An exact and a randomized approach for the satisfiability problem

H. Drias

USTHB, Institut d'informatique, BP 32 El-Alia, 16111 Alger Algeria
E-mail: drias@wissal.dz

ABSTRACT

In this paper, three simple algorithms for the satisfiability problem are presented with their probabilistic analyses. One algorithm called *counting* is designed to enumerate all the solutions of an instance of satisfiability. The second one, namely *E-SAT* is proposed for solving the corresponding decision problem. Both the enumeration and decision algorithms have a linear space complexity and a polynomial average time performance for a specified class of instances. The third algorithm is a randomized variant of *E-SAT*. Its probabilistic analysis yields a polynomial average time performance.

1 Motivation and preliminaries

The satisfiability problem or SAT for short is of special interest because it has a wide variety of applications notably in theorem proving, automated reasoning, logic programming and database consistency. During the last decade and very recently, we have known interesting results on SAT [8,12,13,14,15,16,18]. Perhaps the most known algorithm for solving satisfiability is the Davis and Putnam procedure (DPP) [3]. Several variants of DPP have been studied and have provided interesting results.

This paper contains three algorithms based on the same method. All these algorithms are extremely simple, involving no data structure other than arrays. The first one enumerates all the solutions of an instance of SAT. It is then slightly modified to yield a second algorithm that solves the corresponding decision problem. In the last part of the paper, we describe a simple randomized algorithm for solving satisfiability instances with polynomial expected time.

JCMCC 38 (2001), pp. 209-223

The enumeration algorithm called *counting* consists in searching disjoint subsets of solutions and counting solutions of each group. We can observe many advantages for such algorithm. The method used is conceptually simple; it does not handle heavy data structure. It is easy to implement; it does not have to control phenomena such as the overlapping of clauses on solutions.

The algorithm *E-SAT* for the decision problem is obtained by stopping the algorithm *counting* after encountering the first subset of solutions when the instance is satisfiable. According to the probabilistic constant density model $M(k,n,p)$, both the enumeration and decision algorithms present $O(kr^m)$ average time complexity for the class of $SAT(k,n)$ instances verifying $r(1-p^2)^m - 1 \leq 0$ where k, n, r and p are respectively the number of clauses, the number of variables, the largest length of clauses and the probability for a literal to be present in a clause, m being a constant. On the other hand, Purdom and Brown [19] have demonstrated that under the constant density model, the average time complexity for solving SAT is polynomial when $p > \varepsilon$ or $p < \alpha(\frac{\ln n}{n})^{\frac{3}{2}}$ where ε is any small constant and α any constant. No polynomial average time algorithms were known for satisfiability within the region $\alpha(\frac{\ln n}{n})^{\frac{3}{2}} < p < \varepsilon$. Iwama [15] has reduced this interval to $\alpha(\frac{\ln n}{n})^{\frac{3}{2}} < p < (t\frac{\ln n}{n})^{\frac{1}{2}}$, where t is a constant such that n^t is the number of clauses. Another interesting result of the probabilistic analysis of E-SAT, restricts the unfavorable region for polynomiality to $\alpha(\frac{\ln n}{n})^{\frac{3}{2}} < p < (1-r^{-\frac{1}{m}})^{\frac{1}{2}}$ in case $r < (1-t\frac{\ln n}{n})^{-m}$.

The Monte Carlo type algorithm called *rand-SAT* for randomized satisfiability, follows from the randomization of *E-SAT*. *Rand-SAT* takes instances of SAT as input and yields the right answer for a subset of instances whereas for the remaining ones, it may be defective and produce a result with a probability of failure not exceeding ε. The probabilistic analysis of the algorithm shows an interesting polynomial average time complexity.

1.1 The satisfiability problem and the probabilistic model

Let $\{x_1, x_2, ..., x_n\}$ be a set of Boolean variables, then the set of literals is $\{x_1, x_2, ..., x_n, \overline{x}_1, \overline{x}_2, ..., \overline{x}_n\}$where $\overline{x}_i$ denotes the negation of x_i. A clause is a disjunction of literals. A set of k clauses over n variables represents an instance of satisfiability, the class of such instances is called $SAT(k,n)$. The length of a clause is defined to be the number of literals involved in the clause, r_i denotes the length of the i^{th} clause. When all clauses have the same length equal to r, the class is denoted $r\text{-}SAT(k,n)$. A small example of an instance of satisfiability can be $(\overline{x}_1 + \overline{x}_2 + \overline{x}_5)(x_1 + \overline{x}_2)(\overline{x}_2 + \overline{x}_4)(\overline{x}_1 + x_2 + \overline{x}_3)(\overline{x}_1 + \overline{x}_2 + \overline{x}_3 + x_4)$. A solution to SAT is an assignment of Boolean values to variables such that every clause is satisfied.

The probabilistic model considered in this paper is the known constant density model $M(k,n,p)$ due to Goldberg [12] and used in many papers. More generally a clause is constructed by including a positive literal with probability p ($p < \frac{1}{2}$), a negative literal associated with the same variable with the same probability. Let us consider two different clauses c_i and c_j. A variable in either positive or negative form appears in

both clauses with probability equal to p^2 and appears in none of the two clauses with probability equal to $(1-p)^2$. It occurs in just one clause, in either c_i and c_j, with probability equal to $2p(1-p)$. Hence the probability that a variable does not appear simultaneously in both clauses is equal to $2p(1-p)+(1-p)^2 = 1-p^2$.

2 Counting solutions of satisfiability

Let A be a matrix containing an instance of satisfiability. If we consider the example of section 1.1 then

$$A = \begin{bmatrix} \overline{x}_1 & \overline{x}_2 & \overline{x}_5 & - \\ x_1 & \overline{x}_2 & - & - \\ \overline{x}_2 & \overline{x}_4 & - & - \\ \overline{x}_1 & x_2 & \overline{x}_3 & - \\ \overline{x}_1 & \overline{x}_2 & \overline{x}_3 & x_4 \end{bmatrix}$$

Each row of A represents a clause. The symbol $-$ denotes an absence of literal. Let us modify A as follows

$$B = \begin{bmatrix} \overline{x}_1 & x_1\overline{x}_2 & x_1x_2\overline{x}_5 & - \\ x_1 & \overline{x}_1\overline{x}_2 & - & - \\ \overline{x}_2 & x_2\overline{x}_4 & - & - \\ \overline{x}_1 & x_1x_2 & x_1\overline{x}_2\overline{x}_3 & - \\ \overline{x}_1 & x_1\overline{x}_2 & x_1x_2\overline{x}_3 & x_1x_2x_3x_4 \end{bmatrix}$$

B[i,j] is the conjunction of $\overline{A}$[1,j], $\overline{A}$[2,j], ..., $\overline{A}$[i-1,j] and A[i,j]. A row of B can be viewed as an ORing of Boolean terms, hence as a disjunctive form. The fourth row of B for instance holds the Boolean expression $\overline{x}_1 + x_1x_2 + x_1\overline{x}_2\overline{x}_3$ where $+$ denotes the OR Boolean operator. The equivalence between a row of B and its corresponding row in A can be shown using the absorption law of Boolean algebra ($x + \overline{x}y = x + y$).

Property 2.1. *The items of any row of B represent disjoint sets of solutions for the corresponding clause*

Proof. This assertion is true since in any pair of terms of any row there exists a variable that appears positive in one term and negative in the other one. In the solutions of one term, this variable is assigned Boolean value 1 and in the solutions of the other one, the same variable is assigned Boolean value 0. Both sets share then no common solutions. □

Theorem 2.2. *The conjunction of expressions of rows of B preserves Property 2.1., i.e. it yields a disjunction of terms respecting property 2.1.*

Proof. First let us consider the case of a conjunction of two rows. The generalization to a number of rows greater than two is obvious and can be done by induction. The Boolean product of two rows of B consists in ANDing each term of one row with each term of the other one and then ORing the results of this operation. The maximum number of terms of the final expression is equal to the product of the numbers of terms of each row. If we consider two terms among the non null terms of the result, we notice that they contain a common part which is a term of either one of the two rows and two different parts , which are terms belonging to the other one. Property 2.1 is thus inherited from the latter row. □

From this theorem, we observe that a group of solutions for the whole instance can be built up by merging items of matrix B, one from each row. Therefore, all groups of solutions can be searched by crossing matrix B in a depth-first order for instance. The number of solutions is equal to the sum of the numbers of solutions of the disjoint sets. Algorithm *counting* is designed to yield the number of solutions N.

Algorithm COUNTING

input: an instance of satisfiability

output: N, the number of solutions of the instance

begin

obtain matrix B from the initial instance;

$N := 0$;

call count({},1);

output N;

end;

Procedure count (**var** S: **set of** literals; i: **integer**);

begin

for $j := 1$ **to** r_i **do**

begin

$S := S \cup B[i,j]$ (* append $B[i,j]$ to S *)

if S is non null **then** (* S is null if it involves two

opposite literals *)

if $i = k$ **then** $N = N + 2^{n-|S|}$ **else call** count $(S, i+1)$;

(* $|S|$ denotes the number of literals in S *)

$S := S - B[i,j]$ (* suppress $B[i,j]$ from S *)

end

end;

The first parameter S of procedure *count* contains items of matrix B, one from each row 1 up to $i-1$. Initially S is empty, during the calculation process, it becomes null when $B[i,j]$ involves a literal that has the opposite form of a literal that is already in S. The second parameter, i, indicates that the process is considering an item of the i^{th} row; the depth of the search is equal to i. When $i = k$, a group of solutions is detected. The number of solutions is then updated and the search for another set of solutions continues.

Theorem 2.3. *The space complexity of Algorithm counting is* $0(n)$.

Proof. The data structure that may be used in procedure *count* must hold at any time the current set of literals S. The latter reaches its largest size n when it involves all the literals.□

2.1 Probabilistic analysis of Algorithm COUNTING

Let S be a set of literals determined by procedure *count* and $P(S)$ the probability that S is non null. Let us denote S_i a subset of S containing i items of B, one from each row 1 up to i and item_{ij}, a term of row i. The probability that S_{i+1} is non zero is

$$P(S_{i+1}) = P(S_i) \times P_{i+1} \qquad 0 \leq i \leq k-1$$

where P_{i+1} is the probability that $\text{item}_{i+1,j}$ does not involve a literal that already exists in S_i in opposite form; $P_{i+1} = 1 - p^2$, p being the probability for a literal to be present in a clause (see section 1.1).

$$P(S_{i+1}) = \prod_{j=2}^{j=i+1} P_j = (1-p^2)^i \qquad 0 \leq i \leq k-1$$

Let $t(1..k)$ be the complexity of procedure count({},1) with an instance of clauses from range $[1..k]$ and S_{ij} the set S_i including $item_{ij}$ of the i^{th} row, then

$$t(i..k) = \begin{cases} \sum_{j=1}^{j=r_i} (P(S_{ij})\, t((i+1)..k) + c) & 1 \le i \le k-1 \\ \sum_{j=1}^{j=r_i} (P(S_{ij}) + c) & i = k \end{cases}$$

where c is a constant.

$$t(1..k) = \sum_{j=1}^{j=r_1} (P(S_{1j})\, t(2..k) + c)$$

$$= \sum_{j=1}^{j=r_1} P(S_{1j})\, t(2..k) + \sum_{j=1}^{j=r_1} c$$

$$= \sum_{j=1}^{j=r_1} P(S_{1j}) \left(\sum_{j=1}^{j=r_2} (P(S_{2j})\, t(3..k) + c)\right) + r_1 c$$

$$= \sum_{j=1}^{j=r_1} P(S_{1j}) \sum_{j=1}^{j=r_2} P(S_{2j})\, t(3..k) + \sum_{j=1}^{j=r_1} P(S_{1j}) r_2 c + r_1 c$$

...

...

$$= c \sum_{j=1}^{j=r_1} P(S_{1j}) \sum_{j=1}^{j=r_2} P(S_{2j}) \ldots \sum_{j=1}^{j=r_{k-1}} P(S_{(k-1)j}) \sum_{j=1}^{j=r_k} P(S_{kj}) +$$

$$c \sum_{j=1}^{j=r_1} P(S_{1j}) \sum_{j=1}^{j=r_2} P(S_{2j}) \ldots \sum_{j=1}^{j=r_{k-1}} P(S_{(k-1)j}) +$$

...

...

$$cr_2 \sum_{j=1}^{j=r_1} P(S_{1j}) +$$

$$cr_1$$

Let r be the largest length of the k clauses then

$$t(1..k) \le r(1-p^2)^{(k-1)} \times r(1-p^2)^{(k-2)} \times \ldots \times r(1-p^2) \times rc +$$

$$r(1-p^2)^{(k-2)} \times \ldots \times r(1-p^2) \times rc +$$

...

$$r(1-p^2) \times rc+$$

$$rc$$

$$= \sum_{j=1}^{k} c \prod_{i=1}^{i=j} (r(1-p^2)^{(i-1)}) = \sum_{j=1}^{k} (cr^j (1-p^2)^{\sum_{i=1}^{i=j}(i-1)})$$

Let $t(1..k) \leq \sum_{j=1}^{j=k} \bar{t}(j) = \sum_{j=1}^{k} (cr^j (1-p^2)^{\sum_{i=1}^{i=j}(i-1)})$

Note that as j increases, $\bar{t}(j)$ increases within a small interval then decreases.

$\frac{\Delta(\bar{t}(j))}{\bar{t}(j)} = \frac{\bar{t}(j+1)}{\bar{t}(j)} - 1 = (r(1-p^2)^j - 1)$

$\bar{t}(j)$ reaches the greatest value when $(r(1-p^2)^j - 1) = 0$. Let j_0 be the least integer verifying $r(1-p^2)^{j_0} \leq 1$ then the sum $\sum_{j=1}^{k} (cr^j (1-p^2)^{\sum_{i=1}^{i=j}(i-1)})$ is less than k times the largest term, that is $m \geq j_0 \Rightarrow r(1-p^2)^m - 1 \leq 0$

$$t(1..k) \leq kcr^m \prod_{i=1}^{i=m} (1-p^2)^{(i-1)} \leq ckr^m$$

Theorem 2.4. *The average complexity of Algorithm counting is $O(kr^m)$ for the class SAT(k,n) verifying $r(1-p^2)^m - 1 \leq 0$, where r is the largest length of clauses and m a constant.*

3 Checking satisfiability

The procedure *count* can be stopped before completion in order to answer the satisfiability decision problem. It halts after finding the first non null subset of solutions when the instance is satisfiable and terminates when the instance is unsatisfiable. It is modified as follows

procedure satisfiable(**var** S: set of literals; i: integer);

begin

$j := 1$;

while $(j \leq r_i)$ **and** (**not** satisf) **do**

begin

$S = S \cup B[i,j]$;

```
        if S is non null then
          if i = k then satisf := true else call satisfiable(S, i + 1);
        S := S − B[i, j];
        j := j + 1;
      end;
    return(satisf);
end;
```

We can state the algorithm for satisfiability as

Algorithm E-SAT

input: an instance of satisfiability

output: 'satisfiable' if the instance is satisfiable, 'unsatisfiable' otherwise

```
var satisf : Boolean;
begin
    satisf := false;
    call satisfiable({},1);
    if satisf then output('satisfiable') else output('unsatisfiable')
end;
```

3.1 Probabilistic analysis of Algorithm E-SAT

It is not difficult to see that the average complexity of Algorithm *E-SAT*, namely $tSAT(k)$ is the ratio between the average complexity of Algorithm *counting*, $t(1..k)$ and the average number of non null sets of solutions called $\overline{NS}$.

$$tSAT(k) = \begin{cases} \frac{t(1..k)}{\overline{NS}} & \overline{NS} > 0 \\ t(1..k) & \overline{NS} = 0 \end{cases}$$

3.1.1 Estimation of $\overline{NS}$

The probability that a set S of literals determined by procedure *count* is non null is $P(S_k)$. The total number of these sets is equal to $\prod_{i=1}^{i=k} r_i$. Among these sets, there are sets that are non null, i.e. they do not involve a literal in both positive and negative form. The average number of these sets is given by

$$\overline{NS} = \sum_{i=1}^{\Pi r_i} P(S_k) = (\prod_{i=1}^{i=k} r_i)(1-p^2)^{k-1} <= r^k(1-p^2)^{k-1}$$

Lemma 4.1. *The expected number of non null sets of solutions $\overline{NS}$ determined by Algorithm counting is equal to* $(\prod_{i=1}^{i=k} r_i)(1-p^2)^{k-1}$.

Now let us deduce the average complexity of Algorithm *E-SAT*.

For $\overline{NS} > 0$, let $\overline{tSAT}(j) = \frac{cr^j(1-p^2)^{\sum_{i=1}^{i=j}(i-1)}}{\overline{NS}}$

$\frac{\Delta(\overline{tSAT}(j))}{\overline{tSAT}(j)} = \frac{\overline{tSAT}(j+1)}{\overline{tSAT}(j)} - 1 = (r(1-p^2)^j - 1)$

The same reasoning as for $\overline{l}(j)$ is made to find

for $m \geq j_0 \Rightarrow r(1-p^2)^m - 1 \leq 0$ where j_0 is the least integer verifying $r(1-p^2)^{j_0} - 1 \leq 0$ then $tSAT(k) \leq kcr^m \frac{(1-p^2)^{\sum_{i=1}^{i=m}(i-1)}}{\overline{NS}} \leq kcr^m$

Theorem 4.2. *For the class SAT(k,n) verifying $r(1-p^2)^m - 1 \leq 0$, where r is the largest length of clauses and m a constant, the expected complexity of Algorithm E-SAT is $O(kr^m)$.*

3.1.2 Comparison with previous results

The condition of average polynomiality for algorithm *E-SAT* is

$r(1-p^2)^m \leq 1$ or

$\ln r + m\ln(1-p^2) \leq 0$ or

$1 - p^2 \leq r^{\frac{-1}{m}}$ or

$$\boxed{p \geq (1 - r^{-\frac{1}{m}})^{\frac{1}{2}}}$$

The results in [15] reported that the subregion for p that remains vulnerable to a polynomial average time algorithm is $\alpha(\frac{\ln n}{n})^{\frac{1}{2}} < p < (t\frac{\ln n}{n})^{\frac{1}{2}}$ where α and t are constants and $n^t = k$. In case $(1 - r^{-\frac{1}{m}})^{\frac{1}{2}} < (t\frac{\ln n}{n})^{\frac{1}{2}}$ we have

$1 - r^{-\frac{1}{m}} < t\frac{\ln n}{n}$ or

$r < (1 - t\frac{\ln n}{n})^{-m}$

the unfavorable region will be reduced to $\alpha(\frac{\ln n}{n})^{\frac{1}{2}} < p < (1 - r^{-\frac{1}{m}})^{\frac{1}{2}}$.

4 Instance preprocessing

Preprocessing of clauses speeds up enormously the procedure running time. The preprocessing we propose tends to increase the number of null subsets of solutions obtained by procedure *count* and hence reduce the number of non null subsets of solutions. A null subset is found mostly before reaching the depth k. The search is then aborted and a gain of time is recorded. The technique is based on the following observation. Let us consider two clauses c_i and c_j of length equal to three and having one common literal named x

$c_i = x \quad x_{i2} \quad x_{i3}$

$c_j = x \quad x_{j2} \quad x_{j3}$

with $x = x_{i1} = x_{j1}$

Let us suppose also that the variables associated with literals x_{i2}, x_{i3}, x_{j2} and x_{j3} are all different from each other and from the variable associated with x. When applying procedure *count*, the number of non null terms is equal to five. Let us suppose now a different placement of the literals in the clauses, for instance the following one

$c_i = \quad x_{i2} \quad x_{i3} \quad x$

$c_j = \quad x_{j2} \quad x_{j3} \quad x$

Then the number of non null terms obtained by procedure *count* is equal to nine. (A similar phenomenon can be observed when the literals x and $\overline{x}$ appear respectively in two clauses). What we can deduce is that when the literal x is placed at the head of the clauses, the number of non null terms is smaller than when x is placed in the second or third position in the clauses. Intuitively, in order to achieve a reduced number of non

null terms, one possible placement policy consists in arranging from left to right the literals in the decreasing order of their frequency. Let x and y be two literals, formally we define a partial order R as

$x\ R\ y$ if and only if the occurring of the variable associated with x in the instance is greater than the occurring of the variable associated with y. In case the occurrings are identical for both variables then

$x\ R\ y$ if and only if the gap between the occurrings of the forms of x is lower than the gap between the occurrings of the forms of y.

The placement procedure can be written as

procedure placement;

begin

compute the frequency for each variable and each literal;

determine the order R for the literals;

for each clause c **do**

sort the literals of c according to the order R;

end;

5 A Monte Carlo type algorithm

The randomized algorithm, namely *rand-SAT* derives from the algorithm *E-SAT*. It randomly chooses v vertical paths from the matrix B and tests whether they correspond to solutions. Since in *rand-SAT* a limited number of paths are tested for possible partial solutions, the answer may be mistaken. In compromise the reduction of consulted paths makes the algorithm run faster. In *E-SAT* all paths are checked until finding out a non null term or exploring all the alternatives. This yields an exact solution with an exponential time complexity for some cases. Let us now turn to the details of *rand-SAT*. If one of the vertical paths chosen at random from B corresponds to a non null term thus to a collection of solutions, the algorithm outputs 'satisfiable' and stops. On the contrary if none of the drawn paths denote a subset of solutions, the algorithm outputs 'unsatisfiable' and the probability of making an error is ε.

Algorithm rand-SAT

input: a $SAT(k,n)$ instance and ε a probability of making an error

output: the right answer 'satisfiable' or a likely mistaken answer

'unsatisfiable' with probability of error equal to ε

1- Preprocess the clauses and put them in matrix namely, A

2- convert matrix A into matrix of type B

3- set $v = \frac{\ln \varepsilon}{\ln(1-(1-p^2)^{(k-1)})}$

4-for $i = 1$ to v do

4.1 choose at random from B a vertical path $S = c_1, c_2, ..., c_j, .., c_k$

$1 \leq c_j \leq r_j \qquad 1 \leq j \leq k$, then $sol = B[1,c_1]B[2,c_2]...B[k,c_k]$

4.2 if *sol* is non null then (* if it does not involve opposite literals*)

4.2.1 answer 'satisfiable'

4.2.2 stop

5- answer 'unsatisfiable' with probability of failure equal to ε

6- stop

5.1 Probability of error and polynomial average time

The probability of error that may be produced by algorithm *rand-SAT* is specified by the following theorem.

Theorem 6.1 *Let v be the number of iterations of the loop in rand-SAT. If $v \geq \frac{\ln \varepsilon}{\ln(1-(1-p^2)^{(k-1)})}$ then the probability that algorithm rand-SAT yields an incorrect result is less than ε.*

Proof. Let $S = \{c_1, c_2, ..., c_k\}$ be a vertical path drawn from B, $1 \leq c_j \leq r_j$, $1 \leq j \leq k$. The probability that a path S is non null is $(1-p^2)^{(k-1)}$, the probability that it is null is equal to $1-(1-p^2)^{(k-1)}$ and the probability that v generated paths are null is $(1-(1-p^2)^{(k-1)})^v$. If all the v drawn paths are null whereas the instance is satisfiable, *rand-SAT* fails to answer correctly with a probability equal to $(1-(1-p^2)^{(k-1)})^v$.

Let prove now that $v \geq \frac{\ln \varepsilon}{\ln(1-(1-p^2)^{(k-1)})} \Longrightarrow (1-(1-p^2)^{(k-1)})^v \leq \varepsilon$

$v \geq \frac{\ln \varepsilon}{\ln(1-(1-p^2)^{(k-1)})} \Longrightarrow \ln \varepsilon \geq v \ln(1-(1-p^2)^{(k-1)}) \Longrightarrow$

$\varepsilon \geq (1-(1-p^2)^{(k-1)})^v$.□

Let show now that the average complexity of rand-SAT is polynomial.

Theorem 6.2 *The number v of iterations of the loop of Rand-SAT is O(1) when* $p < (\frac{1}{k})^{\frac{1}{2}}$.

Proof. $v = \frac{\ln \varepsilon}{\ln(1-(1-p^2)^{(k-1)})} = \frac{\ln \varepsilon}{\ln(1-(1-C^{k-1}_1 p^2 + C^{k-1}_2 (p^2)^2 - \ldots))}$

$= \frac{\ln \varepsilon}{\ln(C^{k-1}_1 p^2 - C^{k-1}_2 (p^2)^2 + \ldots + (-1)^{q-1} C^{k-1}_q (p^2)^q + \ldots)}$

$\frac{C^{k-1}_q (p^2)^q}{C^{k-1}_{q+1} (p^2)^{q+1}} = \frac{q+1}{(k-1-q)p^2} > \frac{1}{kp^2} > 1$ when $p < (\frac{1}{k})^{\frac{1}{2}}$

then $v \leq \frac{\ln \varepsilon}{\ln C^{k-1}_1 p^2} = \frac{\ln \varepsilon}{\ln(k-1)p^2} < \frac{\ln \varepsilon}{\ln kp^2}$

Let show now that $\frac{\ln \varepsilon}{\ln kp^2} < c$ where c is a constant

$\ln \varepsilon > c \ln kp^2$

$\ln \varepsilon > \ln(kp^2)^c$

$\varepsilon > k^c p^{2c}$

$p < \frac{\varepsilon^{\frac{1}{2c}}}{k^{\frac{1}{2}}} < (\frac{1}{k})^{\frac{1}{2}}$□

Theorem 6.3 *rand-SAT takes polynomial time on average.*

Proof. We just showed that algorithm *Rand-SAT* is $O(1)$ when $p < (\frac{1}{k})^{\frac{1}{2}}$. On the other hand, *E-SAT* takes polynomial average time when $p \geq (1 - r^{\frac{-1}{m}})^{\frac{1}{2}} \geq (\frac{1}{k})^{\frac{1}{2}}$ for $k > 1$. Therefore under this condition, *Rand-SAT* terminates in polynomial average time since only a number of vertical paths generated by *E-SAT* are visited.□

6 Conclusions

The conclusions we may draw from this study are:

1) The algorithms *counting* and *E-SAT* designed respectively to enumerate the solutions of instances of satisfiability and to solve the corresponding decision problem require polynomial time on average for $SAT(k,n)$ instances verifying $r(1-p^2)^m - 1 \leq 0$, where r is the largest length of clauses, m a constant and p the probability that a literal appears in a clause.

2) The unfavorable region for polynomial average time for testing satisfiability is reduced to $\alpha(\frac{\ln n}{n})^{\frac{3}{2}} < p < (1 - r^{-\frac{1}{m}})^{\frac{1}{2}}$ in case $r < (1 - t\frac{\ln n}{n})^{-m}$ where α and t are constants.

3) A preprocessing of initial instances is proposed in order to speed both the enumeration and decision procedures. Experiments carried out in [4] give evidence of the necessity and importance of such preliminary treatment. As a prospect, it would be interesting to estimate the average gain in time offered by this preprocessing.

4) A Monte Carlo type algorithm, namely rand-SAT, for solving the satisfiability problem is designed. Its average performance is polynomial and outperforms the results given in the literature.

7 REFERENCES

[1] M.T. Chao and J. Franco, Probabilistic analysis of a generalization of the Unit-clause literal selection heuristics for the k satisfiability problrm, Inf Sci, 51(1990) 23-42.

[2] S.A. Coo, An overview of computational complexity. Communication of the ACM, 26, 6(1983).

[3] M. Davis and H. Putnam, A computing procedure for quantification theory, J Assoc Comput Mach. 7(1960) 202-215.

[4] H. Drias, Number of Solutions for SAT instances: A Probability Distribution Study, J info Science and Engineering, 8, 4 (1992) 621-631.

[5] H. Drias, A new algorithm for counting and checking satisfiability problem and a restricted unfavorable region for polynomial average time, in proc of the 30th IEEE Symposium on System Theory (1998) 246-250.

[6] H. Drias, A Monte Carlo algorithm for the satisfiabilility problem, in Proc of IEA-AIE 98, lectures notes in Artificial Intelligence, Springer Verlag Benicassim, Spain (June 1998).

[7] O. Dubois, P. Andre, Y. Boufkhad and J. Carlier, SAT versus UNSAT, DIMACS workshop on satisfiability Testing, New Brunswick, NJ, (Oct 1993).

[8] J. Franco, On the probabilistic performance of algorithms for the satisfiability problem, Info Proc Let, 23 (1986) 103-106.

[9] J. Franco and Y.C. Ho, Probabilistic performance of heuristic for the satisfiability problem, Discrete Appl Math, 22 (1988/1989) 35-51.

[10] G. Gallo and M.G. Scutella, Polynomially solvable satisfiability problems, Info Proc let, 29 (1988) 221-227.

[11] M.R. Garey and D.S. Johnson, Computers and Intractability (Freeman & C 1979).

[12] A. Goldberg, Average case complexity of the satisfiability problem, in proc Fourth workshop on Automated deduction (1979) 1-6.

[13] A. Goldberg , P.W. Purdom and C.A. Brown, Average time analyses of simplified Davis-Putnam procedures, Info Proc let, 15 (1982) 72-75.

[14] J.N. Hooker, Resolution vs cutting plane solution of inference problems: some computational experience, Operations Research Letters, 7(1), (1988).

[15] K. Iwama, CNF satisfiability test by counting and polynomial average time, SIAM J Comp, 18, 2 (1989) 385-391.

[16] D. Mitchell, B. Selman and H.J. Levesque, Hard and easy distributions of SAT problems, in proc of the tenth conference on Artificial Intelligence (AAAI-92), San Jose, CA, (July 1992) 440-446.

[17] P. Purdom, Search rearrangement backtracking and polynomial average time, Artificial Intelligence, 21 (1983) 117-133.

[18] P. Purdom and C.A. Brown, Polynomial average time satisfiability problem, Info Sci, 41 (1987) 23-42.

[19] P. Purdom and C.A. Brown, The pure literal rule and polynomial average time, SIAM J Comp, 14 (1985) 943-953.

[20] R. Solovay and V.A. Strassen, A fast Monte Carlo test for primality, SIAM J. Comp, 6 (1977) 84-8.

[21] L.C. Wu and C.Y. Tang, Solving the satisfiability problem by using randomized approach, Info Proc Let, 41 (1992)187-190.