DIMACS Technical Report 96-38
September 1996

How good are branching rules in DPLL?

by

Ming Ouyang
Department of Computer Science
Rutgers University
New Brunswick, New Jersey 08903

DIMACS is a partnership of Rutgers University, Princeton University, AT&T Research, Bellcore, and Bell Laboratories.

DIMACS is an NSF Science and Technology Center, funded under contract STC–91–19999; and also receives support from the New Jersey Commission on Science and Technology.

ABSTRACT

The Davis-Putnam-Logemann-Loveland algorithm is one of the most popular algorithms for solving the satisfiability problem. Its efficiency depends on its choice of a branching rule. We construct a sequence of instances of the satisfiability problem that fools a variety of "sensible" branching rules in the following sense: when the instance has n variables, each of the "sensible" branching rules brings about $\Omega(2^{n/5})$ recursive calls of the Davis-Putnam-Logemann-Loveland algorithm, even though only $O(1)$ such calls are necessary.

1 The SAT problem

A *truth assignment* is a mapping f that assigns 0 or 1 to each variable in its domain; we shall enumerate all the variables in this domain as $x_1, \ldots, x_n$. The *complement* $\overline{x}_i$ of each such variable x_i is defined by $f(\overline{x}_i) = 1 - f(x_i)$ for all truth assignments f; both x_i and $\overline{x}_i$ are called *literals*; if $u = \overline{x}_i$ then $\overline{u} = x_i$. A *clause* is a set of (distinct) literals and a *formula* is a family of (not necessarily distinct) clauses. A truth assignment *satisfies* a clause if it maps at least one of its literals to 1; the assignment *satisfies* a formula if and only if it satisfies each of its clauses. A formula is called *satisfiable* if it is satisfied by at least one truth assignment; otherwise it is called *unsatisfiable*. The problem of recognizing satisfiable formulas is known as the *satisfiability problem*, or SAT for short.

2 The Davis-Putnam-Logemann-Loveland algorithm

Given a formula $\mathcal{F}$ and a literal u in $\mathcal{F}$, we let $\mathcal{F}|u$ denote the "residual formula" arising from $\mathcal{F}$ when $f(u)$ is set at 1: explicitly, this formula is obtained from $\mathcal{F}$ by

- removing all the clauses that contain u,
- deleting $\overline{u}$ from all the clauses that contain $\overline{u}$,
- removing both u and $\overline{u}$ from the list of literals.

Trivially, $\mathcal{F}$ is satisfiable if and only if at least one of $\mathcal{F}|u$ and $\mathcal{F}|\overline{u}$ is satisfiable.

It is customary to refer to the number of literals in a clause as the *length* (rather than size) of this clause. Clauses of lentgh one are called *unit clauses*. Trivially, if a formula $\mathcal{F}$ includes a unit clause $\{u\}$, then every truth assignment f that satisfies $\mathcal{F}$ must have $f(u) = 1$. Hence $\mathcal{F}$ is satisfiable if and only if $\mathcal{F}|u$ is satisfiable.

A literal u in a formula $\mathcal{F}$ is called *monotone* if $\overline{u}$ appears in no clause of $\mathcal{F}$. Trivially, if u is a monotone literal and if $\mathcal{F}$ is satisfiable, then $\mathcal{F}$ is satisfied by a truth assignment f such that $f(u) = 1$. Hence $\mathcal{F}$ is satisfiable if and only if $\mathcal{F}|u$ is satisfiable.

These observations are the backbone of an algorithm for solving SAT, designed by Davis, Logemann, and Loveland [1] and evolved from an earlier proposal by Davis and Putnam [2]: see Figure 1.

3 Branching rules

Each recursive call of DPLL may involve a choice of a literal u; algorithms for making these choices are referred to as *branching rules*. It is customary to represent each call of `DPLL(`$\mathcal{F}$`)` by a node of a full binary tree: if a call represented by a node γ brings about calls of `DPLL(`$\mathcal{F}|u$`)` and `DPLL(`$\mathcal{F}|\overline{u}$`)`, then the call of `DPLL(`$\mathcal{F}|u$`)` is represented by the left child of γ and the call of `DPLL(`$\mathcal{F}|\overline{u}$`)` is represented by the right child of γ, else γ is a leaf. The shape and size of this tree depends not only on the input formula $\mathcal{F}$, but also on the branching rule B that is used; we shall let $T(\mathcal{F}, B)$ denote the tree.

```
DPLL(𝓕)
{
   while (𝓕 includes a clause of length at most one) {
      if (𝓕 includes an empty clause) return UNSATISFIABLE;
      {v} = any clause of length one;
      𝓕 = 𝓕|v;
   }

   while (there is a monotone literal) {
      v = any monotone literal;
      𝓕 = 𝓕|v;
   }

   if (𝓕 is empty) return SATISFIABLE;

   choose a literal u in 𝓕;

   if (DPLL(𝓕|u)=SATISFIABLE) return SATISFIABLE;
   if (DPLL(𝓕|ū)=SATISFIABLE) return SATISFIABLE;
   return UNSATISFIABLE;
}
```

Figure 1: Definition of the Davis-Putnam-Logemann-Loveland algorithm.

To see how dramatically a choice of B can influence the size of $T(\mathcal{F}, B)$, take the formula with variables $x_1, x_2, \ldots, x_n$ and clauses

$$\{x_i, x_{n-1}, x_n\},\ \{x_i, \overline{x}_{n-1}, x_n\},\ \{x_i, x_{n-1}, \overline{x}_n\},\ \{x_i, \overline{x}_{n-1}, \overline{x}_n\} \quad (i = 1, 2, \ldots, n-2),$$

$$\{\overline{x}_j, \overline{x}_{j+1}, \overline{x}_{j+2}, \ldots, \overline{x}_{n-3}, \overline{x}_{n-2}\} \quad (j = 1, 2, \ldots, n-3).$$

It is easy to see that this formula, $\mathcal{G}$, is unsatisfiable and that, with branching rules

MIN: "let u be the variable with the smallest subscript", and

MAX: "let u be the variable with the largest subscript",

$|T(\mathcal{G}, \mathbf{MIN})| = 2^{n-1} - 1$ but $|T(\mathcal{G}, \mathbf{MAX})| = 7$. However, one might object that, unlike the branching rules commonly used in DPLL, both **MIN** and **MAX** are artificial: they disregard the structure of the (residual) formula from which u is to be chosen.

One branching rule that does *not* disregard the structure of the formula from which u is to be chosen has been proposed by Jeroslow and Wang [7]; it goes as follows. Let $d_k(\mathcal{F}, u)$ be the number of clauses of length k in $\mathcal{F}$ which contain u. The Jeroslow-Wang branching rule (**JW**) associates a weight with each literal u,

$$w(\mathcal{F},u) = \sum_k 2^{-k} d_k(\mathcal{F},u),$$

and chooses the literal with the largest weight. (If there is a tie, then, among the literals with the largest weight, the literal with the smallest subscript is chosen.) In our example,

$$w(\mathcal{G},x_i) = \begin{cases} 1/2 & \text{if } i = 1,\ldots,n-2, \\ (n-2)/4 & \text{if } i = n-1 \text{ or } n, \end{cases}$$

$$w(\mathcal{G},\overline{x}_i) = \begin{cases} (2^i-1)/2^{n-2} & \text{if } i = 1,\ldots,n-3, \\ (2^{n-3}-1)/2^{n-2} & \text{if } i = n-2, \\ (n-2)/4 & \text{if } i = n-1 \text{ or } n. \end{cases}$$

Therefore **JW** chooses x_{n-1}. The residual formulas $\mathcal{G}|x_{n-1}$ and $\mathcal{G}|\overline{x}_{n-1}$ are identical; their clauses are

$$\{x_i,x_n\},\ \{x_i,\overline{x}_n\} \quad (i = 1,2,\ldots,n-2),$$
$$\{\overline{x}_j,\overline{x}_{j+1},\overline{x}_{j+2},\ldots,\overline{x}_{n-3},\overline{x}_{n-2}\} \quad (j = 1,2,\ldots,n-3).$$

Therefore **JW** chooses x_n for both $\mathcal{G}|x_{n-1}$ and $\mathcal{G}|\overline{x}_{n-1}$. Since all of the resulting four residual formulas contain the empty clause, $|T(\mathcal{G},\mathbf{JW})| = 7$.

This branching rule is based on the intuition that shorter clauses are more important than longer ones and, specifically, that clauses of length k are twice as important as clauses of length $k+1$. (The idea of progressively halving the weighting factors was used by Johnson [8] some fifteen years earlier in an approximation algorithm for MAX-SAT.)

Hooker and Vinay [6] proposed a variation (**HV**) on **JW**, which they called "two-sided Jeroslow-Wang rule": among all the literals u such that $w(\mathcal{F},u) \geq w(\mathcal{F},\overline{u})$, choose one that maximizes $w(\mathcal{F},u) + w(\mathcal{F},\overline{u})$.

Van Gelder and Tsuji [10] proposed another variation (**vGT**): among all the literals u such that $w(\mathcal{F},u) \geq w(\mathcal{F},\overline{u})$, choose one that maximizes $w(\mathcal{F},u) * w(\mathcal{F},\overline{u})$.

Dubois, Andre, Boufkhad, and Carlier [5] developed a program called C-SAT, which was shown to be one of the most efficient programs in solving the formulas from the DIMACS benchmarks [3]. Their branching rule (**C-SAT**) is as follows. Define

$$w(\mathcal{F},u) = \sum_k \ln\left(1 + \frac{1}{4^k - 2^{k+1}}\right) d_k(\mathcal{F},u)$$

and

$$W(\mathcal{F},u) = w(\mathcal{F},u) + \sum_{\{u,v\}\in\mathcal{F}} w(\mathcal{F},\overline{v});$$

among all the literals u such that $W(\mathcal{F},u) \geq W(\mathcal{F},\overline{u})$, choose one that maximizes

$$W(\mathcal{F},u) + W(\mathcal{F},\overline{u}) + 1.5\min(W(\mathcal{F},u), W(\mathcal{F},\overline{u})).$$

These four branching rules, **JW**, **HV**, **vGT**, and **C-SAT**, share the following property.

Sensible Property: If all clauses have length three and if v,w are literals such that $d_3(v) < d_3(w)$, $d_3(\overline{v}) < d_3(\overline{w})$, then do *not* choose v.

4 Fooling a family of branching rules

Theorem. *For every nonnegative integer t, there is an unsatisfiable formula $\mathcal{H}$ with $5t+21$ variables such that $|T(\mathcal{H},\mathbf{MAX})| = 111$, but $|T(\mathcal{H},\mathbf{B})| > 2^t$ for every branching rule $\mathbf{B}$ with the Sensible Property.*

Proof. Write $r = 5t$ and consider the following formula, $\mathcal{H}_t$, with variables $x_1,\ldots,x_r$, $\ldots,x_{r+21}$. For each $s = 0,1,\ldots,t-1$, there are eight clauses involving x_{5s+1}, x_{5s+2}, x_{5s+3}, x_{5s+4}, x_{5s+5} and their complements,

$$\{x_{5s+1},x_{5s+2},x_{5s+3}\},\ \{x_{5s+1},x_{5s+3},x_{5s+4}\},\ \{x_{5s+1},x_{5s+4},x_{5s+5}\},\ \{x_{5s+1},x_{5s+2},x_{5s+5}\}$$

and

$$\{\overline{x}_{5s+1},\overline{x}_{5s+2},\overline{x}_{5s+3}\},\ \{\overline{x}_{5s+1},\overline{x}_{5s+3},\overline{x}_{5s+4}\},\ \{\overline{x}_{5s+1},\overline{x}_{5s+4},\overline{x}_{5s+5}\},\ \{\overline{x}_{5s+1},\overline{x}_{5s+2},\overline{x}_{5s+5}\};$$

in addition, there are 22 clauses involving $x_{r+1},\ldots,x_{r+21}$ and their complements,

$$\{x_{r+1},x_{r+7},x_{r+13}\},\ \{x_{r+2},x_{r+8},x_{r+14}\},\ \{x_{r+3},x_{r+9},x_{r+15}\},\ \{x_{r+4},x_{r+10},x_{r+16}\},$$

$$\{x_{r+5},x_{r+11},x_{r+17}\},\ \{x_{r+6},x_{r+12},x_{r+18}\},\ \{\overline{x}_{r+1},x_{r+7},x_{r+13}\},\ \{\overline{x}_{r+2},x_{r+8},x_{r+14}\},$$

$$\{\overline{x}_{r+3},x_{r+9},x_{r+15}\},\ \{\overline{x}_{r+4},x_{r+10},x_{r+16}\},\ \{\overline{x}_{r+5},x_{r+11},x_{r+17}\},\ \{\overline{x}_{r+6},x_{r+12},x_{r+18}\},$$

$$\{\overline{x}_{r+7},x_{r+13},x_{r+19}\},\ \{\overline{x}_{r+8},x_{r+14},x_{r+19}\},\ \{\overline{x}_{r+9},x_{r+15},x_{r+20}\},\ \{\overline{x}_{r+10},x_{r+16},x_{r+20}\},$$

$$\{\overline{x}_{r+11},x_{r+17},x_{r+21}\},\ \{\overline{x}_{r+12},x_{r+18},x_{r+21}\},\ \{\overline{x}_{r+13},\overline{x}_{r+14},x_{r+19}\},\ \{\overline{x}_{r+15},\overline{x}_{r+16},x_{r+20}\},$$

$$\{\overline{x}_{r+17},\overline{x}_{r+18},x_{r+21}\},\ \{\overline{x}_{r+19},\overline{x}_{r+20},\overline{x}_{r+21}\}.$$

These last 22 clauses alone constitute an unsatisfiable formula ([4], bottom of page 56). Altogether, $\mathcal{H}_t$ has $8t+22$ clauses and each of these clauses has length three. It is a routine matter to verify that $|T(\mathcal{H}_t,\mathbf{MAX})| = 111$; we will use induction on t to prove that $|T(\mathcal{H}_t,\mathbf{B})| > 2^t$ for every branching rule $\mathbf{B}$ with the Sensible Property.

Trivially, $|T(\mathcal{H}_0,\mathbf{B})| > 1$. Since

$$d_3(\mathcal{H}_t,x_i) = \begin{cases} 4 & \text{if } i = 5s+1,\ s = 0,\ldots,t-1,\\ 2 & \text{if } i = 5s+j,\ s = 0,\ldots,t-1,\ j = 2,\ldots,5,\\ 1 & \text{if } i = r+1,\ldots,r+6,\\ 2 & \text{if } i = r+7,\ldots,r+12,\\ 3 & \text{if } i = r+13,\ldots,r+21,\end{cases}$$

$$d_3(\mathcal{H}_t,\overline{x}_i) = \begin{cases} 4 & \text{if } i = 5s+1,\ s = 0,\ldots,t-1,\\ 2 & \text{if } i = 5s+j,\ s = 0,\ldots,t-1,\ j = 2,\ldots,5,\\ 1 & \text{if } i = r+1,\ldots,r+21,\end{cases}$$

any branching rule $\mathbf{B}$ with the Sensible Property, given $\mathcal{H}_t$ with $t > 0$, chooses some x_{5s+1} with $0 \le s < t$ to be u. In $\mathcal{H}_t|x_{5s+1}$, the four literals $\overline{x}_{5s+2}$, $\overline{x}_{5s+3}$, $\overline{x}_{5s+4}$, $\overline{x}_{5s+5}$ are monotone;

in $\mathcal{H}_t|\overline{x}_{5s+1}$, the four literals x_{5s+2}, x_{5s+3}, x_{5s+4}, x_{5s+5} are monotone. These monotone literals (together with the clauses containing them) are removed from the formulas by DPLL before the branching rule is consulted again. The resulting formulas,

$$((((\mathcal{H}_t|x_{5s+1})|\overline{x}_{5s+2})|\overline{x}_{5s+3})|\overline{x}_{5s+4})|\overline{x}_{5s+5} \quad \text{and} \quad ((((\mathcal{H}_t|\overline{x}_{5s+1})|x_{5s+2})|x_{5s+3})|x_{5s+4})|x_{5s+5},$$

are identical and isomorphic to $\mathcal{H}_{t-1}$ via σ defined by

$$\sigma(x_i) = \begin{cases} x_i & \text{if } i \leq 5s, \\ x_{i-5} & \text{if } i > 5s+5. \end{cases}$$

Hence $|T(\mathcal{H}_t, \mathbf{B})| = 1 + |T(\mathcal{H}_t|x_{5s+1}, \mathbf{B})| + |T(\mathcal{H}_t|\overline{x}_{5s+1}, \mathbf{B})| = 1 + 2 * |T(\mathcal{H}_{t-1}, \mathbf{B})|$; now, by the induction hypothesis, $|T(\mathcal{H}_t, \mathbf{B})| > 1 + 2 * (2^{t-1}) > 2^t$. □

Acknowledgment

I wish to thank my adviser, Vašek Chvátal, for raising the question of the existence of formulas that fool branching rules, and his encouragement and generous contribution to this work. His careful reading and comments on an earlier version helped me improve the presentation.

References

[1] M. Davis, G. Logemann, and D. Loveland, "A Machine Program for Theorem-Proving", *C. ACM* **5** (1962), 394–397.

[2] M. Davis and H. Putnam, "A computing procedure for quantification theory", *J. ACM* **7** (1960), 201–215.

[3] http://dimacs.rutgers.edu/Challenges/index.html

[4] O. Dubois, "On the r, s-SAT Satisfiability Problem and a Conjecture of Tovey", *Discrete Applied Mathematics* **26** (1990), 51–60.

[5] O. Dubois, P. Andre, Y. Boufkhad, and J. Carlier, "SAT versus UNSAT", *American Mathematical Society* DIMACS Volume Series **26**, to appear.

[6] J. N. Hooker and V. Vinay, "Branching rules for satisfiability", *J. of Automated Reasoning* **15** (1995), 359–383.

[7] R. G. Jeroslow and J. Wang, "Solving Propositional Satisfiability Problems", *Annals of Mathematics and Artificial Intelligence* **1** (1990), 167–187.

[8] D. S. Johnson, "Approximation Algorithms for Combinatorial Problems", *J. Computer and System Sciences* **9** (1974), 256–278.

[9] D. W. Loveland, *Automated Theorem Proving: A Logical Basis*, North-Holland, Amsterdam, 1978.

[10] A. van Gelder and Y. K. Tsuji, "Satisfiability Testing with More Reasoning and Less Guessing", *American Mathematical Society* DIMACS Volume Series **26**, to appear.