# Dynamic Variable Filtering for Hard Random 3-SAT Problems

Anbulagan, John Thornton, and Abdul Sattar

School of Information Technology
Gold Coast Campus, Griffith University
PMB 50 Gold Coast Mail Centre, QLD 9726, Australia
{a.anbulagan, j.thornton, a.sattar}@griffith.edu.au
Telephone: (07) 5552 8730, Fax: (07) 5552 8066
**Keywords:** Constraints, Problem Solving, Search

**Abstract.** The DPL (Davis-Putnam-Logemann-Loveland) procedure is one of the most effective methods for solving SAT problems. It is well known that its efficiency depends on the choice of the branching rule. One of the main improvements of this decision procedure has been the development of better branching variable selection through the use of unit propagation look-ahead (UPLA) heuristics (e.g., [13]). UPLA heuristics perform one of the following actions during two propagations of a free variable at each search tree node: detecting a contradiction earlier, simplifying the formula, or weighing the branching variable candidates. UPLA heuristics can be considered as polynomial time reasoning techniques. In this paper, we propose a new branching rule which does more reasoning to narrow the search tree of hard random 3-SAT problems. We term this reasoning technique the *dynamic variable filtering heuristic*. In our empirical study we develop four different variants of the DPL procedure: two ($ssc34$ and $ssc355$) based on this new heuristic and another two ($Satz215$-0 and $Satz215sT$) based on static variable filtering heuristics. $ssc355$ additionally integrates the Neighborhood Variable Ordering (NVO) heuristic into $ssc34$. We then compare the best known versions of the state-of-the-art $Satz$ DPL procedure ($Satz215$), with each of our four procedures. Our results suggest that improved branching rules can further enhance the performance of DPL procedures. On hard random 3-SAT problems, our best procedure ($ssc355$) outperforms $Satz215$ with an order of magnitude in terms of the number of branching nodes in the search tree. While the run-times for dynamic variable filtering are still uncompetitive with $Satz215$, we have yet to explore the benefits that can be gained from avoiding redundant propagations and we still can improve the performance of the NVO heuristic. A further interesting property of dynamic filtering is that all backtracking can be eliminated during the DPL unit rule process. This property can also be explored in our future work for improving DPL procedure efficiency.

# Dynamic Variable Filtering for Hard Random 3-SAT Problems

**Abstract.** The DPL (Davis-Putnam-Logemann-Loveland) procedure is one of the most effective methods for solving SAT problems. It is well known that its efficiency depends on the choice of the branching rule. One of the main improvements of this decision procedure has been the development of better branching variable selection through the use of unit propagation look-ahead (UPLA) heuristics (e.g., [13]). UPLA heuristics perform one of the following actions during two propagations of a free variable at each search tree node: detecting a contradiction earlier, simplifying the formula, or weighing the branching variable candidates. UPLA heuristics can be considered as polynomial time reasoning techniques. In this paper, we propose a new branching rule which does more reasoning to narrow the search tree of hard random 3-SAT problems. We term this reasoning technique the *dynamic variable filtering heuristic*. In our empirical study we develop four different variants of the DPL procedure: two ($ssc34$ and $ssc355$) based on this new heuristic and another two ($Satz215$-0 and $Satz215sT$) based on static variable filtering heuristics. $ssc355$ additionally integrates the Neighborhood Variable Ordering (NVO) heuristic into $ssc34$. We then compare the best known versions of the state-of-the-art $Satz$ DPL procedure ($Satz215$), with each of our four procedures. Our results suggest that improved branching rules can further enhance the performance of DPL procedures. On hard random 3-SAT problems, our best procedure ($ssc355$) outperforms $Satz215$ with an order of magnitude in terms of the number of branching nodes in the search tree. While the run-times for dynamic variable filtering are still uncompetitive with $Satz215$, we have yet to explore the benefits that can be gained from avoiding redundant propagations and we still can improve the performance of the NVO heuristic. A further interesting property of dynamic filtering is that all backtracking can be eliminated during the DPL unit rule process. This property can also be explored in our future work for improving DPL procedure efficiency.

## 1 Introduction

The satisfiability (SAT) problem is central in mathematical logic, artificial intelligence and other fields of computer science and engineering. In conjunctive normal form (CNF), a SAT problem can be represented as a propositional formula $\mathcal{F}$ on a set of Boolean variables $\{x_1, x_2, ..., x_n\}$. A literal $l$ is then a variable $x_i$ or its negated form $\bar{x_i}$, and a clause $c_i$ is a logical *or* of some literals such as $x_1 \vee x_2 \vee \bar{x_3}$. A propositional formula $\mathcal{F}$ consists of a logical *and* of several clauses, such as $c_1 \wedge c_2 \wedge \ldots \wedge c_m$, and is often simply written as a set $\{c_1, c_2, ..., c_m\}$ of clauses.

Given $\mathcal{F}$, the SAT problem involves testing whether all the clauses in $\mathcal{F}$ can be satisfied by some consistent assignment of truth values $\{true, false\}$ to the variables. If this is the case, $\mathcal{F}$ is satisfiable; otherwise it is unsatisfiable. The SAT problem, as the original of all NP-Complete problems [4], is fundamentally important to the study of computational aspects of decision procedures. When each clause in $\mathcal{F}$ contains exactly $k$ literals, the restricted SAT problem is called $k$-SAT. 3-SAT is the smallest NP-Complete sub-problem of SAT.

One of the best known and most widely used algorithms to solve SAT problems is the DPL (Davis-Putnam-Logemann-Loveland) procedure [6]. Many SAT solvers such as Posit [8], Tableau [5], $Satz$ [13], $Satz214$ [15] and $cnfs$ [7] are based on this procedure. DPL essentially enumerates all possible solutions to a given SAT problem by setting up a binary search tree and proceeding until it either finds a satisfying truth assignment or concludes that no such assignment exists. It is well known that the search tree size of a SAT problem is generally an exponential function of the problem size, and that the branching variable selected by a branching rule at a node is crucial for determining the size of the sub-tree rooted at that node. A wrong choice may cause an exponential increase of the sub-tree size. Hence, the actual performance of a DPL procedure depends significantly on the effectiveness of the branching rule used.

Much of the research on DPL has focussed on finding clever branching rules to select the branching variable that most effectively reduces the search space. In this paper, we too propose a new branching rule based on a *dynamic variable filtering heuristic* as a polynomial time reasoning technique aimed at significantly narrowing the search tree for solving hard random 3-SAT problems. The key idea underlying this new branching rule is to further detect failed literals that would remain undiscovered using a unit propagation look-ahead (UPLA) branching rule, before choosing a branching variable. In other words, we perform more reasoning in the open space between the UPLA heuristic and the MOMS (Maximum Occurrences in clause of Minimum Size) heuristic in the actual DPL branching rule. To test this idea, we use $Satz215$ (the best version of the $Satz$ DPL procedure) where we simply replace its branching rule by a new branching rule. The new rule allows filtering of free variables, and at the same time reduces the sub-problem size at each node until the filtering process is saturated.

We develop two DPL procedures that use a dynamic variable filtering heuristic, and two other DPL procedures that use a static filtering heuristic. We then analyse these four DPL procedures with respect to $Satz215$, using a large sample of hard random 3-SAT problems.

An empirical study of the proposed variants of DPL indicates significant performance improvements can be obtained, with the two dynamic filtering heuristics consistently outperforming $Satz215$ in terms of mean search tree size when solving hard random 3-SAT problems. In some tests, the best dynamic filtering procedure was able to reduce the search tree size by an order of magnitude over $Satz215$, and for the 300 variable problems (1000 problems solved), the best dynamic filtering procedure produced a mean search tree size of 2679 nodes, in contrast to $Satz215$'s mean size of 6634 nodes. Not surprisingly, given the ad-

ditional reasoning involved in the branching rule, the new heuristics proved less competitive in terms of run-times, with $Satz215$ running approximately twice as fast on the largest (300 variable) problems.

Li and Gérard [16] discussed the hardness of proving an unsatisfiable hard random 3-SAT problem with 700 variables, and empirically calculated the approximately optimal number of branching nodes. They conjectured that obtaining this optimal sized tree was not possible using a branching heuristic. However, our results indicate that a dynamic variable filtering heuristic can achieve an optimal number of branching nodes. Therefore our work shows, in principle, that optimal branching can be achieved. The second major issue is whether optimal branching can be achieved efficiently. At present our results show $Satz215$ still performs significantly better than the dynamic filtering heuristics in terms of run-time. However we have not explored the potential gains that could result from avoiding redundant UPLA propagations. This we leave to future work.

The paper is organized as follows: In the next section we present the $Satz$ DPL procedure, one of the best known SAT procedures for solving hard random 3-SAT problems and structured SAT problems. In the subsequent section, we present our new dynamic filtering branching rules which perform additional reasoning to find a best branching variable. In section 4, we present a comparison of results for our new DPL procedures with $Satz215$ on a set of hard random 3-SAT problems. Finally, we conclude the paper with some remarks on current and future research.

## 2 The *Satz* Solver

$Satz$ [13, 14] is one of the best DPL procedures developed for solving both hard random SAT problems and structured SAT problems. Its powerful branching rule allows it to select the best branching variable for generating more and stronger constraints. It is a well-structured program and allows integration of new ideas easily. In 1999, Li further improved $Satz$ to produce $Satz214$ [15], and in 2001, Daniel Le Berre suggested the further detection of implied literals within $Satz214$, resulting in the latest and best version of $Satz$, $Satz215$ [12].

**Definition:** Let $PROP$ be a binary predicate such that $PROP(x, i)$ is true iff $x$ is a variable that occurs both positively and negatively in binary clauses and occurs in at least $i$ binary clauses in $\mathcal{F}$, and let $\mathcal{T}$ be an integer, then $PROP_z(x)$ is defined to be the first of the three predicates $PROP(x, 4)$, $PROP(x, 3)$, $true$ (in this order) whose denotational semantics contains more than $\mathcal{T}$ variables.

In figure 1, we present $Satz$'s branching rule which integrates the UPLA heuristic. The unary predicate $PROP_z$ is used to restrict the number of free variables executed by the heuristic and the parameter $\mathcal{T}$ is empirically fixed to 10. This UPLA heuristic plays a crucial role in the $Satz\text{-}family$ and is used to reach dead-ends earlier with the aim of minimising the length of the current path in the search tree. We distinguish the UPLA heuristic from the conventional unit propagation procedure (UP) that is usually used in DPL as follows: UP is

executed to reduce the size of a sub-problem possessing unit clauses *after* a branching variable selected, while UPLA is integrated in the branching rule and is executed at each search tree node.

Given a variable $x_i$, the UPLA heuristic examines $x_i$ by adding the two unit clauses possessing $x_i$ and $\bar{x}_i$ to $\mathcal{F}$ and independently making two unit propagations. These propagations result in a number of newly produced binary clauses, which are then used to weigh the variable $x_i$. This is calculated in figure 1, using the function $diff(\mathcal{F}_1, \mathcal{F}_2)$ which returns the number of new binary clauses in $\mathcal{F}_1$ that were not in $\mathcal{F}_2$. Let $w(x_i)$ be the number of new binary clauses produced by setting the variable to $true$, and $w(\bar{x}_i)$ be the number of new binary clauses produced by setting the variable to $false$. $Satz$ then uses a MOMS heuristic to branch on the variable $x_i$ such that $w(\bar{x}_i) * w(x_i) * 1024 + w(\bar{x}_i) + w(x_i)$ is the highest. The branching variable selected follows the two-sided Jeroslow-Wang (J-W) Rule [9] designed to balance the search tree.

The UPLA heuristic also allows the earlier detection of the so-called failed literals in $\mathcal{F}$. These are literals $l$ where $w(l)$ counts an empty clause. For such variables, $Satz$ immediately tries to satisfy $\bar{l}$. When there is a contradiction during the second unit propagation, $Satz$ will directly perform backtracking, else the size of the sub-problem is reduced allowing the selection of a set of best branching variable candidates at each node in search tree.

So, during two propagations of a free variable through the UPLA heuristic, three circumstances can occur:

- The free variable selected becomes a candidate for the branching variable.
- Only one contradiction is found during two unit propagations, meaning the size of formula $\mathcal{F}$ will be reduced during the other successful unit propagation process.
- Two contradictions are found during two unit propagations causing the search to backtrack to an earlier instantiation.

## 3 Using Variable Filtering to Narrow the Search Tree

The branching rules used in $Satz$ are powered by the UPLA heuristic. The main objective of using UPLA in $Satz$ is to detect contradictions earlier or to find a set of best branching variable candidates. In reality, $Satz$'s UPLA heuristic performs a series of variable filtering processes at each node as a static variable filtering agency. We therefore term $Satz$'s UPLA heuristic a Static Variable Filtering (SVF) heuristic, because it will only perform between one to three filtering processes at each node (depending on the evaluation of $PROP_z(x)$). During the filtering process, some variables are assigned the value $true$ or $false$ through a forced unit propagation when a contradiction occurs during another unit propagation. Note that UPLA examines a free variable by performing two unit propagations. This process will automatically reduce the size of sub-problem and collects the (almost) best branching variable candidates at each node of the search tree. In the empirical studies presented in [13, 14] the $Satz$ solver using UPLA was shown to outperform a range of other UP heuristics based DPL

```
B := ∅;
For each free variable x_i, do
Begin
   let F' and F'' be two copies of F
   F' := UPLA(F' ∪ {x_i}); F'' := UPLA(F'' ∪ {x̄_i});
   If both F' and F'' contain an empty clause then backtrack();
   else if F' contains an empty clause then x_i := false; F := F'';
   else if F'' contains an empty clause then x_i := true; F := F';
   else
      B := B ∪ {x_i};
      w(x_i) := diff(F', F) and w(x̄_i) := diff(F'', F);
End;

For each variable x_i ∈ B, do M(x_i) := w(x̄_i) * w(x_i) * 1024 + w(x̄_i) + w(x_i);
Branch on the free variable x_i such that M(x_i) is the highest.
```

**Fig. 1.** The *Satz* Branching Rule

procedures. It was concluded that the superior performance of *Satz* was due to its greater use of variable filtering processes.

Our work is based on the insight that the size of a sub-problem during the variable filtering process can be further reduced in the *Satz-family* of DPL procedures. Here, we propose a new heuristic called the *Dynamic Variable Filtering (DVF) heuristic* that further filters the free variables and at the same time reduces the sub-problem size at each node until the filtering process is saturated. We illustrate the new branching rule powered by the DVF heuristic in figure 2.

We expect this new heuristic to perform better than the UPLA heuristics in terms of reducing the search tree size. To verify this, we carried out an empirical study and modified the branching rule of the DPL procedure *Satz*215[1] for our purpose. Four new DPL procedures based on the variable filtering heuristic are proposed. Two of them (*Satz*215-0 and *Satz*215*sT*) are based on the SVF heuristic, and the other two solvers (*ssc*34 and *ssc*355) are based on the DVF heuristic.

***Satz*215-0.** This is same as the *Satz*215 DPL procedure, except we examine all free variables using the SVF heuristic without restriction. This process is executed only once at each node.

***Satz*215*sT*.** This is same as the *Satz*215 DPL procedure, except we refuse to use the $T$ parameter in the branching rule. This DPL procedure performs at most three filtering processes for each variable at each node.

***ssc*34.** This is same as the *Satz*215 DPL procedure, except we replace the branching rule used in *Satz*215 with the DVF heuristic based branching rule. It

[1] available from http://www.laria.u-picardie.fr/∼cli/EnglishPage.html

```
Do
   F_init := F; B := ∅;
   For each free variable x_i, do
   Begin
      let F' and F'' be two copies of F
      F' := UPLA(F' ∪ {x_i}); F'' := UPLA(F'' ∪ {x̄_i});
      If both F' and F'' contain an empty clause then backtrack();
      else if F' contains an empty clause then x_i := false; F := F'';
      else if F'' contains an empty clause then x_i := true; F := F';
      else
         B := B ∪ {x_i};
         w(x_i) := diff(F', F) and w(x̄_i) := diff(F'', F);
   End;
Until (F = F_init);

For each variable x_i ∈ B, do M(x_i) := w(x̄_i) * w(x_i) * 1024 + w(x̄_i) + w(x_i);
Branch on the free variable x_i such that M(x_i) is the highest.
```

**Fig. 2.** The Dynamic Variable Filtering Branching Rule

performs the variable filtering process until the sub-problem cannot be further reduced at each node before a branching variable selected. In fact, *ssc*34 examines the free variables many times using the UPLA heuristic at each node. One might think that this saturation process is very costly, but it is not the case.

***ssc*355.** Since *ssc*34 examines all free variables many times using the UPLA heuristic at each node, we attempt to limit the number of free variables examined by only exploring the neighborhood variables of the current assigned variable. For this purpose, we create the *ssc*355 DPL procedure by integrating a simple Neighbourhood Variable Ordering (NVO) heuristic into *ssc*34. Bessière et. al. [2] proposed a formulation of a dynamic variable ordering heuristic in the CSP domain that takes into account the properties of the neighborhood of the variable. The main objective of our simple NVO heuristic in *ssc*355 is to restrict the number of variables examined by UPLA in the DVF heuristic.

## 4 Comparative Experimental Results

We compare the four DPL procedures we introduced in the previous section with *Satz*215 on a large sample of hard random 3-SAT problems generated by using the method of Mitchell et. al. [18]. Given a set $\mathcal{V}$ 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 $\mathcal{V}$ and negating each with probability 0.5. Empirically, when the ratio $m/n$ is near 4.25 for a 3-SAT problem $\mathcal{F}$, $\mathcal{F}$ is unsatisfiable with a probability 0.5 and is the most difficult to solve. We

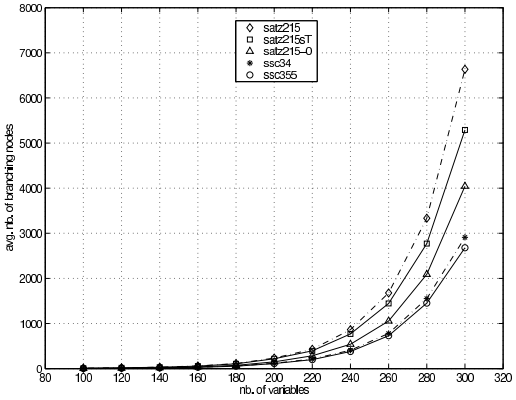


**Fig. 3.** Mean search tree size of each DPL procedure as a function of $n$ for hard random 3-SAT problems at the ratio $m/n$=4.25

vary $n$ from 100 variables to 300 variables incrementing by 20, at ratio $(m/n) = 4.25$, with 1000 problems solved at each point by all the five DPL procedures.

Figure 3 shows the performance of five DPL procedures on all problem instances, where the mean search tree size is computed from the number of branching nodes reported by each procedure. It illustrates that *ssc*355's mean search tree size is the smallest, e.g., for 300 variables, *ssc*355's search tree consists of 2679 nodes, in contrast to $Satz215$'s search tree of 6634 nodes.

Figure 4 also illustrates that *ssc*355's search tree size is smaller than the other DPL procedures on the *unsatisfiable* problem instances. For example, on the hard random unsatisfiable 3-SAT problems, with 300 variables, *ssc*355's mean search tree size consists of 4405 nodes, in contrast to $Satz215$'s mean search tree size of 10328 nodes. Also, the mean search tree size of *ssc*355 achieves the approximate optimal search tree size proposed by Li and Gérard in [16].

Three of the DPL procedures ($Satz215$, $Satz215$-0, and $Satz215sT$) integrate the SVF heuristic. The comparative results presented in figures 3 and 4 show that the mean search tree sizes of $Satz215$-0 and $Satz215sT$ are smaller than $Satz215$. This means that more reasoning at each node has reduced the search tree size.

Overall, in terms of search tree size, the DVF heuristics used in *ssc*34 and *ssc*355 outperform the other SVF heuristic-based DPL procedures. This better performance is explained because the DVF sub-problem at each node is explored

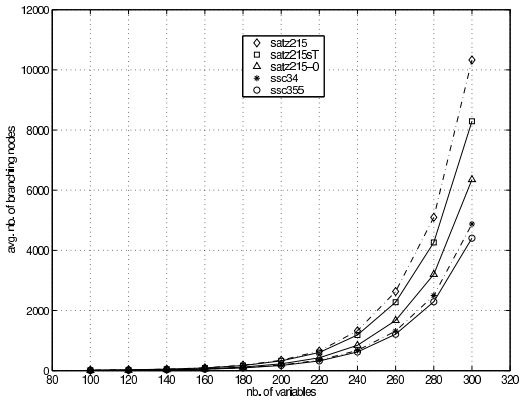


**Fig. 4.** Mean search tree size of each DPL procedure as a function of $n$ for hard random unsatisfiable 3-SAT problems at the ratio $m/n$=4.25

more fully to find a contradiction or to simply reduce the sub-problem size through a unit propagation before selecting a best branching variable.

The integration of a simple NVO heuristic on top of DVF in *ssc*355 yields a promising result compared to *ssc*34 (which only uses the DVF heuristic). In terms of search tree size, the gain of *ssc*355 over *ssc*34 increases with the size of the input problem.

Finally, figure 5 illustrates mean run-time of each DPL procedure on a Dual Xeon PC with a 2.4 GHz CPU. At present our results show that *Satz*215 still performs significantly better than the dynamic filtering heuristics in terms of run-time. However we have not explored the potential gains that could result from avoiding redundant UPLA propagations. A further improvement of the NVO heuristic also promises to ameliorate the run-time of *ssc*355 DPL procedure. This we leave to future work.

## 5 Related and Future Work

In addition to real world benchmark problems, hard random 3-SAT problems are frequently used for testing or comparing DPL procedures. Since hard random 3-SAT problems are difficult to solve in an acceptable time when the number of variables is greater than 500, a challenge to prove that a hard random 3-SAT problem with 700 variables is unsatisfiable, was put forward by Selman et. al. in IJCAI'97 [20].

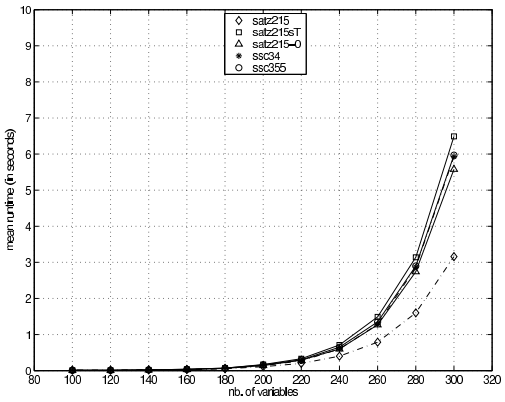


**Fig. 5.** Mean run-time of each DPL procedure as a function of $n$ for hard random 3-SAT problems at the ratio $m/n$=4.25

To answer the challenge, Li and Gérard [16] have studied the limit of branching rules in DPL procedures for solving hard random unsatisfiable 3-SAT problems. After running an empirical study for more than five months, they suggested in their paper that the current state-of-the-art DPL procedures are already close to optimal for solving hard random unsatisfiable 3-SAT problems and that the resolution of the challenge problem with 700 variables could not be proved by a branching rule based DPL procedure.

A final answer to the challenge was realised by Dubois and Dequen [7] with their $cnfs$ solver. This solver integrates a backbone search heuristic (introduced by Monasson et. al. [19] in 1999)into a DPL-based branching rule. Dubois and Dequen's implementation allows the DPL procedure to solve hard random unsatisfiable 3-SAT problems with 700 variables (12 problems solved) in a mean run-time of 26 days using an AMD Athlon PC running at 1 GHz under a Linux operating system. When solving hard random unsatisfiable 3-SAT problems (456 problems solved), $cnfs$'s mean search tree size consists of 12739 nodes, in contrast to $satz214$'s mean search tree size of 18480 nodes [7]. Although the $cnfs$ solver can solve hard random unsatisfiable 3-SAT problems with up to 700 variables, on the other hand its mean search tree size is still far from the approximate optimal mean search tree size calculated by Li and Gérard [16].

After reviewing the results presented in [7, 16], we decided the first step in our research would be to reduce the mean search tree size rather than looking for a more efficient solver. As the result, we have developed the Dynamic Variable

Filtering (DVF) heuristic, for reinforcing the branching rule of a DPL procedure. In effect, we further explore further the open space between the UPLA heuristic process and the MOMS heuristic process in $Satz215$. The results of our empirical study show that DVF heuristic can achieve an optimal number of branching nodes as presented in [16].

In the last stages of finishing this paper, we obtained the new more powerful version of $cnfs$, the $kcnfs$[2] solver. We ran $kcnfs$ on the same hard random unsatisfiable 3-SAT problems with 300 variables used in our earlier study. As a result, $kcnfs$'s mean search tree size came to 7418 nodes (compared to $ssc355$'s mean search tree size of 4405 nodes), although $kcnfs$ performs three times better than $ssc355$ in terms of run-time.

To further investigate the effectiveness of our DVF heuristics, we also compared our techniques with the $2clseq$ solver[1] and $OKsolver$ [11]. $2clseq$ integrates a UPLA heuristic, binary clause reasoning, using intelligent backtracking and a pruneback technique. Hence it performs more reasoning in each node of the search tree and proved very competitive with the other solvers during 2002 SAT competition for solving real-world problems. The second procedure, $OKsolver$, integrates an adaptive density-based heuristic in its branching rule. In 2002 SAT competition, $OKsolver$ won both categories for the random benchmarks (only satisfiable and all problems).

We compared $ssc355$ with $2clseq$ and $OKsolver$ on our hard random 3-SAT problems with 300 variables. The preliminary results of this comparison show that $ssc355$ still performs significantly better than both solvers in terms of run-time and search tree size. When solving the hard random satisfiable 3-SAT problem $v300c1275g4$, the run-times of $2clseq$ and $OKsolver$ are 34.78 and 2.25 seconds with search tree sizes of 683 and 2,612 nodes respectively. In contrast, $ssc355$ solves the problem in 0.02 seconds with a search tree size of 48 nodes. When running the hard random unsatisfiable 3-SAT problem $v350c1488g255$, $2clseq$ had no result after 6000 seconds, and $OKsolver$ solved the problem in 438 seconds with a search tree size of 275,159 nodes. Again $ssc355$ strongly outperformed these techniques, solving the problem in 190 seconds with a search tree of 65,784 nodes.

The work presented in this paper is a first attempt at building an efficient SAT solver than can approach an optimal branching rule. In principle, our results show DVF can obtain optimal results - hence, if the efficiency issues can be addressed, DVF could prove to be a better heuristic than backbone search. In our future work, we envisage at least three further improvements of our current approach. Firstly, it is clear that savings can be made by avoiding redundant unit propagation searches for variables that remain unchanged between iterations of UPLA. The kind of benefits that we hope to obtain have already been exploited in arc-consistency algorithms. Secondly, further improvements of the NVO heuristic appear promising, as our first implementation is fairly simplistic. Finally, we are also looking at integrating a backjumping technique into DVF, exploiting the feature that backtracking only occurs during the UPLA process

[2] available from http://www.laria.u-picardie.fr/∼dequen/sat/

in DVF, compared to $Satz215$, where backtracking can also occur during the UP process.

## 6 Conclusion

In this paper we have proposed a new heuristic, Dynamic Variable Filtering (DVF), for improving the performance of DPL-based procedures. In terms of search tree size, our preliminary results already show that the DVF heuristic outperforms some of the best solvers in the area, while still remaining reasonable efficient in terms of search time.

Our evidence further suggests that the branching rules integrated in DPL procedures can obtain predicted optimal performance by using a DVF heuristic. Finally, we anticipate that the efficiency of DVF can be improved by eliminating the redundant unit propagation, improving the NVO heuristic and integrating a backjumping technique.

## References


1. Bacchus, F. *Enhancing Davis Putnam with Extended Binary Clause Reasoning.* In Proceedings of AAAI Conference, 2002, USA, pp. 613-619.
2. Bessière, C., Chmeiss, A., and Sais, L. *Neighborhood-based Variable Ordering Heuristics for the Constraint Satisfaction Problem.* In Proceedings of Seventh International Conference on Principles and Practice of Constraint Programming, 2001, Paphos, Cyprus, pp. 565-569.
3. Buro M., Kleine-Buning, H. *Report on a SAT Competition.* Technical Report, University of Paderborn, November 1992.
4. Cook, S. A. *The Complexity of Theorem Proving Procedures.* In Proceedings of 3rd ACM Symp. on Theory of Computing, Ohio, 1971, pp. 151-158.
5. Crawford, J. M., and Auton, L. D. Experimental Results on the Crossover Point in Random 3SAT. *Artificial Intelligence Journal*, 1996, Vol. 81, no. 1-2.
6. Davis, M., Logemann, G. and Loveland, D. *A Machine Program for Theorem Proving.* Communication of ACM 5 (1962), pp. 394-397.
7. Dubois, O., and Dequen, G. *A Backbone-search Heuristic for Efficient Solving of Hard 3-SAT Formulae.* In Proceedings of 17th International Joint Conference on Artificial Intelligence, 2001, Seattle, Washington, USA.
8. Freeman, J. W. *Improvements to Propositional Satisfiability Search Algorithms.* Ph.D. Dissertation, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA, 1995.
9. Hooker, J. N., Vinay, V. *Branching Rules for Satisfiability.* Journal of Automated Reasoning, 15:359-383, 1995.
10. Jeroslow, R., Wang, J. *Solving Propositional Satisfiability Problems.* Annals of Mathematics and AI 1 (1990), pp. 167-187.
11. Kullmann, O. *Investigating the behaviour of a SAT solver on random formulas.* Submitted to Annals of Mathematics and AI, 2002.
12. Le Berre, D. *Exploiting the Real Power of Unit Propagation Lookahead.* In Proceedings of Workshop on the Theory and Applications of Satisfiability Testing, 2001, Boston University, MA, USA.

13. Li, C. M., and Anbulagan. *Heuristics Based on Unit Propagation for Satisfiability Problems.* In Proceedings of 15th International Joint Conference on Artificial Intelligence, 1997, Nagoya, Aichi, Japan, pp. 366-371.
14. Li, C. M., and Anbulagan. *Look-Ahead Versus Look-Back for Satisfiability Problems.* In Proceedings of Third International Conference on Principles and Practice of Constraint Programming, 1997, Schloss Hagenberg, Austria, pp. 341-355.
15. Li, C. M. *A Constraint-based Approach to Narrow Search Trees for Satisfiability.* Information Processing Letters, 71, 1999, pp. 75-80.
16. Li, C. M., and Gérard, S. *On the Limit of Branching Rules for Hard Random Unsatisfiable 3-SAT.* In Proceedings of 14th European Conference on Artificial Intelligence, 2000, Berlin, Germany.
17. Marques-Silva, J. P. *The Impact of Branching Heuristics in Propositional Satisfiability Algorithms.* In Proceedings of 9th Portuguese Conference on Artificial Intelligence, 1999.
18. Mitchell, D., Selman, B., and Levesque, H. *Hard and Easy Distributions of SAT Problems.* In Proceedings of 10th AAAI Conference, 1992, San Jose, CA, pp. 459-465.
19. Monasson, R., Zecchina, R., Kirkpatrick, S., Selman, B., and Troyansky, L. *Determining Computational Complexity for Characteristic 'Phase Transitions'.* Nature, 400, 1999, pp. 133-137.
20. Selman, B., Kautz, H., and McAllester, D. *Ten Challenges in Propositional Reasoning and Search.* In Proceedings of 15th International Joint Conference on Artificial Intelligence, 1997, Nagoya, Aichi, Japan.
21. Warners, J. P., van Maaren, H. *Solving Satisfiability Using Elliptic Approximations - Effective Branching Rules.* Discrete Applied Mathematics, 107, 2000, pp. 241-259.