Improving Backtrack Search for SAT by Means of Redundancy

Laure Brisoux, Éric Grégoire, and Lakhdar Saïs

CRIL – Université d'Artois
rue de l'Université SP16
F-62037 Lens Cedex,
France.
{brisoux,gregoire,sais}@cril.univ-artois.fr

Abstract. In this paper, a new heuristic that can be grafted to many of the most efficient branching strategies for Davis and Putnam procedures for SAT is described. This heuristic gives a higher weight to clauses that have been shown unsatisfiable at some previous steps of the search process. It is shown efficient for many classes of SAT instances, in particular structured ones.

1 Introduction

For many years, the problem of propositional satisfiability (SAT) has received little attention in spite of the fact that it is recognized as a central issue in many areas of artificial intelligence and other computer science domains. Recently, there has been a surge in interest in designing new computational techniques to address it. On the one hand, very simple stochastic search techniques have proved efficient in solving large and hard consistent SAT problems (see e.g. [26, 25] [1] [20]). On the other hand, several people have tried to improve the best logically complete techniques for SAT. Currently, the most efficient logically complete techniques are still based on the classical Davis and Putnam procedure[11]. In this paper we refer to the Davis-Logemman-Loveland procedure [10] (in short DPLL). Some new efficient versions of DPLL have been proposed recently, extending its practical scope to a really significant extent. Among them, let us simply mention some of the most efficient ones, namely C-SAT [3] [13], Tableau [9], POSIT [16], Satz [5], Relsat [2], GRASP [19] and DP+TSAT [21, 22]. Their efficiency is based on several principal ingredients. First, a smart use of data structures and programming skills can pay a lot with respect to this particular problem. Some adequate local treatment before and during the search can allow for better results for many classes of problems, in particular in detecting local inconsistencies. Last but not least, the quality of the branching strategy appears to be the key for good performance results. In this paper, it is shown that this strategy can advantageously include a new heuristic, that gives a higher weight to clauses that appeared unsatisfiable at some previous steps of the search process.

Such an heuristic avoids the extra-space and computation required by techniques that use additional ingredients to avoid repetitive search (e.g. adding clauses when unsatisfiability is reached at some level of the search tree [19, 2, 17]).

The paper is organized as follows. First, the addressed formal framework is briefly recalled and DPLL with its main computational features are reviewed. Then this new heuristic is motivated and presented. Experimental results on various standard benchmarks are given to illustrate its efficiency[1]. Finally, the limits and the possible extensions of the approach, together with promising further possible paths of research are discussed.

2 Backtrack Search SAT Algorithms

Let us first recall the formal framework under consideration. SAT consists in checking the satisfiability of a boolean formula in conjunctive normal form (CNF). A CNF formula is a set (interpreted as a conjunction) of clauses, where a clause is a disjunction of literals. A literal is a positive or negated propositional variable.

An interpretation of a boolean formula is an assignment of truth values to its variables. A model is an interpretation that satisfies the formula. Accordingly, SAT consists in finding a model of a CNF formula when such a model does exist or in proving that such a model does not exist. SAT is an NP-complete problem meaning that all algorithm to solve it should be exponential in the worst cases, unless $P = NP$ [7].

However, not all SAT instances do exhibit the same difficulty, with respect to usual algorithms to solve it. Theoretical and experimental results even show good average-case performance for several classes of SAT instances (see e.g. [15]). In the following, we shall refer to a variety of benchmarks for SAT [12][2] and hard random 3-SAT instances, i.e. instances from a random generation model where the number of literals per clause is 3, the sign of each literal is randomly generated (with a probability 0.5) and the ratio of variables to clauses is 4.25 ([27], [9], [4] and [14]). Those last problems prove exponential for resolution [6].

Recently, very simple stochastic search techniques have proved efficient in solving large and hard consistent SAT problems, in particular the above mentioned hard random 3-SAT consistent instances (see e.g. [26, 25] [1] [28] [20]). However, such techniques are logically incomplete since they do not cover the whole search space. Accordingly, they cannot be used as such to prove the inconsistency of SAT instances (see however [21]). Actually, the most efficient logically complete techniques are still based on the classical DPLL procedure (see Fig. 1) [10, 11].

---

[1] Our system is available from `http://www.lifl.fr/~brisoux`

[2] These benchmarks are available from `http://dimacs.rutgers.edu/Challenges/index.html`.

```
Procedure DPLL(S)
  Input :  a set S of clauses
  Output :  a satisfying truth assignment of S if found,
  or a definitive statement that S is inconsistent.
  Begin
    Propagation of unit literals;
    if the empty clause is generated then return (false);
    else if all variables are assigned then return (true)
      else begin
        p := some unassigned literal selected by a branching rule ;
      return (DPLL(S ∧ p ) ∨ DPLL(S ∧ ¬p)) ;
      end ;
  end ;
end ;
```

Fig. 1. DPLL procedure

Some new efficient versions of DPLL have been proposed recently, extending its practical scope to a really significant extent. Among them, let us simply mention the most efficient ones, namely C-SAT [3] [13], Tableau [9], POSIT [16], Satz [5], Relsat [2], GRASP [19] and DP+TSAT [21, 22]. One of the key feature for efficiency in the DPLL procedure lies in its branching rule strategy. Accordingly, many branching rules have been proposed in the literature. Let us review the most efficient of them.

It is widely accepted that literals occurring in the shortest clauses should be preferred to some extent. Actually, for each unassigned literal l, a score $f(l)$ is defined as follows. First, a weight is assigned to each clause translating the above idea. Several measures for this weight have been proposed. Mainly, [18] proposed 2^{-r}, where r is the length of the clause. On the other hand, empirical results led Dubois and Boufkhad to adopt $-\ln(1-\frac{1}{(2^r-1)^2})$ as the recommended weight, increasing the weight difference between clauses of various lengths. Let S be the set of clauses.

$$f(l) = \sum_{\forall c \in S \ s.t. \ l \in c} weight(c)$$

Then the final score of a variable x is given by

$$A(x) = f(x) + f(\neg x) + \alpha \times \min(f(x), f(\neg x)) \tag{1}$$

(where α is empirically set to 1.5 [3]).

Following close motivations, Freeman [16] proposed

$$A'(x) = f(x) + f(\neg x) + \alpha' \times f(x) \times f(\neg x) \tag{2}$$

(where α' is empirically set to 1024).

Let us note that the role of the $\alpha \times \min(f(x), f(\neg x))$ and $\alpha' \times f(x) \times f(\neg x)$ terms is to balance the search tree.

These $A(x)$ and $A'(x)$ functions that are be maximized, play the role of branching rules and prove very efficient. This family of branching rules includes many variants. Let us mention Rauzy's FFIS [24] (First Failed In Shortened clauses) and also variants that attempt to explore at a minimal cost the effect of further additional resolution steps [3] or to even more exploit the power of unit propagation [5].

3 A New Heuristic

DPLL is mainly devised in such way that it attempts to cover the whole search space, exhibiting a model or showing that any branch leads to unsatisfiability.

Accordingly, the intuition behind the heuristic we shall propose is not covered by the above branching strategy. Intuitively, it is as follows.

Whenever some clauses have been shown unsatisfiable at some steps of the search process, this information should not be neglected in the remaining search process. On the contrary, in the development of the other branches of the search tree it could be efficient to try to encounter these situations of unsatisfiability again, everywhere and as soon as possible, modulo the other factors allowing an efficient branching rule to be obtained. This can be done by selecting with a higher priority the literals occurring in these clauses.

Interestingly enough, such an heuristic can be grafted to the above efficient branching strategies. Formally, it is translated through a numerical factor β_c that is introduced in the function f in the following way.

A new generic selection function f is thus given by

$$f(l) = \sum_{\forall c \in s \ s.t. \ l \in c} \beta_c \ weight(c)$$

where β_c is set to 1, initially.

Each time DPLL selects a propositional variable that would immediately lead to inconsistency, each initial clause c of the SAT instance that would be shown unsatisfiable at this step of the search tree has its factor β_c increased by a given value γ, and its importance is thus increased in the further search .

Let us stress that such a transformed evaluation function requires an extra computing cost that is negligible and does not require additional clauses to be recorded. The β_c weight expresses the degree of redundancy of a clause c. Intuitively, increasing the weight of clauses that have been previously shown unsatisfiable direct the search on the probable inconsistent kernels.

In the following section, we give γ the best experimental value with respect to several classes of SAT instances.

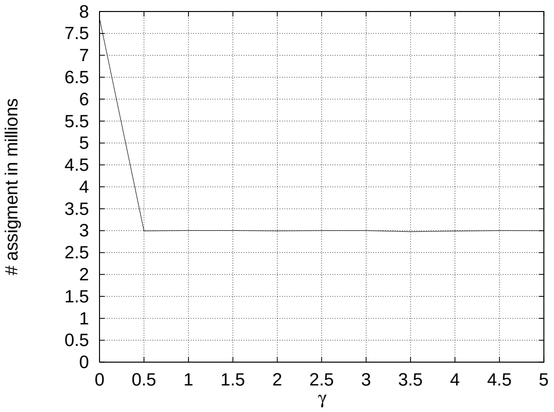


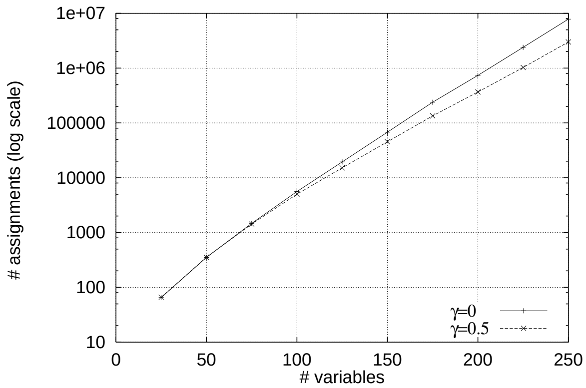
Fig. 2. Average number of assignments w.r.t. γ

4 Experimental Results

The efficiency of most DPLL procedures does not depend on the branching strategy only but also depends on the following other principal ingredients. First, a smart use of data structures and programming skills can pay a lot with respect to this particular problem. Efficient implementations often rely on a code that is often really intricate. Also, some adequate local treatment before and during the search can allow for better results for many classes of problems. In particular, it can allow us to detect inconsistencies that are local in the sense that they can be related to a small subset of clauses in the SAT instance. For instance, the execution of a local search procedure can be inserted as a first step of DPLL (see [20], [8]). This step can provide us a model or even allow for the further detection of inconsistencies that are local in the above sense. Also, a limited number of resolution steps at the root node can allow some of these inconsistencies to be detected. Accordingly, all these facets play a role in the efficiency of the satisfiability checking program.

As the heuristic described in this paper can be grafted to most branching strategies, we have experimented it with the basic one presented above, inside a straightforward implementation, allowing us to make abstraction of coding tricks. Accordingly, it should be clear that the experimental results in this paper are not expected to defeat the performance of the best SAT systems, but simply to illustrate the gain that most systems (including the best ones) could obtain, using this additional heuristic. Let us also stress that this heuristic does not intervene with local treatment at the root.

First, the heuristic has been experimented in the framework of hard random 3-SAT instances. In Figure 2, the average number of assignments obtained for various values of γ is given for 200 3-SAT instances at the threshold with 250 variables, using the standard Jeroslow and Wang 2^{-r} rule weight inside the score function $A(x) = f(x) + f(\neg x) + 1.5 \times \min(f(x), f(\neg x))$. A similar behaviour has been obtained for other 3-SAT hard instances using Boufkhad's weight rule and Freeman's score rule. A significant performance improvement is obtained when this new additional heuristic is used. Quite surprisingly, the gain does not increase in a significant way whenever γ is more than 0.5. We are currently investigating the theoretical explanation of this feature. Accordingly, we settled γ to 0.5.



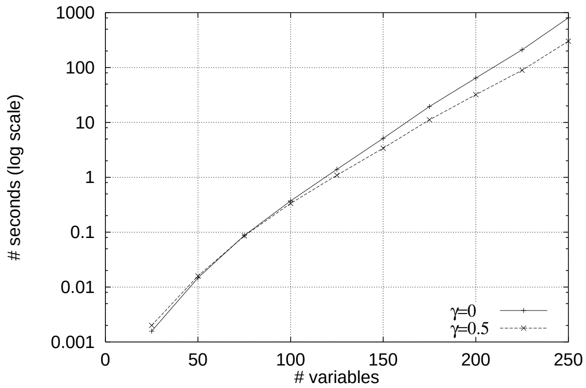


Fig. 3. Results for 3-SAT instances using the $A(x)$ branching rule (1)

In Figure 3, average results for 3-SAT instances[3] (200 instances with a variables/clauses ratio being 4.25) are illustrated; the effect of using the additional heuristic with γ set to 0.5 on the score function $A(x)$ integrating Jeroslow and Wang weight rules and the instances as above is given. Similar figures are ob-

[3] These experimentations have been conducted on 133 Pentium PC's.

tained for Freeman's score rule (2) using the same weight rule. In any case, a significant performance improvement is obtained using the heuristic.

Table 1. Some results for DIMACS instances: weight vs. no-weight results

Dimacs		**Size**		$A(x)$ branching rule (1)			
				$\gamma = 0$		$\gamma = 0.5$	
Instances	Sat.	# Var.	#Cla.	# assignments	seconds	# assignments	seconds
par16-1	Y.	1015	3310	3327186	73.85	898435	28.33
par16-4-c	Y.	324	1292	587249	23.76	282760	12.91
aim-100-2_0-no.1	N.	100	200	27056964	944.58	589	0.06
aim-200-1_6-no-2	N.	200	320	***	> 7200	1459555	211.91
hanoi4	Y.	718	4934	***	> 7200	35899827	3114.00
ssa2670-130	N.	1359	3321	***	> 7200	9023846	483.18
ssa2670-141	N.	986	2315	***	> 7200	16808521	594.90
bf1355-075	N.	2180	6778	***	> 7200	30124	5.36
bf1355-638	N.	2177	4768	***	> 7200	16737	3.48
f7hh.15	Y.	5315	140258	***	> 14400	442332	1073.40
f8h_10	N.	2459	25290	3562510	2159.16	172344	116.75
ii16e1	Y.	1245	14766	***	> 7200	77465	118.98
ii32d2	Y.	404	3153	20630465	5462.36	39993	12.16
2bitadd_12	Y.	708	1702	15508567	1363.66	1413216	109.71
e0ddr2_10by5	Y.	19500	103887	25681427	5498.15	905054	278.16
bw_large.d	Y.	6325	131973	***	> 172800	158265845	73829.20
logistics.d	Y.	2160	27242	***	> 172800	***	> 172800

Dimacs		**Size**		$A'(x)$ branching rule (2)			
				$\gamma = 0$		$\gamma = 0.5$	
Instances	Sat.	# Var.	# Cla.	# assignments	seconds	# assignments	seconds
par16-1	Y.	1015	33.6	3327186	74.81	757032	24.98
par16-4-c	Y.	324	1292	587249	23.30	250183	11.81
aim-100-2_0-no-1	N.	100	200	27703970	932.16	586	0.06
aim-200-1_6-no-2	N.	200	320	***	> 7200	1459555	112.98
hanoi4	Y.	718	4934	***	> 7200	43012777	3892.86
ssa2670-130	N.	1359	3321	***	> 7200	9023846	385.38
ssa2670-141	N.	986	2315	***	> 7200	16808521	628.26
bf1355-075	N.	2180	6778	***	> 7200	29129	5.00
bf1355-638	N.	2177	4768	***	> 7200	16706	3.55
f7hh.15	Y.	5315	140258	***	> 14400	***	> 14400
f8h_10	N.	2459	25290	3562510	2159.16	66625	41.60
ii16e1	Y.	1245	14766	1023	7.05	1023	7.05
ii32d2	Y.	404	3153	87651	23.18	5925	1.76
2bitadd_12	Y.	708	1702	***	> 7200	***	> 7200
e0ddr2_10by5	Y.	19500	103887	19987285	4311.01	1012918	318.45
bw_large.d	Y.	6325	131973	****	> 172800	118425586	59229.61
logistics.d	Y.	2160	27242	****	> 172800	11617271	18196.40

Then, this heuristic has been experimented in an extensive manner with respect to structured problems[4] , using the standard DIMACS benchmarks [12] and the Kautz and Selman's planning instances[5]. Some results are given in Table 1 and discussed below. We kept 0.5 as value for γ. Extensive results on DIMACS benchmarks are available from the authors[6] .

[4] These experimentations have been conducted on 166 Pentium PC's.

[5] available from `ftp://ftp.research.att.com/dist/ai`

[6] `http://www.lifl.fr/~brisoux`

Clearly enough, the empirical results confirm the intuition. Performance improvement is obtained with respect to most (consistent and inconsistent) problems classes. Such a result is even more important for many classes of structured problems. Although our implementation of DPLL on a 166 Pentium is a straightforward one that does not use efficient coding tricks and extra features leading to greater efficiency, it is worth noting that it already performs better for many DIMACS instances [12] than advanced platforms like C-SAT [3], GRASP [19], POSIT [16], Satz [5] and Relsat [2] (comparing the published computing times).

Interestingly very good results are obtained on hard planning problems and confirms new perspectives on handling these problems in an efficient way, using SAT related-techniques.

In conclusion, it proves thus efficient to attempt to direct DPLL towards already discovered inconsistencies in the search process in most cases.

5 Conclusions

Clearly, the DPLL procedure is oriented towards the proof of inconsistency. Branching rules attempt to shorten the search tree as much as possible. Inconsistencies detected in such trees do not necessary rely on the whole path of instantiated variables leading to them. On the contrary, a same cause of inconsistency can appear several time in the search tree. Accordingly, we propose to favor the selection of literals occurring in clauses that have been shown unsatisfiable at some previous steps of the search process without the need of recording additional clauses. Interestingly enough, this can be grafted at almost no computing cost to most existing branching rules, allowing us to balance the importance of this feature with other criteria that are necessary for computational efficiency.

6 Acknowledgments

This work has been supported in part by a *Contrat d'objectif de la Région Nord/Pas-de-Calais*. Many thanks to our colleagues at CRIL, in particular O. Bailleux, Y. Boufkhad, P. Marquis and B. Mazure for fruitful discussions about this paper.

References

1. *Artificial Intelligence*, volume 81, March 1996. Special volume on frontiers in problem solving: phase transition and complexity.
2. R. J. Bayardo Jr. and R. C. Schrag. Using csp look-back techniques to solve real-world SAT instances. In *Proc. of the Fourteenth National Conference on Artificial Intelligence (AAAI'97)*, pages 203–208, 1997.
3. Y. Boufkhad. *Aspects propabilistes et algorithmiques du problème de satisfiabilité*. PhD thesis, Université de Paris VI, 1996.

4. P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the really hard problems are. In *Proc. of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI'91)*, pages 163–169, 1991.
5. Chumin Li and Anbulagan. Heuristic based on unit propagation for satisfiability problems. In *Proc. of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 366–371, 1997.
6. V. Chvátal and E. Szemerédi. Many hard examples for resolution. *ACM*, 35(4):759–768, 1988.
7. S. Cook. The complexity of theorem proving procedures. In *Proc. of Third Annual ACM Symp. on Th. of Computing*, pages 151–158, 1971.
8. J. M. Crawford. Solving satisfiability problems using a combinaison of systematic an local search, 1993. working notes of the DIMACS challenge on SAT organized bye the Center for Discrete Mathematics and Computer Science of Rutgers University.
9. J. M. Crawford and L. D. Auton. Experimental results on the crossover point in random 3-SAT. In *Artificial intelligence*, volume 81, pages 31–57, 1996.
10. M. Davis, G. Logemann, and D. Loveland. A Machine Program for Theorem Proving. *CACM*, 5:394–397, 1962.
11. M. Davis and H. Putnam. A Computing Procedure for Quantification Theory. *Journal of the Association for Computing Machinery*, 7:201–215, 1960.
12. Second SAT challenge organized bye the Center for Discrete Mathematics and Computer Science of Rutgers University, 1993. The benchmarks used in our tests can be obtained by anonymous ftp from Rutgers University Dimacs Center : ftp dimacs.rutgers.edu cd pub/challenge/sat/benchmarks/cnf .
13. O. Dubois, , P. André, Y. Boufkhad, and J. Carlier. SAT versus UNSAT. *[28]*, pages 415–436, 1996.
14. O. Dubois and J. Carlier. Probabilistic Approach to the Satisfiability Problem. *Journal of Theoretical Computer Science*, 81:65–75, 1991.
15. J. Franco and M. Paull. Probabilistic Analysis of the Davis and Putnam procedure for Solving the Satisfiability Problem. In *Discrete Applied Math.*, volume 5, pages 77–87, 1983.
16. J. W. Freeman. *Improvement to propositional satisfiablity search.* PhD thesis, Univ. of Philadelphia, 1995.
17. Jin-Kao Hao and Laurent Tétart. CH-SAT: A Complete Heuristic Procedure for Satisdiability Problems. In *proceedings of the ECAI'96 Workshop on Advances in Propositional Deduction*, pages 27–38, Budapest, Hungary, 13 August 1996.
18. R. G. Jeroslow and J. Wang. Solving propositional satisfiability problems. In *Annals of Mathematics and Artificial Intelligence*, pages 167–187, 1990.
19. João P. Marques Silva. An Overview of Backtrack Search Satisfiability Algorithms. In *Proc. of the Fifth International Symposium on Artificial Intelligence and Mathematics*, Fort Lauderdale, Florida, January 1998. available on `http://rutcor.rutgers.edu/~amai/Proceedings.html`.
20. B. Mazure, L. Saïs, and É. Grégoire. Tabu search for SAT. In *Proc. of the Fourteenth National Conference on Artificial Intelligence (AAAI'97)*, pages 281–285, Rhodes Island, 1997.
21. B. Mazure, L. Saïs, and É. Grégoire. Boosting complete technique thanks to local search methods. In *Annals of Mathematics and Artificial Intelligence*, 1998.
22. B. Mazure, L. Saïs, and É. Grégoire. CRIL Platform for SAT. In *Proc. of the Fifteenth International Conference on Automated Deduction(CADE-15)*, 1998.
23. P. Morris. The breakout method for escaping from local minima. In *Proc. of the Eleventh National Conference on Artificial Intelligence (AAAI'93)*, 1993.

24. A. Rauzy. On the complexity of the david and putnam's procedure on some polynomial sub-classes of sat. Technical Report LaBRI 806–94, Université de Bordeaux 1, 1994.
25. B. Selman, H. A. Kautz, and B. Cohen. Local Search Strategies for satisfiability Testing. In *[28]*, pages 521–531, 1996.
26. B. Selman, D. Mitchell, and H. Levesque. A New Method for Solving Hard Satisfiability Problems. In *Proc. of the Tenth National Conference on Artificial Intelligence (AAAI'92)*, 1992.
27. B. Selman, D. Mitchell, and H. Levesque. Hard and easy distributions of SAT problems. In *Proc. of the Tenth National Conference on Artificial Intelligence (AAAI'92)*, pages 459–465, San Jose, CA, July 1992.
28. M. Trick and D. Johnson. Cliques, Coloring and Satisfiability. In *DIMACS Serie in Discrete Mathematics and Theorical Computer Science*, volume 26, 1996.