RAIRO. RECHERCHE OPÉRATIONNELLE

ROSSELLA PETRESCHI
BRUNO SIMEONE

Experimental comparison of 2-satisfiability algorithms

RAIRO. Recherche opérationnelle, tome 25, n° 3 (1991), p. 241-264

<http://www.numdam.org/item?id=RO_1991__25_3_241_0>

Recherche opérationnelle/Operations Research
(vol. 25, n° 3, 1991, p. 241 à 264)

EXPERIMENTAL COMPARISON OF 2-SATISFIABILITY ALGORITHMS (*)

by Rossella PETRESCHI (1) and Bruno SIMEONE (2)

Abstract. – *We report on the results of an experimental study in which we have compared the performance of four algorithms for 2-SATISFIABILITY on* 400 *randomly generated test problems with up to* 2 000 *variables and* 8 000 *clauses (half of these problems were known* a priori *to be satisfiable). The four algorithms are:* (1) *the "serial" Guess and Deduce algorithm by Deming;* (2) *the "parallel" or "dovetailing" Guess and Deduce algorithm by Even, Itai, and Shamir;* (3) *our Switching algorithm;* (4) *the Strong Components algorithm by Aspvall, Plass, and Tarjan. When the test problems are sampled from a uniform distribution, we have observed the following phenomenon: all the strong components of the implication graph are singletons, except for one (in the unsatisfiable case) or two (in the satisfiable case) "megacomponents".*

In order to eliminate these and other related anomalies, we have built another instance generator which gives rise to strong components with binomially distributed sizes. An interesting outcome of our experiments is that, under both probability models, the serial Guess and Deduce algorithm turns out to be the fastest one, in spite of its nonlinear worst-case complexity; whereas the Strong Components algorithm, whose worst-case complexity is linear, turns out to be the slowest one.

Keywords : 2-satisfiability; algorithms; graphs.

Résumé. – *Nous rapportons les résultats d'une étude expérimentale où nous avons comparé les performances de quatre algorithmes pour la 2-SATISFAISABILITE (« 2-SATISFAISABILITÉ ») appliqués à* 400 *problèmes tests engendrés aléatoirement. Ces problèmes ont jusqu'à* 2 000 *variables et* 8 000 *clauses (la moitié de ces problèmes étaient* a priori *satisfaisable). Les quatre algorithmes sont :* (1) *l'algorithme « Deviner et Déduire » (Guess and Deduce) sériel de Deming;* (2) *l'algorithme « Deviner et Déduire » parallèle de Even, Itai et Shamir;* (3) *notre propre algorithme d'aiguillage (Switching);* (4) *l'algorithme des composantes fortes de Apvall, Plass et Tarjan. Lorsque les problèmes tests sont tirés aléatoirement à partir d'une loi uniforme, nous avons observé le phénomène suivant : toutes les composantes fortes du graphe des implications sont des singletons, excepté une « mègacomposante » ou deux dans les cas insatisfaisable et satisfaisable respectivement.*

Pour éliminer ces anomalies et quelques autres, nous avons construit un autre générateur qui donne des composantes fortes avec une taille suivant la loi binomiale. Une conséquence intéressante de nos expériences est que, avec les deux modèles probabilistes, l'algorithme « Deviner et Déduire » sériel s'est montré le plus rapide, malgré sa complexité non linéaire dans le pire cas; tandis que l'algorithme des composantes fortes, dont la complexité dans le pire cas est linéaire, s'est montré le plus lent.

Mots clés : 2-satisfaisabilité: algorithmes; graphes.

(*) Received August 1988.
(1) Department of Mathematics, "La Sapienza" University, Roma, Italy.
(2) Department of Statistics "La Sapienza" University, Roma, Italy.

Recherche opérationnelle/Operations Research, 0399-0559/91/03 241 24/$ 4.40

1. INTRODUCTION

Let $\varphi = C_1 \cup C_2 \cup \ldots \cup C_m$ be a boolean expression in conjunctive normal form, where each *clause* C_j is a disjunction of literals and each *literal* ζ_i is either a variable x_i or its complement x_i' $(1 \leqq i \leqq n)$.

The *SATISFIABILITY* problem consists in determining whether such an expression φ is true for some assignment of boolean values to the variables $x_1, \ldots, x_n$.

Cook [3] has shown that SATISFIABILITY is complete in polynomial time, even if the expression is restricted to contain at most three literals per clause. However, this is a tightest possible restriction on the number of variables in a clause because SATISFIABILITY with only two literals per clause (2-*SATISFIABILITY* or 2-*SAT* for short) is solvable in polynomial time, as Cook pointed out [3].

In the present paper, we shall deal only with 2-SATISFIABILITY. Simeone [12, 13] pointed out the close connection between the satisfiability of a quadratic boolean expression (*i.e.* one having two literals per clause) and the König-Egerváry (KE) property of graphs. As a consequence, any algorithm for testing the KE property may be used to solve 2-SAT.

Two algorithms for testing the KE property have been proposed by Deming [5] and by Gavril [7]. Actually, they can be viewed as a "sequential" and a "dovetailing" version, respectively, of a basic "guess and deduce" algorithm. When specialized to solve 2-SAT, they are seen to have $O(mn)$ and $O(m)$ time-complexity, respectively, in the worst case.

It should be mentioned that the 2-SAT specialization of Gavril's algorithm turns out to be very similar to an algorithm for 2-SAT outlined by Even, Itai and Shamir in an earlier paper [6].

Aspvall, Plass, Tarjan [2] and Petreschi, Simeone [10] present two graph-theoretic algorithms, whose complexities are $O(n+m)$ and $O(nm)$, respectively.

In this paper we report on experimental results about the expected time of the algorithms in [2, 5, 7, 10]. We chose not to test other classical methods for SATISFIABILITY, such as the Davis-Putnam procedure [4] or the consensus method [11], whose complexities are indeed polynomial in the 2-SAT case, but of higher degree.

2. PRELIMINARY DEFINITIONS AND PROPERTIES

In the present section we introduce some preliminary concepts related to quadratic boolean expressions φ. For convenience, we shall use "0" and "1" instead of "FALSE" and "TRUE", respectively.

DEFINITION 1: A variable x in φ is said to be *forced* to the value α ($\alpha = 0,1$), iff either the expression is unsatisfiable or the variable x has the value α in all solutions.

Without loss of generality, we may assume from now on that

(i) no linear clauses ζ appear in φ;

(ii) whenever the clause $\zeta \cup \eta$ is present in φ, the clauses $\zeta' \cup \eta$, $\zeta \cup \eta'$, $\zeta' \cup \eta'$, as well as other occurrences of $\zeta \cup \eta$, must be absent.

DEFINITION 2: *A boolean expression* φ satisfying (i) and (ii) will be called *irreducible*.

DEFINITION 3: A boolean expression φ is called *pure* if every clause has at least one uncomplemented variable. A pure expression is always satisfiable, since (1...1) is obviously a solution to the equation $\varphi = 1$.

DEFINITION 4: A *switch* on the variable x in φ consists in replacing x by its complement x' and vice versa, wherever they appear.

THEOREM 5: [10] *A boolean expression is satisfiable if it can be transformed into a pure one by a switch on some set S of variables.*

Next we introduce two graphs associated with φ.

DEFINITION 6: [10] The *clause-graph* associated with φ is the undirected graph $G = (V, E)$ where:

$$V = \{ x_1, \ldots, x_n; x'_1, \ldots, x'_n \} \qquad \textit{and}$$

$$E = \{ \langle \zeta, \eta \rangle, \text{for each clause } \zeta \cup \eta \text{ in } \varphi \} \cup \{ \langle x_i, x'_i \rangle \mid i = 1, \ldots, n \} \text{ (*)}.$$

The edges of G are classified into *positive*, *negative*, *mixed* and *null* edges according to whether they have the form $\langle x_i, x_j \rangle$, $\langle x'_i, x'_j \rangle$, $\langle x_i, x'_j \rangle$ or $\langle x_i, x'_i \rangle$, respectively.

(*) Unordered pairs are denoted by $\langle x, y \rangle$, while ordered pairs are denoted by (x, y).

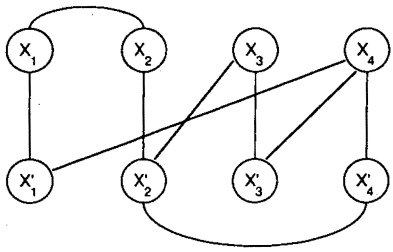

Figure 1. – The clause-graph G associated with Ψ.

Figure 1 shows the clause-graph G associated with the following expression:

$$\Psi = (x_1 \cup x_2)(x'_1 \cup x_4)(x'_2 \cup x_3)(x'_2 \cup x'_4)(x'_3 \cup x_4).$$

DEFINITION 7: [2] The *implication-graph* associated with φ is the digraph $D = (V, A)$ where:

V is defined as above and

$$A = \{ \langle \zeta', \eta \rangle, \langle \eta', \zeta \rangle \text{ for each clause } \zeta \cup \eta \text{ in } \varphi \}$$

$\langle \eta', \zeta \rangle$ is called the *mirror edge* of $\langle \zeta', \eta \rangle$.

The digraph D is isomorphic to the digraph D^- obtained from D by reversing the orientations of all the edges and complementing the names of all the vertices.

Figure 2 shows the digraph D associated with Ψ.

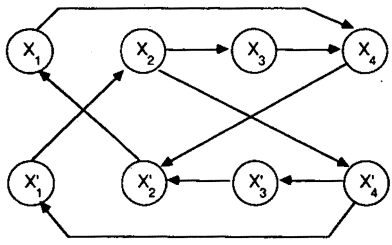

Figure 2. – The implication graph D associated with Ψ.

DEFINITION 8: [10] An *alternating tree rooted at* x'_i is any subgraph of G which is a tree $T(x'_i)$ rooted at x'_i and has the following properties:

(1) x'_i is the root;

(2) if x_j is a vertex of $T(x'_i)$, then its father in $T(x'_i)$ is x'_j;

(3) if x'_j is a vertex of $T(x'_i)$, then its father is a vertex x_r of $T(x'_i)$, such that $\langle x_r, x'_j \rangle$ is a mixed edge of G.

(4) if x_j is a vertex of $T(x'_i)$, and $\langle x_j, x'_r \rangle$ is a mixed edge of G, then x'_r is a vertex of $T(x'_i)$.

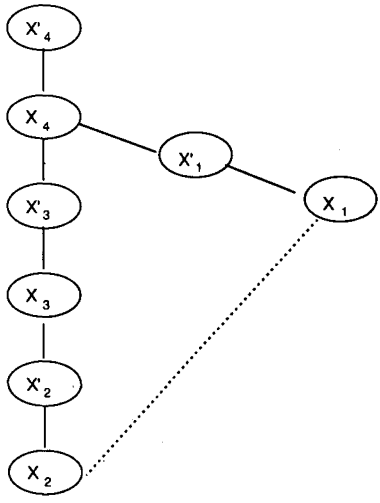

Figure 3. — An alternating tree of G.

For the clause-graph G of figure 1, figure 3 shows an alternating tree rooted at x'_4.

DEFINITION 9: The *join* of two vertices ζ and η of $T(x'_i)$ is their common ancestor which is farthest from the root x'_i.

DEFINITION 10: A *strongly connected component* (or *strong component*, for short) of a digraph is a maximal set C of vertices such that for any two vertices $\zeta, \eta \in C$, there exists a circuit through ζ and η.

A graph is said to be *strongly connected* if it has only one strongly connected component.

The strongly connected components of the digraph D of figure 2 are $\{1, 4, 2'\}$, $\{2, 1', 4'\}$, $\{3\}$, $\{3'\}$.

3. ALGORITHMS

In order to make this paper self-contained, we shall briefly describe the four algorithms in [5, 7, 10, 2], henceforth referred to as *L* (Labelling), *AL* (Alternate Labelling), *S* (Switching), *SC* (Srong Components), respectively.

Algorithm *L*

The idea of the algorithm is to *guess* the value of an arbitrary literal ζ in some solution and to *deduce* the possible consequences of this guess on other variables appearing in the expression. Initially, all clauses are declared unscanned. An arbitrary literal ζ is selected and ζ and ζ' receive the labels 0 and 1, respectively. The labelling is extended to as many literals as possible by repeated applications of the following elementary step.

Step: Take an arbitrary unscanned clause $\zeta \cup \eta$ such that ζ has the label 0, and assign to η and η' the labels 0 and 1 respectively, making sure that η has not previously received the label 1. Declare the clause $\zeta \cup \eta$ scanned.

Three exclusive cases may occur:

1. During the execution of some step, a conflict occurs because one attempts to assign the label 0 to a literal η which has already the label 1. That means that the initial guess on the value of the starting literal ζ was wrong. Then all labels are erased, ζ and ζ' are assigned this time the labels 1 and 0, and the labelling procedure starts again. If a new conflict occurs, the algorithm stops and the expression is unsatisfiable.

2. No conflict occurs and all literals finally have a label. Then by assigning to each literal the value corresponding to its label, one gets a truth assignment.

3. No conflict occurs but the labelling procedure "gets stuck" and some literals remain unlabelled. In this case we say that a *blocking* occurs. This may happen only when, for every unscanned term, literals appearing in the term are either unlabelled or have the label 0. But this allows us to ignore, from now on, the labelled literals and to work only on the reduced expression involving only the unlabelled literals.

Clearly, the reduced expression is consistent if and only if the previous one is such. Then a guess is made on an arbitrary unlabelled literal ω, and the labelling procedure starts again on the reduced expression.

The algorithm ends when all literals have been labelled.

Algorithm *AL*

The basic idea of this algorithm is the same as in algorithm *L*, but the consequences of two alternative guesses on a starting literal ζ are derived in "parallel", rather than in series. One keeps track of these consequences by a "red" labelling (corresponding to the guess $\zeta = 1$) and by a "green" labelling (corresponding to the guess $\zeta = 0$).

Initially all the clauses of the expression are declared "red-unscanned" and "green-unscanned"; then an arbitrary literal ζ is chosen for receiving the red label 1 and the green label 0. Of course ζ' must then receive the red label 0 and the green label 1.

The two labellings are extended as long as possible by executing the following step alternately for the red labelling and for the green one:

Step: An arbitrary unscanned clause $\zeta \cup \eta$ such that label $(\zeta) = 0$ is considered and the labels 1 and 0 are assigned to η and η', respectively, provided that a conflict with previous labellings (if any) of η and η' does not arise. Now clause $\zeta \cup \eta$ is declared (red- or green-) scanned.

If the execution of the above step results, say, in a red conflict, then the red labelling is interrupted.

The expression is not satisfiable when both labellings end up in a conflict.

If at least one labelling (the red labelling, say) does not generate any conflict and all literals are red-labelled, then the expression is satisfiable and the red labelling yields and explicit solution.

It may happen that a labelling, the red one, say, "gets stuck": no conflict has occurred, but there are still literals having no red label. This is possible only when, for each red-unscanned clause, the literals that appear in the clause are either red-unlabelled or have red label 0. In this case the red labels are taken for granted and both the red and the green labellings are restarted on the reduced expression involving only the red-unlabelled literals.

As an example, the table *AL* summarizes the steps of the algorithm with reference to the expression Ψ.

Algorithm *S*

The idea of the algorithm consists in trying to transform the expression into a pure one, if possible, by a sequence of switches.

The algorithm works on the associated graph *G*. An endpoint x_i' of a negative edge $x_i' x_r'$ is selected and the alternating tree $T(x_i')$ is grown. As soon as a new vertex x_h of $T(x_i')$ is generated, one checks whether there is in

TABLE *AL*

Step	Red-scanned	Green-scanned clause	Red labels clause	Green labels
0.	None	None	$x_2=1;\ x'_2=0$ (guess)	$x_2=0;\ x'_2=1$ (guess)
1.	$x'_2 \cup x_3$	–	$x_3=1;\ x'_3=0$	–
2.	–	$x_1 \cup x_2$		$x_1=1;\ x'_1=0$
3.	$x'_3 \cup x_4$	–	$x_4=1;\ x'_4=0$	–
4.	–	$x'_1 \cup x_4$	–	$x_4=1;\ x'_4=0$
5.	$x'_2 \cup x'_4$	–	$x_4=0;\ x'_4=1$ (conflict)	–
6.	–	Stuck	$x_3=0;\ x'_3=1$ (guess)	$x_3=1;\ x'_3=0$ (guess)
Labelling completed				

G any positive edge $\langle x_h, x_k \rangle$ linking x_h to a previously generated vertex x_k of $T(x'_i)$. If this is the case, the variable x_j, corresponding to the join of x_h and x_k, must be forced to 1 [10].

After forcing the join, one forces as many variables as possible, by employing the two following rules:

(1) If ζ is forced to 1, then ζ' is forced to 0.

(2) If ζ_i is forced to 0 and $\langle \zeta_i, \zeta_j \rangle$ is an edge in G, then ζ_j is forced to 1.

If during this process a conflict is generated the algorithm stops pointing out that the expression is not satisfiable. Otherwise one obtains a reduced expression and a new cycle begins. If the construction of $T(x'_i)$ has been completed without detecting any positive edge between vertices of $T(x'_i)$, a switch is performed on all the variables in $T(x'_i)$. In this way one obtains an equivalent expression and a new cycle begins. The procedure is iterated until either a pure expression is obtained or all variables are forced. In both cases a solution of the original expression can be found by inspecting the list of the forced variables and the list of the switched ones.

As example, if wz analyze the alternating tree of figure 3, rooted at x'_4, we see that x_1 and x_2 are linked by a positive edge. Hence the join x_4 must be forced to 1 and consequently x'_4 is forced to 0. According to rule 2, x'_2 must be forced to 1 and x'_1 to 0.

An interesting feature of Algorithm S is that it does not need to scan all the edges in order to yield a certificate of satisfiability, provided that the input includes also the list of all negative edges (*see* Table VI).

Algorithm *SC*

This algorithm rests on the following key result:

THEOREM 11: [2] *The expression* φ *is satisfiable iff in* D *no vertex* x_i *is in the same strong component as its complement* x_i'.

The algorithm works on D and generates the strong components SC of D in reverse topological order [14]. The isomorphism between D and D^- implies that every strong component SC of D has a mirror component SC' consisting of the subgraph induced by the complements of the vertices in SC. Hence Theorem 11 can be restated as follows: "An expression φ is satisfiable iff in D no strong component coincides with its mirror component". The general step of the algorithm consists in processing the strong components of D as follows:

For each strong component SC, one of the following cases occur:

(*a*) SC is already labelled. The algorithm analyzes another component.

(*b*) $SC = SC'$. The algorithm stops. By Theorem 11, the expression is not satisfiable.

(*c*) SC is not labelled yet. The algorithm assigns the label 1 to SC and the label 0 to SC'.

Let us call SC_1 a *predecessor* of SC_2 (SC_2 a *successor* of SC_1) when an edge leads from a vertex in SC_1 to a vertex in SC_2. It is easy to prove that every component labelled 1 has only 1-labelled components as successors and every component labelled 0 has only 0-labelled components as predecessors. Thus if we assign to each vertex the value of the component containing it we get a solution to φ.

As an example, we consider again Ψ and the associated digraph of figure 2. Tarjan's algorithm for finding the strong components yields the spanning arborescence of figure 4; then the strongly connected components of D are $\{x_2 x_1' x_4'\}$, $\{x_3\}$ and their mirror components.

It should be noticed that all the four algorithms, whenever φ is satisfiable, output a solution of the equation $\varphi = 1$.

4. EXPERIMENTAL RESULTS I: UNIFORM DISTRIBUTION

4.1. Experimental design

In order to compare the performance of the four algorithms outlined in Sec. 3, we have tested them on 400 randomly generated problems with up to 2000 variables and 8000 clauses. The expressions given as input to the

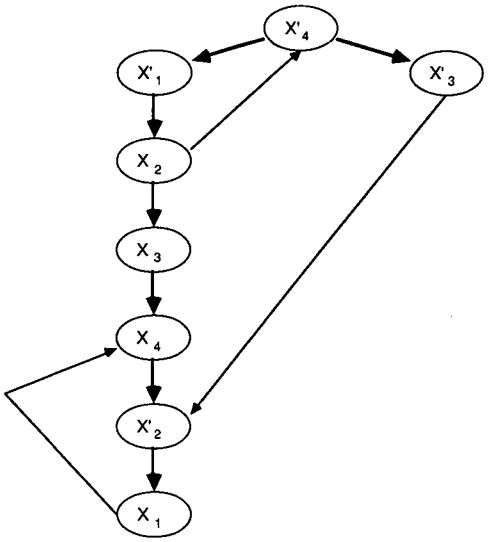

Figure 4. – Spanning arborescence associated with the graph of figure 2.

algorithms presented in Sect. 3, are generated by a routine that outputs a pair of m-arrays A and B representing the m clauses of a quadratic boolean expression φ in n variables.

In our first set of experiments, the expressions have been generated according to a uniform probability distribution model and, whenever we sample from a finite set, we always assume a uniform probability distribution over such set.

Since we restrict ourselves to irreducible expressions, the number of clauses in φ is at most $\binom{n}{2}$. Actually, given two integers n and m $\left(1 \leqq m \leqq \binom{n}{2}\right)$, m pairs of indexes $(i_1,j_1),\ldots,(i_m,j_m)$, where $i_h<j_h$ for all h, are randomly drawn without repetitions out of all possible $\binom{n}{2}$ such pairs. In all our experiments, we have chosen $m=4n$. For each pair (i_h,j_h) an integer r, $1 \leqq r \leqq 4$, is randomly chosen to tell whether the corresponding clause has the form $x_{ih} \cup x_{jh}$, $x'_{ih} \cup x_{jh}$, $x_{ih} \cup x'_{jh}$, $x'_{ih} \cup x'_{jh}$.

The striking phenomenon that we have empirically observed is that the expressions which were randomly generated by the above procedure were

always unsatisfiable when $m=4n$ and were unsatisfiable with a single exception when $m \geqq 1.5n$. This is in accordance with the probabilisitic results in [8].

Therefore, in order to generate unsatisfiable expressions, we have just used the above model. On the other hand, we wanted to explore the performance of the four algorithms also on satisfiable expressions. In order to obtain expressions known *a priori* to be satisfiable, we have modified the above generator so as to get only pure expressions (to this aim, it is enough to let the random integer r range from 1 to 3). Next, an integer k is randomly selected in the interval $[0, n]$ and k randomly chosen variables are switched.

In such a way an irreducible quadratic boolean expression is generated and one solution of the corresponding expression is obtained by setting all the switched variables equal to 0 and all the other variables equal to 1.

In our experiments, both in the unsatisfiable and in the satisfiable case, test problems with $n=100, 200, \ldots, 2000$ variables and $m=4n$ clauses were generated. For each problem size, a sample of 5 test problems has been generated and the average CPU time has been estimated from such a sample.

4.2. Performance indicators

All algorithms

TIME	CPU time in 10^{-6} sec IBM 3090. The CPU time refers only to the actual solution of the expression and does not include generation time.

Labelling and Alternate Labelling

LABEL	No. of labelled variables
BLOCK	No. of blockings
CONFL	No. of conflicts
FORCE	No. of detected forced variables

Switching:

TREE	No. of alternating trees generated
EDGE	No. of edges scanned
SWITCH	No. of switched trees
FORCE	No. of detected forced variables

Strong components:

STRONG	No. of strong components generated

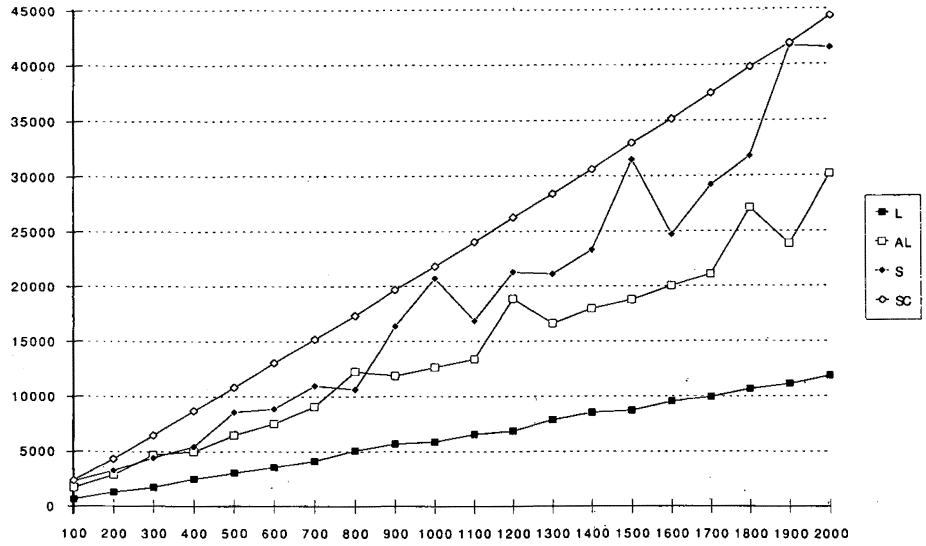

Figure 5.

Figure 6.

The results of our experiments are shown in figures 5, 6, 7, 8 and in Tables I, II, III, IV, V, VI, VII.

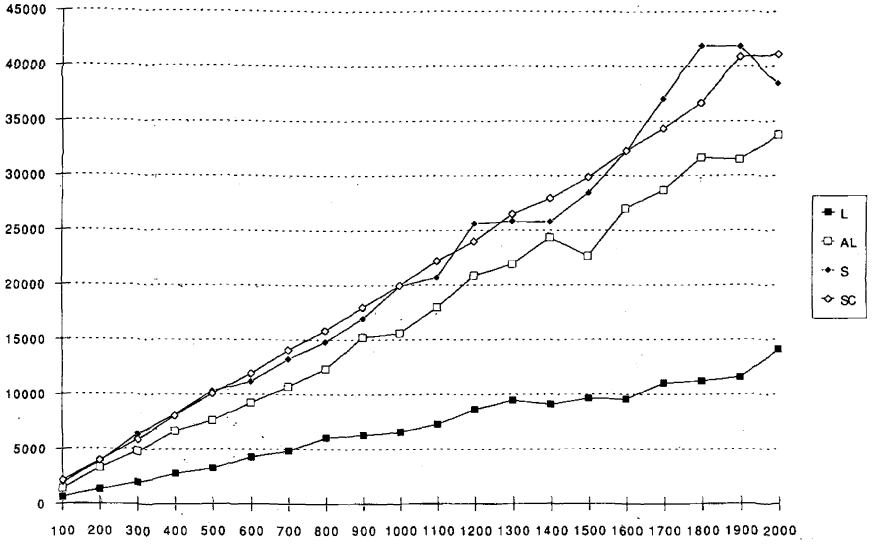

Figure 7.

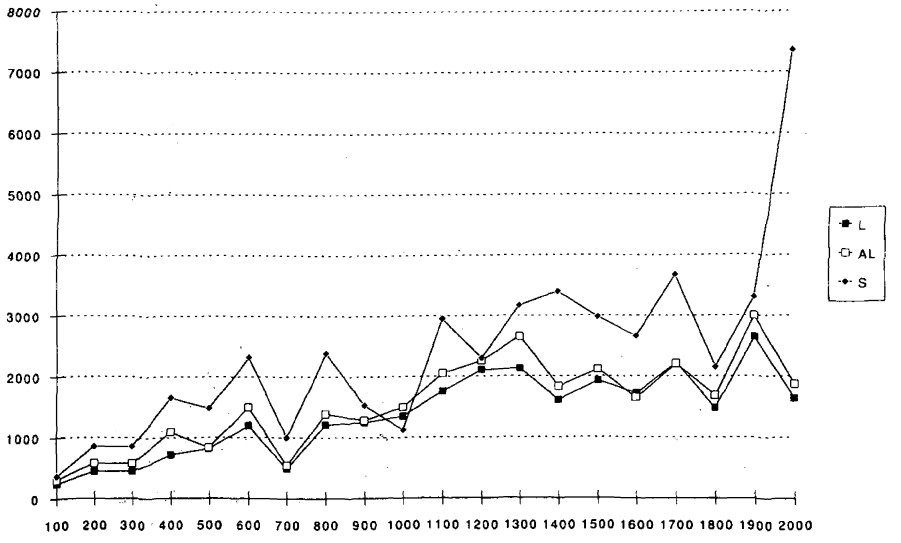

Figure 8.

TABLE I

Satisfiable

	UNIFORM DISTRIBUTION				BINOMIAL DISTRIBUTION			
Var.	$TIME_L$	$TIME_{AL}$	$TIME_S$	$TIME_{SC}$	$TIME_L$	$TIME_{AL}$	$TIME_S$	$TIME_{SC}$
100	643.6	1698.2	2283.8	2365.8	647.2	1424.0	1939.0	2159.6
200	1298.0	2863.0	3255.2	4321.6	1316.2	3262.4	3851.2	3921.8
300	1731.0	4701.0	4425.6	6473.4	1974.6	4799.8	6372.0	5874.2
400	2490.4	4998.2	5431.4	8688.8	2763.4	6623.8	8146.6	8045.6
500	3018.4	6460.6	8574.2	10822.8	3276.8	7614.0	10300.4	10064.0
600	3571.2	7535.0	8871.6	13074.6	4280.4	9228.6	11152.2	11915.0
700	4135.0	9105.0	10963.2	15205.2	4855.6	10716.0	13255.8	14074.0
800	5097.0	12275.8	10665.8	17368.8	6051.8	12368.0	14803.6	15839.6
900	5718.4	11879.4	16366.6	19674.4	6298.4	15238.0	16971.0	17986.2
1000	5882.6	12646.6	20731.0	21805.8	6561.2	15605.4	19931.0	19968.6
1100	6567.8	13383.8	16853.6	24021.2	7262.2	17954.8	20687.0	22212.6
1200	6860.0	18865.0	21280.4	26280.8	8613.8	20867.2	25653.0	24003.6
1300	7888.6	16625.6	21108.2	28390.6	9454.2	21910.4	25842.2	26488.6
1400	8550.8	17967.4	23324.8	30625.6	9084.0	24387.6	25797.8	27952.4
1500	8737.4	18798.0	31500.2	33002.8	9612.4	22650.8	28440.2	29886.6
1600	9559.2	20061.6	24754.8	35157.2	9588.0	27017.4	32204.2	32294.4
1700	9941.8	21097.2	29217.6	37480.8	11032.2	28722.2	37017.8	34327.4
1800	10658.4	27126.2	31786.0	39794.6	11209.2	31690.6	41873.2	36648.4
1900	11094.8	23845.2	41690.6	41907.4	11564.6	31499.4	41784.2	38454.8
2000	11856.0	30153.8	41516.4	44371.0	14123.2	33743.4	40849.6	41090.4

TABLE II

Unsatisfiable

	UNIFORM DISTRIBUTION				BINOMIAL DISTRIBUTION			
Var.	$TIME_L$	$TIME_{AL}$	$TIME_S$	$TIME_{SC}$	$TIME_L$	$TIME_{AL}$	$TIME_S$	$TIME_{SC}$
100	240.0	285.4	388.2	2963.8	240.8	307.8	368.0	2972.0
200	365.6	412.8	335.4	5494.6	473.8	557.4	443.0	5519.4
300	264.2	291.2	473.2	8223.6	446.2	580.6	855.8	8083.2
400	348.4	397.8	737.2	10940.2	716.2	1093.4	1656.6	9546.6
500	451.8	518.6	610.2	13666.4	818.4	835.0	1484.2	13137.4
600	419.4	462.8	591.2	16363.8	1193.0	1495.2	2319.0	14722.2
700	703.6	784.6	722.6	19127.4	489.6	540.8	997.2	18573.8
800	515.4	520.0	915.6	21736.8	1210.2	1387.2	3292.4	20649.0
900	674.6	702.0	832.8	24533.2	1251.0	1281.4	1530.8	23407.6
1000	746.6	813.2	1010.4	27409.0	1351.2	1499.0	1121.2	25879.2
1100	849.4	888.2	853.2	30014.8	1759.2	2053.8	2953.0	28334.6
1200	811.2	859.0	761.8	33221.0	2109.6	2256.4	2301.6	31192.4
1300	859.0	744.2	861.4	35733.8	2130.8	2655.4	3171.6	32524.6
1400	627.4	713.8	1058.2	38811.4	1610.8	1832.6	3400.0	36842.8
1500	969.2	986.4	978.2	41669.2	1929.4	2113.4	2980.2	41442.4
1600	812.2	875.5	1042.8	44789.8	1725.2	1654.4	2664.2	41927.2
1700	857.4	986.0	1517.6	47690.6	2228.4	2207.6	3678.6	45115.2
1800	911.8	782.8	1493.2	50326.2	1476.2	1681.4	2153.6	47901.4
1900	976.2	1084.	1294.8	53532.6	2646.4	2997.4	3301.4	50900.4
2000	716.8	727.4	1166.4	56622.8	1637.2	1870.0	7364.4	52532.0

TABLE III

	UNIFORM DISTRIBUTION				BINOMIAL DISTRIBUTION			
	Satisfiable		*Unsatisfiable*		*Satisfiable*		*Unsatisfiable*	
Var.	$LABEL_L$	$LABEL_{AL}$	$LABEL_L$	$LABEL_{AL}$	$LABEL_L$	$LABEL_{AL}$	$LABEL_L$	$LABEL_{AL}$
100	105.5	149.8	49.0	49.0	108.8	129.2	51.6	51.6
200	225.8	269.6	78.2	78.2	233.6	297.4	99.4	99.4
300	303.4	426.2	55.8	55.8	352.2	447.2	95.0	95.0
400	437.4	473.0	73.0	73.0	488.0	608.2	144.8	158.4
500	529.2	605.4	99.8	99.8	577.8	714.8	150.6	150.6
600	625.4	709.8	86.8	86.8	752.0	849.6	218.6	228.8
700	722.0	857.2	148.6	148.6	850.4	994.8	96.8	96.8
800	891.2	1103.4	102.2	102.2	1044.2	1133.2	244.6	244.6
900	997.6	1116.8	133.6	133.6	1093.2	1367.6	225.2	225.2
1000	1029.8	1191.0	150.0	150.0	1162.0	1443.2	245.4	258.2
1100	1151.6	1245.6	168.8	168.8	1264.6	1261.0	338.0	344.4
1200	1203.2	1693.6	166.0	166.0	1502.4	1878.6	389.6	391.4
1300	1376.6	1557.8	167.2	139.4	1651.4	1977.0	402.2	421.4
1400	1490.8	1679.8	134.6	134.6	1606.8	2163.2	321.6	321.6
1500	1520.2	1751.6	193.8	193.8	1693.0	2087.0	363.6	363.6
1600	1676.4	1871.4	165.4	165.4	1697.4	2415.0	327.8	287.8
1700	1736.4	1967.4	190.0	190.0	1938.8	2564.2	385.0	385.0
1800	1858.6	2413.8	181.6	146.4	1973.8	2771.6	292.8	292.8
1900	1946.6	2231.4	199.0	199.0	2048.6	2811.0	503.0	503.0
2000	2079.0	2692.2	137.4	137.4	2443.2	3024.6	299.2	295.0

TABLE IV

	UNIFORM DISTRIBUTION								BINOMIAL DISTRIBUTION							
	Satisfiable				*Unsatisfiable*				*Satisfiable*				*Unsatisfiable*			
	BLOCK		CONFL		BLOCK		CONFL		BLOCK		CONFL		BLOCK		CONFL	
Var.	L	AL	L	AL	L	AL	L	AL	L	AL	L	AL	L	AL	L	AL
100	5.8	6.6	0.2	0.8	0.0	0.0	2.0	2.0	1.2	1.2	1.4	2.2	0.0	0.0	2.0	2.0
200	16.2	19.2	0.6	1.0	0.0	0.0	2.0	2.0	3.0	3.2	1.4	2.4	0.0	0.0	2.0	2.0
300	19.0	21.6	0.2	1.0	0.0	0.0	2.0	2.0	3.4	3.8	2.2	3.0	0.2	0.2	2.2	2.2
400	25.2	29.8	1.0	1.0	0.0	0.0	2.0	2.0	5.2	6.0	1.8	3.2	0.4	0.4	2.0	2.4
500	36.4	43.8	0.6	1.0	0.0	0.0	2.0	2.0	4.8	5.2	2.2	3.6	0.0	0.0	2.0	2.0
600	38.2	43.4	0.6	1.0	0.0	0.0	2.0	2.0	4.2	4.6	1.6	3.2	0.2	0.2	2.0	2.2
700	51.0	60.4	0.2	1.0	0.0	0.0	2.0	2.0	5.0	6.0	2.4	3.4	0.0	0.0	2.0	2.0
800	62.6	75.2	1.0	0.8	0.0	0.0	2.0	2.0	4.8	5.0	2.2	3.2	0.0	0.0	2.0	2.0
900	65.6	75.6	0.6	1.0	0.0	0.0	2.0	2.0	4.2	7.0	1.2	3.0	0.0	0.0	2.0	2.0
1000	74.8	86.4	0.4	1.0	0.0	0.0	2.0	2.0	4.0	6.0	1.4	3.4	0.2	0.2	2.0	2.0
1100	78.4	90.8	0.8	1.0	0.0	0.0	2.0	2.0	7.0	12.0	2.0	2.4	0.2	0.2	2.0	2.2
1200	90.4	102.8	0.2	1.0	0.0	0.0	2.0	2.0	8.6	10.8	3.6	4.6	0.2	0.2	2.0	2.0
1300	74.8	112.8	0.6	1.0	0.2	0.2	2.0	2.0	8.4	10.8	2.8	3.2	0.4	0.4	2.2	2.2
1400	74.8	118.8	0.6	1.0	0.0	0.0	2.0	2.0	7.6	8.2	1.8	4.0	0.0	0.0	2.0	2.0
1500	74.8	129.8	0.2	1.0	0.0	0.0	2.0	2.0	6.0	6.4	2.4	3.8	0.0	0.0	2.0	2.0
1600	74.8	138.2	0.8	1.0	0.0	0.0	2.0	2.0	7.0	8.6	1.4	3.0	0.2	0.2	2.2	2.0
1700	74.8	145.6	0.4	1.0	0.0	0.0	2.0	2.0	9.4	12.0	3.2	4.0	0.0	0.0	2.0	2.0
1800	74.8	142.6	0.4	1.0	0.2	0.2	2.0	2.0	11.8	16.2	2.8	4.0	0.0	0.0	2.0	2.0
1900	74.8	147.8	0.6	1.0	0.0	0.0	2.0	2.0	8.8	10.0	2.0	3.2	0.0	0.0	2.0	2.0
2000	74.8	161.2	0.4	1.0	0.0	0.0	2.0	2.0	11.2	14.2	2.4	4.4	0.2	0.2	2.2	2.0

TABLE V

	UNIFORM DISTRIBUTION						BINOMIAL DISTRIBUTION					
	Satisfiable			*Unsatisfiable*			*Satisfiable*			*Unsatisfiable*		
	FORCE			FORCE			FORCE			FORCE		
Var.	L	AL	S	L	AL	S	L	AL	S	L	AL	S
100	18.4	75.4	92.8	24.4	29.2	20.4	51.0	100.0	100.0	32.6	35.6	25.2
200	72.4	179.8	180.4	42.0	50.0	22.6	98.2	142.8	183.2	57.4	60.0	36.8
300	57.0	222.6	222.4	32.2	39.6	37.4	164.8	277.8	289.8	67.6	67.6	66.6
400	369.0	369.0	369.0	38.2	44.6	60.4	173.8	333.8	375.2	53.0	107.4	112.0
500	180.6	454.6	454.8	62.2	63.4	46.2	289.4	451.2	478.8	58.4	108.4	87.6
600	336.4	554.4	554.4	47.0	53.2	48.4	404.0	546.6	579.2	46.2	172.4	152.6
700	128.2	637.0	637.0	79.8	86.4	46.6	409.2	620.4	681.0	54.6	68.2	47.2
800	577.4	577.0	721.2	47.2	73.2	74.8	535.8	730.0	775.0	142.0	160.8	179.6
900	492.2	820.4	820.4	54.8	83.2	58.2	397.4	738.6	869.0	78.4	155.6	88.6
1000	183.6	911.4	911.8	65.2	84.8	80.2	322.4	849.0	964.0	66.8	174.2	93.0
1100	601.6	1004.2	1005.0	68.6	108.4	52.2	472.2	860.6	1027.4	141.8	260.0	169.0
1200	0.2	870.2	875.2	80.2	113.6	62.2	591.6	1018.8	1154.0	145.0	251.6	167.6
1300	706.6	1178.8	1178.8	62.8	85.4	46.8	755.4	1032.4	1237.0	156.4	251.2	200.0
1400	769.0	1275.2	1275.4	84.0	84.0	69.4	454.8	1084.8	1350.6	197.6	230.8	177 6
1500	273.2	1362.4	1362.4	86.8	139.8	75.4	518.4	1328.0	1456.8	148.2	213.8	156.8
1600	1159.	1454.6	1454.6	85.2	107.2	68.6	451.0	1126.2	1528.2	167.8	213.2	163.4
1700	622.2	1547.8	1547.8	126.	126.8	109.6	692.2	1211.8	1615.8	121.6	291.2	264.6
1800	658.0	1313.8	1649.4	82.8	89.2	112.6	687.6	1162.0	1711.2	165.4	182.0	132.4
1900	695.8	1744.6	1744.8	98.8	108.4	102.2	469.6	1669.8	1811.8	244.2	297.2	114.8
2000	733.8	1463.4	1829.4	57.4	88.8	82.2	756.4	1498.8	1912.4	136.2	182.2	298.8

TABLE VI

	UNIFORM DISTRIBUTION						BINOMIAL DISTRIBUTION					
	Satisfiable			*Unsatisfiable*			*Satisfiable*			*Unsatisfiable*		
Var.	TREE	EDGE	SWITCH	TREE	EDGE	SWITCH	TREE	EDGE	SWITCH	TREE	EDGE	SWITCH
100	3	118.2	2	1.4	22	0.4	3.6	55	1.6	1.4	2.2	0.4
200	2.2	133	1.2	1.4	28.4	0.2	12	276	8.4	1.4	28.4	0.4
300	3.4	220	2.4	0.6	61.4	0	16	472.4	12	1.6	61.4	0.4
400	4.6	106.6	3.6	2.6	108.8	0.6	23.4	581.4	19.4	2.6	108.8	1.6
500	5.6	402.8	4.6	2.2	141.6	1.2	20.8	757	16.6	2.1	141.6	1.0
600	4	243.4	3	3.8	158.6	2.8	21.8	648.4	18.2	3.8	158.6	2.8
700	10.2	379.6	9.2	2.6	109.2	1.6	25.2	813	20	2.6	109	1.6
800	6.6	174	5.6	2	138.8	0.8	32.6	897.8	26.6	1.8	138.8	1
900	9.2	873	8.2	1.4	159	0.4	26.6	1032.6	21.8	1.4	159	0.4
1000	9.4	1318.8	8.4	2.6	63.8	1.2	32.4	1328.4	26.8	2.6	63.8	1.6
1100	11	582.8	10	3.8	241.4	2.8	36.4	1322.8	31.6	3.8	241.4	2.6
1200	13	1396.6	12.2	3.4	161.4	2.4	44.2	1939	36.2	3.4	161.4	2.4
1300	9.6	865.4	8.6	4.4	227.2	3.2	47	1818.4	39.4	4.4	227.2	3.2
1400	10.4	992.2	9.4	6.8	285.2	5.6	38	1501	32.8	6.8	285.2	5.6
1500	5.8	2018.8	14.8	7.6	256.6	6.6	40.6	1726.2	34.2	7.6	256.6	7
1600	8.6	940.4	7.6	3.6	231.8	2.4	51	2209.2	44.6	3.6	231.8	2.4
1700	15.6	1356.2	14.6	3	192.6	2	59.8	2886.2	49.6	3.4	238	2.4
1800	12.8	1548.6	11.8	5.2	377.4	4.2	67	3437.8	59	3	192.6	2
1900	12.6	2817.2	7.2	4.2	419	2.8	60.2	3240.2	53	5.2	377.4	4.2
2000	15.4	2631.6	14.4	4.2	419	2.8	63.8	2845.2	56	4.2	419	2.8

TABLE VII

	UNIFORM DISTRIBUTION		BINOMIAL DISTRIBUTION	
Var.	*Satisfiable* STRONG	*Unsatisfiable* STRONG	*Satisfiable* STRONG	*Unsatisfiable* STRONG
100	31.6	3.8	8.0	1.2
200	72.4	8.2	40.8	1.8
300	96.4	11.4	63.6	2.4
400	130.8	14.8	92.8	14.4
500	175.6	17.4	91.6	7.0
600	191.2	25.8	92.0	15.2
700	253.6	22.4	147.6	8.4
800	289.6	37.0	125.2	11.6
900	294.4	38.6	140.0	13.4
1000	336.4	130.	37.4	16.4
1100	369.6	42.6	212.0	17.2
1200	394.4	45.8	193.2	14.6
1300	444.8	50.8	202.0	30.4
1400	465.6	61.4	223.6	20.0
1500	530.4	57.4	148.0	4.6
1600	560.4	63.0	265.2	25.2
1700	582.8	65.2	299.6	22.6
1800	588.0	74.2	322.8	17.4
1900	610.8	68.4	286.4	23.0
2000	677.2	78.4	348.0	35.4

4.3. Analysis of the results

(1) The first, and perhaps the most important, observation is that 2-SATISFIABILITY is a well-solved problem: even the slowest algorithm took only 44 ms (on a IBM 3090) to solve the largest problem (2 000 variables and 8 000 clauses)!

(2) In the satisfiable case, our experiments show a clear-cut ranking of the four algorithms with respect to running times: L is unquestionably the fastest, followed by AL, S, and SC (see *fig.* 5).

(3) In the unsatisfiable case, the running times of L, AL, and S are roughly comparable, while the running time of SC is by far larger; except for SC, the running times were much smaller in the unsatisfiable case than in the satisfiable one (see *fig.* 6, where the SC-graph is oversized and hence is not shown).

(4) In the satisfiable case, the running times of SC and L grow quite regularly with the problem size. In fact, they are very well fitted by a straight line: actually we found that $\text{TIME}_L = 5.94\,n$ and $\text{TIME}_{SC} = 21.99\,n$, the squared correlation coefficients being $R_L^2 = 0.999$ and $R_{SC}^2 = 1$, respectively. On the other hand, the graph of the running times of AL and S as a function of n

is more irregular, but anyhow lies between two straight lines corresponding to L and SC (see *fig.* 5).

In the unsatisfiable case, the behaviour of SC is as regular as in the satisfiable case. The other three algorithms, instead, behave very irregularly, and exhibit frequent nonmonotonicities (see *fig.* 6). At any rate, their complexity turns out to be sublinear: actually, the best fit among the eleven functional forms supported by the statistical package SPSS/PC was found to be $\text{TIME}_L = 22.39\, n^{0.49}$ with $R_L^2 = 0.844$; $TIME_{AL} = 33.61\ n^{0.44}$ with $R_{AL}^2 = 0.803$; $\text{TIME}_S = 37.04\ n^{0.46}$ with $R_S^2 = 0.845$, respectively. Thus, roughly speaking, the running times are proportional to the square root of n. Furthermore, the running times of L andAL are seen to be highly correlated.

In conclusion, the experimental average complexity of both L and S is lower than their worst-case complexity, and in any case is bounded above by a linear function of n.

(5) In the satisfiable case the vast majority of the variables turned out to be forced.

Let us analyze how variable forcing is propagated in both graph models. To this purpose, it is convenient to introduce some definitions.

DEFINITION 12: In the implication graph, a literal η is *reachable* from the literal ζ if there is a directed path from η to ζ.

In the clause graph, η is *reachable* from ζ if there is an even alternating path with endpoints ζ and η and ending with a matching edge.

In either graph, we denote by $C(\zeta)$ the set of literals that are reachable from ζ and their complements. Notice that whenever ζ is forced to 0, all literals in $C(\zeta)$ are also forced.

Under the uniform probability distribution model, the set $C(\zeta)$ turned out to be usually quite large: this explains the large percentage of forced variables, as well as the unsatisfiability of almost all rendomly generated quadratic expressions.

From Table III, one can see that S has the ability to detect almost all forced variables (91% of all variables). AL detects slightly less forced variables, while L recognizes on the average only 40% of all variables to be forced. SC as such does not recognize forced variables.

(6) A direct comparison between L and AL shows that the latter algorithm, in spite of its $O(m)$ worst-case complexity, is more than twice slower than the former one, whose worst-case complexity is $O(mn)$.

The point is that L "capitalizes on luck", while AL follows a more "pessimistic" approach and is less affected by random factors, which may increase its running time in the worst-case, but may also decrease it on the average. Actually, for L to reach its $O(mn)$ worst-case complexity, the following events must take place:

(1) Every time a guess is made, it is always the wrong one;

(2) Every time a wrong guess is made, the resulting conflict is detected very late;

(3) Every time a conflict takes place, the alternating guess results in a very early blocking.

However, under the uniform model, things do not go that way:

(1) A guess is successful in about 50% of the cases;

(2) Every time a wrong guess is made, the resulting conflict is detected rather early, since a conflict is due to "local obstructions" [13];

(3) Every time a conflict takes place, a certain literal ζ is recognized as being forced; as a consequence, a large set $C(\zeta)$ of literals must be forced [*see* (5)].

A typical history of the behaviour of the two algorithms is shown in test No. 3 with 2000 variables. Algorithm L finds a conflict right after 65 variables have been labelled. The opposite guess ends up with a blocking after labelling 1839 variables (which, of course, must be forced). From this point on, a sequence of 136 blockings follows, such that between any two of them only one or two variables are labelled. On the other hand, in Algorithm AL a "red" conflict occurs after 65 variables have been labelled. At this moment the red labelling is discontinued, while the green one goes on until 1839 variables (as before) are labelled. From now on, 154 red or green blockings take place; almost always between any two of them only one variable (occasionally two) is labelled.

After each such blocking, AL wastes time by performing both opposite guesses, while L always makes a right one. At the end, 2065 variables are labelled by L and 2240 by AL.

This explains only in part the difference between the running times of L and AL (29878 and 59116 μs, respectively). The rest is due to the overhead for the alternate use of two labellings.

(7) An analysis of the behavior of Algorithm SC shows that its running time is actually proportional to n, whatever the input structure is. As a matter on fact, in the satisfiable case, SC must necessarily generate all strong

components. It turns out that there are always two large (mirror) strong components; the remaining ones are (mirror) singletons.

For example, in the above mentioned test problem with 2000 variables, the two large components include 1658 nodes each, while the remaining 684 components are singletons. In the unsatisfiable case, the two large components collapse into a single component including more than half vertices of the implication graph. Clearly this is the component containing a pair of mirror literals. Hence, also in the unsatisfiable case a large percentage of nodes must be generated and thus the running time does not differ too much from the time required in the satisfiable case.

(8) The running time of Algorithm S strongly depends on the number of switching operations, even though in our implementation switching is done "virtually", *i.e.* by keeping track of commuted variables. It is interesting to point out that in the unsatisfiable case no switching at all is required in about 80% of the test problems. When this happens, S is usually faster than L.

5. EXPERIMENTAL RESULTS II: BINOMIAL DISTRIBUTION

5.1. Drawbacks of uniform distribution

As mentioned in the previous section, in all our experiments with the uniform probability model, in the satisfiable case, we have observed a number of peculiar phenomena:

– All strong components of the implication digraph turned out to be singletons, with the exception of two mirror "megacomponents" (only one in the unsatisfiable case).

– The number of conflicts detected by L was always either 0 or 1, while in AL was almost always 1.

– The number of forcing trees in S was always one.

An analysis of these phenomena suggests that there is a big "continent" of literals which are reachable from each other, surrounded by an "archpelag" of unreachable literals.

A theoretical explanation of these phenomena can be found in the works of Hansen, Jaumard, Minoux [8] and Karp [9].

In view of the above facts, we have come to the conclusion that the uniform probability model is not the best one for the purpose of the experimental evaluation of 2-SAT algorithms.

Thus we have looked for a different probability model which would ensure more balanced sizes of the strong components.

5.2. Binomial generator

The basic idea is to generate directly the strong components and then to link them. Let us start with the generator of satisfiable expressions. As we have seen, in this case Algorithm *SC* can label the strong components so that: (i) if some component gets the label 0, then its disjoint mirror component gets the label 1, and vice-versa; (ii) no edge goes from a component labelled 1 to a component labelled 0.

Hence one can partition the vertex set of the implication graph into two "halves" S_0 and S_1 such that $\zeta \in S_0 \Rightarrow \zeta' \in S_1$ and no edge goes from S_1 to S_0. Each strong component must be contained either in S_0 or in S_1. The generator consists of the following steps:

(1) Generate S_0 by complementing a random set of variables; let S_1 be the complement of S_0.

(2) Generate at random a partition $\{C_1, \ldots, C_p\}$ of S_0.

(3) Introduce in each C_h a random hamiltonian circuit, so as to make C_k a strong component.

(4) Generate at random $m-n+s$ edges (where s is the number of singleton components) so that

(*a*) each such edge has probability 1/2 of having both its endpoints in S_0 and probability 1/2 of going from S_0 to S_1;

(*b*) if C_i, $C_j \in S_0$, $i<j$ and an edge goes from C_i to C_j, then its orientation is reversed.

(5) Complete the graph by adding, for each edge, its mirror edge.

Clearly, the distribution of the sizes of the strong components produced by this generator is binomial.

In order to obtain a generator of instances that are unsatisfiable with probability close to 1, we have modified Step 4 so as to allow also for edges going back from S_1 to S_0.

Let p and $q=1-p$ be the probability that an edge goes from S_0 to S_1 and from S_1 to S_0, respectively. The obvious idea is to choose $p=q=1/2$. However, we have empirically found that this choice gives rise to a megacomponent, similarly to the uniform case. In order to avoid megacomponents while still achieving unsatisfiability with probability close to 1, we have chosen $p=5/7$, $q=2/7$.

5.3. Analysis of the results

In order to obtain a direct comparison between the results obtained with the two generators, we have adopted the same experimental plan for both probability models.

An analysis of the outcomes of 100 satisfiable and 100 unsatisfiable test problems, which were randomly generated according to the binomial model, leads to the following observations.

(1.) The running times are roughly of the same order of magnitude as in the uniform model. However, while we have observed an increase in the running times of L (about 10% in the satisfiable case and 97% in the unsatisfiable case, resp.), AL (about 18% in the satisfiable case and 104% in the unsatisfiable case, resp.) and S (about 12% in the satisfiable case and 222% in the unsatisfiable case, resp.), we have observed a slight decrease in the running time of SC (about 8% in the satisfiable case and 4% in the unsatisfiable case, resp.).

(2.) In the satisfiable case, the ranking of the four algorithms is the same, although the performances of AL, S and SC get closer to each othee (see *fig.* 7).

(3.) In the unsatisfiable case, the big gap between SC and the other three algorithms is confirmed. Once more the ranking is L, AL, S, and SC (see *fig.* 8).

(4.) The growth of the running times as a function of the number of variables follows the same pattern as in the uniform case. The only differences we have observed are that in the satisfiable case, the growth-rate appears to be more regular for all algorithms, while in the unsatisfiable case the growth of SC is slightly more irregular.

(5.) In the satisfiable case, the percentage of forced variables is again quite large, and actually even larger than in the uniform case. Also under the binomial model S has the greatest ability to detect forced variables, followed by AL and L.

(6/7.) No significant difference has been observed with respect to the uniform case (*see* the corresponding points in Sec. 4).

(8.) The strong dependence of the running times of S on the number of switching operations is confirmed. Actually, for this very reason in the unsatisfiable case S behaves less well than under the uniform model, since now the nonoccurrence of switching operations is quite rare.

5.4. Advantages of binomial distribution

Our experiments confirm that, when one makes use of the binomial distribution, many anomalies observed under the uniform model (*see* Sec. 5.1) indeed disappear:

- The sizes of the strong components of the implication digraph are more balanced, and in fact they are binomially distributed.
- The number of conflicts detected by L and AL is often greater than 1.
- The number of forcing trees in S is almost always larger than 1.

In this case, rather than a single big "continent" of forced literals, there are several smaller continents, which is, of course, more natural.

6. CONCLUSIONS

The main conclusions of our experimental work may be summarized as follows:

1. 2-SAT is a well-solved problem. Instances with 2 000 variables and 8 000 clauses were solved in few tens of milliseconds on a mainframe. Hence, in view of the linear growth of running times, solving 2-SAT problems with several tens or hundreds of thousand variables and clauses does not constitute a problem even within the limits of current technology.

2. There is an evident ranking of the four algorithms with respect to running times, namely L, AL, S, and SC. It is worth emphasizing that the fastest one, L, has a non-linear worst-case complexity. The above ranking was independent of the probability model.

3. The boolean expressions generated under the uniform probability model exhibit strong and peculiar structural properties which affect the behavior of the four algorithms. This led us to the conclusion that the uniform model is not the ideal one for testing 2-SAT algorithms. Our binomial generator was designed so as to eliminate the above anomalies, and our experimental results confirm that this goal has been attained.

In view of the above considerations, it is worth looking into the following question which has an obvious relevance to the design of computational experiments on general satisfiability: it is still true that higher order boolean expressions generated under the uniform model tend to have a peculiar structure which may affect the behavior of SAT algorithms?

ACKNOWLEDGEMENTS

We are grateful to Professors M.A.H. Dempster, J. Franco, and B. Jaumard for their valuable suggestions; to Prof. M. Maravalle for carrying out the regressions; to Drs. A. Ciammetti, C. Cavazzani, C. Campetto, and D. Alimonti for their assistance in writing, debugging, and running the computer programs.

BIBLIOGRAPHIE

1. A. V. Aho, J. E. Hopcroft and J. D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, MA, 1974.
2. B. Aspvall, M. F. Plass and R. E. Tarjan, A Linear Time Algorithm for Testing the Thrue of Certain Quantified Boolean Formulas, *Inf. Proc. Lett.*, 1979, *8*, pp. 121-123.
3. S. Cook, The Complexity of Theorem Proving Procedures, *Proc. 3rd ACM Symp. on Theory of Comput.*, 1971, pp. 151-158.
4. M. Davis and H. Putnam, A Computing Procedure for Qualification Theory, *J. of ACM*, 1960, *7*, pp. 201-215.
5. K. W. Deming, Independence Numbers of Graphs – an Extension of the Konig-Egervary Property, *Discr. Math.*, 1979, *27*, pp. 23-24.
6. S. Even, A. Itai and A. Shamir, On the Complexity of Timetable and Multicomodity flow Problems, *SIAM J. Comput.*, 1976, *5*, pp. 691-703.
7. F. Gavril, Testing for Equality Between Maximum Matching and Minimum Node Covering, *Inf. Proc. Lett.*, 1977, *6*, pp. 199-202.
8. P. Hansen, B. Jaumard and M. Minoux, A Linear Expected-Time Algorithm for Deriving all Logical Conclusion Implied by a set of Boolean Inequalities, *Math. Prog.*, 1986, *34*, pp. 223-231.
9. R. M. Karp, The Transitive Closure of a Random Digraph, *Random structures and algorithms*, 1990, *1*, pp. 73-93.
10. R. Petreschi and B. Simeone, A Switching Algorithm for the Solution of Quadratic Boolean Equations, *Inf. Proc. Lett.*, 1980, *11*, pp. 199-202.
11. W. V. Quine, A Way to Simplifying Truth Functions. *Amer. Math. Monthly*, 1955, *52*, pp. 627-631.
12. B. Simeone, Quadratic Zero-Uno Programming, Boolean Functions and Graphs, *Ph. D. Diss.*, Univ. of Waterloo, Ontario, 1979.
13. B. Simeone, Consistency of Quadratic Boolean Equations and the Konig-Egervary Property for Graphs, *Ann. Discr. Math.*, 1985, *25*, pp. 281-290.
14. R. E. Tarjan, Depth First Search and Linear Graph Algorithms, *Siam. J. Comput.*, 1972, *1*, (2), pp. 146-160.