An efficient probabilistic algorithm for an important NP complete problem in mathematics

Cristian Dumitrescu

Abstract. In this article I describe an efficient, randomized algorithm (section 3) that I think solves the 3- SAT problem (known to be NP complete) with high probability in polynomial time, and a bit of the history of the problem under consideration. In the last section I present an interesting application, based on an idea that belongs to Godel. The appendix contains a Markov chain model for the dynamics of the algorithm.

Keywords. The Satisfiability Problem, Markov chains, random walk with absorbing barriers.

Introduction. In this article I propose an algorithm that has the potential of changing the way we do mathematics (based on an idea that belongs to Godel), and also has many applications in many fields of activity. I build on the work of Professor Uwe Schoning, but my algorithm is more efficient. Ultimately, my algorithm is inspired by nature, since it can be related to gene acquisition/deletion and speciation in the greater theory of evolution.

Section 1. Useful notions that are used for the analysis of the algorithm.

A Boolean expression is said to be in conjunctive normal form (CNF) if it is of the form $E_1 \wedge E_2 \wedge E_3 \wedge \ldots\ldots\ldots \wedge E_k$, and each E_i, called a clause (or conjunct), is of the form $\alpha_{i1} \vee \alpha_{i2} \vee \alpha_{i3} \ldots\ldots\ldots \vee \alpha_{ir}$, where each α_{ij} is a literal, either x or $\neg x$, for some variable x.

A Boolean expression is said to be in disjunctive normal form (DNF) if it is of the form $F_1 \vee F_2 \vee F_3 \ldots\ldots\ldots \vee F_k$, and each F_j, called a clause (or disjunct), is of the form $\beta_{j1} \wedge \beta_{j2} \wedge \beta_{j3} \wedge \ldots\ldots\ldots \wedge \beta_{jr}$, where each β_{jk} is a literal, either y or $\neg y$, for some variable y.

A Boolean expression in CNF form is called satisfiable if there is some assignment of 0's and 1's to the variables that gives the expression the value 1.

The satisfiability problem is to determine, given a Boolean expression, whether it is satisfiable.

An expression is said to be 3 - CNF if each clause has exactly three distinct literals.

Theorem 1 (see reference [1]). L_{3SAT}, the satisfiability problem for 3 - CNF expressions, is NP - complete.

The Hamming distance $d_H(\mathbf{x}, \mathbf{y})$ between two vectors **x**, **y** is the number of components in which they differ. It is known that the Hamming distance $d_H(\mathbf{x}, \mathbf{y})$

satisfies the conditions for a metric.

Related to the theory of symmetric random walks (in one dimension), we have the following theorem.

Theorem 2 (see reference [2]). Limit theorem for first passages. For fixed t, the probability that the first passage through r occurs before epoch $t \cdot r^2$ tends to $P = \sqrt{\frac{2}{\pi}} \cdot \int_{\frac{1}{\sqrt{t}}}^{\infty} e^{-\frac{1}{2}s^2} ds = 2 \cdot \left(1 - N\left(\frac{1}{\sqrt{t}}\right)\right)$, as $r \to \infty$, where N is the normal distribution function. We note that when $t \to \infty$, then P tends to 1.

Section 2. The description of Schoning's algorithm.

Input: a formula in 3-CNF with n variables.

Guess an initial assignment for the n variables, uniform at random.

Repeat 3n times:

> *If the formula is satisfied by the actual assignment: stop and accept.*
>
> *Let C be some clause not being satisfied by the actual assignment.*
>
> *Pick one of the 3 literals in the clause at random, and flip its value in the current assignment.*

Schoning proves (see reference [3]) that the complexity of k-SAT (with this algorithm) is within a polynomial factor of $\left(2 \cdot \left(1 - \frac{1}{k}\right)\right)^n$. This means that this algorithm does not have direct practical value, since the expected time needed to hit a solution grows exponentially with the number of variables.

Section 3. The presentation of the DEA class of algorithms.

3.1. The proposed dual expression algorithm (DEA2).

Basically, given a 3-CNF expression, we want to construct a 2-CNF path through all the 3-clauses from the 3-CNF expression. In the following, I will explain what I mean by a 2-CNF path though the 3-CNF expression. Each 3-clause in the original 3-CNF expression has the form $x_i \vee x_j \vee x_k$, where each x_i represents a variable or the negation of one of the original variables that appear in the given 3-CNF expression. For example, the 3-clause $(a \vee \neg b \vee \neg c)$ contains all the 2 – clauses $(a \vee \neg b)$, $(a \vee \neg c)$, and $(\neg b \vee \neg c)$.

By the active 2-CNF chain, I mean a conjunction of 2 – clauses so that each 2 – clause is chosen from a corresponding 3 – clause from the original 3-CNF expression. In the current state (during the execution of the algorithm), the length of the active 2-CNF chain could be equal to the number of clauses or less than the number of clauses in the original 3-CNF expression.

The function Test2CNF will check whether the conjunction of all the 2 – clauses from the active 2-CNF chain (in the current state) is satisfiable or not. We note that we do not work with the original variables, the function Test2CNF is just a 2-CNF tester/solver, it will not look for a satisfiable assignment in the original variables unless all the 3 – clauses from the original 3-CNF expression are satisfied.

We also notice that 2-SAT can be solved in linear time. The reason why I call it DEA2 (dual expression algorithm) has to do (intuitively) with the fact that the 3 – clauses from the original 3-CNF expression, for example, $(a \vee \neg b \vee \neg c)$ can also be written as $(a \vee \neg b) \vee (a \vee \neg c) \vee (\neg b \vee \neg c)$, and I look at the 2-clauses as some sort of new variables (but the way you look at it does not affect the understanding of the algorithm, but this is the way I thought about it when I invented the algorithm).

We also assume that we order the clauses in the 3-CNF expression such that when we talk about the next unsatisfied clause (for example), or the first clause satisfying a certain condition, we know which clause we are talking about. When we choose a clause at random, it does not matter.

Here is the algorithm, the **dual expression algorithm (DEA2)**:

Input: a formula in 3-CNF with n variables x_i.

Choose a 2 – clause from the first 3 – clause (of the given 3-CNf expression) and add it to the active 2-CNF chain.

Repeat A(n) times (where A(n) is a polynomial discussed later):

> *Call the routine Test2CNF for the conjunction of all the 2 – clauses that are currently part of the active 2-CNF chain.*
>
> *If Test2CNF finds that the active 2-CNF chain is satisfied (I will sometimes say that the chain is consistent), and if all the 3 – clauses from the original 3-CNF expression are satisfied, then the original 3-CNF expression is satisfied, and we finished, return "expression satisfiable".*
>
> *If Test2CNF finds that the active 2-CNF chain is satisfied (I will sometimes say that the chain is consistent), but not all the 3 – clauses from the original 3-CNF expression are satisfied (or not considered yet), then choose a random unsatisfied 3 – clause, and choose a random 2 – clause from it (from the three possible) and add this 2 – clause to the active 2-CNF chain.*
>
> *If Test2CNF finds that the active 2-CNF chain is not satisfied (I will sometimes say that the chain is inconsistent), then discard a 2 – clause (chosen at random) from the active 2-CNF chain.*
>
> *Repeat cycle.*

If a solution has not been found yet, return "expression unsatisfiable".

In a slightly different version of this algorithm, if Test2CNF finds consistency, then we look at all 2- clauses (there are three of them) from an unsatisfied 3 - clause such that the 3 – clause can be satisfied while maintaining consistency of the active 2-CNF chain (we try to force consistency). The modified version of the DEA2 algorithm can be written as follows:

Input: a formula in 3-CNF with n variables x_i.

Choose a 2 – clause from the first 3 – clause (of the given 3-CNF expression) and add it to the active 2-CNF chain.

Repeat A(n) times (where A(n) is a polynomial discussed later):

> *Call the routine Test2CNF for the conjunction of all the 2 – clauses that are currently part of the active 2-CNF chain.*
>
> *If Test2CNF finds that the active 2-CNF chain is satisfied (I will sometimes say that the chain is consistent), and if all the 3 – clauses from the original 3-CNF expression are satisfied, then the original 3-CNF expression is satisfied, and we finished, return "expression satisfiable".*
>
> *If Test2CNF finds that the active 2-CNF chain is satisfied (I will sometimes say that the chain is consistent), but not all the 3 – clauses from the original 3-CNF expression are satisfied (or not considered yet), then choose at random an unsatisfied 3 – clause, and choose another 2 – clause from it (from the three possible, in order) and add this 2 – clause to the active 2-CNF chain only if the 2-CNF chain stays consistent, otherwise do not add any 2 – clause to the active 2-CNF chain. Basically, we add a 2- clause to the active 2-CNF chain only if we are sure that the chain stays consistent.*
>
> *If Test2CNF finds that the active 2-CNF chain is not satisfied (I will sometimes say that the chain is inconsistent), then discard a 2 – clause (chosen at random) from the active 2-CNF chain.*
>
> *Repeat cycle.*

If a solution has not been found yet, return "expression unsatisfiable".

3.2. The proposed dual expression algorithm (simplified version, DEA1).

We will consider now a simplified version of the algorithm, a version that is more intuitive and easy to understand. We will call a literal, a variable or its negation. For example, x or $\neg x$ are literals. We also note that the negation of the literal x is $\neg x$, and the negation of the literal $\neg x$ is x. Given a 3-CNF expression, by a 1-CNF path through all the clauses of the 3-CNF expression we mean a conjunction of literals, one from each clause, such that once a literal appears in the 1-CNF path, its negation does not appear in the conjunction.

By the active 1-CNF chain, I mean a conjunction of literals so that each literal is chosen from a corresponding 3 – clause from the original 3-CNF expression. In the

current state (during the execution of the algorithm), the length of the active 1-CNF chain could be equal to the number of clauses or less than the number of clauses in the original 3-CNF expression.

The function Test1CNF will check whether the conjunction of all literals from the active 1-CNF chain (in the current state) is satisfiable or not. We note that in this case we work with the original variables. The function Test1CNF basically checks that if one literal appears in the active 1-CNF chain, then its negation does not appear in it.

Here is the algorithm, the **dual expression algorithm (DEA1)**:

Input: a formula in 3-CNF with n variables x_i.

Choose a literal from the first 3 – clause (of the given 3-CNf expression) and add it to the active 1-CNF chain.

Repeat A(n) times (where A(n) is a polynomial discussed later):

> *Call the routine Test1CNF for the conjunction of all the literals that are currently part of the active 1-CNF chain.*
>
> *If Test1CNF finds that the active 1-CNF chain is satisfied (I will sometimes say that the chain is consistent), and if all the 3 – clauses from the original 3-CNF expression are satisfied, then the original 3-CNF expression is satisfied, and we finished, return "expression satisfiable".*
>
> *If Test1CNF finds that the active 1-CNF chain is satisfied (I will sometimes say that the chain is consistent), but not all the 3 – clauses from the original 3-CNF expression are satisfied (or not considered yet), then choose a random unsatisfied 3 – clause, and choose a random literal from it (from the three possible) and add this literal to the active 1-CNF chain.*
>
> *If Test1CNF finds that the active 1-CNF chain is not satisfied (I will sometimes say that the chain is inconsistent), then discard a literal (chosen at random) from the active 1-CNF chain.*
>
> *Repeat cycle.*

If a solution has not been found yet, return "expression unsatisfiable".

In a slightly different version of this algorithm, if Test1CNF finds consistency, then we look at all the literals (there are three of them) from an unsatisfied 3 - clause such that the 3 – clause can be satisfied while maintaining consistency of the active 1-CNF chain (we try to force consistency). The modified version of the DEA1 algorithm can be written similarly to the modified version of the DEA2 algorithm (the terminology emphasizes this analogy).

Section 4. Godel's letter to von Newmann.

For general implications, related to efficiently solving NP – complete problems, see

[4]. An interesting application is related to the problem of automated theorem proving using an efficient algorithm for NP – complete problems.

We know that we can solve the following problem in polynomial time:

Given two well formed formulas α and β, in a given axiomatic system (like ZFC), is β a ZFC – proof of α?

Therefore, the following problem is in NP (it can be easily proved):

Given a formula α, and a number n, is there a ZFC – proof of size at most n for α?

Any efficient solution for NP-complete problems would make automated theorem proving a reality. We can have an automated system that would tell us (with probability as close to 1 as we want) that no solution to a given problem exists, that can be written in (for example) less than 10000 pages, or hit upon (find) such a proof.

In a letter in 1956, Godel asked John von Newmann whether there was a general method to find proofs of size n, using time that increases only as n or n^2. If such a method existed, Godel argued that this *" would have consequences of the greatest magnitude. That is to say, it would clearly indicate that the mental effort of the mathematician in the case of yes or no questions could be completely replaced by machines. One would indeed have to simply select an n so large that, if the machine yields no result, there would then also be no reason to think further about the problem. "*.

This is not just a problem of optimization, or applied mathematics. I think that this problem should be the focus of attention for the core of the mathematicians, a problem the solution of which could transform mathematics and fulfill (to some extent) Hilbert's dream, by following an idea that belongs to Godel.

Another interesting path is to consider quantum algorithms, quantum random walks, in particular, but we will not go into this issue here.

Conclusions. For all practical purposes, we can assume that P = NP, even if the conjecture P ≠ NP might be true, if we exclude randomized algorithms. This article can be considered a review article, but the ideas expressed in section 3, the DEA algorithm are original though.

Appendix. In this appendix, we will give one example of the DEA2 algorithm dynamics for a very simple 3 – CNF expression, and we will present an approximate Markov chain model for the DEA2 algorithm (a similar analysis can be made for DEA1).

A1. Example. We will give an example of the dynamics of the DEA2 algorithm, in a very simple case. We consider the 3 – CNF expression:

$$E = (a \vee b \vee c) \wedge (\neg a \vee \neg b \vee c) \wedge (\neg a \vee b \vee \neg c) \wedge (a \vee \neg b \vee \neg c).$$

We choose the 2-clause $(a \vee b)$ from the first 3-clause of the 3-CNF expression and

we add it to the active 2-CNF chain. The active 2-CNF chain will be, at this moment:

$C \equiv (a \vee b)$

We call the function Test2CNF. The active 2-CNF chain (at this moment) is satisfiable (I will also say consistent). Assume that we choose the 2-clause, $(\neg a \vee \neg b)$ from the second 3-clause of the 3-CNF expression E, and we add it to the active 2-CNF chain C. We have then:

$C \equiv (a \vee b) \wedge (\neg a \vee \neg b)$

We call the function Test2CNF. The active 2-CNF chain (at this moment) is satsifiable (consistent). Assume that we choose the 2-clause $(\neg a \vee b)$ from the third 3-clause of the 3-CNF expression E, and we add it to the active 2-CNF chain C. We have then:

$C \equiv (a \vee b) \wedge (\neg a \vee \neg b) \wedge (\neg a \vee b)$

We call the function Test2CNF. The active 2-CNF chain (at this moment) is satisfiable. Assume that we choose the 2-clause $(a \vee \neg b)$ from the fourth 3-clause of the 3-CNF expression E, and we add it to the active 2-CNF chain C. We have then:

$C \equiv (a \vee b) \wedge (\neg a \vee \neg b) \wedge (\neg a \vee b) \wedge (a \vee \neg b)$

We call the function Test2CNF. This time the active 2-CNF chain is not satisfiable (not consistent). Now we have to discard a 2-clause from the active 2-CNF chain. Assume that we discard the 2-clause $(\neg a \vee b)$, taken from the third 3-clause of the 3-CNF expression. We have then:

$C \equiv (a \vee b) \wedge (\neg a \vee \neg b) \wedge (a \vee \neg b)$

We call the function Test2CNF. The active 2-CNF chain is satisfiable. Assume that we choose the 2-clause $(\neg a \vee \neg c)$ from the third 3-clause of the 3-CNF expression (because at this moment this is the clause that is not satisfied yet), and we add it to the active 2-CNF chain. We have then:

$C \equiv (a \vee b) \wedge (\neg a \vee \neg b) \wedge (a \vee \neg b) \wedge (\neg a \vee \neg c)$

We call the function Test2CNF. The active 2-CNF chain is satisfiable. We also notice that all the 3-clauses from the 3-CNF expression E are satisfied. As a consequence, the original 3-CNF expression E is satisfiable, and we can finally ask Test2CNF to give us a solution in terms of the original variables $a = 1, b = 0, c = 0$, for example.

A2. The Markov chain approximate model (for DEA2). We will call a correct 3 - clause, a clause from where a correct 2 - clause has been chosen in the current active 2-CNF chain. A 3 - clause from which an incorrect 2 - clause has been chosen will be called an incorrect clause. We will consider the pairs (i, j), where i is the number of correct clauses from which the 2 - clauses have been chosen in the current active 2-CNF chain, and j is the number of incorrect clauses from which the 2 - clauses have been chosen in the current active 2-CNF chain. In this case, $i + j$ will be the length of

the current active 2-CNF chain, and in general we have $i + j \leq N$, where N is the number of clauses in our 3 – CNF expression. We note that when we talk about correct and incorrect clauses, the assumption is that a solution does exist, otherwise this terminology is not meaningful.

We assume that in the current state of the Markov chain, Test2CNF has found consistency, then (following the algorithm) we choose another 2 - clause from an unsatisfied 3 - clause, and we add it to the current active 2-CNF chain. We have the following transition probabilities for the Markov chain:

$p\big((i,j),(i+1,j)\big) = \frac{2}{3}$. We choose the correct 2- clause from the unsatisfied 3 – clause with probability $\frac{2}{3}$.

$p\big((i,j),(i,j+1)\big) = \frac{1}{3}$. We choose the incorrect 2 - clause from the unsatisfied 3 - clause with probability $\frac{1}{3}$.

If in the current state of the Markov chain, Test2CNF has found inconsistency, then (following the algorithm) we discard a (random) 2 - clause from the current active 2-CNF chain. We have the following transition probabilities for the Markov chain:

$p\big((i,j),(i-1,j)\big) = \frac{i}{i+j}$ We discard a 2 - clause from a correct 3 - clause with probability $\frac{i}{i+j}$.

$p\big((i,j),(i,j-1)\big) = \frac{j}{i+j}$. We discard a 2- clause from an incorrect 3 - clause with probability $\frac{j}{i+j}$.

We assume that the Markov chain hits consistency with probability α, and it hits inconsistency with probability β, where $\alpha + \beta = 1$. We need to properly define α and β (and prove that they exist). This problem is not addressed in the current version of this article. At the moment, we can assume that we consider all the runs of the algorithm in which we have T calls to the function Test2CNF, and in αT of them the function finds that the active chain is consistent, and in βT of them the function finds that the active chain is inconsistent (how probable that is, this question is not answered here). The approximate model will be a Markov chain with transition probabilities:

$p\big((i,j),(i+1,j)\big) = \frac{2\alpha}{3}$.
$p\big((i,j),(i,j+1)\big) = \frac{\alpha}{3}$.
$p\big((i,j),(i-1,j)\big) = \frac{i\beta}{i+j}$.
$p\big((i,j),(i,j-1)\big) = \frac{j\beta}{i+j}$.

Since we know that $i + j \leq N,\ i \geq 0,\ j \geq 0$, that means that the Markov chain dynamics takes place on a 2 – dimensional grid inside the triangle OAB, where O(0,0), A(N, 0), and B(0, N). When the state representing our current state moves on

the grid inside the triangle OAB (one unit up, down, left or right), we consider the projection on the i – axis. The motion of this projection will not be governed by a Markov chain, but we can couple it with the dynamics of a Markov chain in such a manner that if this Markov chain (a birth and death chain, in fact) is expected to reach the point A(N,0) in linear time (linear in N), then the projection mentioned above will also be expected to reach A(N,0) in linear time. This birth and death chain will have the transition probabilities (we do not need the value of p(i,i)):

$p(i,i+1)) = \frac{2\alpha}{3} = p_i.$
$p(i,i-1) = \beta = q_i$, *we note that* $\beta \geq \frac{i\beta}{i+j}$.
$p(i,i) = 1 - p_i - q_i.$

We note that this birth and death chain will have the transition probability towards A(N,0) equal to the corresponding transition probability of the projection mentioned above, and the transition probability away from A(N,0) will be greater or equal to the corresponding transition probability of the projection. As a consequence, if this chain is expected to reach A(N,0) in linear time, then the projection will also reach A(N,0) in expected linear time.

Let t(i, i+1) be the random time it takes for the Markov chain to go from i to i+1. An analysis based on conditioning on the first transition and iteration leads us to the relation:

$$E\big(t(i,i+1)\big) = 1 + \frac{q_i}{p_i} + \frac{q_i \cdot q_{i-1}}{p_i \cdot p_{i-1}} + \cdots + \frac{q_i \cdot q_{i-1} \cdots q_1}{p_i \cdot p_{i-1} \cdots p_1}.$$

This formula is valid for any birth and death chain. We note that $\frac{\beta i}{i+j} \leq \beta$, and if $\beta < \frac{2\alpha}{3}$, then $\frac{q_i}{p_i} < 1$, and the chain will hit A(N, 0) in linear time in N. Starting from anywhere inside the triangle OAB (following a distribution that can be easily calculated), and staying inside the triangle, the Markov chain will reach the point A(N, 0) in expected linear time in N. The condition $\beta < \frac{2\alpha}{3}$, together with the equation $\alpha + \beta = 1$ will lead us to the conclusion that $\beta < \frac{2}{5}$ is a sufficient condition, so that our Markov chain will hit A(N,0) in expected linear time in N. If our Markov chain (the approximation considered here) hits inconsistency less than 40% of the time, then it will find a solution in expected linear time in N, the number of clauses in the 3-CNF expression. We note that in the second version of the DEA algorithm (considered above), where we avoid inconsistency as much as possible, the probability of hitting inconsistency could easily be less than 40%. of the time. Since N is at most cubic in n (the number of variables x_i), we see that our algorithm is at most biquadratic in n (taking into consideration the time needed for each call of Test2CNF), the number of variables x_i (but in most cases much smaller time is sufficient). In fact, the expected time to hit a solution (if it exists) has the order of magnitude $n \cdot N$ (up to a constant), where n is the number of x_i variables and N is the number of clauses in the 3 – CNF expression. We can the take A(n) from section 3 of the form $C \cdot n \cdot N$, where C is a constant. In the most interesting cases, when $\frac{N}{n}$ is around 4.5, the DEA2 algorithm is quadratic.

The power of the DEA algorithm is in the fact that a current active 2-CNF chain is connected to a whole class of assignments for the x_i variables, in other words, we work with many x_i assignments in parallel, at every step of the algorithm.

References (in the order in which they appear mentioned in the article)**:**

[1]. J. E. Hopcroft, J. D. Ullman, *" Introduction to Automata Theory, Langiages, and Computation "*, Addison - Wesley Publishing Company, 1979.

[2]. W. Feller, *" An Introduction to Probability Theory and Its Applications "*, John Wiley & Sons, 1968.

[3]. Uwe Schoning, *" A probabilistic Algorithm for k-SAT and Constraint Satifaction Problems"*, Research Supported by the ESPRIT Basic Research, 1991.

[4]. L. Fortnow, *" The Golden Ticket, P, NP, and The Search For The Impossible "*, Princeton University Press, 2013.

Cristian Dumitrescu,
119 Young St., Ap. 11,
Kitchener, Ontario N2H 4Z3,
Canada.

Email: cristiand43@gmail.com
cristiand41@hotmail.com

Tel : (519) 574-7026