Adding a New Conflict Based Branching Heuristic in two Evolved DPLL SAT Solvers

Renato Bruni, Andrea Santori

Università di Roma "La Sapienza", Dip. di Informatica e Sistemistica,
Via Michelangelo Buonarroti 12 - 00185 Roma, Italy,
E-mail: renato.bruni@dis.uniroma1.it, santori.andrea@libero.it

Abstract. The paper is concerned with the computational evaluation of a new branching heuristic, called *reverse assignment sequence* (RAS), for evolved DPLL Satisfiability solvers. Such heuristic, like several other recent ones, is based on the history of the conflicts obtained during the solution of an instance. A score is associated to each literal. When a conflict occurs, some scores are incremented with different values. The branching variable is then selected by using the maximum score. This branching heuristic is introduced in two evolved DPLL solvers: ZChaff and Simo. Experiments on several benchmark series, both satisfiable and unsatisfiable, demonstrate advantages of the proposed heuristic.

1 Introduction

Given a propositional formula, determining whether there exists a truth assignment for its propositional variables such that the formula evaluates to true is called the propositional Satisfiability problem, commonly abbreviated as SAT. Extensive references can be found in [4, 12, 20]. Many problems arising from different fields, such as artificial intelligence, logic circuit design and testing, cryptography, database systems, software verification, are usually encoded as SAT. Moreover, SAT carries considerable theoretical interest as the original NP-complete problem [5, 8]. From the practical point of view, this implies that investing on the cleverness of the solution algorithm can result in very large savings in computational times. The above has motivated a wide stream of research in practically efficient SAT solvers. As a consequence, many algorithms for solving the SAT problem have been proposed, based on different techniques (see for instance [6, 7, 9, 12, 15]). Computational improvements in this field are impressive, see e.g. [15]. A solution method is said to be *complete* if it guarantees (given enough time) to find a solution if it exists, or prove lack of solution otherwise. Incomplete, or *stochastic*, methods, on the contrary, cannot guarantee finding the solution, although they may scale better than complete methods on some large satisfiable problems. Most of the best complete solvers are based on so-called Davis-Putnam-Logemann-Loveland (DPLL) enumeration techniques. From the initial relatively simple DPLL backtracking algorithm described in [6], SAT solvers have evolved experimenting several more sophisticated branching and backtracking frameworks, and eventually incorporating the best ones. Noteworthily examples have been non-chronological backtracking and conflict-driven learning [1, 17]. These techniques greatly improve the efficiency of DPLL algorithms, expecially for structured SAT instances. Subsequently, a further generation of solvers paying special attention to implementation aspects appeared: SATO [22], Chaff [19], BerkMin [11] and several others, sometimes referred to as *chaff-like* solvers [15]. Such kind of solvers nowadays appear to probably be the most competitive in solving real-world problems.

As a matter of fact, a relevant influence on computational behavior is given by the *branching rule*, or *branching heuristic*, that is how to chose the next variable assignment. Different branching heuristics for the same basic algorithm may result in completely different computational results [18, 21]. Early branching heuristics (e.g. Böhm [3], MOM [12], Jeroslow-Wang [14]) have often been viewed as greedy trials of simplifying as much as possible current subproblem, for instance by satisfying the most clauses. Such heuristics are based on *a priori* statistics on the instance, and have a certain effectiveness in the case of randomly generated problems. However, they usually cannot capture hidden problem's structure, and real world problems typically are quite well structured. In order to tackle such problems, heuristics based on the history of the search, and in particular on the history of conflict, have been studied . Examples are VSIDS heuristic of Chaff [19], the

adaptive branching rule of ACS [2], BerkMin decision making strategy [11], the dynamic selection of branching rules [13].

We propose here a branching heuristic having two main aims. The first consists in trying to assign at first the more constrained variables, which are also those on which we learned more. The second consists in trying to reverse every sequence of assignments which have led to a conflict, by satisfying at first clauses which have become empty. The proposed heuristic, called *reverse assignment sequence* (RAS), is introduced in two modern DPLL SAT solvers, in order to verify its empirical behavior. We specifically selected ZChaff [19] and Simo [10], since their structure and features are quite different, and therefore the effect of their modification can be considered enough representative for the family of DPLL solvers. Experiments on several benchmark series, both satisfiable and unsatisfiable, show that the proposed branching heuristic improves in the two solvers both speed and number of decisions.

2 The RAS Branching Rule

For DPLL-based algorithm, the search evolution is often represented as the exploration of a *search tree*, where each node subproblem is obtained by assigning a variable. Being SAT an NP-complete problem theoretically implies that, for satisfiable instances, choosing at every step the right branch in the search tree would lead to the solution in a polynomial number of assignments [8]. Unfortunately, unless P=NP, it seems unlikely that some practical algorithm doing this in polynomial time may in general exist. Moreover, the problem of choosing such assignment for DPLL algorithms has been proven to be NP-hard and coNP-hard [16]. Therefore, the (heuristic) policy governing the choice of the variable assignments is generally called *branching heuristic*. Different branching heuristics may produce drastically different sized search trees for the same basic algorithm. Conflict-based branching heuristics generally keep, for each variable x_i, a counter, or *score* s_i, or sometimes two counters, for the two possible truth assignments, or phases, of x_i. Score s_i is incremented when x_i is somehow involved in a *conflict*, i.e. an empty clause is derived by current truth assignments. Branching variables are selected according to the values of such scores. Counters are often periodically proportionally reduced, both for avoiding overflow problems, and for giving to earlier history of the search progressively less importance than recent history. For instance, ZChaff [19] heuristic (called VSIDS, variable state independent decaying sum) uses for each variable two scores initialized to the number of occurrences of each literal in the instance. Whenever a new clause is *learned*, the counter of each of its literal is incremented. The variable assignment corresponds to the literal having maximum score. Also, BerkMin [11] heuristic uses one score for each variable. Whenever a conflict occurs, the scores of all variables contained in the clauses that are responsible for the conflict are increased. The variable assignment corresponds to the literal whose variable has maximum score among those contained in the last added clause that is unresolved. Conflict based branching heuristics have the advantages of requiring low computational overhead and of being often able to detect the hidden structure of a problem. They therefore generally produce good results on large real-world instances.

The proposed branching heuristic, called *reverse assignment sequence* (RAS), proceeds as follows. For each variable $x_i,\ i = 1, \ldots, n$, we use two counters s_i^0 and s_i^1 for the two possible phases of x_i. Counters are therefore associated to the two possible literals $l_0(x_i) = \neg x_i$ and $l_1(x_i) = x_i$. When branching is needed, we assign, as usual, variable x_i at value $v \in \{0, 1\}$, as follows.

$$x_i = v, \quad \text{with } s_i^v = \max(s_1^0, s_1^1, \ldots, s_n^0, s_n^1)$$

The main issue clearly is how scores $\{s_1^0, s_1^1, \ldots, s_n^0, s_n^1\}$ are incremented. We do this by trying to pursue two different aims. The first aim consists in assigning at first the more *difficult* variables, in the sense of the more constrained ones. This because, when assigning them in the upper levels of the search tree, either we should discover unsatisfiability earlier, or we should remain with only *easy* variables to assign in the lower levels of the search tree, and therefore little backtrack should be needed to reach a satisfying assignment. Whenever a new *learned clause* $C_l = \{l_v(x_{l1}), \ldots, l_v(x_{lh})\}$ is added to the clause set by effect of a conflict, what we have actually discovered is that variables $\{x_{l1}, \ldots, x_{lh}\}$ contained in C_l are a bit more constrained than other variables. In fact, C_l represents

just an explicitation of such constraint, that is already implied by the original clauses. Therefore, we increment the scores as follows

$$s_i^v \leftarrow s_i^v + p_l, \quad \forall l_v(x_i) \in C_l$$

(where $a \leftarrow a + b$ means that new value of a is obtained by adding b to its old value). The effect can also be viewed as trying to satisfy C_l. Note that, so far, this is also ZChaff's policy. Our second aim, on the other hand, consists in trying to reverse every sequence of assignments which leads to a conflict. Whenever a sequence of assignment produces an *empty clause*, this sequence risks to be repeated again in the search tree, leading again to the same conflict. The use of learned clauses, together with the increment of the scores of their literals, can only partially solve the problem. We therefore try to satisfy the *failed clause* $C_f = \{l_v(x_{f1}), \ldots, l_v(x_{fk})\}$ (the clause which has become empty) by incrementing again the scores as follows.

$$s_i^v \leftarrow s_i^v + p_f, \quad \forall l_v(x_i) \in C_f$$

Similarly to other conflict based heuristics, scores are initialized to the number of occurrences of each literal in the instance, and periodically divided.

However, since increasing scores has a cost, and moreover implies an even higher cost for reordering the scores in order to choose the higher value, we apply some simplifications to the above algorithm. In fact, adding p_f to only one of the counters corresponding to the failed clause C_f, and in particular to the last assigned one, decreases computational overhead while maintaining most of the positive features. Several other alternatives were tested, but the above proposed one appears more stable. The following example illustrates in detail counters updating performed after a typical conflict.

Example: Consider an instance containing, among others, the following two clauses:

$$C_a = (\neg x_1 \vee x_3 \vee x_5) \qquad C_b = (x_2 \vee \neg x_4 \vee \neg x_5)$$

Imagine that we have already assigned $\{x_1$ to 1, x_2 to 0, x_3 to 0 and x_4 to $1\}$. We now have C_a reduced to a unit clause, which force assigning $\{x_5$ to $1\}$. So far C_b becomes empty, and we learn $C_l = (\neg x_1 \vee x_2 \vee x_3 \vee \neg x_4)$, while C_f is in this case C_b. Counters are updated as follows:

$$s_1^0 \leftarrow s_1^0 + p_l \quad s_2^1 \leftarrow s_2^1 + p_l \quad s_3^1 \leftarrow s_3^1 + p_l \quad s_4^0 \leftarrow s_4^0 + p_l + p_f$$

3 Computational Analysis

RAS branching heuristic is introduced in two evolved DPLL solvers: ZChaff [19] and Simo [10], obtaining what we called BrChaff and BrSimo. Experiments are conducted on a Pentium III 733MHz PC with 256Mb RAM and using MS VC++ compiler. Note that some libraries may be different using other compilers, therefore results may vary (we experienced it) but maintaining about the same average results on each series. Parameters p_l and p_f are respectively fixed at 1 and 2. We report running times in seconds and number of decisions. Time limit was set at 7200 sec. (2 hours), when exceeded we report "-". When the two versions of a solver solve different problems, totals are extended only to problems solved by both versions (unless this has no sense), and this is indicated by "*". Mostly of the considered benchmark series are real-world problems, therefore structured. We also considered one randomly generated series from 2003 competition, where we omitted for brevity the central part of the names (e.g. hardnm-L19-02-S125896754.shuffled-as.sat03-916 → hrdnm-L19-02-sat03-916). The series are either all satisfiable, or all unsatiafiable, or mixed. The use of the RAS branching rule demonstrate advantages in almost the totality of considered cases. In particular, on the Beijing series (logic circuit problems) we obtain great running times improvements for both solvers; on the Des-encryption series (cryptography problems) great running times improvements for both solvers (Simo in particular becomes able to solve many instances); on the FVP series (hardware verification problems) great running time improvements for ZChaff and some running time improvement for Simo; on the Miters series (equivalence checking problems) moderate running time improvements for ZChaff and sensible running time improvement for Simo; on the Barrel series (bounded model checking problems) improvements only in the number of decisions for ZChaff, but great running time improvements for Simo; on the Hardnm series (randomly generated problems) we obtain great running times improvements for both solvers.

Beijing		ZChaff		BrChaff		Simo		BrSimo	
Problem	Sol	Decisions	Time	Decisions	Time	Decisions	Time	Decisions	Time
2bitadd_10	U	172675	360.42	59483	70.14	-	-	-	-
2bitadd_11	S	9110	3.66	14655	9.42	290	0.04	1345	0.11
2bitadd_12	S	10228	3.83	9173	4.61	324	0.04	1837	0.17
2bitcomp_5	S	50	0.00	50	0.00	66	0.01	65	0.00
2bitmax_6	S	329	0.01	404	0.01	104	0.01	136	0.01
3bitadd_31	-	-	-	-	-	-	-	-	-
3bitadd_32	-	-	-	-	-	-	-	-	-
3blocks	S	872	0.02	980	0.04	426	0.48	1338	0.45
4blocks	S	6552	1.60	6462	2.32	80917	207.40	71438	206.93
4blocksb	S	1994	0.42	1521	0.28	5625	11.96	4508	4.83
e0ddr2-10-by-5-1	S	25156	54.48	10105	17.35	-	-	-	-
e0ddr2-10-by-5-4	S	2876	3.28	584	0.55	25617	154.85	744	1.78
enddr2-10-by-5-1	S	2656	3.68	2244	3.62	2647	9.35	732	1.72
enddr2-10-by-5-8	S	664	0.78	586	0.74	-	-	765	1.72
ewddr2-10-by-5-1	S	1730	2.30	1814	2.61	3039	12.33	766	1.77
ewddr2-10-by-5-8	S	310	0.25	310	0.22	3030	12.40	817	1.77
Total		235202	434.71	108371	111.92	121301*	408.86*	83726*	219.54*

Table 1: Comparison on logic circuit problems.

Des-encryption		ZChaff		BrChaff		Simo		BrSimo	
Problem	Sol	Decisions	Time	Decisions	Time	Decisions	Time	Decisions	Time
cnf-r3-b1-k1.1-comp	S	15799	16.21	13356	8.90	-	-	-	-
cnf-r3-b1-k1.2-comp	S	11173	6.62	10865	7.65	-	-	-	-
cnf-r3-b2-k1.1-comp	S	809	0.22	887	0.26	-	-	52956	39.91
cnf-r3-b2-k1.2-comp	S	666	0.16	612	0.14	-	-	498996	657.58
cnf-r3-b3-k1.1-comp	S	413	0.10	367	0.10	-	-	71594	89.65
cnf-r2-b3-k1.2-comp	S	420	0.11	422	0.11	-	-	20950	20.70
cnf-r3-b4-k1.1-comp	S	428	0.13	379	0.12	-	-	31394	50.25
cnf-r3-b4-k1.2-comp	S	428	0.13	379	0.12	-	-	31394	50.26
cnf-r4-b1-k1.1-comp	-	-	-	-	-	-	-	-	-
cnf-r4-b1-k1.2-comp	-	-	-	-	-	-	-	-	-
cnf-r4-b2-k1.1-comp	S	354831	2187.71	264884	1565.59	-	-	-	-
cnf-r4-b2-k1.2-comp	S	221764	1106.45	157293	751.58	-	-	-	-
cnf-r4-b3-k1.1-comp	S	132080	539.47	96646	351.12	-	-	-	-
cnf-r4-b3-k1.2-comp	S	118457	458.63	56830	167.84	-	-	-	-
cnf-r4-b4-k1.1-comp	S	64161	196.18	67229	228.94	-	-	-	-
cnf-r4-b4-k1.2-comp	S	64161	196.31	67229	228.84	-	-	-	-
Total		985590	4708.43	737378	3311.32	-	-	-	-

Table 2: Comparison on cryptography problems.

FVP		ZChaff		BrChaff		Simo		BrSimo	
Problem	Sol	Decisions	Time	Decisions	Time	Decisions	Time	Decisions	Time
1dlx_c_mc_ex_bp_f	U	3586	0.12	3730	0.14	-	-	2778567	293.76
2dlx_ca_mc_ex_bp_f	U	45130	4.13	35761	3.62	-	-	-	-
2dlx_cc_mc_ex_bp_f	U	60084	6.43	54278	7.34	-	-	-	-
9vliw_bp_mc	U	3619845	644.04	2144050	349.84	-	-	-	-
Total		3728645	654.71	2237819	360.93	-	-	-	-

Table 3: Comparison on hardware verification problems.

Miters		ZChaff		BrChaff		Simo		BrSimo	
Problem	Sol	Decisions	Time	Decisions	Time	Decisions	Time	Decisions	Time
c1355-s	U	9875	1.11	34707	17.42	-	-	-	-
c1355	U	15783	2.56	28896	8.90	-	-	-	-
c1908-s	U	29833	7.56	17169	3.39	-	-	-	-
c1908	U	28687	7.47	30461	8.94	-	-	-	-
c1908_bug	S	15841	3.09	23721	5.68	480	0.24	305	0.17
c2670-s	U	20507	1.98	29180	3.85	-	-	-	-
c2670	U	36645	3.97	26795	3.67	-	-	-	-
c2670_bug	S	3802	0.11	2461	0.07	-	-	-	-
c3540-s	U	130424	140.36	115457	106.47	-	-	-	-
c3540	U	92076	71.22	91554	69.99	-	-	-	-
c3540_bug	S	50	0.01	50	0.01	-	-	2240	2.47
c432-s	U	1467	0.05	1354	0.05	890461	146.08	1309852	144.09
c432	U	1345	0.05	1319	0.04	684995	114.69	830546	86.28
c499-s	U	41934	6.79	14037	1.21	-	-	-	-
c499	U	30722	4.02	23977	3.20	-	-	-	-
c5315-s	U	187191	77.70	131179	42.22	-	-	-	-
c5315	U	133662	42.38	125974	37.87	-	-	-	-
c5315_bug	S	32850	1.94	13109	0.66	-	-	-	-
c6288-s	-	-	-	-	-	-	-	-	-
c6288	-	-	-	-	-	-	-	-	-
c7552-s	U	230642	102.53	217503	94.93	-	-	-	-
c7552	U	250398	120.24	301945	174.39	-	-	-	-
c7552_bug	S	3225	0.18	12827	0.84	-	-	822670	157.92
c880-s	U	21166	5.18	10577	1.17	-	-	-	-
c880	U	21166	5.18	10577	1.17	-	-	-	-
Total		1339291	605.67	1264729	586.14	1575936*	261.01*	2140703*	230.54*

Table 4: Comparison on combinational equivalence checking problems.

Barrel		ZChaff		BrChaff		Simo		BrSimo	
Problem	Sol	Decisions	Time	Decisions	Time	Decisions	Time	Decisions	Time
barrel2	U	3	0.00	3	0.00	-	-	3	0.00
barrel3	U	48	0.01	48	0.01	-	-	31	0.02
barrel4	U	201	0.05	164	0.04	-	-	52	0.10
barrel5	U	8856	1.22	6510	1.57	-	-	7977	15.94
barrel6	U	28110	5.71	30522	12.85	-	-	24463	102.18
barrel7	U	66959	21.20	50730	22.62	-	-	53277	388.52
barrel8	U	116858	55.02	106328	79.50	-	-	75655	999.11
barrel9	U	649532	374.02	342396	366.84	-	-	-	-
Total		870567	457.32	536701	483.03	-	-	-	-

Table 5: Comparison on bounded model checking problems.

Hardnm		ZChaff		BrChaff		Simo		BrSimo	
Problem	Sol	Decis.	Time	Decis.	Time	Decis.	Time	Decis.	Time
hrdnm-L19-01-sat03-915	S	123124	59.60	77950	47.43	128749	69.70	332827	191.34
hrdnm-L19-02-sat03-916	S	360213	182.44	54090	57.70	-	-	963348	548.11
hrdnm-L19-03-sat03-917	S	235179	165.96	163985	105.13	-	-	4078653	1568.44
hrdnm-L22-01-sat03-920	S	-	-	69329	25.29	-	-	-	-
hrdnm-L22-02-sat03-921	S	1106442	830.05	397023	1040.09	-	-	-	-
hrdnm-L22-03-sat03-922	S	-	-	697296	2945.82	-	-	-	-
hrdnm-L23-01-sat03-925	S	470570	517.55	195864	159.58	-	-	-	-
hrdnm-L23-02-sat03-926	S	-	-	320977	527.62	-	-	-	-
hrdnm-L23-03-sat03-927	S	198048	95.70	154258	105.23	-	-	-	-
hrdnm-L25-01-sat03-930	S	-	-	576653	1025.36	-	-	-	-
hrdnm-L25-02-sat03-931	S	-	-	-	-	-	-	-	-
hrdnm-L25-03-sat03-932	S	-	-	830900	2653.42	-	-	-	-
Total		2493576*	1851.30*	1043170*	1515.16*	-	-	-	-

Table 6: Comparison on randomly generated problems.

4 Conclusions

The branching heuristic has a relevant influence on computational behavior of DPLL SAT solvers. Conflict based branching heuristics have the advantages of requiring low computational overhead and of being often able to detect the hidden structure of a problem. The proposed conflict based heuristic, called *reverse assignment sequence* (RAS), has two main aims: to assign at first the more constrained variables, and to reverse every sequence of assignments which have led to a conflict, by satisfying at first clauses which have become empty. RAS heuristic is introduced in two modern DPLL SAT solvers, ZChaff and Simo, in order to verify its empirical behavior. Experiments on several benchmark series, both satisfiable and unsatisfiable, demonstrate advantages of the proposed heuristic.

References

1. R. Bayardo, R. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of 14th National Conference on Artificial Intelligence* (AAAI), 1997.
2. R. Bruni, A. Sassano. Restoring Satisfiability or Maintaining Unsatisfiability by finding small Unsatisfiable Subformulae. In *Procedings of Theory and Applications of Satisfiability Testing* (SAT2001), 2001.
3. M. Buro, H. Kleine Büning. Report on a SAT Competition. *Bulletin of the European Association for Theoretical Computer Science*, 49, 143–151, 1993.
4. V. Chandru and J.N. Hooker. *Optimization Methods for Logical Inference*. Wiley, New York, 1999.
5. S.A. Cook. The Complexity of Theorem-Proving Procedures. In *Proceedings of Third Annual ACM Symposium on Theory of Computing*, 1971.
6. M. Davis, G. Logemann, D. Loveland. A machine program for theorem proving. *Communications of the ACM* 5, 394-397, 1962.
7. M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM* **7**, 201–215, 1960.
8. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and co., San Francisco, 1979.
9. I.P. Gent, H. van Maaren, T. Walsh editors. *SAT 2000*, IOS Press, Amsterdam, 2000.
10. E. Giunchiglia, M. Maratea, A. Tacchella. Look-Ahead vs. Look-Back techniques in a modern SAT solver. In *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing* (SAT2003), 2003.
11. E. Goldberg, Y. Novikov. BerkMin: a Fast and Robust SAT-Solver. In Proceedings of Design Automation & Test in Europe (DATE 2002), 2002.
12. J. Gu, P.W. Purdom, J. Franco, and B.W. Wah. Algorithms for the Satisfiability (SAT) Problem: A Survey. *DIMACS Series in Discrete Mathematics* American Mathematical Society, 1999.
13. M. Herbstritt, B. Becker. Conflict-based Selection of Branching Rules in SAT-Algorithms. In E. Giunchiglia, A. Tacchella eds., *Sixth International Conference on Theory and Applications of Satisfiability Testing -Selected Papers*, LNAI 2919, Springer, 2003.
14. R.E. Jeroslow and J. Wang. Solving Propositional Satisfiability Problems. *Annals of Mathematics and AI* **1**, 167–187, 1990.
15. D. Le Berre, L. Simon. The Essentials of the SAT 2003 Competition. In E. Giunchiglia, A. Tacchella eds., *Sixth International Conference on Theory and Applications of Satisfiability Testing -Selected Papers*, LNAI 2919, Springer, 2003.
16. P. Liberatore. On the complexity of choosing the branching literal in DPLL. *Artificial Intelligence*, 116(1-2):315326, 2000.
17. J.P. Marques-Silva, K.A. Sakallah. Conflict Analysis in Search Algorithms for Propositional Satisfiability. In *Proceedings of IEEE International Conference on Tools with Artificial Intelligence*, 1996.
18. J. P. Marques-Silva. The Impact of Branching Heuristics in Propositional Satisfiability Algorithms. In Proceedings of the 9th Portuguese Conference on Artificial Intelligence (EPIA), 1999.
19. M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, S. Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 39th Design Automation Conference*, 2001.
20. K. Truemper. *Effective Logic Computation*. Wiley, New York, 1998.
21. L. Zhang, S. Malik. The Quest for Efficient Boolean Satisfiability Solvers. In *Proceedings of CADE 2002 and CAV 2002*, 2002.
22. H. Zhang and M.E. Stickel. Implementing the Davis-Putnam Method. In I.P. Gent, H. van Maaren, and T. Walsh eds. *SAT 2000*, IOS Press, Amsterdam, 2000.