An Efficient Algorithm for 3-SAT

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, Hamming distance, random walk with absorbing barriers.

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 ¬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 ¬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 proposed dual expression algorithm (DEA).

We also notice that 2-SAT can be solved in linear time (one of the proofs is based on theorem 2 above). We can then look at any clause E in a 3 – CNF expression of the form $x_i \vee x_j \vee x_k$, where each x_i is a literal, either x or $\neg$x, for some variable x. If we then write $z_{ij} = x_i \vee x_j$, then we can write E as $z_{ij} \vee z_{ik} \vee z_{jk}$. An unsatisfied clause has at most one of the z_{ij}'s set to 1 (in our algorithm, an unsatisfied clause will have all z_{ij} set to 0). In a satisfied clause, written as $z_{ij} \vee z_{ik} \vee z_{jk}$, at least two of the z_{ij} 's are 1. Before the presentation of the algorithm, we need some notation. We write Test2CNF for the function that tests if a certain 2 – CNF expression has a solution (and we know that it works in linear time). We start with n variables x_i, with the negations $\neg x_i$, we have 2n symbols. That means that the z_{ij}'s must represent no more than $\binom{2n}{2}$ symbols. We will call the z_{ij}, the **dual variables**. When given a 3 – CNF expression, we can always write it with the help of the dual variables. In this form, we will call it the dual 3 – CNF expression (not to be confused with duality between CNF and DNF).

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

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

We write the corresponding dual 3 – CNF expression in the z_{ij} variables. There will be at most $\binom{2n}{2}$ z_{ij}- variables involved in the dual expression. We can form a

*binary vector with these z_{ij} - variables, call it the **dual vector**. The length of the dual vector will be the number of clauses in the 3 – CNF expression.*

Guess an initial assignment for the z_{ij}- variables , uniform at random. Basically, we randomly choose one z_{ij} from each clause and set its value to 1.

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

> *Call the routine Test2CNF for the conjunction of all the z_{ij} that are currently assigned value 1 (this will be a 2 – CNF expression in the x_i – variables, and we will call it the active chain of z_{ij}).*
>
> *If Test2CNF finds that this conjunction of all the z_{ij}'s (that are currently assigned value 1) is satisfied, and if all clauses are satisfied, then stop and accept.*
>
> *If Test2CNF finds that this conjunction of all the z_{ij}'s (hat are currently assigned value 1) is satisfied, but not all clauses are satisfied, then find the first unsatisfied clause, and flip a random z_{ij} from that clause that currently has value 0 (change its value from 0 to 1).*
>
> *If Test2CNF finds that this conjunction of all the z_{ij}'s (that are currently assigned value 1) is not satisfied, then choose a random z_{ij} that is currently set to 1 and flip its value to 0. In other words, if Test2CNF finds inconsistency, then discard a random z_{ij} from the chain of z_{ij}'s that are currently set to 1.*
>
> *Repeat cycle.*

The difference between this and Schoning's algorithm is that when we choose at random a z_{ij}- variable and assign it the value 1 (flip its value from 0 to 1), we are right with probability $\frac{2}{3}$, in other words, the probability of decreasing the Hamming distance between this dual vector and the possible dual vector solution is $\frac{2}{3}$. The polynomial A(n) can be taken as $C \cdot N^2$, where C is a large constant, and N is the number of clauses in the 3 – CNF expression (we will see in the appendix that even $C \cdot N$ is sufficient). We do not have more than $\binom{2n}{2}$ z_{ij}- variable involved in the dual expression. It is difficult to find an exact and suitable mathematical model for this algorithm (in terms of random walks or Markov chains, for example). The problem is that when Test2CNF finds inconsistency, it pushes the random walk one unit (in terms of the Hamming distance) away from the possible solution, with high probability, but if Test2CNF does not find inconsistency too often, the chain will hit the target in polynomial time with high probability. If the probability that Test2CNF finds consistency is high enough, then the algorithm hits on a solution with high probability in polynomial time (this can be easily proved). In the appendix, an approximate model is considered. In a slightly different version of this algorithm, if Test2CNF finds consistency, then we look at all unsatisfied clauses such that they can be satisfied while maintaining consistency. The modified version of the DEA algorithm can be written as follows:

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

*We write the corresponding dual 3 – CNF expression in the z_{ij} variables. There will be at most $\binom{2n}{2}$ z_{ij}- variables involved in the dual expression. We can form a binary vector with these z_{ij} - variables, call it the **dual vector**. The length of the dual vector will be the number of clauses in the 3 – CNF expression.*

Guess an initial assignment for the z_{ij}- variables , uniform at random. Basically, we randomly choose one z_{ij} from each clause and set its value to 1.

Repeat A(n) times:

> *Call the routine Test2CNF for the conjunction of all the z_{ij} that are currently assigned value 1 (this will be a 2 – CNF expression in the x_i – variables, and we will call it the active chain of z_{ij}).*
>
> *If Test2CNF finds that this conjunction of all the z_{ij}'s (that are currently assigned value 1) is satisfied, and if all clauses are satisfied, then stop and accept.*
>
> *If Test2CNF finds that this conjunction of all the z_{ij}'s (hat are currently assigned value 1) is satisfied, but not all clauses are satisfied, then find the first unsatisfied clause, and flip a random z_{ij} from that clause that currently has value 0 (change its value from 0 to 1). If the conjunction of all the z_{ij}'s (hat are currently assigned value 1, including the last one flipped) is not satisfied, then reset the last z_{ij} flipped, back to its previous value 0 and look for the next z_{ij} in that clause, or the next unsatisfied clause and flip a random z_{ij} from that clause that currently has value 0 (change its value from 0 to 1). Repeat this process until there are no more clauses to check. Basically, we are trying to avoid inconsistency, when possible, at every step.*
>
> *If Test2CNF finds that this conjunction of all the z_{ij}'s (that are currently assigned value 1) is not satisfied, and there are no more clauses to check, then choose a random z_{ij} that is currently set to 1 and flip its value to 0. In other words, if Test2CNF finds inconsistency, then discard a random z_{ij} from the chain of z_{ij}'s that are currently set to 1.*
>
> *Repeat cycle.*

Yet in another version of the DEA algorithm, each time the routine Test2CNF is called, find the solution in term of the x_i variables and update the corresponding z_{ij} variables in all the clauses. In this case, the associated Markov chain will move in jumps of more than one unit (in terms of the Hamming distance to a possible solution), and will not be analyzed here. An approximate Markov chain model for the DEA algorithm is possible, and it will be given in the appendix. I hope that the DEA algorithm (or some version of it) will find enough interest among programmers, in

order to be properly tested in a wide variety of situations. There is something interesting here that has not been tested before.

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 DEA algorithm dynamics for a very simple 3 – CNF expression, and we will present an approximate Markov chain model for the DEA algorithm.

Example. We will give an example of the dynamics of the DEA 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 change the variable. For convenience, in this appendix we use the notation z{x, y} = x ∨ z, and ¬ x is the negation of x.

The expression E will become:

$E = (z\{a,b\} \vee z\{a,c\} \vee z\{b,c\}) \wedge (z\{\neg a,\neg b\} \vee z\{\neg a,c\} \vee z\{\neg b,c\}) \wedge$
$(z\{\neg a,b\} \vee z\{\neg a,\neg c\} \vee z\{b,\neg c\}) \wedge (z\{a,\neg b\} \vee z\{a,\neg c\} \vee z\{\neg b,\neg c\}).$

Now, we will follow the DEA algorithm. We choose one z{x,y} at random from every clause and form the current active chain of z{x,y}'s. We assume that we chose the conjunction:

$E_1 = z\{a,b\} \wedge z\{\neg a,\neg b\} \wedge z\{\neg a,b\} \wedge z\{a,\neg b\}.$

Call Test2CNF, which will tell us that E_1 is not satisfiable (the equation $E_1 = 1$ has no solution). Following the algorithm, we discard a random z{x,y} from the current active chain of z{x,y}'s.

Assume that we discarded z{¬a,b}.

We are left with the conjunction:

$E_2 = z\{a,b\} \wedge z\{\neg a,\neg b\} \wedge z\{a,\neg b\}$, which is our current active chain of z{x,y}'s.

Call Test2CNF, which will tell us that E_2 is satisfiable (the equation $E_2 = 1$ has solution a = 1, b = 0, for example).

Following the DEA algorithm, we choose a random z{x,y} from the third clause, which is not yet satisfied, and add it to the current active chain of z{x,y}'s.

Assume that we chose z{¬a,¬c}, and add it to the current active chain of z{x,y}'s.

We form the new active chain of z{x,y}'s:

$E_3 = z\{\neg a,\neg c\} \wedge z\{a,b\} \wedge z\{\neg a,\neg b\} \wedge z\{a,\neg b\}$

Call Test2CNF, which will tell us that E_3 is satisfiable (for example a = 1, b = 0, c = 0 satisfies $E_3 = 1$. We see that all clauses are satisfied, so this is a solutiom to our problem (a = 1, b = 0, c = 0 is a solution to E = 1).

The Markov chain approximate model. We will call a correct clause, a clause from where a correct z_{ij} has been chosen in the current active chain of z_{ij}'s. A clause from which an incorrect z_{ij} 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 z_{ij}'s have been chosen in the current active chain of z_{ij}'s, and j is the number of incorrect

clauses from which the z_{ij}'s have been chosen in the current active chain of z_{ij}'s (we note that when we refer to the z_{ij}'s, we do not mean any connection with the pair (i,j), just a matter of notation). In this case, $i + j$ will be the length of the current active chain of z_{ij}'s, and in general we have $i + j \leq N$, where N is the number of clauses in our 3 – CNF expression.

We assume that in the current state of the Markov chain, Test2CNF has found consistency, then (following the algorithm) we choose another z_{ij} from an unsatisfied clause, and we add it to the current active chain of z_{ij}'s. 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 z_{ij} from the unsatisfied clause with probability $\frac{2}{3}$.

$p\big((i,j),(i,j+1)\big) = \frac{1}{3}$. We choose the incorrect z_{ij} from the unsatisfied 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 z_{ij} from the current active chain of z_{ij}'s. 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 z_{ij} from a correct clause with probability $\frac{i}{i+j}$.

$p\big((i,j),(i,j-1)\big) = \frac{j}{i+j}$. We discard a z_{ij} from an incorrect 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. 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, 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.

The power of the DEA algorithm is in the fact that a current active chain of z_{ij} 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