. This can be done as follows. As we agreed above, in every ED literals are listed in the ascending order of variable indexes. Hence pair-wise comparison of variable indexes in the last literal of each ED will allow to find *d(n)* in $O\ m \log n\ = O\ m^2$ steps (each step here is comparison of two binary digits). Then we keep subtracting 1 from *d(n)* in order to obtain ED (2), which will take another $O\ n \log n\ = O\ m^2$ steps.

In the end, ED (2) will be built in $O\ m^2$ steps.

Now, moving from left to right, we take the next ED from CNF K, build its duplicate (2), which we'll denote as (2*d*), and compare all numbers in literals from "the next ED" to the numbers in literals from its duplicate (2*d*). For each literal $x^a d\ i$ from "the next ED" there is a literal *xd(i)* from (2*d*). Let's transform *xd(i)* to $X^a d\ i$ (note how we move from lowercase *x* to uppercase *X*). To do this, we will need to compare, for each literal from "the next ED", binary words of length O(log *n*) with at most *n* binary words of the same length. With each ED having at most *n* literals, this will require $O\ n \cdot n \log n\ = O\ m^3$ steps for one ED and

$O\ m^4$ steps for all EDs in CNF K.

If (2d) still has lowercase letters *x*, we replace each of them with *. For example, with *n*=4 the following expression is possible:

$$*1\bar{X}10*11X100\ . \qquad (3)$$

Then, in the expression derived from (2*d*), replace all symbols * with $\bar{x}$ (here, *x* is lowercase and we regard $\bar{x}$ as a placeholder for zero. To the new expression (without *), we keep adding 1 (which we'll denote as lowercase *x* thus building all EDs until we arrive at the ED, in which there will only be lowercase *x* in front of the binary notations of variable indexes in literals that used to have * to their left (literals containing uppercase *X* are not changing).

For example, from (3) we subsequently derive $\bar{x}1\,\bar{X}10\,\bar{x}11X100$, $\bar{x}1\,\bar{X}10\,x11\,X100$, $x1\,\bar{X}10\,\bar{x}11\,X100$, $(x1\bar{X}10x11X100)$. Expression (3) represents ED $\bar{x}10\,x100$, which does not contain literals on the 1st and the 3rd variables. Using the above described method, we have built four EDs, each containing literals on all variables (although some of them are denoted by uppercase *X*).

What we have left to do is to find the number of steps required to obtain all EDs such that each of them contains literals on all *n* variables. Duplicate EDs (2) will be written at most *m* times, each ED having O(*n* log*n*) symbols. This results in $O\ mn\log n\ = O\ m^3$ steps. Also, each ED in CNF K will yield no more than 2^{n-2} EDs containing literals on all *n* variables. At most *m* EDs in the CNF will then limit the number of final EDs to $m{\cdot}2^{n-2}$. With inequality (1) in mind, we have $m\cdot 2^{n-2} \le m^9$.

Then writing all of the final EDs will involve $m^9 \cdot O\ m^3\ = O\ m^{12}$ steps.

As we see, writing all the EDs containing literals on all *n* variables can be done in polynomial time (in the length of CNF K).

Let us now exclude repeating EDs.

To do this, we compare the first ED with all the others, marking those that are equal to it – by making the opening parenthesis bold. In this comparison, we only

need to check for equality x^a and x^b that are found in front of the same binary string – since literals in each ED are listed in the increasing order of variable indexes (remember, they were derived from ED (2)).

After tackling this first ED, we proceed to the next unmarked ED and compare it to all the unmarked ones, again marking those that are equal to it. This will require $O\ m^{18}$ comparisons between two *n*-EDs, and $n \cdot O\ m^{18} = O\ m^{19}$ comparisons between symbols x^a.

Upon completion of this process, all unmarked EDs will be distinct and they will form perfect CNF. Let the number of these distinct EDs be *d*. Then if $d = 2^n$, i.e. when comparing binary notation of *d* with (2) we find 1 in position *(n+1)*, then the original CNF is not satisfiable. Otherwise, if $d \neq 2^n$, the CNF is satisfiable.

So, when inequality (1) holds, satisfiability of CNF K is determined in polynomial time (in its length *m*), equal to the total of all steps outlined above.

We can rewrite (1) in equivalent form: $m \geq 2^{n-2}/8$. Then we now have to consider and efficiently solve the satisfiability problem for those CNFs whose length *m* satisfies condition

$$m < 2^{n-2}/8. \qquad (4)$$

First, let's prove that in case $n \leq 26$ the satisfiability problem is easily solved. From (4), we arrive at $m<8$, i.e., CNF K in question contains at most 7 EDs. Then sets K1 and K2 (or K3 and K4) will be such that the total of the number of EDs in them is not greater than 7. Even if the addenda are 3 and 4, it is trivial to determine in polynomial time whether an executing assembly exists for the given CNF. Therefore from now on, we'll assume that $n>26$, along with inequality (4).

Remark 3. It may seem like the case just considered is not worth attention. Indeed, if *n* is bound by a constant, then the computational complexity, as a function of *n*, will also be bound, regardless of how quickly that function grows.

However, the significance of it is in showing how the efficiency of computation follows from the properties of partitioning a 7-set. The same properties are at work if in (1) coefficient 8 is replaced with 16, 32, etc, deriving the similar findings for n=50, n=98, etc. It already follows from here that P ≠ NP is impossible. Indeed, suppose P≠NP. Then for any deterministic algorithm that checks satisfiability of a CNF, there can be found at least one n such that this algorithm will require non-polynomial time to check satisfiability of some CNF E with n variables. Then we choose, instead of coefficient 8 in inequality (1), minimal coefficient 2^q, so that $3 \cdot 2^q + 2 \geq n$. From here we have $(n\text{-}2) / 2^q \leq 3$, and thus satisfiability of E can be determined in polynomial time.

(Another proof of impossibility of P≠NP is given in [4], which also references a negative review – however, that review later turned out to have errors).

Let us attend to the theorem's proof. As a reminder, inequality (4) holds and n>26. For given CNF K, we perform its efficient conversion [1] into 3-CNF H. This efficient conversion is based upon the following equivalence, in satisfiability terms:

$$x \vee y = \; x \vee z \quad y \vee \overline{z} \;\; ,$$

where z is the new variable not found in the set of variables of CNF K. The conversion is applied to all s-ED, $s \geq 4$, that are either in K or created during the conversion process.

In the 3-CNF H, some 3-EDs and 2-EDs may contain literals both with negation and without it. Let's then apply to these EDs the following conversion, which, again, preserves satisfiability:

$$\overline{x}_i \vee \overline{x}_j \vee x_k = \; \overline{x}_i \vee \overline{x}_j \vee \overline{y}_1 \quad x_k \vee y_1 \;\; ,$$

$$\overline{x}_i \vee x_j \vee x_k = \; \overline{x}_i \vee \overline{y}_2 \quad x_j \vee x_k \vee y_2 \;\; ,$$

$$\overline{x}_i \vee x_j = \; \overline{x}_i \vee \overline{y}_3 \quad x_j \vee y_3 \;\; ,$$

where y_i are newly introduced variables, unique to each converted ED.

As a result, we obtain 3-CNF E, which is satisfiability-equivalent to 3-CNF H. In 3-CNF E, every ED contains literals either only with negation or only without it; we will call E divisible. Note that for divisible 3-CNF E

E2=E3 and E1=E4,

and also from Remark 2 follows

Remark 4. For a divisible CNF E, both sets E1 and E2 contain literals on all variables because any newly introduced variable cannot be a pure inclusion.

Let's agree to denote new variables being introduced as y_i (in CNF K, we used variables $x_1, x_2, \ldots, x_n$) . Let's further agree that in all EDs of the 3-CNF E we write all literals on new variables in the right-most position; if there are more than one such literal, the right-most position will be taken by the one with the greatest variable index.

Let CNF E have $n+p$ variables, where p is the number of newly introduced ones, and E's length be r. Obviously, $n + p \leq r$. It is easy to prove that p and r have an upper bound that is polynomial in m. Below we will be expressing time complexities as polynomials in r, hence also polynomials in m, since composition of polynomials is a polynomial.

Lemma 2. If cardinality of either E2 or E1 does not exceed $8\lfloor log\ r \rfloor$, then satisfiability of CNF E is determined in time polynomial in r, the length of E.

Proof. Let the above inequality hold for E2 (the case with E1 is considered analogously).

Let's build set A of all variables, literals on negations of which are present in all EDs in E2; A is not empty and the number of variables in it is not greater than $t=24\lfloor log\ r \rfloor$. As noted previously, CNF E has $n+p$ variables, where n is the number of variables x_i in the original CNF K and p is the number of y_i introduced during conversion of CNF K into 3-CNF; we also agreed that in the 3-CNF E

every ED has the literal on the new variable (if any) written in the right-most position, if there is more than one such literal (but there can't be more than three), then the right-most position is taken by the one with the greatest index. Then if we consider all EDs in CNF E that contain new variables, pair-wise comparison of their right-most literals will allow to find binary representation of *p*: out of two EDs we choose the one whose right-most literal has a greater index, then the chosen one is compared against next ED and so forth. Comparison of two EDs requires O(log *p*) steps. Hence we'll arrive at binary representation of *p* in $O(\log p \cdot r) = O(r^2)$ steps. Then, by subtracting 1 from *p* and using ED (2), let's build the following ED

$$x1x2\ldots xd\ i\ \ldots xd\ n\ \ y1y2\ldots yd\ \ j\ \ldots yd\ \ p\ \ , \qquad (5)$$

where *d*(*s*) has the same meaning as in (2). This process will take $O(p\log p) + O(n\log n) = O(r^2)$ steps.

Finally, let's build duplicate ED (5), which we'll denote (5*d*), in $O(r^2)$ steps. For each ED in E2, we'll mark in (5*d*) those *xd*(*i*) or *yd*(*j*), literals on which the ED contains. This will require $3r \cdot O(d\ n) + d\ p) = O(r^2)$ steps. If upon completion, (5*d*) has unmarked literals, we delete them in *O*(*r*) step, eventually obtaining ED

$$xd\ i_1\ xd\ i_2\ xd\ i_k\ \ yd\ \ j_1\ yd\ \ j_2\ \ldots yd\ i_s\ \ , \qquad (6)$$

which contains literals on all variables from A (here, $i_t \le n, a\ j_q \le p$).

Let's now move onto the important issue of finding all possible subsets of set A, besides the empty set – there'll be no more than $2^t - 1 < r^{24}$ such subsets. For this, we'll employ algorithm suggested in [5, p.34], which is based on binary notation. First, we create (*k*+*s*)-long binary assembly with a single 1: (0, 0,.., 0, 1).

Then we build all the other ones, of the same length, by adding 1 (in binary) to the previous assembly – until we have 1's in all (k+s) positions.

When building the first assembly, we look through the literals in ED (6) from right to left and upon encountering literal $yd\ \ j_s$, write 1, along with a comma to its left and closing parenthesis to its right; then we write all the zero components upon encountering $zd(s)$ (where z is either x or y). Thus the first assembly will take O(k+s)=O(r) steps to build. Addition of 1 will require O(r) steps. In the end, all the aforementioned binary assemblies will be built in $2^t \cdot O\ \ r\ \ = O\ \ r^{25}$ steps.

Lastly, we need to set all variables in every subset of A to 0, and the remaining variables in CNF E to 1. To do this, we take one of the created binary assemblies and consider it side-by-side with ED (6), writing out those literals from (6) whose positions correspond to 1's in the assembly. Then we build duplicate ED (5), which will have literals identical to those we have just written out. In those literals, let's replace variables with uppercase letter O. For example, we may arrive at expression like ($x1\ O10\ x11\ldots\ xd(n\text{-}1)\ Od(n)\ O1\ y10\ldots yd(p)$); in these expressions, lowercase x and y will be regarded as 1's. With these conventions, we have a set of values for all variables in CNF E, where zero values (uppercase O's) are assigned only to variables derived using the specific chosen binary assembly of the length k+s.

Let's count the number of steps needed to obtain all sets of values for variables in CNF E. Writing out all the relevant literals when considering side-by-side one binary assembly and ED (6) takes O(d+s)=O(r^3) steps. Hence building all sets will take $r^{24} \cdot O\ \ r^3\ \ = O\ \ r^{27}$ steps.

Once we're done creating this set of inputs, we need to find the value of CNF E on these inputs and pay attention to those, on which E is 1. To find the value of

E on one input, requires comparison of one binary word, at most 3·O(log r) long, with at most 3r words of the same length, which will take O(3· log r ·3r)=O(r^2) steps. Then the number of steps required to find the value of E on all inputs is $r^{24} \cdot O\ r^2\ = O\ r^{26}\ .$

Let's now make sure that the above algorithm will find the executing assembly for 3-CNF E, if one exists. According to Remark 2, for given divisible 3-CNF E both sets E1 and E2 contain literals on all variables, with E1 containing all literals without negation, E2 – all literals with negation. This means that executing assembly for E must contain both 0's and 1's. Our algorithm goes through all possible sets of variables in literals of E2, and all these variables are assigned the value of 0. Then if the CNF is satisfiable, we will find all the variables whose values in the executing assembly are 0. Lastly, let's note that the subset of variables with the value of 1 (which is a complement of the set of "zero variables") will yield 1 in all EDs in E1 – assuming there exists an executing assembly for E. Thus we have shown that the executing assembly, if it exists, will indeed be found. Since all time complexities we've outlined are polynomial in r (the maximum one being $O\ r^{27}\ $), and sum of polynomials yields a polynomial, this concludes the proof of the lemma.

Let us attend to the final step in proving the theorem. If either E2 or E1 is such that the condition of Lemma 2 holds, we are done. Otherwise, we will use the following algorithm to find the executing assembly of E.

Let's partition E2 into classes E_i so that for each i (except, perhaps, one) $|E_i| = 8\lfloor \log r \rfloor$; the number of such classes is $v(r)$=O(r / log r). Then according to Lemma 2, we can efficiently solve the satisfiability problem for each CNF E1 E_i obtaining T(i) – sets of executing assemblies – in the process, which will re-

quire $v\ r\ \cdot O\ r^{27}\ = O\ r^{28}$ steps (as a reminder, $O\ r^{27}$ is the highest-power polynomial from Lemma 2 proof). If any of the T(*i*) is empty, E is not satisfiable.

Suppose none of the T(*i*) is empty. Then for each pair *a* and *b* from T(1) and T(2) respectively, we build binary assembly *c*, whose components are conjunctions of the relevant (i.e., those in the same position) components in *a* and *b*. All such binary assemblies *c* collectively form set D□ (1). (From now on, we'll call assemblies *c* conjunctive; if *a* and *b* are the same, their conjunctive assembly is equal to either of them). Since cardinality of each T(*i*) is not greater than r^{24}, cardinality of D□ (1)cannot be greater than r^{48}. To obtain one conjunctive assembly, we'll need O(log *r*) steps (we go through pairs of components in *a* and *b* one-by-one, performing conjunction on them). Thus building D□ (1) will take O(r^{49}) steps.

For each such set D□ (1) we check satisfiability of CNF E1□ E1□E2, coming by D(1), the set of all executing assemblies. Since checking one input of E for satisfiability requires O(r·log$r$) = O(r^2) steps, D(1) will be built in r^{48}·O(r^2)=O(r^{50}) steps. From Lemma 1, we have |D(1)| ≤ |T(1)|, as E1 and E2 contain all variables found in CNF E. If D(1) is empty, then E is not satisfiable; if D(1) is not empty and *i*=2, then E is satisfiable and the proof is concluded.

If D(1) is not empty and *i*≥3, then we build set D□ (2) of all conjunctive assemblies from the assemblies found in D(1) and T(3). For each such assembly, we check satisfiability of CNF E1□ E1□E2□E3, denoting the set of all executing assemblies as D(2) (again, from Lemma 1: |D(2)|≤ |D(1)|) and so forth.

Building one set D(*i*) takes O(r^{50}) steps. The number of sets D(*i*) is v(r) –2, and cardinality of each of them is not greater than r^{48}. Then building them all will take (v(r) –2)·O(r^{50})=O(r^{51}) steps. If at least one of them is empty, CNF E is not satisfiable, and satisfiable otherwise. Since all time complexities outlined are po-

lynomial, and the sum of polynomials is a polynomial, we have thus proved the theorem.

Satisfiability problem is known to be NP-complete, hence we arrive at the

COROLLARY. NP = P.

References

1. Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи. М.: Мир, 1982
2. Мощенский В.А., Тихоненко Т.П. // Вестник БГУ. Сер. 1. 1999. N 1
3. Чень Ч., Ли. Р. Математическая логика и автоматическое доказательство теорем. М.: Наука, 1983
4. Мощенский В.А. Из неравенства классов NP и P вытекает противоречие // Международный конгресс по информатике: информационные системы и технологии. Часть 2. Мн.: БГУ, 2011, сс. 318-320
5. Липский В. Комбинаторика для программистов. М.: Мир, 1988
6. Мощенский В.А. Подсчет числа нулевых наборов и проблема выполнимости. Мн.: Изд. центр БГУ, 2005