A Constraint-Based Approach to Narrow Search Trees for Satisfiability

Chu Min LI[1]

LaRIA, Université de Picardie Jules Verne, 5 Rue du Moulin Neuf, 80000 Amiens France

Keywords: computational complexity, automatic theorem proving, DP procedure, 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 (SAT) problem consists in testing whether clauses in $\mathcal{F}$ can all be satisfied by some consistent assignment of truth values (1 or 0) to variables. If it is the case, $\mathcal{F}$ is said satisfiable; otherwise, $\mathcal{F}$ is said unsatisfiable. If each clause exactly contains r literals, the subproblem is called r-SAT problem. 3-SAT is the smallest NP-complete sub-problem of SAT.

The Davis-Putnam-Loveland procedure (DPL) [2] is the best complete method to solve SAT problems. It constructs a binary search tree for solving $\mathcal{F}$, each recursive call constituting a node of the tree. It is well known that given a problem size, certain problems are much harder than others, necessitating construction of a much larger tree.

Our purpose here is to improve our understanding of why a problem of a given size is harder than another by experimentally studying the shape of trees. The knowledge obtained is used to motivate improvements in DPL. We use hard random 3-SAT problems in our experimentation for two reasons: (i) these problems are very difficult to solve on average especially when the ratio of clause number to variable number is approximatively equal to 4.25 [9, 1], so they are suitable to evaluate a DPL; (ii) these problems are independent of any particular domain, so a technique to

[1]E-mail: cli@laria.u-picardie.fr

solve these problems is applicable to a particular domain, augmented by special techniques related to the domain such as symmetry detection, learning, or intelligent backtracking.

Our experimentation shows that the mean height of a search tree is somewhat irrelevant in the hardness of random 3-SAT problems when using a DPL and if a search tree is larger, it is only because the search tree is wider. The experimental result is of practical interest, because one of objectives when implementing a DPL procedure actually is minimizing the mean height of search trees. We argue that the essential objective should be minimizing the width (instead of the mean height) of search trees, which roughly implies using constraints to find a contradiction as early as possible. The argument extends a simplification hypothesis [6] saying that one should simplify a SAT problem as quickly as possible when branching in a search tree.

The paper is organized as follows. Section 2 briefly presents *Satz* [7], the highly optimized and publicly available DPL procedure used in our experimentation. Section 3 studies the mean height and the width of search trees for hard and easy random 3-SAT problems at the clause-to-variable ratio 4.25. Section 4 presents our constraint-based improvements to Satz validated by experimental results. Section 5 discusses related work and section 6 concludes.

2 About *Satz*

We sketch DPL procedure in Figure 1.

Roughly speaking, *Satz* is a very simple DPL procedure which next branches on the variable reducing the largest number of clauses in $\mathcal{F}$ at every node. More precisely, let $w(x)$ be the number of clauses reduced when x is assigned 1 and $w(\bar{x})$ the number of clauses reduced when x is assigned 0, x is weighted by the equation suggested by Freeman [4]:

$$H(x) = w(\bar{x}) * w(x) * 1024 + w(\bar{x}) + w(x) \quad (1)$$

Satz branches on x such that x is the largest.

The run times and search tree sizes of *Satz* are quite competitive. In fact, *Satz* is the fastest DPL procedure on random 3-SAT problems of which we are aware. It is also among the best for structured problems, even compared with those employing sophisticated look-back searching techniques [8].

procedure DPL($\mathcal{F}$)

```
Begin
   if F is empty, return "satisfiable" else F:=UnitPropagation(F);
   if F contains an empty clause, return "unsatisfiable".

/* branching rule */
   select a variable x in F.  If DPL(F ∪ {x}) is "satisfiable"
   then return "satisfiable", otherwise return the result of DPL(F ∪ {x̄}).
End.

procedure UnitPropagation(F)
Begin
   While there is no empty clause and a unit clause l exists, satisfy l and simplify F.
   Return F.
End.
```

Figure 1: The Very Simple DPL Procedure *Satz*

3 Mean Height Versus Width of a Search Tree

We call *height* of a search tree leaf the number of nodes in the path from the root to the leaf. The *mean height* of a search tree is the sum of heights of all leaves divided by the number of leaves. We call *width* of a search tree level at a depth the number of nodes in the level. The *width* of a search tree is the width of its widest level. Obviously, the search tree size is determined by its width and mean height.

In order to study the hardness of random 3-SAT problems at the ratio $m/n = 4.25$, where m is the number of clauses and n is the number of variables, we solve a large sample of these problems using *Satz* by fixing n at 4 values from 250 to 400 in increments of 50. At each point we divide the solved problems into two classes of same size: the class $\mathcal{H}$ of hard problems and the class $\mathcal{E}$ of easy problems, under the condition that every problem in class $\mathcal{H}$ is harder to solve than any problem in class $\mathcal{E}$. We study the mean height and the width of search trees for problems in class $\mathcal{H}$ and in class $\mathcal{E}$ respectively.

Figure 2 shows the width of search tree levels as a function of depth for problems at $n = 300$ in class $\mathcal{H}$ and in class $\mathcal{E}$. We observe from Figure 3 that the mean height of class $\mathcal{H}$ is the same as that of class $\mathcal{E}$. However the width of class $\mathcal{H}$ is substantially larger than that of class $\mathcal{E}$. We do not distinguish satisfiable and unsatisfiable problems here because they give the same shape of figures. The experimental results on other points give exactly the same conclusions.

The widest level of a search tree is at the mid-depth approximatively equal to $n/20$. Above

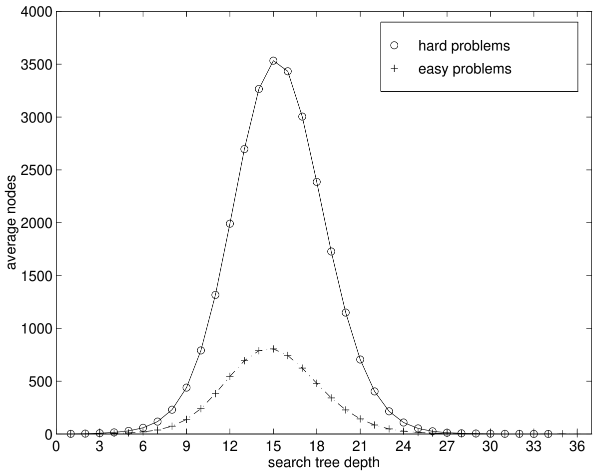

Figure 2: width of search tree levels as a function of depth for problems with 300 variables and 1275 clauses (1000 problems are solved)

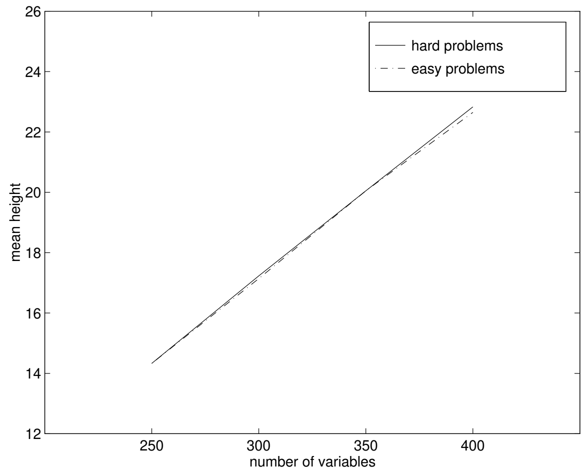

Figure 3: Mean Height of Satz's search trees for hard and easy random 3-SAT problems at the ratio 4.25

the widest level, a node generally has two sons representing two subproblems generated from the node. If it happens that some nodes are leaves representing a dead-end the search tree becomes less wide and the problem is easy, otherwise if no or few nodes above the mid-depth are leaves, the search tree becomes very wide and the problem is hard to solve. However, all problems, hard or easy, have the same behavior below the widest level where more than a half of nodes at each level are leaves, so that the node number is monotonously decreased.

4 Constraint-based improvements to Satz

The experimental results presented in the last section suggest that in order to speed up a DPL procedure one should reduce the width of a search tree instead of reducing its mean height. Reducing the width of a search tree means to reach a dead-end as early as possible.

When there are little binary clauses in $\mathcal{F}$, it is unlikely to reach a dead-end immediately, as is the case for most nodes above the mid-depth. All one can do in this case is to generate a maximum number of strong constraints for deeper branches. On the other hand, when there are many binary clauses, it may be easy to reach a dead-end, as is the case below the mid-depth where the probability for a node to be a dead-end is more than 0.5.

Our approach is based on these considerations.

4.1 Preparing and Generating more and stronger constraints

First we add a preprocessing of the input formula by adding resolvents of length ≤ 3.

For example, if $\mathcal{F}$ contains two clauses $x_1 \vee x_2 \vee x_3$ and $\bar{x}_1 \vee x_2 \vee \bar{x}_4$ then we add a clause $x_2 \vee x_3 \vee \bar{x}_4$ to $\mathcal{F}$. The new clauses can in turn be used to produce other resolvents of length ≤ 3. The process is performed until saturation. Clearly, the added resolvents become explicit constraints in $\mathcal{F}$ as other clauses.

Then we modify the branching rule of Satz in order to generate more and stronger constraints. Roughly speaking, a constraint is stronger if it suppresses more solutions. For example, a binary clause is a stronger constraint than a ternary clause, so the distinction between a binary and a ternary clause is usual when designing a branching rule. Our approach is still more precise since it is based on a distinction between binary clauses.

If $\mathcal{F}$ has n variables, it has 2^n possible solutions. Two binary clauses sharing a complementary literal such as $x_1 \vee x_2$ and $\bar{x}_1 \vee x_3$ remove 2^{n-1} solutions while two binary clauses sharing an identical literal or having no common variable such as $x_1 \vee x_2$ and $x_1 \vee x_3$ or $x_1 \vee x_2$ and $\bar{x}_3 \vee x_4$ remove $2^{n-1} - 2^{n-3}$ and $2^{n-1} - 2^{n-4}$ solutions, respectively. Clearly binary clauses sharing complementary literals remove much more solutions and have more chances to lead to a dead-end where all solutions are removed. So a DPL procedure should branch next on the variable generating subproblems having more binary clauses sharing complementary literals.

For this purpose we define the weight of the literal x by:

$$w(x) = \sum_{l \vee l' \textit{ is produced by } x=1} [f(\bar{l}) + f(\bar{l}')]$$

where $f(\bar{l})$ is the number of binary occurrences of $\bar{l}$ in $\mathcal{F}$ if there is a sufficient number of binary clauses in $\mathcal{F}$ otherwise it is the number of weighted occurrences of $\bar{l}$ in $\mathcal{F}$, an occurrence in a clause of length $r > 2$ being counted as $5^{-(r-2)}$ binary occurrence. $w(\bar{x})$ is similarly defined. We then obtain the weight of variable x by equation 1.

Clearly the more refined weight of x looks further forward by measuring the impact of the newly produced binary clauses when branching on x. The distinction between binary clauses is particularly important for some hard structured problems of real-world such as planning problems which often have a huge quantity of binary clauses.

Remark: The notion "a sufficient number of binary clauses" is empirical in our implementation. We count the number of variables having at least 4 binary occurrences and occuring both positively and negatively in binary clauses. If it is more than 10, we estimate that the number of binary clauses in $\mathcal{F}$ is sufficient.

4.2 Looking further forward to search a dead-end

Unit propagation is a look-ahead searching technique. If the satisfaction of a literal l reduces many clauses, i.e. it introduces many strong constraints by unit propagation, it probably leads to an imminent dead-end which can be reached by further unit propagations. Our looking further forward technique realizes this idea.

If $UnitPropagation(\mathcal{F} \cup \{l\})$ reduces more than T clauses, then for every variable y in the newly produced binary clauses occuring both positively and negatively in binary clauses, execute $UnitPropagation(\mathcal{F} \cup \{l\} \cup \{y\})$ and $UnitPropagation(\mathcal{F} \cup \{l\} \cup \{\bar{y}\})$. If both propagations reach a dead-end, $\bar{l}$ should be satisfied. These two propagations are called *unit propagations of second level*. T is empirically fixed to 65 for hard random 3-SAT problems.

Unit propagations of second level have a great impact on search tree size. Recall that it is usually admitted that an effective branching rule should be "two-sided" [6] [3] [4] [7], i.e., $w(x)$ and $w(\bar{x})$ should be kept balanced for the branching variable x. Equation 1 is used in Satz for this purpose. If $w(x) \gg w(\bar{x})$ or $w(\bar{x}) \gg w(x)$, x is generally not selected as branching variable so that the probable dead-end implied by x or $\bar{x}$ is missed. Unit propagations of second level allow to find many of these dead-ends with moderate cost.

4.3 Experimental results

Table 1 compares our approach with original version of Satz on hard random 3-SAT problems. Note that the gain of our approach grows with the problem size. Our approach improves the performance of Satz on the four benchmarks tested in [8]. Particularly, our approach enables Satz to solve within one hour the problems ii32d3 in DIMACS[2], bf2670-244 and bf2670-400 in UCSC[3]

[2]available from ftp://dimacs.rutgers.edu/pub/challenge/satisfiability

[3]available from ftp://dimacs.rutgers.edu/pub/challenge/sat/contributed/UCSC

that have a huge quantity of binary clauses and could not be solved by the original Satz within two hours.

Table 1: Mean run time (in second on a Macintosh G3 300 Mhz under PPCLinux system) and mean number of backtracking(t _size) of Satz and the improved version for hard random 3-SAT problems at clause-to-variable ratio=4.25. The gain of the new version is given in the last line

	300 vars 1000 problems		350 vars 500 problems		400 vars 300 problems		450 vars 200 problems	
	time	*t_size*	*time*	*t_size*	*time*	*t_size*	*time*	*t_size*
satz	12.56	22390	77.64	119248	486	636526	2631	2993061
new satz	9.29	7800	52.02	39908	304	207822	1550	918533
gain	35%	187%	49%	199%	60%	206%	70%	226%

5 Related Work

The adding of resolvents of length ≤ 3 improves the performance of Satz in most cases, especially for unsatisfiable problems. Nevertheless, it may trouble the resolution of some huge satisfiable structured problems of real-world. Table 2 compares the new version of Satz with and without resolvent adding on Kautz and Selman's planning logistics problems[4].

Table 2: impact (in second) of resolvent adding on Kautz & Selman's planning logistics problems

(#vars, #clauses)	logistics.a (828, 6718)	logistics.b (843, 7301)	logistics.c (1141, 10719)	logistics.d (2160, 27242)
with adding	7	0.3	2	> 7200
without adding	0.8	72	68	290

Satz with resolvent adding, although faster on logistics.b and logistics.c, becomes much slower on the largest problem logistics.d. However a randomization approach combined with a rapid restart strategy proposed and realized in Satz by Gomes *et al.* [5] appears to remedy the situation: instead of branching on *the best* variable, the randomized version of Satz (with resolvent adding) branches on a variable randomly selected among the best *variables* and solves logistics.d in 95 seconds. The randomization also enables Satz to solve 3bit-adder-31 and 3bit-adder-32 problems in Beijing challenge benchmark[5] that were beyond the reach of Satz and other backtrack-style procedures [5].

[4]available by request to kautz@research.att.com

[5]available from http://www.cirl.uoregon.edu/crawford/beijing

Hooker and Vinay [6] propose a simplification hypothesis: a branching rule works better when it creates simpler subproblems, i.e. those with fewer and shorter clauses. The hypothesis implies branching next on the variable allowing to reduce a maximum number of clauses. The two-sided Jeroslow-Wang rule [6], C-Sat [3], Posit[4], Tableau [1] and Satz might be classified under simplification hypothesis.

Our approach is motivated by a constraint hypothesis: more and stronger there are constraints, more opportunities there are to reach a dead-end. A constraint hypothesis can be considered as an extension of simplification hypothesis in the sense that while a simplification hypothesis considers unit propagation for only one branch below the current branching point, a constraint hypothesis motivates the preparation of constraints for unit propagations in deeper branches. Note that the simplicity of obtained subproblems is not an objective under a constraint hypothesis.

Preparing constraints is particularly important above the mid-depth of a search tree, because a dead-end found there would considerably narrow the search tree. Furthermore, when satisfaction of a literal introduces many strong constraints, an imminent dead-end is probably implied and reached by unit propagations of second level, which remedies a drawback of two-sided considerations when designing branching rules.

6 Conclusion

We found that all search trees constructed by *Satz* for hard random 3-SAT problems of a given size have the same mean height and a harder problem corresponds to a wider search tree. This experimental result suggests a way to speed up a DPL procedure by reducing the width of a search tree instead of its mean height.

We then extend the branching rule of Satz by distinguishing binary clauses removing more solutions than others and integrate unit propagations of second level to detect dead-ends when satisfaction of a literal introduces many strong constraints. These constraint-based improvements to Satz, together with the adding of resolvents of length ≤ 3 into the input formula, enable Satz to reach dead-end earlier so as to narrow a search tree and speed up the resolution.

References

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

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

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

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

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

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

[7] Li C.M., Anbulagan, *Heuristics Based on Unit Propagation for Satisfiability Problems*, In Proceedings of IJCAI-97, ISBN 1-55860-480-4, Page 366-371, Nagoya, Japan, August 1997.

[8] Li C.M., Anbulagan, *Look-ahead versus look-back for satisfiability problems*, In Proceedings of CP-97, Springer-Verlag, LNCS 1330, Page 342-356, Austria, 1997.

[9] Mitchell D., Selman B., Levesque H., *Hard and Easy Distributions of SAT Problems*, In Proceedings of the 10th National Conference on Artificial Intelligence (AAAI-92), San Jose, CA, July 1992, pp. 459-465.