Conflict Analysis in Search Algorithms for Satisfiability

João P. Marques Silva
Cadence European Laboratories
IST/INESC
1000 Lisboa, Portugal

Karem A. Sakallah
Department of EECS
University of Michigan
Ann Arbor, Michigan 48109-2122

Abstract

This paper introduces GRASP (Generic seaRch Algorithm for the Satisfiability Problem), a new search algorithm for Propositional Satisfiability (SAT). GRASP incorporates several search-pruning techniques, some of which are specific to SAT, whereas others find equivalent in other fields of Artificial Intelligence. GRASP is premised on the inevitability of conflicts during search and its most distinguishing feature is the augmentation of basic backtracking search with a powerful conflict analysis procedure. Analyzing conflicts to determine their causes enables GRASP to backtrack non-chronologically to earlier levels in the search tree, potentially pruning large portions of the search space. In addition, by "recording" the causes of conflicts, GRASP can recognize and preempt the occurrence of similar conflicts later on in the search. Finally, straightforward bookkeeping of the causality chains leading up to conflicts allows GRASP to identify assignments that are necessary for a solution to be found. Experimental results obtained from a large number of benchmarks indicate that application of the proposed conflict analysis techniques to SAT algorithms can be extremely effective for a large number of representative classes of SAT instances.

1 Introduction

The propositional satisfiability problem (SAT) appears in many contexts in Artificial Intelligence, including Constraint Satisfaction and Automated Reasoning. SAT has also been extensively studied and applied in other fields of Computer Science, as for example in Design Automation of Digital Electronic Circuits. Though well-researched and widely investigated, it remains the focus of continuing interest because efficient techniques for its solution can have great theoretical and practical impact. Over the years, many algorithmic solutions have been proposed for SAT, the most well known being the different variations of the Davis-Putnam procedure [3]. The best known version of this procedure is based on a backtracking search algorithm that, at each node in the search tree, elects an assignment and prunes subsequent search by iteratively applying the *unit clause* and the *pure literal* rules.

Most of the recently proposed improvements to the basic Davis-Putnam procedure [2, 5, 8, 11] can be distinguished based on their decision making heuristics or their use of preprocessing or relaxation techniques. Common to all these approaches, however, is the chronological nature of backtracking. Nevertheless, non-chronological backtracking techniques have been extensively studied and applied to different areas of Artificial Intelligence, particularly Truth Maintenance Systems (TMS), Constraint Satisfaction Problems (CSP) and Automated Reasoning, in some cases with very promising experimental results. (Bibliographic references to work in these areas can be found in [10].) In recent years, extensive research has been directed towards the development of local-search algorithms for SAT [9]. Generally, these algorithms are incomplete, i.e. they may not find a solution and cannot prove unsatisfiability of an instance of SAT. Nevertheless, local-search algorithms have been shown to be extremely effective on specific classes of *satisfiable* instances of SAT.

This paper introduces GRASP (***Generic seaRch Algorithm for the Satisfiability Problem***), a new search algorithm for SAT. GRASP incorporates several search-pruning techniques, some of which are specific to SAT, whereas others find equivalent in other fields of Artificial Intelligence. Experimental results obtained from a large number of benchmarks [6] indicate that application of the proposed conflict analysis techniques to SAT algorithms can be extremely effective for a large number of representative classes of SAT instances.

Several features distinguish the conflict analysis procedure in GRASP from others used in TMSs and CSPs. First, conflict analysis in GRASP is tightly coupled with Boolean Constraint Propagation (BCP) [5] and the causes of conflicts need not necessarily correspond to decision assignments. Second, clauses can be added to the original set of clauses, and the number and size of added clauses is user-controlled. This is in explicit contrast to nogood recording techniques developed for TMSs and CSPs. Third, GRASP employs techniques to prune the search by analyzing the implication *structure* generated by BCP. Exploiting the "anatomy" of conflicts in this manner has no equivalent in other areas.

The remainder of this paper is organized in four sections. In Section 2, we describe the overall architecture of GRASP. Further details of the algorithm can be found in [10]. A large number of experimental results on a wide range of benchmarks are presented and analyzed in Section 3. In particular, GRASP is shown to outperform several state-of-the-art SAT algorithms [1, 2, 4, 5, 8, 9, 11, 7] on most, but not all, benchmarks. Furthermore, the experimental results shown strongly suggest that for several practical classes of instances of SAT, local-search algorithms may be inadequate. This is particularly significant whenever the instances of SAT

```
// Global variables:     CNF formula φ
//                       Variable assignment A
// Return value:         FAILURE or SUCCESS
// Auxiliary variables:  Backtracking level β
//
GRASP()
{
    return (Search (0, β) != SUCCESS) ?
        FAILURE : SUCCESS;
}

// Input argument:      Current decision level d
// Output argument:     Backtracking level β
// Return value:        CONFLICT or SUCCESS
//
Search (d, &β)
{
    if (Decide (d) == SUCCESS)
        return SUCCESS;
    while (TRUE) {
        if (Deduce (d) != CONFLICT) {
            if (Search (d + 1, β) == SUCCESS)
                return SUCCESS;
            else if (β != d) {
                Erase(); return CONFLICT;
            }
        }
        if (Diagnose (d, β) == CONFLICT) {
            Ease(); return CONFLICT;
        }
        Erase();
    }
}
```

Figure 1: Description of GRASP

are likely to be unsatisfiable. This is usually the case, for example, in Automated Theorem Proving and in several Electronic Design Automation tasks.

2 Search Algorithm Template

The general structure of the GRASP search algorithm is shown in Figure 1. We assume that an initial CNF formula φ and an initial assignment A of variable-value pairs are given at decision level 0. This initial assignment, which may be empty, may be viewed as an additional problem constraint and causes the search to be restricted to a subcube of the n-dimensional Boolean space. As the search proceeds, both φ, A and the decision level are modified. The recursive search procedure consists of four major operations:

1. `Decide()`, which chooses a decision assignment at each stage of the search process. Decision procedures are commonly based on heuristic knowledge. For the results given in Section 3, the following greedy heuristic is used:
   At each node in the decision tree evaluate the number of clauses directly satisfied by each assignment to each variable. Choose the variable and the assignment that directly satisfies the largest number of clauses.
   Other decision making procedures have been incorporated in GRASP, as described in [10].
2. `Deduce()`, which implements BCP and (implicitly) maintains the resulting implication graph.
3. `Diagnose()`, which identifies the causes of conflicts, can backtrack non-chronologically and can augment the CNF formula with additional clauses. (See [10] for the details of `Deduce()` and `Diagnose()`.)
4. `Erase()`, which deletes the assignments at the current decision level.

We refer to `Decide()`, `Deduce()` and `Diagnose()` as the *Decision*, *Deduction* and *Diagnosis* ***engines***, respectively. Different realizations of these engines lead to different SAT algorithms. For example, the Davis-Putnam procedure can be emulated with the above algorithm by defining a decision engine, requiring the deduction engine to implement BCP and the pure literal rule, and organizing the diagnosis engine to implement chronological backtracking.

3 Experimental Results

In this section we present experimental results for GRASP. Several benchmarks are used and GRASP is compared with other state-of-the-art and publicly available SAT programs [1-5, 7-9, 11]. In all cases, either the source code or the executable was provided by the respective author.

GRASP is implemented in the C++ programming language and was compiled with GCC 2.7.2. The CPU times for all programs were scaled to the CPU times on a SUN SPARC 5/85 machine. All SAT programs were run with a CPU time limit of 10,000 seconds. In order to evaluate the different programs, the DIMACS and UCSC benchmarks were used [6]. The UCSC benchmarks represent one practical application of SAT algorithms to the field of Electronic Design Automation, thus being of key significance for experimentally evaluating SAT algorithms.

For the results shown below, GRASP was configured to use the decision engine described in Section 2, to implement non-chronological backtracking and to limit the size of recorded clauses to 20 or fewer literals. (Additional configuration details can be found in [10].)

For the tables of results the following definitions apply. A benchmark suite is partitioned into classes of related benchmarks. In each class, **#M** denotes the total number of class members; **#S** denotes the number of class members for which each program terminated in less than the allowed 10,000 CPU seconds; and **Time** denotes the total CPU time, in seconds, taken to process all members of the class.

The results obtained for the DIMACS and the UCSC benchmarks are shown in Table 1. For the DIMACS benchmarks we can conclude that GRASP performs better than the other algorithms in a large number of classes of benchmarks. Furthermore, for the UCSC benchmarks GRASP performs significantly better than all the other programs, all of which abort a large number of problem instances and require much larger CPU times. The UCSC benchmarks are characterized by extremely sparse CNF formulas for which the conflict analysis procedure of GRASP works particularly

Benchmark Class	#M	GRASP[10]		POSIT[5]		C-SAT[4]		H2R[8]		NTAB[2]		SATO[7]		TEGUS[11]		DPL[1,3]		GSAT[9]	
		#S	Time	#S	Time	#S	Time	#S	Time	#S	Time	#S	Time	#S	Time	#S	Time	#S	Time
AIM-100	24	24	1.8	24	1,290	24	4.0	23	21,571	18	39,569	20	60,390	24	107.9	21	58,510	n/a	n/a
AIM-200	24	24	10.8	24	117,991	24	4.2	9	150,004	11	69,410	9	150,095	23	14,059	9	156,196	n/a	n/a
BF	4	4	7.2	4	20,037	1	30,634	3	10,200	2	27,900	1	35,695	2	26,654	0	40,000	n/a	n/a
DUBOIS	13	13	34.4	13	77,189	4	95,485	7	73,729	5	47,952	7	71,528	5	90,333	5	96,977	n/a	n/a
II32	17	17	7.0	17	650.1	17	454.7	14	36,029	17	697.0	16	10,004	17	1,231	15	21,520	9	83,814
PRET	8	8	18.2	8	40,691	4	41,398	4	40,342	0	80,000	4	40,430	4	42,579	4	41,429	n/a	n/a
SSA	8	8	6.5	8	85.3	7	15,781	6	20,006	6	20,024	5	30,092	6	20,230	0	80,000	n/a	n/a
AIM-50	24	24	0.4	24	0.4	24	47.2	24	2.3	24	24.3	24	12.7	24	2.2	24	10.7	n/a	n/a
II8	14	14	23.4	14	2.3	13	10,118	11	30,005	13	11,411	14	0.4	14	11.8	7	84,189	12	27,647
JNH	50	50	21.3	50	0.8	50	30.3	50	5.8	50	10.9	50	11.0	50	6,055	50	40.0	n/a	n/a
PAR8	10	10	0.4	10	0.1	10	3.2	10	0.6	10	0.7	10	0.2	10	1.5	10	0.8	5	50,005
PAR16	10	10	9,844	10	72.1	10	824.4	10	264.8	10	591.5	10	10,447	10	9,983	10	11,741	0	100,000
II16	10	9	10,311	9	10,120	5	50,545	3	75,940	6	10,126	7	85,522	10	269.6	2	83,933	9	11,670
HANOI	2	1	14,480	1	10,117	1	16,550	1	10,733	1	15,840	0	20,000	1	11,641	0	20,000	0	20,000
HOLE	5	4	12,704	4	937.9	5	1,025	4	11,182	5	1,244	5	362.2	3	21,301	4	11,404	n/a	n/a
F	3	0	30,000	0	30,000	0	30,000	0	30,000	0	30,000	0	30,000	0	30,000	0	30,000	0	30,000
G	4	0	40,000	0	40,000	0	40,000	0	40,000	0	40,000	0	40,000	0	40,000	0	40,000	2	20,079
PAR32	10	0	100,000	0	100,000	0	100,000	0	100,000	0	100,000	0	100,000	0	100,000	0	100,000	0	100,000
BF0432	21	21	47.6	21	55.8	21	1,861	21	101.4	21	1,863	1	207,036	19	53,852	0	210,000	n/a	n/a
BF1355	149	149	125.7	64	946,127	61	914,863	121	352,175	58	975,862	0	1490,000	53	993,915	0	1490,000	n/a	n/a
BF2670	53	53	68.3	53	2,971	43	154,565	41	143,357	30	253,336	8	450,992	25	295,410	0	530,000	n/a	n/a
SSA0432	7	7	1.1	7	0.2	7	32.0	7	2.2	7	103.9	0	70,000	7	1,593	0	70,000	n/a	n/a
SSA2670	12	12	51.5	12	2,826	6	95,355	0	120,000	0	120,000	0	120,000	0	120,000	0	120,000	n/a	n/a
SSA6288	3	3	0.2	3	0.0	3	15.9	3	7.9	3	13.3	0	30,000	3	17.5	0	30,000	n/a	n/a
SSA7552	80	80	19.8	80	60.0	78	20,217	80	97.5	80	166.0	1	791,076	80	3406	0	800,000	n/a	n/a

Table 1: Results on the DIMACS and UCSC SAT benchmarks

well.

Additional experiments measuring the effect of non-chronological backtracking and clause recording on the amount of search conducted by GRASP can be found in [10].

4 Conclusions

This paper describes a configurable algorithmic framework for solving SAT that incorporates procedures for conflict analysis. Experimental results indicate that conflict analysis and its by-products, non-chronological backtracking and identification of equivalent conflicting conditions, can contribute decisively for efficiently solving a large number of classes of instances of SAT. For this purpose, the proposed SAT algorithm is compared with other state-of-the-art algorithms.

References

[1] P. Barth, "A Davis-Putnam Based Enumeration Algorithm for Linear pseudo-Boolean Optimization," Technical Report MPI-I-95-2-003, Max-Planck-Institut für Informatik, 1995.

[2] J. Crawford and L. Auton, "Experimental Results on the Cross-Over Point in Satisfiability Problems," in *Proc. National Conference on Artificial Intelligence (AAAI-93)*, pp. 22-28, 1993.

[3] M. Davis and H. Putnam, "A Computing Procedure for Quantification Theory," *Journal of ACM*, vol. 7, pp. 201-215, 1960.

[4] O. Dubois, P. Andre, Y. Boufkhad and J. Carlier, "SAT versus UNSAT," *Second DIMACS Implementation Challenge*, D. S. Johnson and M. A. Trick (eds.), 1993.

[5] J. W. Freeman, *Improvements to Propositional Satisfiability Search Algorithms*, Ph.D. Dissertation, CIS Department, University of Pennsylvania, May 1995.

[6] D. S. Johnson and M. A. Trick (eds.), *Second DIMACS Implementation Challenge*, 1993. DIMACS benchmarks available in ftp://Dimacs.Rutgers.EDU/pub/challenge/sat/benchmarks/cnf. UCSC benchmarks available in /pub/challenge/sat/contributed/UCSC.

[7] S. Kim and H. Zhang, "ModGen: Theorem proving by model generation," in *Proc. National Conference of American Association on Artificial Intelligence (AAAI-94)*, pp. 162-167, 1994.

[8] D. Pretolani, "Efficiency and Stability of Hypergraph SAT Algorithms," *Second DIMACS Implementation Challenge*, D. S. Johnson and M. A. Trick (eds.), 1993.

[9] B. Selman, H. J. Levesque and D. Mitchell, "A New Method for Solving Hard Satisfiability Problems," in *Proc. National Conference on Artificial Intelligence (AAAI-92)*, 1992.

[10] J. P. M. Silva and K.A. Sakallah, "Conflict Analysis in Search Algorithms for Satisfiability," Technical Report RT-4-96, INESC, May 1996.

[11] P. R. Stephan, R. K. Brayton and A. L. Sangiovanni-Vincentelli, "Combinational Test Generation Using Satisfiability," Memorandum no. UCB/ERL M92/112, EECS Department, University of California at Berkeley, October 1992.