Hidden *Gold* in Random Generation of SAT Satisfiable Instances

Thierry Castell
IRIT - Universite Paul Sabatier
118, route de Narbonne
31062 Toulouse Cedex
France

Michel Cayrol
IRIT - Universite Paul Sabatier
118, route de Narbonne
31062 Toulouse Cedex
France

Abstract

Evaluation of incomplete algorithms that solve SAT requires to generate hard satisfiable instances. For that purpose, the kSAT uniform random generation is not usable. The other generators of satisfiable instances generate instances that are not intrinsically hard, or exhaustive tests have not been done for determining hard and easy areas. A simple method for generating random hard satisfiable instances is presented. Instances are empirically shown to be hard for three classical methods: the "Davis-Putnam" procedure (which is complete), and the two incomplete methods: GSAT and the Break Out Method. Moreover, a new method for escaping from local minima is presented.

1 Introduction

A crucial problem in practical AI development is the problem of satisfiability (called SAT) of a finite set of propositional clauses. SAT is a NP-complete problem. Which AI system does not use a satisfiability test or equivalent (Constraint Satisfaction Problem, Graph coloring, ...)? Formal reasoning is limited because there is no efficient propositional theorem prover. On one hand, the complete methods that solve SAT have not really progressed since the resolution method and Davis-Putnam procedure, except that it has been shown that the choice of the heuristics is central [Dubois *et* al., 1993]. On the other hand, the incomplete methods have been really developed thanks to the results of GSAT for the random instances [Selman *et al.,* 1993].

A threshold phenomenom has been brought to light for the generation of random instances [Cheeseman *et al.,* 1991], making possible the generation of random hard instances. Satisfiable instances are needed for evaluating the incomplete methods. But only 50% of the hardest instances are satisfiable. It has been shown [Chvatal and Szemeredi, 1988] that the unsatisfiable instances are hard (for the resolution) but it has not been shown [Cha and Iwama, 1995] that the satisfiable instances are difficult (for the local search for example). In works on evaluation of incomplete methods, [Konolige, 1994] showed the inefficiency of GSAT on random structured instances, compared with Davis-Putnam procedure. But with few changes, GSAT becomes efficient [Kask and Dechter, 1995]. May be it is due to the fact that these structured instances are not *intrisically hard.* But, what is it the definition of *intrisically hard?* So, a main problem about SAT is to determine if a set of instances is hard. There are no general definition and determination method of the complexity of a set of instances (except when a polynomial class is recognized). For this reason, empirical methods are used for showing the difficulty of the instances produced by a generator. A *hard* set of instances is a set of instances which are hard for the best known algorithms, namely the "Davis-Putnam" procedure [Davis *et al,* 1962], GSAT with random walk [Selman and Kautz, 1993] and the Break Out Method [Morris, 1993].

2 The kSAT Uniform Random Generation

An instance of kSAT is produced by the uniform random generation if each clause of the instance is randomly and independently picked out. These instances are generated from two parameters: the number of variables and the ratio. The ratio is defined as the number of clauses divided by the number of variables. The uniform generation of kSAT instances is an interesting theoretical problem. A threshold phenomenom has been brought to light for the probability to pick out a satisfiable instance as a function of the ratio [Cheeseman *et al.,* 1991] [Crawford and Auton, 1993]. The threshold seems to appear when the probability to generate a satisfiable instance is equal to 1/2. The value of this critical ratio has been determined (empirically) as 4.25 for 3SAT, 9.8 for 4SAT etc [Mitchell *et* al., 1992] [Dubois *et al.,* 1993]. A problem is to determine theoretically this threshold. Near the threshold, if the ratio decreases, the probability to pick out a satisfiable instance quickly tends to 1; if the ratio increases, the probability to pick out a satisfiable instance quickly tends to 0. The more important the number of variables, the more quickly the probability tends to 0 (or 1). Moreover, it has been shown (empirically) that the hardest instances are generated for a ratio value equal to the threshold. For other values, the instances are easier.

This generation method is used for evaluating complete or incomplete theorem provers. In the hardest region, it has been proved that inconsistent instances are difficult (for the resolution) [Chvatal and Szemeredi, 1988], But there is no theoretical result for the difficulty of the satisfiable instances. Futhermore, for generating hard instances, the higher the number of variables, the more the value for the

ratio is precise. A little variation on the ratio can produce only satisfiable or only insatisfiable instances.

For example, if the theoretical value of the critical ratio for 1000 variables is equal to 4.255 then the uniform generation would produced, for 1000 variables and with a ratio equal to 4.25, more than 50% of satisfiable instances. In this case, the evaluation of a local search algorithm under the assumption "50% satisfiable" is a big mistake.

With a lot of variables, the complete methods are unusable. Thus it is impossible to know the rate of success of the incomplete methods. The uniform generation is not a good means for evaluating the incomplete methods if the *theoretical precise value* of the threshold is not known.

3 Satisfiable Instances

The algorithm kSAT_GEN presented in [Cha and Iwama, 1995] produces only random satisfiable instances. The clauses are randomly constructed in order to be satisfied by a given model. There are several generation parameters: number of variables, ratio and literal distribution (i.e. the number of occurrences of each literal in the generated set of clauses). In [Cha and Iwama, 1995], there is no result on the relation between the parameters and the difficulty of the instances. For not producing only easy instances, the studies on the uniform random generation have shown the importance of the value of the parameters. There are a lot of parameters for kSAT_GEN, then it will be difficult to make exhaustive tests.

Two other methods are proposed [Cha and Iwama, 1995] for generating always inconsistent sets of clauses or sets of clauses having one and only one model. The basic steps of these generators are the reversal resolution principle and the reversal subsumption simplification (a clause Cl subsumes a clause C2 iff $Cl \subseteq C2$). If the generation is reversed, a resolution proof can be easily obtained. The time of the generation of the sets of clauses must be limited. So there is a short proof, by resolution, of the empty clause for the inconsistent sets of clauses. And there is a short proof, by resolution, for each prime implicate for the sets of clauses having only one model. The smaller the generation time, the smaller the resolution proofs! This kind of instances is not intrinsically hard, because the resolution proof of the prime implicates is short. A very good resolution-based theorem prover would be able to solve easily this kind of instances.

Let x be a propositional symbol; x is called a *positive* literal and $\neg x$ is called a *negative* literal. One possibility for generating only consistent sets of clauses is to leave out the clauses that have only positive litterals, from a set of clauses generated by uniform random generation (this idea appears in [Morris, 1993] [Rauzy, 1995]). For these instances, a model is obtained by assigning false to all the variables. Unfortunately, this method often produces easy instances. It is due to the fact that the clauses with only positive literals are left out, so there more negative literals than positive literals. Consequently, the Davis-Putnam procedure easily finds a model (with a good heuristic). Moreover, [Rauzy, 1995] has noticed that only "nearly Horn-renamable" instances are generated. To generate hard instances, new parameters must be introduced.

Each clause has a negative literal. An "intelligent" algorithm would be able to show this property. To solved that problem, some literals of the instances have to be renamed (swap x and $\neg x$ for some x), and then some clauses without negative literal appear.

4 How to be Hard

Let us consider the case where the clauses without negative literals are left out. In order to control the proportion of positive and negative literals, [Rauzy, 1995] introduces parameters for determining the probability to pick out a given sign of a clause. For 3SAT, there are four kinds of sign for a clause: (i) all the literals in the clause are positive, (ii) two literals are positive and one is negative, (iii) one literal is positive and two are negative, (iv) all the literals are negative. So there are four parameters for the generation: δ the probability to pick out a clause (i), γ for a clause (ii), β for a clause (iii) and a for a clause (iv).

The probability to pick out each sign of a clause is $\alpha=\delta=1/8$ and $\beta=\gamma=3/8$ for the uniform random generation. With a probability equal to $\delta=0$ to pick out a clause which sign is (i), and with given probabilities for the others, [Rauzy, 1995] proposed two equations for generating instances (before renaming) with approximative^ the same number of positive and negative literals:

(1) $\alpha+\beta+\gamma=1$ (since clauses with only positive literals are rejected)

(2) $3\alpha+\beta=\gamma$ (for the equilibrium between positive and negative literals)

For the sake of simplicity, a restrictive strategy is chosen in our work. Only one parameter is introduced: the probability to pick out a positive literal, called *posp.* The following algorithm is proposed:

```
function kSAT_satisfiable(k, nbv, ratio, posp)
input: k = number of literals per clause, nbv = number of
variables, ratio = value for number of variables/number of
clauses, posp = probability to pick out a positive literal
(0 ≤ posp < 1).
output: a satisfiable instance of kSAT
B := ∅; nbc := 0;
while (nbc < nbv*ratio) do
    {x_1, ..., x_k} := a random part of the set of the nbv
    variables;
    Clause := ∅;
    for i := 1 to k do
        let p be a random number among [0,1[;
        if (p ≤ posp) then Clause := Clause∪{x_i};
        else Clause := Clause∪{¬x_i};
    if a negative literal belongs to Clause then
        B := B∪{Clause}; nbc := nbc+1;
rename B; %swap x and ¬x for some propositional symbol x %
return B;
```

5 Hidden Gold

In order to obtain instances of 3SAT consisting of sets of clauses with the same expected total number of positive and negative literals, in the same method as [Rauzy, 1995], *posp* must be equal to $(\sqrt{5}-1)/2 \approx 0.61803$.

Indeed, with the generator, the values of α,β,γ in function of *posp* are: $\alpha=(1-posp)^3/(1-posp^3)$, $\beta=3posp.(1-posp)^2/(1-posp^3)$ and $\gamma=3posp^2(1-posp)/(1-posp^3)$.

In order to satisfy equations (1) and (2) there is only one solution for $posp$: $(\sqrt{5}-1)/2$. We've just found the expected gold, this value is equal to

$$1/the\_golden\_ratio$$

6 Search Algorithms

The generator kSAT_satisfiable has been tested on three classical different methods. This generator has also been used to evaluate two new personal versions of the Break Out method.

6.1 Complete Method

We use the classical "Davis-Putnam" procedure (DP), It is an enumeration of the interpretations. With sophisticated heuristics, it is the most efficient complete algorithm for the uniform random instances of kSAT. For our tests, an implementation with heuristics of C-SAT is used [Dubois *et al.,* 1993]. C-SAT generally solves in few seconds hard instances (from uniform random generation) having 200 variables. Instances having 300 variables are solved in less than one hour. Instances having more than 500 variables require some days, years or much more ...

6.2 Incomplete Methods

These methods are incomplete because they cannot always find a model. But they can solve very large problems [Selman *et ah,* 1992]. At the present time, the best algorithms for SAT are based on the hill-climbing algorithm. For a given interpretation, these methods try to decrease, by local changes on the interpretation, the number of falsified clauses. A local change in the interpretation is an inversion of the truth value of a variable, called *flip* of the variable. For these methods, the crucial problem is to escape from a local minimum.

We used the Break Out Method (BOut) [Morris, 1993] and GSAT with random walk [Selman and Kautz, 1993]. BOut escapes from a local minimum by weigthing the falsified clauses on that local minimum. The results presented in [Cha and Iwama, 1995] show that this method seems to be better than GSAT which is supposed to be the best incomplete method for solving SAT. GSAT [Selman *et al.,* 1992] is the more popular incomplete algorithm for SAT. There exists some refinements, in particular a notion of weighting [Selman and Kautz, 1993] as for BOut. For the tests, we used the version with random walk. A random walk is an elementary improvement for escaping from local minima: with a given probability, a variable that appears in a falsified clause is flipped.

6.3 Escaping from a Local Maxima

Break Out Method with Jump

Now an improvement of BOut is presented. It is called the Break Out Method with Jump (BOJ). On a local minimum very often the same clauses are falsified. To avoid that, on a local minimum, *all the variables that appear in the falsified clauses are flipped.* Thus several flips have to be done for falsifying again these clauses.

Another change is introduced. In some cases, the current interpretation tends to the opposite of a model; in order to solve that, the following property is used: when the sum of the weights of the clauses that are totally satisfied (each literal of the clause is satisfied) is lower than the sum of the weights of the falsified clauses, if all the variables are flipped, the new interpretation is better.

The Break Out Method with Jump is represented by the following algorithm:

```
function BOJ(B) : boolean
input : a set B of clauses.
output : true if B is satisfiable.
let I an initial random interpretation;
until (I becomes a model of B) do
    if (the sum of the weights of the totally satisfied
    clauses is lower than the sum of the weights of the
    falsified clauses)
    then all the variables of I are flipped;
    if (I is not a local minimum) then
        Poss_flips:=the set of the variables that minimize
        the sum of the weights of the falsified clauses;
        flip on I a variable from Poss_flips;
    else
        add a weight to the falsified clauses;
        flip on I all the variables that appear in the
        falsified clauses;% jump %
return True;
```

A Specialized Algorithm: *Mirror*

This variation of BOut called *Mirror* is presented because its efficiency is a mystery for us. Its principle is simple. It is BOut with only one new operation. On a local minimum, falsified clauses are normally weigthed and immediately after *all the variables are flipped* (it is the reason for the name *Mirror).* In the hardest areas of the other incomplete algorithms, this method is amazingly efficient. But it has a lot of problems on obvious areas. Moreover this method seems to be efficient only with our generator.

7 Empirical Results

The figures appear at the end of the paper.

kSATjsatisfiable has been tested with a lot of values for nbv, ratio and *posp,* for the 3SAT case. Each point is

calculated from a sample of 100 instances. Only general results for 100 variables are presented (the results are similar for 50, 150 and 200 variables). Here we are interested in the behaviour of the different methods as well as the easy and hard instances of the generator. Comparing the algorithms is outside the scope of the paper.

The complexity of Davis-Putnam procedure is measured in terms of the sum of the average number of calls, the number of unit propagations and the number of pure literal simplifications (a literal is a pure literal for a set of clauses if its complementary literal does not appear in this set). The complexity of the incomplete methods is measured in terms of the average number of tested interpretations. This measurement is more precise than the number of flips. Indeed, in order to choose a best neighbour, a local search method inspects the interpretations in the neighbourhood of the current interpretation (explicitly or implicitly). Therefore, this measurement enables us to differenciate algorithms which do not have the same kind of neighbourhood, or which do not explore the search space in the same way.

The most classical parameters are chosen for the incomplete algorithms. The initial weight of the clauses is fixed to one and a unit weight is added on local minima. The probability of the random walk for GSAT is equal to 1/2. In order to avoid a too quick restart, no restart is realized. The programs are stopped after 100 000 flips, but GSAT with random walk is stopped after 200 000 flips. It is a high value compared to values used in experimentations with uniform random generation. Experimentations indicate that GSAT with random walk needs in average 27 654 flips and almost no restart in order to prove the consistency of the instances composed of 200 variables generated at the threshold by the standard model [Selman *et al.,* 1994].

7.1 The Hardest Areas

As for the ratio of instances solved (figure 1), for DP it is equal to 100%, which is not surprising since DP is a complete method. The performances of GSAT with random walk are poor: close to 0% of instances solved for a ratio equal to 8 and *posp* = 0.8. BOut behaves better but still have big weaknesses. BOJ behaves quite well, almost all instances are solved. Moreover, BOJ has the same behaviour of the other algorithms, but the failure starts only at around 200 variables. As for complexity (figure 2), GSAT, BOut and BOJ have similary curves. The complexity of these algorithms grows with the ratio. It is more simple for DP, the form of the complexity is easy-hard-easy. It is like the uniform random generation. The hardest areas are located at a ratio=4.4 and $posp \approx 0.64$. However, the same "ridge" of hardness is present in DP: when *ppos* increases with the ratio.

We could conclude that this model for generating consistent instances generates hard instances and that the region of the hardest instances follows a ridge which is common to all methods: *when ppos* increases with the ratio. But we are immediately going to check that this conclusion is erroneous.

7.2 Calling the conclusions into question

We must not conclude too quickly, even if some evidences are showed. We are going to show that the presence of a common ridge in the complexity curves is not directly linked with the hardness of instances but with the methods used. To show this, we are going to make again the tests with *Mirror,* figure 3. Moreover, when describing the DP procedure used in the experimentations, we omitted an important factor: the presence of a heuristics in the choice of the first subtree explored (heuristics of S-SAT [Dubois *et al.,,* 1993]). We are also going to make again the tests with DP, but with a choice of the first subtree explored which is opposed to that proposed by the heuristics and which is random, figure 4.

The curves always show the presence of a ridge, but here *posp* decreases when the ratio increases! For a high *posp* value and a high ratio, we observe that the generated instances are easy for *Mirror.* 100% of the instances are solved quickly. And for the hardest area *of Mirror* (low *posp* value and a high ratio), the generated instances are easy for BOut, BOJ and GSAT: 100% are solved quickly.

When the first subtree explored by DP is selected randomly then the two ridges appear, figure 4.

8 Conclusion

With the kSAT-consistent function, it is easy to generate hard satisfiable instances. These instances turn out to be practically much harder than instances generated by the standard random model. 50 variables are enough to make GSAT+random walk and the Break Out Method little efficient. The generated instances have no structure and yet they are very hard to solved (by almost the most popular algorithms). As for the efficiency of local search, the results of *Mirror* bring into light gaps of classical approaches. There exists instances generally simple to solve *(byMirror)* which may be extremely difficult for methods such as GSAT and the Break Out Method. An important result of our tests is that incomplete algorithms must be evaluated with all the different areas, and not only on the hardest areas of a given algorithm. If *Mirror* was tested only on the hardest area of BOut or GSAT, a mistake would be made on its efficiency.

New operations must be integrated to local search in order to make it more robust. The flip of the whole set of variables (mirror) is one of our proposals, but up to now this operation gives interesting results on this model of generation only. In order to capture the good characteristics of BOut, GSAT and *Mirror,* the BOJ algorithm has been developed. On our generator, BOJ improves BOut and seems also better on other kinds of instances.

We venture to conjecture that for the kSAT-consistent function, the generated instances which are the hardest to solve are, independently from the method used, located on a ratio = 4.4 and $posp \approx 0.64$. For a given algorithm, the

presence of a zone harder than this point brings into light the gaps of the evaluated method.

Generally speaking, the kSAT-consistent function shows that, as complete methods, local search has weaknesses and allows for instances that are not solvable in a reasonable amount of time. As for the uniform generation, the simplified model generation of satisfiable instances must be intensively studied.

Acknowledgments

To Elsa, Claudette for helping us with English, Jerome for helping us finishing the paper and the members of RESSAC.

References

[Cha and Iwama, 1995] B. Cha and K. Iwama. Performance Test of Local Search Algorithms Using New Types of Random CNF Formulas. *Proc. of IJCAI-95,* pages 304-310, 1995.

[Cheeseman *et al,* 1991] P. Cheeseman, B. Kanefsky and W.M. Taylor.Where the Really Hard Problems Are. *Proc. of IJCAI-9l,* pages 331-337, 1991.

[Chvatal and Szemeredi, 1988] V. Chvatal and E. Szemeredi. Many Hard Examples for Resolution. *JACM,* 35 (4):759-768, 1988.

[Crawford and Auton, 1993] J.M. Crawford and L.D. Auton. Experimental Results on the Cross over Point in Satisfiability Problems. *Proc. of AAA1-93,* pages 21-27, 1993.

[Davis *et ah,* 1962] M. Davis, G. Logemann and D. Loveland. A Machine Program for Theorem Proving. *JACM,* (5):394-397, 1962.

[Dubois *et al.,* 1993] O. Dubois, P. Andre, Y. Boufkhad and J. Carlier. SAT versus UNSAT. *2nd DIMACS Challenge Workshop,* 1993.

[Kask and Dechter, 1995] K. Kask and R. Dechter. GSAT and Local Consistency. *Proc. of 1JCA1-95,* pages 616-621, 1995.

[Konolige, 1994] K. Konolige. Easy to be Hard: difficult problems for greedy algorithms. *Proc. of KR-94,* pages 374-378, 1994.

[Mitchell *et ah,* 1992] D. Mitchell, B. Selman.and H. Levesque. Hard and Easy Distribution of SAT problems. *Proc. of AAAI-92,* 1992.

[Morris, 1993] P. Morris. The Breakout Method For Escaping From Local Minima. *Proc. ofAAAI-93,* 1993.

[Rauzy, 1995]-A. Rauzy. On the Random Generation of 3-SAT Instances. *Technical Report 1060-95, LaBRl-URA CNRS 1304,* Universite Bordeaux I, 1995.

[Selman and Kautz, 1993] B. Selman and H.A. Kautz. Domain Independent Extensions to GSAT: Solving Large Structured Satisfiability Problems. *Proc. of IJCAI-93,* pages 290-295, 1993.

[Selman *et ah,* 1992] B. Selman, H. Levesque and D. Mitchell. A new Method for Solving Hard Satisfiability Problems. *Proc. AAAI-92,* pages 337-343, 1992.

[Selman *et ah,* 1994] B. Selman, H.A. Kautz, B. Cohen. Noise strategies for improving local search. *Proc. of AAAI-94,* 1994.

Figures

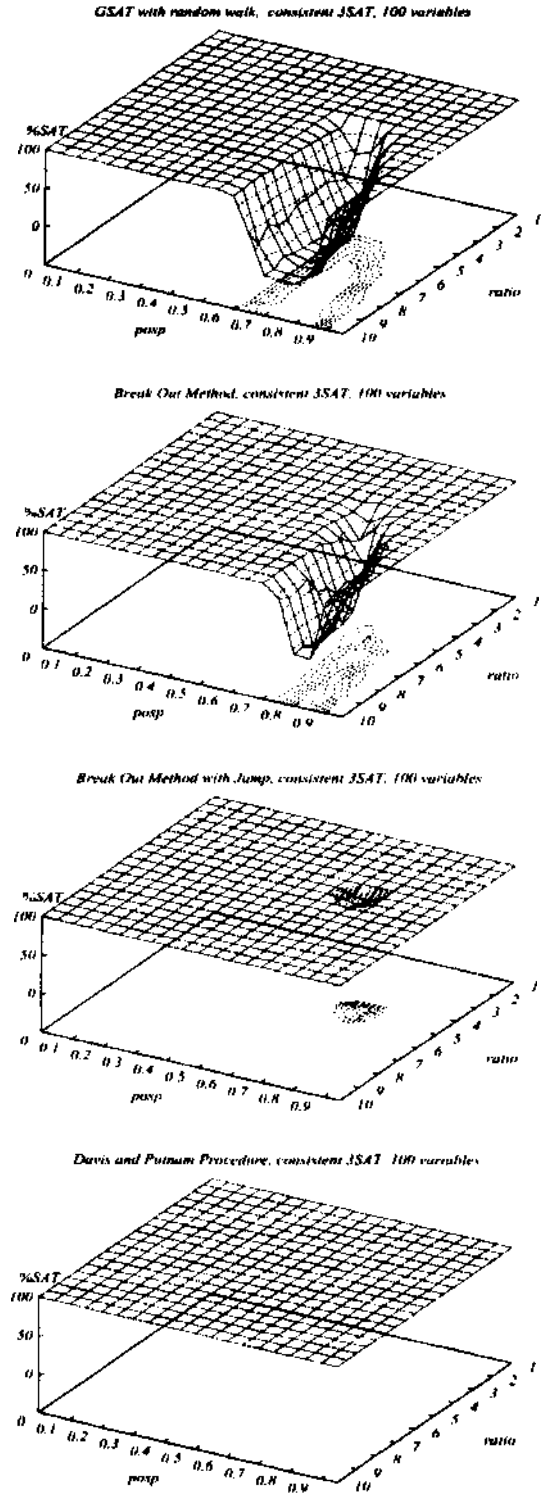

Figure 1: number of solved instances

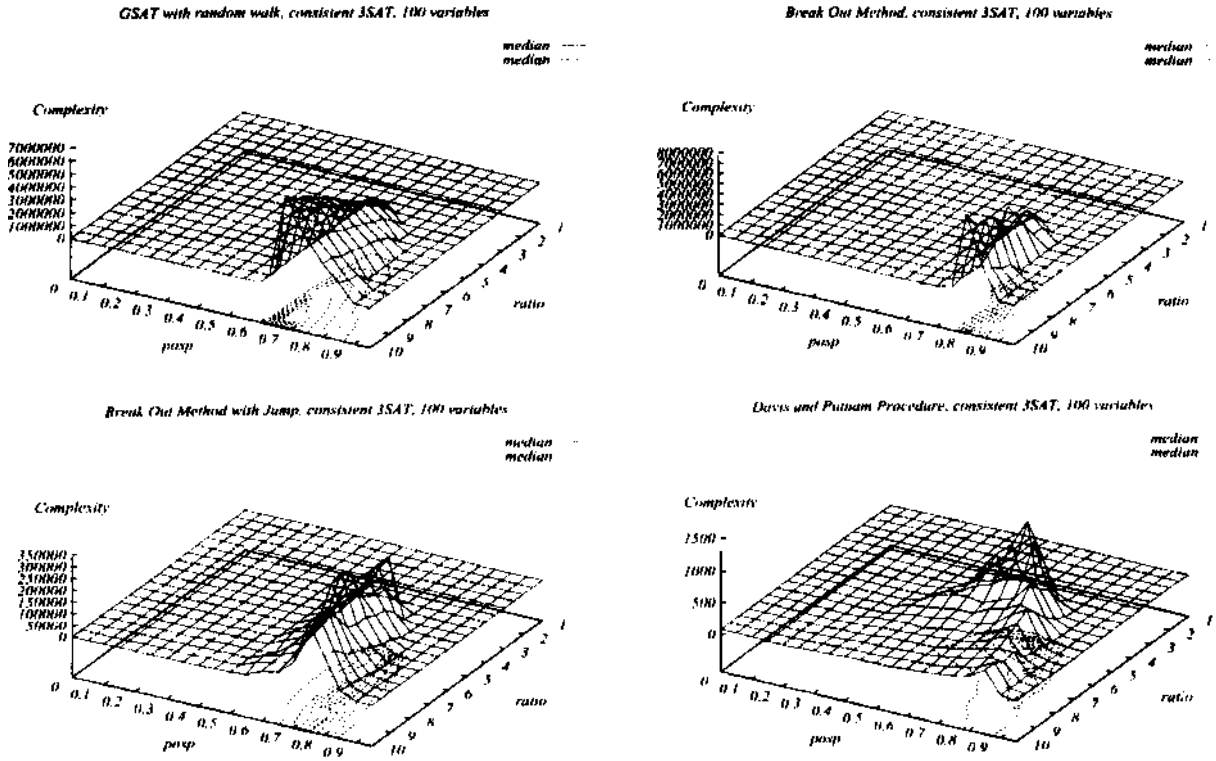

Figure 2: complexity (median curves)

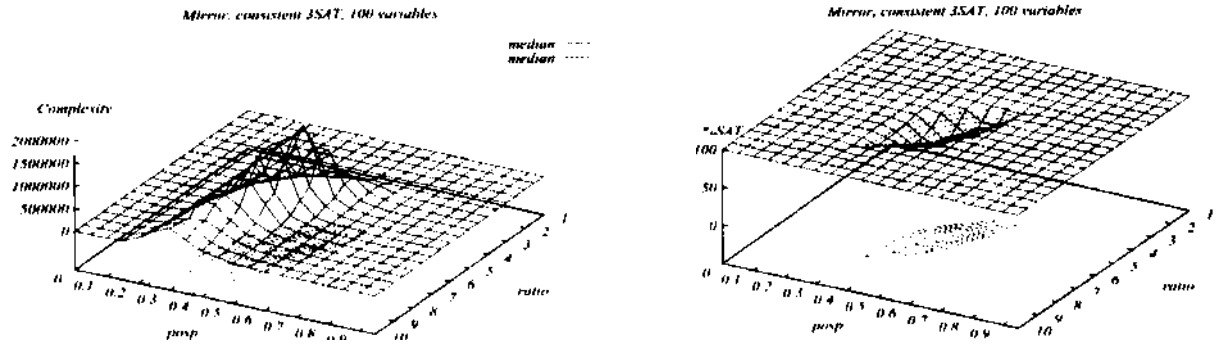

Figure 3: results for *Mirror*

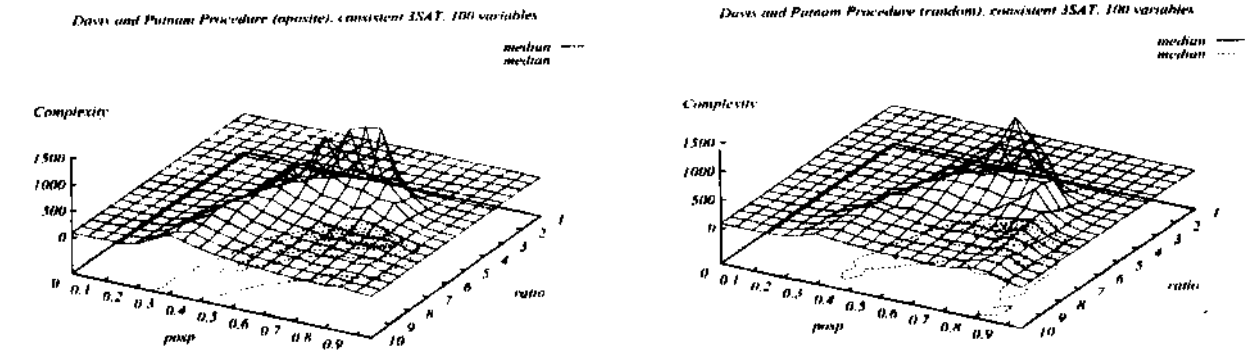

Figure 4: Davis-Putnam with opposite and random choice for the first subtree explored