Heuristics Based on Unit Propagation for Satisfiability Problems

Chu Min Li & Anbulagan
LaRIA, Univ. de Picardie Jules Verne, 33, Rue St. Leu, 80039 Amiens Cédex, France
fax: (33) 3 22 82 75 02, e-mail: {cli@laria.u-picardie.fr, Anbulagan@utc.fr}

Abstract

The paper studies new unit propagation based heuristics for Davis-Putnam-Loveland (DPL) procedure. These are the novel combinations of unit propagation and the usual "Maximum Occurrences in clauses of Minimum Size" heuristics. Based on the experimental evaluations of different alternatives a new simple unit propagation based heuristic is put forward. This compares favorably with the heuristics employed in the current state-of-the-art DPL implementations (C-SAT, Tableau, POSIT).

1 Introduction

Consider a propositional formula 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 F can all be satisfied by some consistent assignment of truth values (1 or 0) to the variables. If it is the case, F is said satisfiable; otherwise, F is said unsatisfiable. If each clause exactly contains r literals, the subproblem is called r-SAT problem.

SAT problem is fundamental in many fields of computer science, electrical engineering and mathematics. It is the first NP-Complete problem [Cook, 1971] with 3-SAT as the smallest NP-Complete subproblem.

The Davis-Putnam-Loveland procedure (DPL) [Davis *et al.*, 1962] is a well known complete method to solve SAT problems, roughly sketched in Figure 1.

DPL procedure essentially constructs a binary search tree, each recursive call constituting a node of the tree. Recall that all leaves (except eventually one for a satisfiable problem) of a search tree represent a dead end where an empty clause is found. The branching variables are generally selected to allow to reach as early as possible a dead end, i.e. to minimize the length of the current path in the search tree.

The most popular SAT heuristic actually is Mom's heuristic, which involves branching next on the variable having Maximum Occurrences in clauses of Minimum Size [Dubois *et al.*, 1993; Freeman, 1995; Pretolani, 1993; Crawford and Auton, 1996; Jeroslow and Wang, 1990]. Intuitively these variables allow to well exploit the power of unit propagation and to augment the chance to reach an empty clause. Recently another heuristic based on Unit Propagation (UP heuristic) has proven useful and allows to exploit yet more the power of unit propagation [Freeman, 1995; Crawford and Auton, 1996; Li, 1996]. Given a variable x, a UP heuristic examines x by respectively adding the unit clause x and $\bar{x}$ to F and independently makes two unit propagations. The real effect of the unit propagations is then used to weigh x.

procedure DPL(F)
Begin
`if` F `is empty, return "satisfiable";`

`F:=UnitPropagation(`F`); If` F `contains an empty clause, return "unsatisfiable".`

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

procedure UnitPropagation(F)
Begin
`While there is no empty clause and a unit clause` l `exists in` F`, assign a truth value to the variable contained in` l `to satisfy` l `and simplify` F`.`
`Return` F`.`
End.

Figure 1: DPL Procedure

However, since examining a variable by two unit propagations is time consuming, two major problems remain open: should one examine all free variables by unit propagation at every node of a search tree? if not, what are the variables to be examined at a search tree node?

In this paper we try to experimentally solve these two problems to obtain an optimal exploitation of UP heuristic. We define a $PROP$ predicate at a search tree node whose denotational semantics is the set of variables to be examined at a search tree node, i.e. x is to be examined if and only if $PROP(x)$ is true. By appropriately changing $PROP$, we experimentally analyse the behaviour of different UP heuristics. We write 12 DPL procedures which are different only in $PROP$ and run these procedures on a very large sample of hard random 3-SAT problems.

We begin in section 2 by describing the 12 DPL procedures and summarizing the experimental results on these programs. In section 3 we compare a pure UP heuristic and a pure Mom's heuristic and show the superiority of UP heuristics. In section 4 we study different restrictions of UP heuristics. In section 5 we discuss the related work and compare the best DPL procedure in our experimentation with three state-of-the-art DPL procedures. Section 6 concludes the paper.

2 UP Heuristics Driven by $PROP$

Let $diff(F_1, F_2)$ be a function which gives the number of clauses of minimum size in F_1 but not in F_2, we show a generic branching rule in Figure 2, where the equation defining $H(x)$ is suggested in [Freeman, 1995] and the weight 5 for uniformizing clauses of different length is empirically optimal.

For each free variable x such that $PROP(x)$ is true do
let F' and F'' be two copies of F
Begin
    F' := UnitPropagation($F' \cup \{x\}$);
    F'' := UnitPropagation($F'' \cup \{\bar{x}\}$);
    If both F' and F'' contain an empty clause then
        return "F is unsatisfiable".
    If F' contains an empty clause then $x := 0$,
    $F := F''$ else if F'' contains an empty
    clause then $x := 1$, $F := F'$;
    If neither F' nor F'' contains an empty clause
    then let $w(x)$ denote the weight of x
        $w(x) := diff(F', F)$ and $w(\bar{x}) := diff(F'', F)$
End;

If all variables examined above are valued or
$PROP(x)$ is false for every x then
For each free variable x in F do
    let r_i be the length of the clause C_i
        $w(x) := \Sigma_{\bar{x} \in C_i} 5^{-r_i}$ and $w(\bar{x}) := \Sigma_{x \in C_i} 5^{-r_i}$

For each variable x do
    $H(x) := w(\bar{x}) * w(x) * 1024 + w(\bar{x}) + w(x)$

Branching on the free variable x such that $H(x)$ is the greatest.

Figure 2: A Generic Branching Rule Driven by $PROP$

The essential reason to use UP heuristics instead of Mom's one is that Mom's heuristic may not maximize the effectiveness of unit propagation, because it only takes binary clauses (if any) into account to weigh a variable, although some extensions try to also take longer clauses into account with exponentially smaller weights (e.g. 5 ternary clauses are counted as 1 binary clauses). A UP heuristic allows to take all clauses containing a variable and their relations into account in a very effective way to weigh the variable. As a secondary effect, it allows to detect the so-called *failed literals* in F which when satisfied falsify F in a single unit propagation. However since examining a variable by two unit propagations is time consuming, it is natural to try to restrict the variables to be examined. For this purpose we use $PROP$ predicate defined at a search tree node.

The success of Mom's heuristic suggests that the larger the number of binary occurrences of a variable is, the higher its probability of being a good branching variable is, implying that if one should restrict UP heuristics by means of $PROP$, he should restrict UP heuristics to those variables having a sufficient number of binary occurrences. For this reason, all restrictions on $PROP$ studied in this paper are defined according to the number of binary occurrences of a variable, so that the resulted UP heuristics rely on combinations of unit propagation and Mom's heuristics.

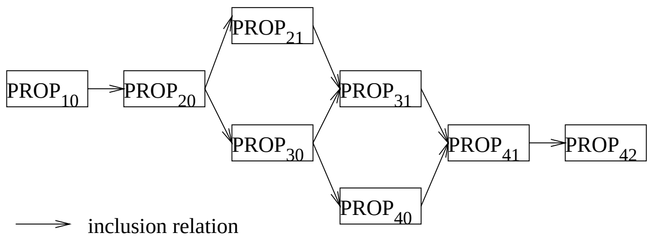

Figure 3: $PROP$ predicate hierarchy by inclusion relation. $PROP_{ij}(x)$ is true iff x has i binary occurrences of which at least j negative and j positive.

The first $PROP$ predicate in our experimentation is called $PROP_a$ and has empty denotational semantics, the resulted branching rule using a pure Mom's heuristic, and the second is called $PROP_0$ whose denotational semantics is the set of all free variables, the resulted UP heuristic is in its pure form and plays its full role:

$PROP_a$: $PROP_a(x)$ is false for every free variable x;

$PROP_0$: $PROP_0(x)$ is true for every free variable x.

Between $PROP_a$ and $PROP_0$, there are many other possible $PROP$ predicates. Figure 3 defines 8 predicates constituting a hierarchy by inclusion relation of their denotational semantics.

$PROP_{3141}$ is defined to be $PROP_{31}$, but for nodes under a fixed depth of a search tree, it is defined to be

$PROP_{41}$. Let T be a constant, the last $PROP$ predicate is named $PROP_z$ and is defined to be the first of the three predicates $PROP_{41}$, $PROP_{31}$ and $PROP_0$ (in this order) which has at least T variables in its denotational semantics.

Each $PROP$ predicate results in a DPL procedure, $PROP_a$, $PROP_0$, $PROP_{3141}$ and $PROP_z$ respectively giving $Sata$, $Sat0$, Sat_{3141} and $Satz$, and $PROP_{ij}$ giving Sat_{ij}. These programs are different only in $PROP$ predicate, except $Sat0$ which need not count the occurrences of variables.

We run the 12 programs (compiled using gcc with optimization) on a PC with a 133 Mhz Pentium CPU under Linux operating system on a very large sample of random 3-SAT problems generated by using the method of Mitchell et al.[Mitchell *et al.*, 1992]. Given a set V of n Boolean variables $\{x_1, x_2, ..., x_n\}$, we randomly generate m clauses of length 3. Each clause is produced by randomly choosing 3 variables from V and negating each with probability 0.5. Empirically, when the ratio m/n is near 4.25 for a 3-SAT formula F, F is unsatisfiable with a probability 0.5 and is the most difficult to solve. We vary n from 140 variables to 340 variables incrementing by 20, for each n the ratio clauses-to-variables (m/n) is set to 4.0, 4.1, 4.2, 4.25, 4.3, 4.4, 4.5. At each ratio and by each program, if $n < 280$ then 1000 problems are solved, if $280 \leq n \leq 300$ then 500 problems are solved, if $n = 320$ then 300 problems are solved, and if $n = 340$ then 100 problems are solved. A problem is solved successively by all the 12 DPL procedures before another to ensure the same environment to all programs. Due to the lack of space, we only present the experimental results for the ratio $m/n = 4.25$ in Figures 4, 5, and 6, where the DPL procedures corresponding to the curves are listed in the same order from top to bottom. The experimental results on the other ratios give exactly the same conclusions.

3 A Pure UP Heuristic Versus a Pure Mom's Heuristics: $Sat0$ vs $Sata$

$Sat0$ systematically examines all the variables by unit propagation at all nodes, using a pure UP heuristic, while $Sata$ does not examine any variable so and employs a pure Mom's heuristic. One might believe that $Sat0$ would be simply too slow, but it is not the case. *$Sat0$ is much faster than Sata.* In fact from Figures 4 and 5, all DPL procedures using a UP heuristic in our experimentation are substantially better than $Sata$ in terms of search tree size and real run time.

Note that Mom's heuristic used in $Sata$ is similar to the so-called two-sided Jeroslow-Wang rule [Hooker and Vinay, 1995], with the only difference that a clause of length i is counted as 5 clauses of length i+1 instead of 2. Our experiments suggest that 5 is better than 2. 5 is also similar to the exponential factors in C-SAT [Dubois *et al.*, 1993] where 5.71 ternary clauses are counted as 1 binary clause.

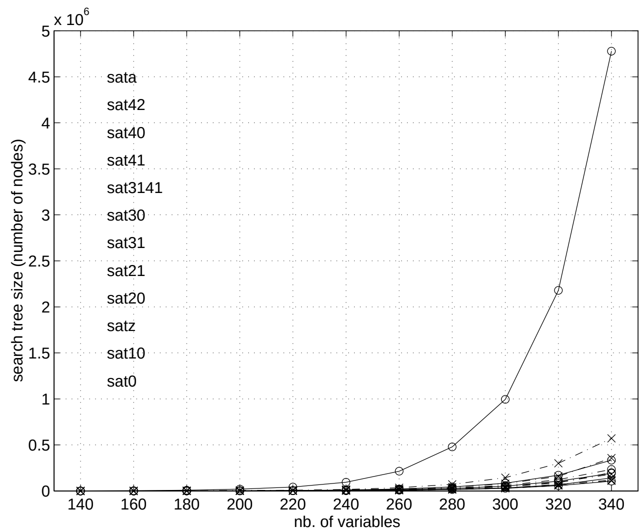

Figure 4: Mean search tree size of each program as a function of n for hard random 3-SAT problems at the ratio $m/n = 4.25$

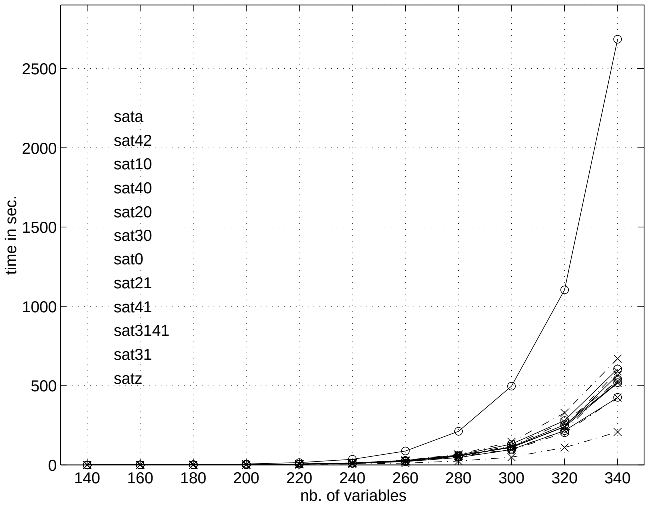

Figure 5: Mean run time of each program as a function of n for hard random 3-SAT problems at the ratio $m/n = 4.25$

$Sat0$ actually is slower than five other programs based on balanced restrictions of variables to be examined by unit propagation, but not substantially so (except $Satz$). The surprisingly good performance of $Sat0$ confirms the power of UP heuristics for selecting the next branching variable and suggests that its effect for detecting failed literals is only secondary.

4 Restricted UP Heuristics

Figure 6 illustrates the number of variables examined by different restricted UP heuristics at a node.

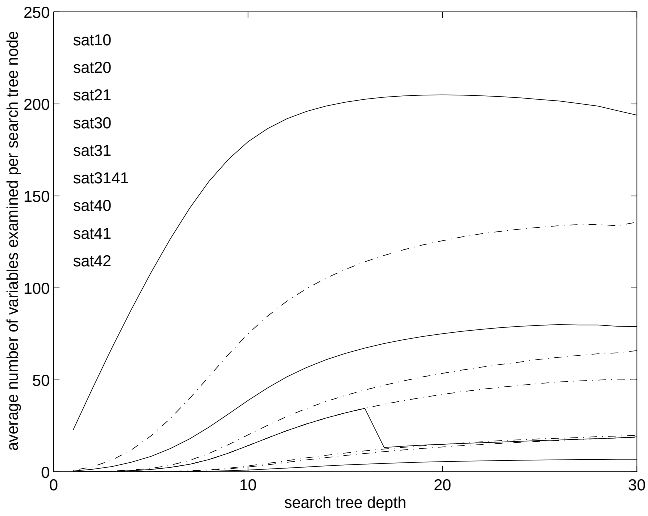

Figure 6: Average number of variables examined at a search tree node in a given depth when solving hard random 3-SAT problems of 300 variables and 1275 clauses (500 problems are solved) for 9 programs

4.1 Restriction by total number of binary occurrences of a variable

Four programs Sat_{10}, Sat_{20}, Sat_{30} and Sat_{40} realize this type of restrictions. While a classical Mom's heuristic selects *the* next branching variable having maximum binary occurrences, the restricted UP heuristics examine a set of variables having more binary occurrences than others, including the variable having maximum binary occurrences. From Figure 4, it is clear that the more variables are examined, the smaller the search tree size is.

4.2 Balanced restriction by total number of binary occurrences of a variable

Four programs Sat_{21}, Sat_{31}, Sat_{41} and Sat_{42} realize this type of restrictions. The $PROP$ predicates require that a variable occurs both positively and negatively in binary clauses to balance the search tree. We compare the duet Sat_{i0} and Sat_{i1} (i=2, 3, 4) and observe that Sat_{i1} examines strictly fewer variables than Sat_{i0} and is faster than it in spite of a slightly larger search tree. In particular, Sat_{41} and Sat_{40} examine almost the same number of variables (see Figure 6), but the balanced restriction gives a faster DPL procedure.

We pay special attention to $PROP_{31}$ and $PROP_{41}$ since they seem to be the best balanced restrictions.

4.3 Dynamic restriction as a function of search tree depth

Sat_{3141} realizes this restriction. A general observation when solving 3-SAT problems using a DPL procedure is that there are more and more binary clauses when descending from the search tree root and the denotational semantics of a $PROP$ predicate such as $PROP_{31}$ becomes larger and larger. Furthermore, the nodes are more numerous near the leaves and the branching variables play a less important role there. It appeared that one could restrict more the variables to be examined by unit propagation near the leaves without important loss on the search tree size so as to obtain some gain in terms of real run time.

POSIT's UP heuristic (called BCP-based heuristic) [Freeman, 1995] realizes this idea: under the level 9 of a search tree, at most 10 variables are examined by unit propagation.

Sat_{3141} uses $PROP_{31}$ from the top of a search tree, but under the depth empirically fixed to $n * 4/70$, it uses $PROP_{41}$, where n is the number of variables in the initial input 3-SAT problem. Note that if $n \geq 160$, $n * 4/70 \geq 9$, so Sat_{3141} generally strengthens the restriction later than POSIT.

From Figures 4 and 5 Sat_{3141} is not better than Sat_{31}, although it makes many fewer unit propagations to examine variables (see Figure 6), suggesting that the search tree depth is rather irrelevant to the restriction of UP heuristics.

4.4 Dynamic restriction by number of variables to be examined

The relatively poor performance of Sat_{42} seems due to the small number of variables examined at each node (see Figure 6), though these variables have many binary occurrences. A careful analysis shows that even Sat_{31}, the best one up to now, examines few or no variables at some nodes, especially near the root where there are few binary clauses, although these nodes are more determinant for the final search tree size. $PROP_z$ is then introduced to ensure that at least T variables are examined at each node, T being empirically fixed to 10. Near the root, all free variables are examined to exploit the full power of UP heuristic. As soon as the number of variables occurring both negatively and positively in binary clauses and having at least 4 (3) binary occurrences is larger than T, only these variables are examined to select the next branching variable.

5 Related Work

C-SAT [Dubois *et al.*, 1993] examines some variables by unit propagations (called local processing) near the bottom of a search tree to rapidly detect failed literals there. Pretolani also uses a similar approach (called pruning method) based on hypergraphs in H2R [Pretolani, 1993]. But the local processing and the pruning method as are respectively presented in [Dubois *et al.*, 1993] and [Pretolani, 1993] do not contribute to the heuristic to select the next branching variable. We find the first effective exploitation of UP heuristic in POSIT [Freeman, 1995] and Tableau [Crawford and Auton, 1996] which use a similar idea as in C-SAT to determine the variables to be examined at a node by unit propagation: *x is to be examined iff x is among the k most weighted variables by a Mom's heuristic.*

The main difference of *Satz* with Tableau and POSIT is that *Satz* does not specify a upper bound k of the number of variables to be examined at a node by unit propagation. Instead, *Satz* specifies a lower bound. In fact, *Satz examines many more variables by an optimal combination of unit propagation and Mom's heuristics.*

Given the depth of a node, Table 1 illustrates the average number of variables examined ($\#examined\_vars$) at the node by *Satz*, with the depth of the root being 0. In order to compare with C-SAT, Tableau and POSIT we also give the theoretical value of k_C (for C-SAT), k_T (for Tableau) and k_P (for POSIT) at the node, respectively according to the definitions of k in [Dubois *et al.*, 1993; Crawford and Auton, 1996; Freeman, 1995].

depth	$\#free\_vars$	$\#examined\_vars$	k_C	k_T	k_P
1	298.24	298.24	0	263	265
2	296.52	296.52	0	227	230
3	294.92	293.89	0	193	198
4	292.44	292.21	0	141	149
5	288.60	282.04	0	61	72
6	285.36	252.14	0	0	10 or 3
7	281.68	192.82	0	0	10 or 3
8	277.54	125.13	0	0	10 or 3
9	273.17	71.51	0	0	10 or 3
10	268.76	40.65	0	0	10 or 3
11	264.55	26.81	0	0	10 or 3
12	260.53	21.55	0	0	10 or 3
13	256.79	19.80	0	0	10 or 3
14	253.28	19.24	0	0	10 or 3
15	249.96	19.16	0	0	10 or 3
16	246.77	19.28	0	0	10 or 3
17	243.68	19.57	0	0	10 or 3
18	240.68	19.97	0	0	10 or 3
19	237.73	20.46	0	0	10 or 3
20	234.82	20.97	0	0	10 or 3

Table 1: Average number of variables examined in *Satz* at a node in a given depth when solving a hard random 3-SAT problem of 300 variables and 1275 clauses (500 problems are solved) compared with theoretical value of k in C-SAT, Tableau and POSIT

It is clear that *Satz* examines many more variables at each node than any of C-SAT, Tableau or POSIT. Near the root, *Satz* examines all free variables. Elsewhere *Satz* examines a sufficient number (T) of variables.

We compare C-SAT, Tableau, POSIT and *Satz* on a large sample of hard random 3-SAT problems on a SUN Sparc 20 workstation with a 125 MHz CPU. The 3-SAT problems are generated from 3 sets of n variables and m clauses at the ratio $m/n = 4.25$, n steping from 300 variables to 400 variables by 50.

We use an executable of C-SAT dated July 1996. The version of Tableau used here is called *3tab* and is the same used for the experimentation presented in [Crawford and Auton, 1996]. POSIT is compiled using the provided *make* command on the SUN Sparc 20 workstation from the sources named *posit*-1.0.*tar.gz*[1]. Table 2 shows the performances of the 4 DPL procedures on problems of 300, 350, and 400 variables, where *time* standing for the real mean run time is reported by the unix command /usr/bin/time and *t_size* standing for search tree size (number of nodes) is reported (or computed from number of branches reported) by the DPL procedures.

	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) and mean search tree size of C-SAT, Tableau, POSIT and *Satz* on ratio m/n=4.25

Table 2 shows that *Satz* is faster than the above cited versions of C-SAT, Tableau and POSIT, *Satz*'s search tree size is the smallest, and *Satz*'s run time and search tree size grow more slowly. Table 3 shows the gain of *Satz* compared with the cited version of C-SAT, Tableau and POSIT at the ratio m/n=4.25. Each item is computed from Table 2 using the following equation:

$$gain = (value(system)/value(Satz) - 1) * 100\%$$

where *value* is real mean run time or real mean search tree size and *system* is C-SAT, Tableau or POSIT. From Table 3, it is clear that the gain of *Satz* grows with the size of the input formula.

	300 vars 300 problems		350 vars 250 problems		400 vars 100 problems	
System	*time*	*t_size*	*time*	*t_size*	*time*	*t_size*
C-SAT	126%	51%	152%	58%	216%	77%
Tableau	132%	31%	175%	45%	276%	66%
POSIT	68%	89%	133%	130%	198%	200%

Table 3: The gain of *Satz* vs. C-SAT, Tableau and POSIT in terms of run time and search tree size on the ratio m/n=4.25 computed from Table 2

The central strategy of *Satz* is to try to reach an empty clause as early as possible. Further along the line, we make two relatively small resolvents-driven improvements in *Satz*. The first improvement is the preprocessing of the input formula by adding some resolvents of length ≤ 3, The second improvement consists in refining yet more the heuristic H in the nodes where all free variables are examined by unit propagation. Refer to Figure 2, when $PROP_z$ is equal to $PROP_0$ we define $w(x)$ as the number of resolvents the newly produced binary clauses would result in in F' by a single step of resolution. $w(\bar{x})$ is similarly defined.

[1] publicly available via anonymous ftp to ftp.cis.upenn.edu in pub/freeman/ directory

Satz improved in this way solves many real-world or structured SAT problems where previous heuristics were not successful. For example, Table 4 shows the performance of the 4 DPL procedures on the well-known Beijing challenging problems[2], where a problem that can not be solved in less than 2 hours is marked by "> 7200" and the version of Tableau is called *ntab*[3]. It is clear that *Satz* is much more efficient and solves many more problems in less than two hours.

Problem	Satz	C-SAT	Posit	ntab
2bitadd_10	> 7200	> 7200	> 7200	> 7200
2bitadd_11	201	> 7200	0.3	> 7200
2bitadd_12	0.4	6379	0.05	> 7200
2bitcomp_5	0.03	0.1	0.01	0.4
2bitmax_6	0.07	3.7	0.01	1.6
3bitadd_31	> 7200	> 7200	> 7200	> 7200
3bitadd_32	4512	> 7200	> 7200	> 7200
3blocks	2.0	4.3	1.8	1468
4blocksb	8.2	118	49	> 7200
4blocks	1542	> 7200	> 7200	> 7200
e0ddr2-10-by-5-1	215	> 7200	> 7200	> 7200
e0ddr2-10-by-5-4	232	> 7200	3508	236
enddr2-10-by-5-1	> 7200	> 7200	> 7200	> 7200
enddr2-10-by-5-8	229	> 7200	> 7200	92
ewddr2-10-by-5-1	339	> 7200	283	> 7200
ewddr2-10-by-5-8	279	> 7200	> 7200	119

Table 4: Run time (in sec.) of Beijing challenging problems

6 Conclusion

We found that UP heuristic is substantially better than Mom's one even in its pure form realized by $PROP_0$ where all free variables are examined at all nodes. In its restricted forms based on combinations of unit propagation and Mom's heuristics, the more variables are examined, the smaller the search tree is, confirming the advantages of UP heuristic, but too many unit propagations slow the execution. The combinations realized by $PROP_{41}$ and $PROP_{31}$ represent good compromises.

A dynamic restriction such as $PROP_{3141}$ which strengthens the restriction under a fixed depth of a search tree fails to work better than the static restriction $PROP_{31}$. We design the dynamic restriction along another line: $PROP_z$ ensures that at least T candidates are examined by unit propagation at *every* node of a search tree by successively using $PROP_{41}$, $PROP_{31}$ and $PROP_0$, giving the very efficient and very simple DPL procedure called *Satz*.

Satz is favorably compared with several current state-of-the-art DPL implementations (C-SAT, Tableau and POSIT) on a large sample of hard random 3-SAT problems and the recent Beijing SAT benchmarks. The good performance of *Satz* on the structured or real-world SAT problems shows that UP heuristic can tackle new problems or problem domains where Mom's heuristics were not successful and enhances the belief that if a DPL procedure is efficient for random SAT problems, it should be also efficient for a lot of structured ones.

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

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

Acknowledgments

We thank Olivier Dubois, James M. Crawford and Jon W. Freeman for kindly providing us their DPL procedures and anonymous referees for their comments which helped improve this paper.

References

[Chvatal and Szemeredi, 1988] V. Chvatal and E. Szemeredi. Many Hard Examples for Resolution. *Journal of ACM*, 35(4):759–768, October 1988.

[Cook, 1971] S. A. Cook. The Complexity of Theorem Proving Procedures. In *3rd ACM Symp. on Theory of Computing*, pages 151-158, Ohio, 1971.

[Crawford and Auton, 1996] J. M. Crawford and L. D. Auton. Experimental Results on the Crossover Point in Random 3-SAT. *Artificial Intelligence*, 81, 1996.

[Davis *et al.*, 1962] M. Davis, G. Logemann, and D. Loveland. *A machine program for theorem proving. Communication of ACM*, 5(7):394-397, July 1962.

[Dubois *et al.*, 1993] Olivier Dubois, P. Andre, Y. Boufkhad and Jacques Carlier. *SAT versus UNSAT. Second DIMACS Challenge: Cliques, Coloring and Satisfiability*, Rutgers University, NJ, 1993.

[Freeman, 1995] Jon W. Freeman. Improvements to Propositional Satisfiability Search Algorithms. Ph.D. thesis, Department of computer and Information science, Univ. of Pennsylvania, Philadelphia, PA, 1995.

[Hooker and Vinay, 1995] J. N. Hooker and V. Vinay. Branching Rules for Satisfiability. *Journal of Automated Reasoning*, 15:359-383, 1995.

[Jeroslow and Wang, 1990] R. Jeroslow and J. Wang. Solving Propositional Satisfiability Problems. *Annals of Mathematics and AI*, 1:167-187, 1990.

[Li, 1996] ChuMin LI. Exploiting Yet More the Power of Unit Clause Propagation to Solve 3-SAT Problem. In *ECAI'96 Workshop on Advances in Propositional Deduction*, pages 11-16, Budapest, Hungary, 1996.

[Mitchell *et al.*, 1992] D. Mitchell, B. Selman, H. Levesque. Hard and Easy Distributions of SAT Problems. In *AAAI'92*, pages 459–465, San Jose, CA, 1992.

[Pretolani, 1993] Daniele Pretolani. Satisfiability and hypergraphs. Ph.D. thesis, Dipartimento di Informatica, Università di Pisa, 1993.