On the Limit of Branching Rules for Hard Random Unsatisfiable 3-SAT

Chu Min Li[1] and **Sylvain Gérard**[1]

Abstract. We study the limit of branching rules in Davis-Putnam (DP) procedure for hard random unsatisfiable 3-SAT and try to answer the question: what would be the search tree size if every branching variable were the best possible? The issue is of practical interest because many efforts have been spent for designing better branching rules. Our experimental results suggest that the branching rules used in the current state-of-the-art DP procedures are already close to the optimal for hard random unsatisfiable 3-SAT, and in particular, that the first of the ten challenges for propositional reasoning and search formulated by Selman et al. in [14], namely, proving a hard 700 variable random 3-SAT formula is unsatisfiable, probably cannot be answered by DP procedure unless something significantly different from branching can be made effective for hard random unsatisfiable 3-SAT.

1 Introduction

Consider a propositional formula $\mathcal{F}$ in Conjunctive Normal Form (CNF) on a set of Boolean variables $\{x_1, x_2, ..., x_n\}$, the satisfiability problem (SAT) consists in testing whether clauses in $\mathcal{F}$ can all be satisfied by some consistent assignment of truth values (1 or 0) to the variables. If each clause exactly contains r literals, the sub-problem is called r-SAT. 3-SAT is the smallest NP-complete sub-problem of SAT.

SAT is fundamental in many fields of computer science, electrical engineering and mathematics. The Davis-Putnam procedure (DP) [3] is the best complete method to solve SAT, which is roughly sketched in Figure 1.

DP procedure essentially constructs a binary search tree through the space of possible truth assignments until it either finds a satisfying truth assignment or concludes that no such assignment exists, each recursive call constituting a node of the tree. It is well known that the search tree size is generally exponential as a function of the problem size, and the branching variable selected at a node is crucial for the size of the subtree rooting at the node, since a wrong choice may cause an exponential increase of the subtree size.

Many researches on DP procedure concentrate on finding a heuristic to select the branching variable to minimize the search tree size. In this paper, we empirically approximate the minimal search tree size of DP procedure for hard random unsatisfiable 3-SAT in case the best branching variable is selected at every node. We restrict ourselves on unsatisfiable formulas, since the truth value assigned to a branching variable has no importance to the search tree size for a unsatisfiable formula and more importantly the minimal search tree for a satisfiable formula is simply a literal chain if the truth value is appropriately assigned to the branching variables.

procedure DP($\mathcal{F}$)
Begin
if $\mathcal{F}$ is empty, return "satisfiable";
$\mathcal{F}$:=UnitPropagation($\mathcal{F}$);
if $\mathcal{F}$ contains an empty clause, return "unsatisfiable";

/* branching rule */
select a variable x in $\mathcal{F}$ according to a heuristic H,
if the calling of DP($\mathcal{F} \cup \{x\}$) returns "satisfiable", return "satisfiable", otherwise return the result of calling DP($\mathcal{F} \cup \{\bar{x}\}$);
End

procedure UnitPropagation($\mathcal{F}$)
Begin
while there is no empty clause and a unit clause l exists in $\mathcal{F}$, satisfy l and simplify $\mathcal{F}$;
return $\mathcal{F}$;
End

Figure 1. DP Procedure

We study the limit of branching rules for hard random unsatisfiable 3-SAT, because branching actually is the only effective technique to solve this problem. Since our major objective is to study the limit of branching rules, it is natural to focus on it to isolate the impact of branching rules.

The experiments are very time consuming and take more than 5 months on 6 PCs with 300 Mhz pentium CPU under Linux.

Our results suggest that a DP procedure, even with an optimal branching rule able to select the best branching variable at every node, probably would not be substantially better than the current state-of-the-art ones for hard random unsatisfiable 3-SAT. In particular, we feel that the first of the ten challenges for propositional reasoning and search formulated by Selman et al. in [14], namely, proving a hard 700 variable random 3-SAT formula is unsatisfiable, probably cannot be answered by a DP procedure unless something significantly different from branching can be made effective for hard random unsatisfiable 3-SAT.

The paper is organized as follows. Section 2 presents *Satz*, the fastest DP procedure of which we are aware for random 3-SAT. Section 3 presents our approach based on Satz to approximate the min-

[1] LaRIA, Université de Picardie Jules Verne, 5 Rue du Moulin Neuf, 80000 Amiens, France, email: {cli, gerard}@laria.u-picardie.fr

ECAI 2000. *14th European Conference on Artificial Intelligence*
Edited by Werner Horn
Published in 2000 by IOS Press

imal search tree size of DP procedure for hard random unsatisfiable 3-SAT. The experimental results are also presented. In section 4 we try to validate the approach by comparing the approximate value with the computed real mean minimal search tree size of DP procedure for small hard random unsatisfiable 3-SAT. Section 5 discusses related work and section 6 concludes.

2 About Satz

Satz is a DP procedure with a powerful branching rule which selects the variable allowing to generate more and stronger constraints. The strategy is motivated as follows. All leaves (except one for a satisfiable formula) of a DP search tree represent a dead end where an empty clause is found. In order to minimize the search tree, a DP procedure should try to reach the dead end as early as possible.

Let $w(l)$ measure the constraints introduced in $\mathcal{F}$ when satisfying the literal l. $w(l)$ is obtained by running UnitPropagation($\mathcal{F} \cup \{l\}$) and roughly corresponds to the number of new binary clauses in $\mathcal{F}$ when branching on l. Satz branches on the variable x such that $w(\bar{x}) * w(x) * 1024 + w(\bar{x}) + w(x)$ is the greatest. So the branching rule is "two-sided", i.e. $w(\bar{x})$ and $w(x)$ are balanced for a branching variable x. Two exceptions are treated in Satz:

1. if $w(l)$ counts an empty clause, DP procedure should branch next on l then immediately backtrack and satisfy $\bar{l}$;
2. if $w(l) \gg w(\bar{l})$, l generally is not selected as branching variable since l and $\bar{l}$ are not balanced. However if $w(l)$ is very large, $\mathcal{F}$ would have many new strong constraints after branching on l, which probably implies a contradiction in $\mathcal{F}$. If the contradiction can be easily discovered by further unit propagations, DP procedure should branch next on l then immediately backtrack and satisfy $\bar{l}$.

The two cases are not treated as branching points in Satz, but as a simplification of $\mathcal{F}$ by directly satisfying $\bar{l}$, which considerably reduces the search tree size, since the search tree for $\mathcal{F} \cup \{\bar{l}\}$ is at most as large as that for $\mathcal{F}$. For our convenience in this paper, we call the simplification subprocedure of Satz *FurtherSimplification($\mathcal{F}$)* and use it in our approach, since any formula $\mathcal{F}$ will be simplified by the subprocedure before branching. For more details about the branching rule of Satz, see [11, 10].

The original version of Satz as presented in [11] was already the fastest procedure for hard random 3-SAT. Recall that random 3-SAT formulas are hard if $m/n \approx 4.25$ [13, 2], where m is the number of clauses, and n the number of variables. Table 1 taken from [11] compared Satz with 3 other state-of-the-art DP procedures: C-SAT [4], Tableau [2], and Posit [5] on hard random 3-SAT (including satisfiable formulas).

Constant effort is made to improve Satz. Table 2 extracted from [10] displays the performance of the newest version of Satz compared with an older version which was already improved from the original one reported in table 1.

Considering the amount of effort spent for designing better branching rules, our question now is: can Satz be still substantially improved by just improving its branching rule? Or more generally, assuming every branching variable in a search tree is the best possible, is the search tree size substantially smaller than that of Satz? We try to answer this question for hard random unsatisfiable 3-SAT in the next section.

Table 1. Mean run time (in second on SUN Sparc 20 with 125 Mhz CPU) and mean number of backtrackings (t_size) of C-SAT, Tableau, Posit and Satz for hard random 3-SAT ($m/n = 4.25$)

	300 vars 300 problems		350 vars 250 problems		400 vars 100 problems	
System	$time$	t_size	$time$	t_size	$time$	t_size
C-SAT	77	49567	512	275303	3818	1624869
Tableau	79	43041	558	253366	4544	1524551
Posit	57	61797	474	400588	3592	2751611
$Satz$	34	32780	203	174337	1207	916569

Table 2. Mean run time (in second on a Macintosh G3 300 Mhz under PPCLinux system) and mean number of backtrackings (t_size) of Satz and the improved version for hard random 3-SAT ($m/n = 4.25$). The gain of the new version is given in the last line

	350 vars 500 problems		400 vars 300 problems		450 vars 200 problems	
	$time$	t_size	$time$	t_size	$time$	t_size
Satz	77.64	119248	486	636526	2631	2993061
New Satz	52.02	39908	304	207822	1550	918533
gain	49%	199%	60%	206%	70%	226%

3 Approximating the Minimal Search Tree Size for Hard Random Unsatisfiable 3-SAT

We use Satz to approximate the minimal search tree size for hard random unsatisfiable 3-SAT because its tree size is by far the smallest compared with other DP procedures. See tables 1 and 2. At a branching point, we compare all possible branching variables by running Satz with each of them as the branching variable. After the first forced branching variable, i.e., below the branching point, Satz works under the normal conditions. Then we collect the k best variables and respectively branch on them. The same procedure is recursively called for the two children of the branching point. The smallest search tree size among the k trees rooting at the branching point is returned. We call the procedure *ApproximateMinSize* and sketch it in Figure 2.

Procedure ApproximateMinSize($\mathcal{F}$)
Begin
min:=-1; $\mathcal{F}$:=UnitPropagation($\mathcal{F}$);
$\mathcal{F}$:=FurtherSimplification($\mathcal{F}$);
if $\mathcal{F}$ contains an empty clause return 0;
for every free variable x in $\mathcal{F}$
  run Satz($\mathcal{F} \cup \{x\}$) and Satz($\mathcal{F} \cup \{\bar{x}\}$);
let S be the set of the k best variables for Satz;
for every variable x in S do
  begin
    $size_{pos}$:=ApproximateMinSize($\mathcal{F} \cup \{x\}$);
    $size_{neg}$:=ApproximateMinSize($\mathcal{F} \cup \{\bar{x}\}$);
    if ((min=-1) or ($min > size_{neg} + size_{pos} + 1$))
      $min := size_{neg} + size_{pos} + 1$;
  end
return min;
End

Figure 2. Approximating the minimal search tree size

Note that the same simplification subprocedure FurtherSimplification($\mathcal{F}$) as in Satz is used to simplify $\mathcal{F}$ before branching, which considerably reduces the size of the constructed seach trees by fixing some variables in $\mathcal{F}$.

When k is equal to the number of all free variables, we have the exact minimal search tree rooting at the branching point, since the procedure has compared all possible trees. Otherwise the search trees whose branching variables at the branching point are not in the set of the k variables are eliminated. Since the k variables are selected by comparing the real search trees of Satz rooting at the branching point, we believe that the eliminated search trees are likely larger and the "heavy-tailed distribution" phenomenon (if any) as combatted in [6] is avoided.

We conduct three experiments by varying k. The larger k is, the fewer search trees ApproximateMinSize procedure eliminates. In this case, the approximate minimal search tree size obtained by ApproximateMinSize is more exact. Unfortunately, the experiments would take too much time for large k. We first study the case $k = 1$. For larger k, we concentrate on the tree root and its two children since the branching variables there are the most crucial for the search tree size.

3.1 $k = 1$: The approximate minimal search tree size

We say x_1 is better for Satz than x_2 if the sum of the two tree sizes for Satz($\mathcal{F} \cup \{x_1\}$) and Satz($\mathcal{F} \cup \{\bar{x}_1\}$) is smaller than that for Satz($\mathcal{F} \cup \{x_2\}$) and Satz($\mathcal{F} \cup \{\bar{x}_2\}$). When $k = 1$, ApproximateMinSize becomes a DP procedure using Satz to measure the impact of branching next on a variable x. At every node, the best variable x for Satz is selected to branch on. The obtained search tree is obviously smaller than the tree constructed by Satz. We run ApproximateMinSize procedure ($k = 1$) on random unsatisfiable 3-SAT formulas with $m/n = 4.25$ and vary n from 100 to 300 incrementing by 20. For each n, we solve 300 formulas and give the average tree size. Figure 3 compares the tree size of ApproximateMinSize ($k = 1$) with the newest Satz for 100 to 300 variable formulas.

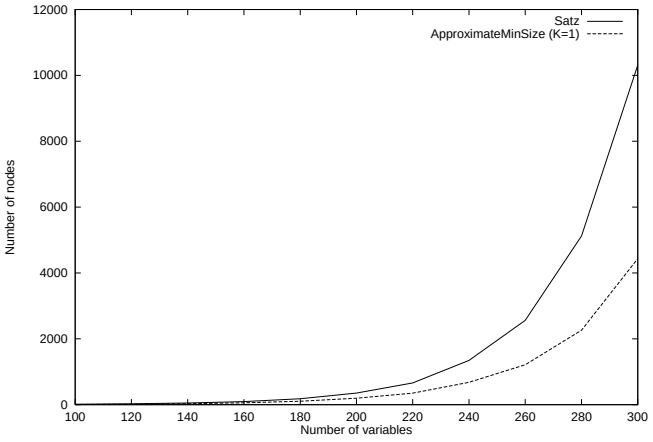

Figure 3. Approximate minimal search tree size and search tree size of Satz

We do not notice a substantial tree size difference between Satz and ApproximateMinSize. The tree size complexity of Satz is $O(2^{(n/21.14)-1})$, while the tree size complexity of ApproximateMinSize is $O(2^{(n/22.18)-1.4})$.

Assuming the complexities displayed in figure 3 can be extended to 700 variables, the tree size of Satz would be $O(2^{32.11})$ for hard 700 variable random unsatisfiable 3-SAT, while the tree size of ApproximateMinSize would be $O(2^{30.16})$. In other words, other things being equal, ApproximateMinSize would be only roughly 4 times faster than Satz.

We run Satz for solving a likely unsatisfiable hard 700 variable random 3-SAT formula (which the stochastic method W-SAT [15] does not find satisfiable), Satz does not return after 100 hours of run time on a 300 Mhz pentium CPU under Linux. A DP procedure with the same search tree size complexity as ApproximateMinSize is unlikely to be able to solve this formula in reasonable time.

3.2 $k \geq 1$: Evidences for ApproximateMinSize procedure

The fact that x_1 is better for Satz than x_2 does not mean that the minimal tree rooting at x_1 is certainly smaller than the minimal tree rooting at x_2. However, we believe that it is often the case for hard random unsatisfiable 3-SAT, and that if the minimal search tree rooting at x_2 is smaller, the difference should not be significant, i.e. if a DP procedure branches on other variables than the best one for Satz, it probably would not give a significantly smaller tree. Below we provide some evidence for our belief.

We run ApproximateMinSize ($k \geq 1$) to take into account the branching variables other than the best for Satz and study three cases: (i) $k = 4$ at the root and $k = 1$ at other nodes; (ii) $k = 2$ at the root and the two children of the root and $k = 1$ at other nodes; (iii) $k = 8$ at the root and $k = 1$ at other nodes. Note that in case (i), 4 search trees are compared and in cases (ii) and (iii) 8 search trees are compared.

We run ApproximateMinSize procedure in the three cases for the same random unsatisfiable 3-SAT formulas as in the case $k = 1$, but limit the execution to $n = 240$ (200 for the case (iii)). Figure 4 compares the approximate minimal tree size obtained in case $k = 1$ with the three cases $k \geq 1$.

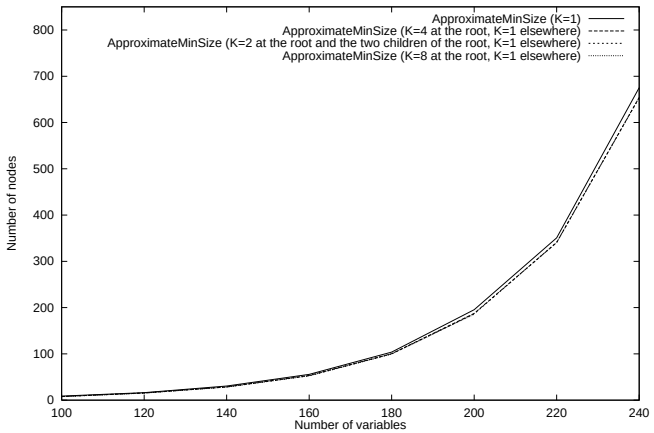

Figure 4. Approximate minimal search tree sizes in cases k=1 everywhere, k=4 at the tree root and 1 elsewhere, k=2 in the first two levels and 1 elsewhere, k=8 at the tree root and 1 elsewhere,

As is expected the three cases $k \geq 1$ give better results, but the difference is not significant. ApproximateMinSize procedure has the same tree size complexity $O(2^{(n/22.18)-1.5})$ in the three cases $k \geq 1$, which is to be compared with the tree size complexity $O(2^{(n/22.18)-1.4})$ in case $k = 1$.

Figure 4 enforces our belief that branching variables other than the best one for Satz probably would not do significantly better for reducing the search tree size and the approximate minimal search tree size obtained by ApproximateMinSize in case $k = 1$ is close to the real minimal search tree size for hard random unsatisfiable 3-SAT.

4 More Evidences for ApproximateMinSize Procedure

Finding the best branching variable at a node of a DP search tree has been proven both NP-Hard and Co-NP-hard [12], so it is unlikely to obtain the exact minimal search tree size when solving a formula of reasonable size. Nevertheless, we can empirically get the minimal tree size for small random unsatisfiable 3-SAT and measure how close is the approximate minimal search tree size to the exact one.

Given a SAT formula $\mathcal{F}$, the procedure GetMinSize sketched in figure 5 compares *all* possible search trees and returns the minimal search tree size. To make the comparison with Satz and ApproximateMinSize procedures meaningful, we use the same subprocedure FurtherSimplification($\mathcal{F}$) to simplify $\mathcal{F}$ before branching by fixing some free variables in $\mathcal{F}$ using experimental unit propagations, which considerably reduces the reported minimal search tree size and makes GetMinSize able to treat larger formulas.

Procedure GetMinSize($\mathcal{F}$)
Begin
   return OptimizedGetMinSize($\mathcal{F}$,
                  ApproximateMinSize($\mathcal{F}$));
End

Procedure OptimizedGetMinSize($\mathcal{F}$,$MinSize$)
Begin
$\mathcal{F}$:=UnitPropagation($\mathcal{F}$);
$\mathcal{F}$:=FurtherSimplification($\mathcal{F}$);
if $\mathcal{F}$ contains an empty clause, return 0;
if ($MinSize \leq 0$) return -1;
$ApproxMin$:= ApproximateMinSize($\mathcal{F}$);
if ($ApproxMin < MinSize$) $MinSize := ApproxMin$;
for every free variable x in $\mathcal{F}$ do
   begin
      $size_{pos}$:=OptimizedGetMinSize($\mathcal{F} \cup \{x\}$,
                                 $MinSize - 1$);
      if ($size_{pos} <> -1$)
         $size_{neg}$:=OptimizedGetMinSize($\mathcal{F} \cup \{\bar{x}\}$,
                        $Minsize - size_{pos} - 1$);
      if (($size_{pos} <> -1$ and $size_{neg} <> -1$)
          and ($MinSize > size_{neg} + size_{pos} + 1$))
         $MinSize := size_{neg} + size_{pos} + 1$;
   end
return $MinSize$;
End

Figure 5. Computing the real minimal search tree size

The parameter $MinSize$ in the procedure OptimizedGetMinSize specifies the largest size of a search tree that a DP procedure is allowed to construct to solve the formula $\mathcal{F}$. The construction is stopped when $MinSize$ is reached. $MinSize$ is initialized by the approximate minimal search tree size for the same formula, then it is modified every time a smaller search tree is obtained. Note that the procedure ApproximateMinSize gives the size of a real search tree, so the minimal search tree cannot be larger than it, which allows to stop the construction of many trees.

In spite of the high optimization, GetMinSize procedure cannot give the minimal search tree size for some 75 variable formulas in reasonable time. We then restrict ourselves on search trees whose branching variables occur in binary clauses, assuming these variables are almost always better. The restriction allows GetMinSize procedure to give the minimal search tree size for 80 variable formulas.

We run Satz, ApproximateMinSize procedure ($k = 1$ and $k = 4$ at the root) and GetMinSize procedure for random unsatisfiable 3-SAT formulas with $m/n = 4.25$ and vary n from 50 to 80 (70 for unrestricted GetMinSize) incrementing by 5. For each n, we solve 300 formulas and give the average tree size. Figure 6 compares the results.

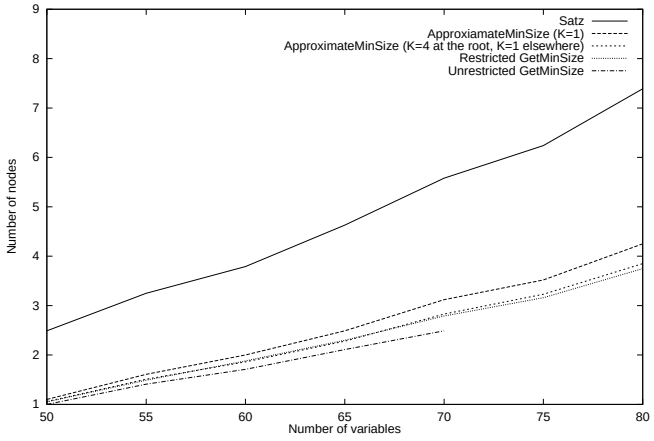

Figure 6. Real mean minimal search tree size versus approximate search tree size

We do not notice a large difference between the results of ApproximateMinSize procedure and GetMinSize procedure, especially compared with the search tree size of Satz. The restricted GetMinSize and ApproximateMinSize ($k = 4$) even give almost the same result. These results give an empirical evidence for the pertinence of ApproximateMinSize procedure. Note that if GetMinSize did not use FurtherSimplification($\mathcal{F}$), it would give a much larger minimal search tree size for $\mathcal{F}$.

From figures 4 and 6, we could predict a probable proportionally smaller difference between the real minimal tree size and the approximated one for larger formulas. Intuitively, the variables in a small formula have very close ties so that there could be a "key" branching variable making the minimal search tree clearly smaller than all other trees. On the other hand, the ties between variables are much weaker in a large formula and there could be a lot of search trees only slightly larger than the minimal one, so that ApproximateMinSize procedure has more chance to approach the minimal search tree size for large formulas.

5 Related Work

Branching rules are a key factor in the success of DP procedure to solve SAT. There are many papers in the literature that focus on the heuristics for branching used in DP algorithm, either proposing new heuristics or analysing properties of existing ones.

A common basis of many effective heuristics is MOM's heuristic which involves branching next on the variable having Maximum Occurrences in clauses of Minimal Size [4, 5, 2, 8]. The heuristic used in Satz is a combination of MOM's heuristics with unit propagation and is called UP heuristic. UP heuristics substantially improve MOM's ones. They go back to C-SAT which tries to discover contradictions at a node near the bottom of a tree by experimental unit propagations. The first uses of UP heuristics are due to Freeman in Posit [5] and Crawford and Auton in Tableau [2].

Analyses of existing heuristics often lead to introduce new ones. Li and Anbulagan have made a systematic empirical study of UP heuristics in [11] and proposed the optimal UP heuristic used in Satz. Hooker and Vinay [7] performed a probabilistic and experimental analysis of several heuristics and proposed a positive two-sided jeroslow-Wang rule. They also proposed a simplification hypothesis providing motivations for a class of heuristics. Although the researches on branching rules are rather empirical, a theoretical study can be found in [9] for some foundations of branching heuristics.

Besides the effort spent for designing better branching rules, Liberatore has shown that it is unlikely to obtain a branching rule able to select the best branching variable to obtain a minimal search tree, since the problem is both NP-hard and Co-NP-hard [12]. Our results suggest that even if there existed such a branching rule, it probably couldnot make a DP procedure substantially better than Satz for random unsatisfiable 3-SAT.

Beame et al. [1] have established a lower bound $2^{\Omega(n/\Delta^{4+\varepsilon})}$ for any DP search tree size for random unsatisfiable 3-SAT with $\Delta = m/n$. The lower bound ($2^{n/326.2539}$ for $\Delta = 4.25$) is far from the empirical approximate minimal search tree size $2^{n/22.18}$ obtained in our experiments. We believe it is also far from the real minimal search tree size.

Selman et al. [14] formulated ten IJCAI challenges in propositional reasoning and search. The first challenge is to develop a way to prove that a hard 700 variable random 3-SAT formula is unsatisfiable. Our results suggest that the challenge probably cannot be answered by DP procedure unless something significantly different from branching can be made effective.

Methods other than DP are obviously encouraged. In fact, the two other challenges in [14] for systematic search (thus for proving unsatisfiability) are to demonstrate that a propositional proof system more powerful than resolution as well as integer programming can be made practical for SAT. The answers to these two challenges would probably imply an answer to the first.

6 Conclusion

We use a special DP-like procedure called ApproximateMinSize to approximate the minimal DP search tree size for hard random unsatisfiable 3-SAT under the hypothesis that every branching variable is the best possible. ApproximateMinSize uses Satz, the fastest DP procedure for random 3-SAT of which we are aware, to select the branching variables. We provide evidence in two aspects for our approach.

First, ApproximateMinSize uses Satz to select several branching variables at the root and the two children of the root, constructs a different search tree for each of them, and compares these trees to give the smallest search tree size. The experimental results do not show a significant difference from the case where ApproximateMinSize always branches next on the best variable for Satz, meaning that other branching variables probably would not do significantly better.

Second, we compute the real mean minimal search tree size of DP procedure for small hard random unsatisfiable 3-SAT and compare the real value with the approximate value obtained by ApproximateMinSize. There is only a very small difference. We predict a probable proportionally even smaller difference for larger formulas.

Our experimental results suggest that the branching rules used in the current state-of-the-art DP procedures are already close to the optimal for hard random unsatisfiable 3-SAT, and in particular, that the first of the ten IJCAI-97 challenges for propositional reasoning and search, namely, proving a hard 700 variable random 3-SAT formula is unsatisfiable, probably cannot be answered by a DP procedure unless something significantly different from branching can be made effective for hard random unsatisfiable 3-SAT.

Acknowledgements

We would like to thank the referees for their comments which helped improve this paper and Henry Kautz for informing us the work of Beame et al [1].

REFERENCES

[1] Beame P., Karp R., Pitassi T., Saks M., *On the Complexity of Unsatisfiability Proofs for Random k-CNF Formulas*, STOC98.

[2] Crawford J.M., Auton L.D., *Experimental Results on the Crossover Point in Random 3-SAT*,Artificial Intelligence, no. 81, 1996.

[3] Davis M., Logemann G., Loveland D., *A Machine Program for Theorem Proving*, In Commun. ACM 5, 1962, pp. 394-397.

[4] Dubois O., Andre P., Boufkhad Y., Carlier J., *SAT Versus UNSAT*. Second DIMACS Challenge: Cliques, Coloring and Satisfiability. Rutgers University, NJ, 1993.

[5] Freeman J.W., *Improvements to Propositional Satisfiability Search Algorithms*, Ph.D. Thesis, University of Pennsylvania, 1995.

[6] Gomes C.P., Selman B., Kautz H. *Boosting Combinatorial Search Through Randomization*, In proceedings of AAAI'98, 1998.

[7] Hooker J.N., Vinay V. *Branching rules for satisfiability*, Journal of Automated Reasoning, 15:359-383, 1995.

[8] Jeroslow R., Wang J. *Solving propositional satisfiability problems*, Annals of Mathematics and AI 1(1990), pp. 167-187.

[9] Kullmann O, *Heuristics for SAT algorithms: Searching for some foundations*, submitted to Discrete Applied Mathematics, 23 Pages.

[10] Li C.M., *A constraint-based approach to narrow search trees for satisfiability*, Information Processing Letters 71 (1999) 75-80.

[11] Li C.M., Anbulagan, *Heuristics Based on Unit Propagation for Satisfiability Problems*, In Proceedings of IJCAI-97, Page 366-371, Nagoya, Japan, August 1997.

[12] Liberatore P., *On the complexity of choosing the branching literal in DPLL*, Artificial Intelligence, Vol 116, Issue 1-2, Pages 315-326, 2000.

[13] Mitchell D., Selman B., Levesque H., *Hard and Easy Distributions of SAT Problems*, In Proceedings of AAAI-92, San Jose, CA, July 1992, pp. 459-465.

[14] B. Selman, H. Kautz, D. McAllester *Ten Challenges in Propositional Reasoning and Search*, in proceedings of IJCAI-97, Nagoya, Aichi, Japan, August 1997.

[15] Selman, B., Kautz H., Cohen B., *Noise strategies for local search*, In Proc. of AAAI-94, pp. 337-343. AAAI Press/The MIT Press, 1994.