

BRIDGING THE SAT SOLVER PERFORMANCE GAP BETWEEN RANDOM AND REAL-WORLD INSTANCES

First Last

Our Institution

Abstract. The performance of SAT solvers based on the Davis-Putnam-Logemann-Loveland procedure [6, 5] is sensitive to the choice of branch variable selection heuristic [14]. Heuristics that do well on random instances tend to do poorly on real-world instances and vice versa. In this paper, we attempt to bridge this bifurcation with the notion of *clause weighting*, which attempts to capture the difficulty of satisfying a given clause in a portion of the search space. We show how this notion can be used to uniformly modify several well-known heuristics, namely, one-sided and two-sided Jeroslow-Wang [13, 2, 12], MOMS [3], and Bohm [3]. We conducted a performance study using random and real-world benchmark suites. The modified heuristics provide a statistically significant improvement in the runtimes and the number of nodes expanded on the random instances while performing at least as well on the real-world instances, as compared to their non-modified counterparts.

1 Introduction

This paper presents the notion of *clause weighting* and how it can be used to improve the performance of solvers of the satisfiability problem. We begin with notation and definitions. We use v , with or without subscripts, to denote propositional variables. A *literal* is either v or $\neg v$. A *clause* is a sequence of literals combined with the logical *or* (\vee) operator. For example, $(v_1 \vee \neg v_3 \vee v_4)$ is a clause. C , with or without decorations, is used to denote a clause. We use $|C|$ to denote the number of literals in C . A *formula* is said to be in conjunctive normal form (CNF) if it comprises a set of clauses that are linked together with the logical *and* (\wedge) operator. For example, $(v_1 \vee \neg v_3 \vee v_4) \wedge (\neg v_2 \vee v_4) \wedge (\neg v_1 \vee v_2 \vee \neg v_4)$ is a CNF formula. We use F , with or without decorations, to denote a CNF formula. The *satisfiability problem* (SAT) asks: Given an arbitrary propositional logic formula that is in the conjunctive normal form, is there an assignment of true/false values to the variables in the formula that make all the clauses in the formula evaluate to true? SAT was proved to be NP-complete by Steven Cook [4] and is one of the well-known NP-complete problems [8].

In this paper, we focus on our attention on satisfiability solvers based on the Davis-Putnam-Logemann-Loveland (DPLL) procedure [6, 5]. The DPLL procedure uses backtracking to systematically explore the exponential search space of possible truth value assignments to the variables in the input formula. DPLL relies on a branch variable selection heuristic to pick the next variable it will assign. In its simplest form, the DPLL

procedure on input formula F while using branch variable selection heuristic H can be stated as follows:

Algorithm 1.1: $DPLL(F, H)$

```

procedure  $DPLL(F, H)$ 
  if ( $F$  is empty)
  then { comment: No unsatisfied clauses remain
         return true
  else if ( $F$  has an empty clause with no literals in it)
  then { comment: The empty clause is unsatisfiable
         return false
  else {
    Use the heuristic  $H$  to pick a branch literal  $x$ 
    Obtain formula  $F'$  by assigning  $x$  to true in  $F$ 
    comment: removing any clauses of  $F$  that contain literal  $x$ , and
    comment: deleting any  $\neg x$  literals from any remaining clauses
    if ( $DPLL(F', H)$ )
    return true
  else {
    Obtain formula  $F'$  by assigning  $x$  to false in  $F$ 
    comment: removing any clauses of  $F$  that contain literal  $\neg x$ , and
    comment: deleting any  $x$  literals from any remaining clauses
    return  $DPLL(F', H)$ 
  }
  }

```

Most complete SAT solvers are based on the DPLL procedure, and several DPLL-based SAT solvers are available for public use including Zchaff [16], and MiniSat [7]. Modern solvers feature sophisticated branch variable selection heuristics [14, 16, 11], efficiently implemented unit propagation [17] and clause learning [15, 18]. The choice of branch variable selection heuristic has a significant impact on the performance of DPLL-based solvers, as was clearly shown in [14].

2 Branch Variable Selection Heuristics

Many branch variable selection heuristics, here onwards referred to as branching heuristics or heuristics, have been proposed in the literature including Jeroslow-Wang [13, 2, 12], MOMS [3] and Bohm [3]. Branching heuristics tend to favor variables from shorter clauses as this leads to increased unit propagation thereby leading to a shorter formula after the assignment is made. The performance of DP-based SAT solvers is sensitive to the choice of branching heuristic. No published heuristic is fastest for all input formulae. The heuristics mentioned above are generally better at solving real-world instances than random instances. For completeness, we describe the heuristics considered in our experiments.

2.1 Jeroslow-Wang Heuristic

For any given literal x in formula F , let $h(x) = \sum_{x \in C \wedge C \in F} 2^{-|C|}$. The one-sided Jeroslow-Wang (JW) heuristic selects the literal x which maximizes the value of function h . The two-sided Jeroslow-Wang (2JW) heuristic picks literal x such that $h(x) + h(\neg x)$ is maximized and then picks literal x such that $h(x) \geq h(\neg x)$. Note that this heuristic favors variables that occur in shorter clauses over those that appear in longer ones (on the average).

2.2 Bohm Heuristic

Let m be the maximum clause length in formula F . Let $h_i(x)$ be the number of clauses of size i that contain x . Let $H_i(v) = \alpha \cdot \max \{h_i(v), h_i(\neg v)\} + \beta \cdot \min \{h_i(v), h_i(\neg v)\}$. For each variable v , let $f(v) = [H_1(v), H_2(v), \dots, H_m(v)]$. The Bohm heuristic picks variable v that leads to the *lexicographically* maximal vector $f(v)$. The values α and β are typically chosen to be 1 and 2 respectively.

2.3 MOMS Heuristic

MOMS stands for Maximum Occurrences on clauses of Minimum size. Let m be the minimum clause length in formula F . For literal x , let $h(x)$ to be the number of occurrences of x in clauses of length m .

2.4 Clause Weighting

Our notion of *clause weighting* attempts to capture the difficulty of satisfying a clause in a portion of the DPLL search space. A similar idea is used in the VSIDS heuristic of the Zchaff solver [16]. In VSIDS, the rating for each literal is initially set to zero. When a clause is learned, the counter for the literal is increased. VSIDS ratings are periodically decreased by dividing by a constant. In our approach, we assign ratings (weights) to *clauses*. All clauses in the formula start out with the same weight. When a clause becomes false during the search process, we increment the weight of the clause. Clause weights can be used in the computation of variable ratings in the JW, 2JW, Bohm and MOMS heuristics to obtain modified heuristics JW+, 2JW+, Bohm+ and MOMS+, respectively. The modified variable functions for Jeroslow Wang, Bohm and MOMS are given below. We use $weight(C)$ to denote the weight of clause C .

Clause Weighted Jeroslow Wang Heuristic For any given literal x in formula F , let $h^+(x) = \sum_{x \in C \wedge C \in F} 2^{-|C|} \cdot weight(C)$. The clause weighted one-sided Jeroslow-Wang (JW+) heuristic selects the literal x which maximizes the value of function h^+ . The clause weighted two-sided Jeroslow-Wang (2JW+) heuristic picks literal x such that $h^+(x) + h^+(\neg x)$ is maximized and then picks literal x such that $h^+(x) \geq h^+(\neg x)$.

Clause Weighted Bohm Heuristic Let m be the maximum clause length in formula F . Let $h_i^+(x) = \sum_{x \in C \wedge C \in F \wedge |C|=i} weight(C)$, and let $H_i^+(v) = \alpha \cdot \max \{h_i^+(v), h_i^+(\neg v)\} + \beta \cdot \min \{h_i^+(v), h_i^+(\neg v)\}$. For each variable v , let $f^+(v) = [H_1^+(v), H_2^+(v), \dots, H_m^+(v)]$. As before, the clause weighted Bohm heuristic picks variable v that leads to the *lexicographically* maximal vector $f^+(v)$.

Clause Weighted MOMS Heuristic Let m be the minimum clause length in formula F . Let $h^+(x) = \sum_{x \in C \wedge C \in F \wedge |C|=m} weight(C)$. The clause weighted MOMS heuristic picks x which maximizes the value of h^+ .

3 Experimental performance study

We conducted an experimental performance study of the effectiveness of clause weighting. We compared the performance of heuristics JW+, 2JW+, Boehm+ and MOMS+ with the performance of JW, 2JW, Boehm, and MOMS, respectively, using two different benchmark suites. Our random benchmark suite contained 33 files generated using the *makewff* program [1]. These files contain random formulae with 305 variables generated in the hard region [9]. The real world benchmark suite contains 83 instances that were obtained from the SIM library [10], which has SAT encodings for 167 problems from many different domains.

The results were obtained under Red Hat Enterprise Linux 3 on a quad Intel Xeon 3.6 GHz machine with 4 Gigabytes of RAM. The SIM source was compiled with the gcc 3.2.3 compiler. Each heuristic was allocated a maximum CPU runtime of 3600 seconds per instance. For each instance-heuristic pair, we recorded two common SAT solver performance metrics, namely, CPU time and the number of nodes generated.

We analyzed the performance results for all instances using paired t-tests. We found a statistically significant decrease in the runtimes for the modified heuristics. We also found a statistically significant decrease in the number of nodes generated for the modified heuristics. The modified heuristics generated an average of 57% fewer nodes than their non-modified counterparts. The average runtime for the modified heuristics was 51% faster than the average runtime for the non-modified heuristics. It is therefore clear that the overhead of the clause weighting computation is justifiable.

We then analyzed the results broken down by instance type: Random or real-world. The results for the random instances show that the modified and the non-modified heuristics terminated in the allocated time for all instances. The modified heuristics produce statistically significant improvements on both the runtimes and number of nodes generated. For the random instances, the percentage average improvement per instance-heuristic pair is 55% on the runtime and 63% on the number of nodes generated.

The results for the real-world instances show that the modified heuristics were able to solve 9 instances more than the non-modified heuristics in the allocated time. These data points were removed from the data set to facilitate the statistical analysis. For the real-world instances, the *clause weighted* modifications to the heuristics resulted in non-significant improvements in the average runtime and the average number of nodes generated.

We also performed two-sample approximate t-tests to compare the performance improvements of *clause weighting* in the random and real-world test suites. We found that the differences between the performance improvements in the random and the real-world benchmark suites are statistically significant, both for the runtimes and for the numbers of nodes generated. In other words, the implementation of *clause weighting* results in a significantly better decrease in the runtimes of the random instances than the runtimes of the real-world instances. The same is true of the decrease in the number of the nodes generated.

4 Conclusions

We have presented the notion of *clause weighting* which attempts to capture the difficulty of satisfying a clause in a portion of the DPLL search space. We have described how *clause weighting* can be used to uniformly modify four published branching variable selection heuristics namely, one-sided and two-sided Jeroslow-Wang, MOMS, and Bohm. The non-modified versions of these heuristics are known to perform relatively better on real-world instances than random instances. The results from our experimental performance study show that the *clause weighted* versions of these heuristics perform significantly better on random instances than the non-modified counterparts, while preserving their original performance on real-world instances. This improvement is observed for two common SAT solver performance metrics: runtime and number of nodes generated. Thus *clause weighting* bridges the prior gap between the performance of these heuristics on real-world instances and the performance of these heuristics on random instances.

5 Acknowledgements

The authors gratefully acknowledge Dr. First Last's assistance with the statistical analysis of the performance data.

References

1. Walksat home page at <http://www.cs.rochester.edu/u/kautz/walksat/>.
2. P. Barth. A davis-putnam enumeration procedure for linear pseudo-boolean optimization. Technical Report MPI-I-2-003, Max Planck Institute, January 1995.
3. M. Buro and H. Kleine-Buning. Report on a sat competition. Technical report, University of Paderborn, November 1992.
4. Steven Cook. On the complexity of theorem-proving procedures. In *Proceedings of the Third ACM Symposium on the Theory of Computing (STOC)*, pages 151–158, 1971.
5. Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.
6. Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, July 1960.
7. Niklas Een and Niklas Sorensson. Minisat - a sat solver with conflict-clause minimization. In *Proceedings of the Theory and Applications of Satisfiability Testing (SAT'05)*, 2005.

8. Michael Garey and David Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co., 1979.
9. Ian P. Gent and Toby Walsh. The SAT phase transition. In *Proceedings of the Eleventh European Conference on Artificial Intelligence (ECAI'94)*, pages 105–109, 1994.
10. E. Giunchiglia, M. Maratea, A. Tacchella, and D. Zambonin. Evaluating search heuristics and optimization techniques in propositional satisfiability. In *Proceedings of International Joint Conference on Automated Reasoning (IJCAR'01)*, Siena, June 2001. to appear.
11. E. Goldberg and Y. Novikov. BerkMin: A fast and robust SAT-solver. In *Design, Automation, and Test in Europe (DATE '02)*, pages 142–149, March 2002.
12. J. N. Hooker and V. Vinay. Branching rules for satisfiability. *Journal of Automated Reasoning*, 15:359–383, 1995.
13. R. J. Jeroslow and J. Wang. Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, 1:167–188, 1990.
14. Joao Marques-Silva. The impact of branching heuristics in propositional satisfiability algorithms. In *Proceedings of the 9th Portuguese Conference on Artificial Intelligence*, volume 1695 of *Lecture Notes in Computer Science*, 1999.
15. Joao P. Marques-Silva and Karem A. Sakallah. GRASP - A New Search Algorithm for Satisfiability. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 220–227, November 1996.
16. Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, June 2001.
17. Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, 2001.
18. L. Zhang, C. F. Madigan, M. W. Moskewicz, and S. Malik. Efficient conflict driven learning in a Boolean satisfiability solver. In *International Conference on Computer-Aided Design (ICCAD'01)*, pages 279–285, November 2001.