

Lookahead Saturation with Restriction for SAT

Anbulagan¹ and John Slaney^{1,2}

¹Logic and Computation Program, National ICT Australia Ltd., Canberra, Australia

²Computer Sciences Laboratory, Australian National University, Canberra, Australia
{anbulagan, john.slaney}@nicta.com.au

Abstract. We present a new and more efficient heuristic by restricting lookahead saturation (LAS) with NVO (neighbourhood variable ordering) and DEW (dynamic equality weighting). We report on the integration of this heuristic in Satz, a high-performance SAT solver, showing empirically that it significantly improves the performance on an extensive range of benchmark problems that exhibit hard structure.

1 Introduction

During the last decade, many new techniques have been proposed to enhance the performance of the DPLL procedure for solving various hard real-world problems represented in conjunctive normal form (CNF). One of the main improvements of this decision procedure has been the development of better branching variable selection through the use of unit propagation (UP) heuristics [1], which detect failed literals through a one-step lookahead. The effect of integrating the UP heuristic into DPLL is to prune the search tree earlier.

In this paper, we provide a new heuristic, DEW-NVO-LAS, which restricts lookahead saturation (LAS) with NVO (neighbourhood variable ordering) and DEW (dynamic equality weighting). DEW weighs equality literals during NVO-restricted lookahead saturation, firstly to restrict the variables to be propagated through the lookahead process, and secondly so that the next branching variable chosen can be the one having the highest score. We report on the integration of the DEW-NVO-LAS heuristic into Satz, showing empirically that it significantly improves Satz’s performance on a range of benchmark problems, such as bounded model checking, cryptographic key search, FPGA routing, equivalence checking in circuits, and, particularly, the challenging 32-bit parity learning problems. The same problems are used for a comparative study between Dew.Satz, the DEW-NVO-LAS-enhanced Satz solver, and other state-of-the-art SAT solvers.

2 Lookahead Saturation with Restriction

Lookahead saturation (LAS) based DPLL was studied in [2]. The key idea underlying LAS is to choose a branching variable which is really the best from an irreducible sub-formula at a given node of search tree. LAS is very similar to the “singleton arc consistency” (SAC) algorithm in CSP reasoning [3].

Intuitively, although a reasoning-intensive process such as LAS can reduce the search tree size enormously, this increased efficiency is outweighed by the cost in terms of runtime. For that reason, we restrict the LAS process using the NVO and DEW heuristics. While the NVO heuristic concentrates on restricting the number of variables to be examined in the next iterative lookahead process by considering only the neighbours of the currently assigned variable in the currently size-reduced clauses, the DEW heuristic restricts the number of literals to be examined during the iterative lookahead process. DEW alone is not particularly useful in this regard, it must be incorporated into NVO-LAS to be really effective.

The basic concept of DEW is as follow. Whenever the binary equality clause $x_i \Leftrightarrow x_j$, which is equivalent to 2 CNF clauses $\bar{x}_i \vee x_j$ and $x_i \vee \bar{x}_j$, occurs in the formula at a node, Satz needs to perform the lookahead process on x_i , \bar{x}_i , x_j , and \bar{x}_j . As result, variables x_i and x_j will be associated the same weight, (i.e. 3 following the computation at line 25 of Algorithm 1). Clearly, the processing of x_j and \bar{x}_j is redundant, so avoid it by assigning the implied literal \bar{x}_j (x_j 's) the weight of its parent literal \bar{x}_i (x_i 's), and then by restricting the lookahead process to literals with weight zero. By doing so, we save two lookahead processes.

To clarify the concept, we present a concrete example. Consider the following simple formula with binary equality clauses: $(x_1 \Leftrightarrow x_2) \wedge (x_2 \Leftrightarrow x_3) \wedge (x_1 \Leftrightarrow x_4)$. The Satz solver evaluates iteratively each variable of the formula by two forced unit propagations, where there is no failed literal found. Each literal of the formula gets the same weight, i.e. 3. Intuitively, we do not need to lookahead on variables x_2 , x_3 and x_4 after performing lookahead on x_1 : all three get the weight of the parent x_1 . The effect of the DEW heuristic is that the weight of each implied literal accumulates dynamically during the lookahead process, and if it is greater than zero then no lookahead process is done on that literal. The DEW heuristic is executed only whenever binary equality clauses occur in the current state formula.

Our main observation is that DEW benefits markedly from NVO-LAS. We integrate DEW-NVO-LAS heuristic into Satz, and call the new solver by `Dew_Satz`. Intuitively, the merged heuristic will enhance the performance of NVO-LAS by avoiding the redundant lookahead process, which is computed by DEW. At the same time the DEW heuristic benefits from NVO-LAS as this dynamically bounds the number of variables to be weighed. The two mutually compatible heuristics work together to improve lookahead-based DPLL. `Dew_Satz` also inherits from Satz a preprocessor for saturating the input clauses under resolution with the restriction to clauses of length ≤ 3 , removing subsumed clauses and tautologies along the way. In certain cases, the preprocessor may remove some equality clauses.

Algorithm 1 sketches the branching rule of `Dew_Satz`. The procedure `Compute_DEW(x_i)` is called for weighting the implied literals of the parent variable x_i . The function `UP(\mathcal{F}_i)` at line 7 (10) of Algorithm 1 is executed if $w(x_i)=0$ ($w(\bar{x}_i)=0$). When there is no conflict found during the two unit propagations, then variable x_i will be piled into the branching variable candidates stack \mathcal{B} .

Algorithm 1 DEW-NVO-LAS-BranchingRule(\mathcal{F})

```
1: Push each variable  $x_i \in \mathcal{V}$  to NVO.STACK;
2: repeat
3:    $\mathcal{B} := \emptyset$ ;  $\mathcal{F}_{init} := \mathcal{F}$ ;
4:   for each variable  $x_i \in$  NVO.STACK do
5:     Let  $\mathcal{F}'_i$  and  $\mathcal{F}''_i$  be two copies of  $\mathcal{F}$ ;
6:     if  $w\{x_i\} = 0$  then
7:        $\mathcal{F}'_i := \text{UP}(\mathcal{F}'_i \cup \{x_i\})$ ;
8:     end if
9:     if  $w\{\bar{x}_i\} = 0$  then
10:       $\mathcal{F}''_i := \text{UP}(\mathcal{F}''_i \cup \{\bar{x}_i\})$ ;
11:    end if
12:    if empty clause  $\in \mathcal{F}'_i$  and empty clause  $\in \mathcal{F}''_i$  then
13:      return "unsatisfiable";
14:    else if empty clause  $\in \mathcal{F}'_i$  then
15:       $\mathcal{F} := \mathcal{F}''_i$ ; NVO( $x_i$ );
16:    else if empty clause  $\in \mathcal{F}''_i$  then
17:       $\mathcal{F} := \mathcal{F}'_i$ ; NVO( $x_i$ );
18:    else
19:       $w(x_i) := \text{diff}(\mathcal{F}'_i, \mathcal{F})$ ;  $w(\bar{x}_i) := \text{diff}(\mathcal{F}''_i, \mathcal{F})$ ;
20:       $\mathcal{B} := \mathcal{B} \cup \{x_i\}$ ; Compute.DEW( $x_i$ );
21:    end if
22:  end for
23: until  $\mathcal{F} = \mathcal{F}_{init}$ 
24: for each variable  $x_i \in \mathcal{B}$  do
25:    $\mathcal{W}(x_i) := w(x_i) * w(\bar{x}_i) + w(x_i) + w(\bar{x}_i)$ ;
26: end for
27: NVO( $x_i$ );
28: return  $x_i$  with highest  $\mathcal{W}(x_i)$  to branch on;
```

3 Experimental Results

The 32-bit parity problem instances are considered as a challenging problem [4]. To answer the challenge, equality reasoning has been integrated differently in different solvers [5–8]. EqSatz uses equality reasoning in the search process while Lsat and March_eq use it in their preprocessors.

In Table 1, we present the performance of Dew_Satz on par16* and the challenging par32* instances in comparison with the following state-of-the-art solvers: EqSatz, Satz (ver. Satz215), zChaff (ver. 2004.11.15), March_eq (ver. March_eq.010), Lsat (ver. 1.1). It is important to observe that Dew_Satz can solve the 32-bit parity problem in the range of 411 to 17,564 seconds. It solved the par32-5 and par32-5-c instances without using the preprocessor (with preprocessing, these instances took 27 and 29 hours respectively). The results of Dew_Satz refute the pessimistic view that lookahead-based DPLL must perform poorly on such highly structured problems.

In order to evaluate further the performance of Dew_Satz versus other solvers used above, we extended the empirical study to include some well-known circuit-

Instance (#Vars/#Cls)	Satz	Dew_Satz	EqSatz	Lsat	March_eq	zChaff
par16*	12.97	1.26	0.55	0.56	0.17	6.12
par32-1 (3176/10227)	>24h	12,918	242	126	0.22	>24h
par32-2 (3176/10253)	>24h	5,804	69	60	0.27	>24h
par32-3 (3176/10297)	>24h	7,198	2,863	183	2.89	>24h
par32-4 (3176/10313)	>24h	11,005	209	86	1.64	>24h
par32-5 (3176/10325)	>24h	17,564	2,639	418	8.07	>24h
par32-1-c (1315/5254)	>24h	10,990	335	270	2.63	>24h
par32-2-c (1303/5206)	>24h	411	13	16	2.19	>24h
par32-3-c (1325/5294)	>24h	4,474	1,220	374	6.65	>24h
par32-4-c (1333/5326)	>24h	7,090	202	115	0.45	>24h
par32-5-c (1339/5350)	>24h	11,899	2,896	97	6.44	>24h

Table 1. CPU time (in seconds) comparison. ”>24h” shows that the problem cannot be solved in 24 hours.

Problem	Dew_Satz	Satz	EqSatz	March_eq	zChaff
barrel6	4.13	271	0.17	0.13	2.95
barrel7	8.62	1,896	0.23	0.25	11
barrel8	72	>5,000	0.36	0.38	44
barrel9	158	>5,000	0.80	0.87	66
longmult10	64	736	385	213	872
longmult11	79	998	480	232	1,625
longmult12	97	1,098	542	167	1,643
longmult13	127	1,246	617	53	2,225
longmult14	154	1,419	706	30	1,456
longmult15	256	1,651	743	23	392
queueinvar12	1.26	0.81	0.67	2.19	0.22
queueinvar14	2.50	1.96	1.17	4.19	0.42
queueinvar16	1.11	1.05	1.06	3.44	0.35
queueinvar18	11	12	1.02	25	1.76
queueinvar20	13	19	1.58	40	3.13
cnf-r3*(8)	10.88	(1) 12,032	2,992	271	11.37
bart*(21)	0.88	0.35	(17) 85,403	(1) 6,838	16
homer*(15)	3,054	(15) 75,000	(15) 75,000	(15) 75,000	(1) 9,245
lisa*(14)	2,955	1,721	5,788	1,211	(3) 30,349
hwb-n20*(3)	177	148	188	46	1,355
hwb-n22*(3)	771	637	716	144	3,700
hwb-n24*(3)	3,457	3,115	4,170	1,115	(3) 15,000
pb-sat*(12)	(2) 13,818	(2) 15,793	(4) 24,478	7,869	(4) 21,099
pb-unsat*(12)	19,547	22,269	(4) 24,293	(1) 19,659	(8) 44,144
philips-org	697	3,845	1974	>5,000	>5,000
philips	295	1,086	2401	726	>5,000

Table 2. CPU time (in seconds) on realistic benchmark problems.

related benchmark problems. All problems used in the study are taken from SATLIB (www.satlib.org), where some of them are used in previous SAT competitions. Some of the problem instances contain more than 600,000 variables. The problems cnf-r3*, bart*, lisa* and pb-sat* are satisfiable, and the others are unsatisfiable. The timebound for this experiment is 5,000 seconds per problem instance. Table 2 shows the runtimes of Dew_Satz, Satz, EqSatz, March_eq and zChaff on these problems. The numbers of instances of some problems are indicated in brackets after the problem names, and the number of instances on which each solver *failed* is also indicated in brackets before the total time. We count 5,000 as the increment in runtime for an unsolved instance. The experiments were conducted on Intel Pentium 4 PCs with 3 GHz CPU, under Linux.

In general, this extended study further confirms the superior performance of Dew_Satz in comparison with the other four solvers. Where Dew_Satz fails to solve 2 instances in the given timebound (instance pb-sat-40-4-02 needs 6,005 seconds and instance pb-sat-40-4-03 needs 12,958 seconds), Satz, EqSatz, March_eq and zChaff fail on 20, 40, 18 and 21 instances respectively of the 108 given.

The empirical results also show that unit propagation based lookahead in DPLL is still a powerful technique. Simply enhancing it with a straightforward heuristic allows us to solve many more hard problems, as shown by the results above.

Acknowledgments

This work was funded by National ICT Australia (NICTA). National ICT Australia is funded through the Australian Government's *Backing Australia's Ability* initiative, in part through the Australian Research Council.

References

1. Li, C.M., Anbulagan: Heuristics based on unit propagation for satisfiability problems. In: Proceedings of 15th IJCAI, Nagoya, Aichi, Japan (1997) 366–371
2. Anbulagan: Extending unit propagation look-ahead of DPLL procedure. In: Proc of 8th PRICAI, Auckland, New Zealand, Springer, LNAI 3157 (2004) 173–182
3. Bessière, C., Debruyne, R.: Theoretical analysis of singleton arc consistency. In: ECAI-04 Workshop on Modeling and Solving Problems with Constraints, Valencia, Spain (2004) 20–29
4. Selman, B., Kautz, H., McAllester, D.: Ten challenges in propositional reasoning and search. In: Proceedings of 15th IJCAI, Nagoya, Aichi, Japan (1997) 50–54
5. Warners, J.P., van Maaren, H.: A two-phase algorithm for solving a class of hard satisfiability problems. *Operations Research Letters* **23** (1998) 81–88
6. Li, C.M.: Integrating equivalency reasoning into Davis-Putnam procedure. In: Proceedings of 17th AAAI, USA, AAAI Press (2000) 291–296
7. Ostrowski, R., Grégoire, E., Mazure, B., Sais, L.: Recovering and exploiting structural knowledge from CNF formulas. In: Proceedings of 8th CP. (2002) 185–199
8. Heule, M., van Maaren, H.: Aligning CNF- and equivalence-reasoning. In: Proceedings of 7th SAT, Vancouver, Canada (2004)