Parallel Execution of Stochastic Search Procedures on Reduced SAT Instances*

Wenhui Zhang, Zhuo Huang, and Jian Zhang

Key Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences
P.O.Box 8718, Beijing, China
{zwh,hz,zj}@ios.ac.cn

Abstract. We present a technique for checking instances of the satisfiability (SAT) problem based on a combination of the Davis-Putnam (DP) procedure and stochastic methods. We first use the DP procedure to some extent, so as to partition and reduce the search space. If the reduction does not lead to an answer, a stochastic algorithm is then used to search each subspace. This approach is proven to be efficient for several types of SAT instances. A parallel implementation of the method is described and some experimental results are reported.

1 Introduction

The Satisfiability (SAT) problem in the propositional logic is very well known. It is a key problem in computer science. It also plays an important role in real world applications such as planning, hardware verification and testing.

Since SAT is an NP-complete problem, we do not expect that there is an efficient algorithm which can determine the satisfiability of every instance of the problem. But it is still worthwhile to do research on practical techniques which can solve many SAT instances efficiently.

We may distinguish between two types of SAT algorithms: complete search and incomplete search. A complete search algorithm gives an answer to a SAT instance no matter it is satisfiable or not. An incomplete search algorithm may give an answer only when the SAT instance is satisfiable.

One of the most efficient complete search algorithm for SAT is the Davis-Putnam (DP) procedure. Many systems based on this procedure have been implemented (such as [6,9]) and many interesting problems have been solved by these tools. A major problem with DP is that it may have to go through a very large search space.

A typical incomplete search algorithm for SAT combines local search and random walks [5,2]. The WSAT tool [7] is based on such a method, and is claimed to be more efficient than DP-based tools on some SAT instances.

* This work was supported by the National Science Fund for Distinguished Young Scholars (No. 60125207) and the "973" project (No. G1998030600).

However, WSAT may come into difficulties on big SAT instances with many variables. The main purpose of our work is to improve the efficiency by combining the DP procedure and incomplete SAT algorithms. In order to utilize computational resources distributed in networked workstations, we have also implemented our approach as a distributed system.

2 The Basic Approach

We first briefly describe the DP procedure. Let S be a set of clauses and a be a literal occurring in S. Let $S|a$ be the result of removing clauses containing a and removing $\neg a$ from the rest of the clauses. The principle of the DP-procedure is as follows.

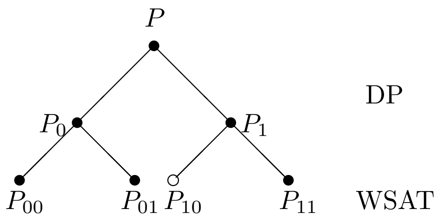
– S is satisfiable if S is empty.
– $S \cup \{a\}$ is unsatisfiable if $\neg a \in S$ is a unit clause.
– if $S = S' \cup \{a\}$, checking the satisfiability of S is reduced to checking the satisfiability of $S'|a$.
– if none of the above rule is applicable, select a literal a, check the satisfiability of $S|a$ and the satisfiability of $S|\neg a$. If either of them is satisfiable, S is satisfiable.

The last rule is called the branching rule and the strategy for selecting a literal is referred to as the branching strategy. We adopt the following strategy in our implementation of DP. To select a literal for branching, we use the look ahead strategy on the 5 best and the 5 worst propositional variables based on the weight of variables appearing in shortest clauses. (The definition of weights is omitted due to the limited space of this paper.) In addition, we require that both of the positive and negative literals must appear in the clauses, in order for a variable to be selected.

In the hybrid approach, we first use the DP procedure partially, and produce some subproblems. Then the subproblems are given to WSAT. There are two parameters for controlling the number of subproblems. One is the maximum depth to be searched by DP, denoted by $MaxDep$; the other is the maximum number of subproblems, denoted by $MaxSubP$. Each of them determines the other. For instance, if the depth is 4, the number is 16; if the number is 16, the depth for each subproblem is 4; if the number is 15, the depth for the first 14 subproblems is 4 and the depth for the last subproblem is 3.

If a subproblem is proven to be satisfiable within the given depth, the satisfiability checking is finished. If all subproblems are proven to be unsatisfiable within the depth, the satisfiability checking is also finished. Otherwise, the subproblems which have not yet been proven to be unsatisfiable are recorded in files. (The number of such problems may be less than $MaxSubP$.) In each subproblem, the propositional variables are renumbered consecutively from 1 to the number of remaining variables. These subproblems are given to WSAT in a loop until a solution is found or the maximum number of repetitions is reached. In the current implementation, the information of renumbering is not saved. This information could be important for producing a model of the original problem.

The following picture illustrates our approach. Here $MaxDep = 2$. Applying the DP procedure to the original problem P, we get two subproblems P_0 and P_1. Applying DP again to them, we get the subproblems P_{00}, P_{01}, P_{10} and P_{11}. Suppose that P_{10} is found to be unsatisfiable. Then we shall apply WSAT to the remaining three subproblems, i.e. P_{00}, P_{01} and P_{11}.



Experimental Results

We have applied our approach to the following test suites[1] and achieved good results. (We can only choose satisfiable problem instances since WSAT does not give meaningful results on unsatisfiable instances.)

- SAT encoding of logistics planning problem [5,4] referred to as LOGISTICS instances. There are four instances in this test suite. These instances are relatively easy for WSAT. Our approach is better for the most difficult one of these problems (i.e. *logistics.d.cnf*).
- SAT encoding of the All-Interval Series problem [2] referred to as AIS instances. There are also four instances in this test suite. The first three instances can be solved by WSAT within a few seconds. The last one is harder.
- SAT encoding of the quasigroup problem instances referred to as QG instances. There are 22 instances in the package and 10 of them are satisfiable. Instead of using the downloaded QG instances, we use the more compact instances generated by SATO [9].

The left part of Table 1 shows the sizes of the SAT instances.

Test 1: For each of the instances described in Table 1, 12 tests were carried out. Each test was carried out on a SUN BLADE 1000 with 750 MHz and 512 MB. The maximum number of subproblems ($MaxSubP$) was set to 15.

The right part of Table 1 shows the sizes of the subproblems produced by the DP-procedure. The ratio between the subproblem size and the problem size varies a lot. The ratio is small for QG instances and big for the LOGISTICS instances. The instances *qg1-07.cnf*, *qg2-07.cnf*, *qg6-09.cnf*, and *qg7-09.cnf* were solved by the DP-procedure and the columns for the subproblem sizes are irrelevant to these instances. We will not consider these problem instances later.

[1] See *http://www.intellektik.informatik.tu-darmstadt.de/SATLIB/benchm.html*

Table 1. Description of SAT Instances

Problem Instance	NumOf Variables	NumOf Clauses	NumOfCls in Subproblems Average	Smallest	Biggest
logistics.a	828	6,718	6,562	6,550	6,596
logistics.b	843	7,301	7,117	7,115	7,137
logistics.c	1,141	10,719	10,508	10,505	10,539
logistics.d	4,713	21,991	20,945	20,007	21,472
ais6	61	581	440	396	537
ais8	113	1,520	1,209	1,058	1,454
ais10	181	3,151	2,595	2,265	3,061
ais12	265	5,666	4,792	4,212	5,552
qg1-07	343	8,578	-	-	-
qg1-08	512	23,106	9,074	6,030	12,064
qg2-07	343	11,254	-	-	-
qg2-08	512	28,945	8,442	6,244	9,735
qg3-08	512	17,689	3,670	3,269	4,506
qg4-09	729	28,751	6,947	4,117	8,976
qg5-11	1,331	64,377	14,797	7,482	20,249
qg6-09	729	28,463	-	-	-
qg7-09	729	28,751	-	-	-
qg7-13	2,197	125,923	31,353	21,285	45,452

We applied WSAT to each problem instance, with the maximum number of trials set to 3000. The maximum number of trials for each of the subproblems was set to 200. The shortest time, the longest time and the average time for finding a solution were calculated. The time for using the DP-procedure was calculated separately. These calculated values are presented in Table 2.

Table 2. Result of Test 1

Problem	NumOf	hybrid SAT					WSAT			
Instance	Subprobs	DP	H-Avg	H-Min	H-Max	F	W-Avg	W-Min	W-Max	F
logistics.a	15	1.5	1.2	0.2	3.0		1.1	0.2	3.6	
logistics.b	15	1.6	0.7	0.1	1.5		0.6	0.1	1.0	
logistics.c	15	2.0	3.4	0.3	12.8		5.8	1.4	15.5	
logistics.d	15	3.3	9.4	0.7	28.4		14.6	0.9	50.1	
ais6	15	0.3	1.1	1.0	1.3		0.0	0.0	0.0	
ais8	15	0.4	0.2	0.0	0.4		0.3	0.0	0.6	
ais10	15	0.5	3.6	0.2	7.7		1.6	0.1	3.6	
ais12	15	1.0	18.7	9.6	34.6		121.5	23.3	227.4	
qg1-08	15	2.1	936.6	146.8	2201.2		3637.7	148.0	7270.0	5
qg2-08	15	2.0	1571.0	231.2	4004.6	1	5831.0	828.3	8749.3	9
qg3-08	10	0.6	4.5	4.0	5.0		27.9	1.7	81.7	
qg4-09	15	1.6	9.8	4.4	16.5		386.9	6.8	856.2	
qg5-11	3	1.7	45.6	0.9	100.3		9460.1	9455.7	9465.1	12
qg7-13	13	5.2	5298.5	5290.1	5310.5	12	15261.1	15208.2	15273.0	12

Table 2 shows the following information (with running times measured in seconds):

- the number of subproblems (with $MaxSubP = 15$);
- the performance of the hybrid approach:
  - the running time of the DP-procedure,
  - H-Avg: the average running time of WSAT on the subproblems,
  - H-Min: the minimum running time of WSAT on the subproblems,
  - H-Max: the maximum running time of WSAT on the subproblems;
  - the number of failures (i.e. when no solution is found);
- the performance of WSAT without DP-reduction (i.e., applying WSAT directly on the original problem instance):
  - W-Avg: the average running time,
  - W-Min: the minimum running time,
  - W-Max: the maximum running time;
  - the number of failures (i.e. when no solution is found).

Based on the experimental results, we compare our approach with the original WSAT procedure as follows:

- For the SAT instances that can be solved with WSAT within a few seconds, our approach does not have much advantage, since we have to add the time usage of the DP-procedure to that of WSAT.
- For the medium hard instances (e.g. with time usage between 10 and 1000 seconds in W-Avg), the advantage of our approach is obvious.
- For many of the hardest instances, we may achieve much better results. The instances *qg4-09.cnf* and *qg5-11.cnf* are witness to such a good result.
- Generally, the advantages of partitioning a problem into subproblems compared to using WSAT alone is that each subproblem is much smaller than the original problem. A consequence of this is that the time needed for each trial of such a subproblem with WSAT is much shorter. For instance, 10 trials for the quasigroup instance *qg7-13.cnf* on our computer need 31.0 seconds. After partitioning the problem into 13 subproblems (3 subproblems are proved to be unsatisfiable within depth 4), the times for 10 trials for each of the subproblems range from 7.9 to 14.6 seconds. Another advantage is that a solution of such a subproblem is expected to be found with much less time, if this subproblem indeed has a solution, as demonstrated by the above experimental results.

3 The Parallel Approach

We have implemented the approach as a distributed system for checking different subproblems in parallel. The system (called PSAT) consists of one master and a number of slaves. The pseudo-code of PSAT is as follows:

```
PSAT(args,input-file)
{
    master: {
        arguments-split();          /* split args into args-1 and args-2 */
        broadcast(args-2);           /* broadcast args-2 to all slaves */
        DP(args-1,input-file);             /* apply the DP procedure */
        get-all-subproblems();
        while (1) {
            listen-to-slaves();
            if (slave-report()) {
                if (success) abort-all;
                else if (found-unfinished-subproblems()) send-new-work();
                else send-stop();
                if (all-finished()) exit;
            }
        }
    }
    slave: {
        receive(args-2);
        while (unfinished) {
            receive-new-work();
            if (have-new-work) {
                WSAT(args-2,new-work);  /* apply WSAT to new-work */
                report-to-master();
            }
            else exit;
        }
    }
}
```

The task of the master is as follows:

1. Apply the DP-procedure.
   If a model is obtained within the depth, the instance is satisfiable (and the program is terminated). Otherwise we check whether there are subproblems. If there is no subproblem, the instance is unsatisfiable (and the program is terminated).
2. Establish connection with the slaves and distribute the subproblems to them. In our system, the files containing subproblems do not need to be communicated to different computers. They are stored in one place which can be read by different computers via NFS file-system. The only information to be passed to the slaves is the identification of the subproblems. In this way, we have avoided heavy communication of the clauses of the subproblems.
3. Get the results from the slaves.

The task of a slave is to use the tool WSAT on the received subproblem, trying to find a model for this subproblem. In the pseudo-code, args-1 refers to the arguments needed by DP (*MaxDep* and *MaxSubP*), while args-2 refers to the arguments of WSAT (e.g., the number of trials). The input file contains the SAT instance and DP is the procedure for partitioning and reducing the instance. The subproblems are identified by the parameter *new-work* and are to be solved by WSAT on different workstations. The tool PSAT is implemented by using *Message Passing Interface (MPI)* which is the de facto standard of parallel programming.

Test 2: We use the same test suites for the testing of our distributed system. The experiments were carried out on a Linux cluster which consists of 8 nodes connected by Fast Ethernet Switch (D-Link DES1008). Each node has two 1GHz Pentium III processors with 2048 MB memory. A problem is partitioned in such a way that the number of subproblems is the same as the number of slaves. Since there is one master process, we can have 15 slave processes. Thus we set *MaxSubP* to be 15.

For each of the instances, 12 tests were carried out. For each of the tests:

- WSAT was applied to the instance with a given new seed. The maximum number of trials was 2000 (the number was reduced from 3000 to 2000 in order to reduce the testing time, and this has increased the percentage of failed tests for the hard instances).
- PSAT was applied to the same instance with the same seed for all WSAT applications within PSAT. The maximum number of trials for each subproblem was 200.

Table 3. Result of Test 2

Problem	NumOf	PSAT					WSAT			
Instance	Jobs	DP	P-Avg	P-Min	P-Max	F	W-Avg	W-Min	W-Max	F
logistics.a	15	0.5	0.3	0.2	0.4		1.1	0.2	1.9	
logistics.b	15	0.5	0.1	0.4	0.6		0.7	0.2	1.4	
logistics.c	15	0.7	0.9	0.4	1.6		3.6	0.2	9.5	
logistics.d	15	1.5	1.2	0.7	1.6		23.7	8.4	54.2	
ais6	15	0.0	0.0	0.0	0.0		0.0	0.0	0.0	
ais8	15	0.1	0.1	0.0	0.1		0.1	0.0	0.5	
ais10	15	0.2	0.4	0.2	1.0		1.6	0.3	3.4	
ais12	15	0.3	2.5	0.2	7.4		74.9	2.4	178.2	
qg1-08	15	0.9	55.2	1.2	103.4		1620.8	135.9	2785.4	4
qg2-08	15	0.9	89.9	4.0	293.4		4320.2	2211.6	4773.7	11
qg3-08	15	0.3	0.3	0.2	0.9		27.2	2.0	75.8	
qg4-09	15	0.6	1.3	0.6	2.4		107.8	6.3	322.9	
qg5-11	3	1.1	7.7	1.2	18.5		8210.6	7166.4	8825.0	12
qg7-13	13	3.2	720.9	713.9	724.6	12	10732.2	9233.6	10941.7	12

Table 3 shows the same types of information as Table 2. With respect to the approach presented in section 2, we had expected that we might get a speed up by a factor of around 15, since we have 15 slaves. However the situation is more complicated.

The processors used in this test are different from that used in the previous tests, and the result is not directly comparable. We have compared the ratio between the time usage with DP reduction (only counting the time usage of WSAT in the approach) and the time usage without DP reduction of this table and that of Table 2. For the easy SAT instances, the speed up was much lower than expected. On the other hand, the speed up was better for many of the other instances, such as qg1-08 and qg2-08.

Apparently there is no fixed factor for the speed up with the parallel approach due to the nature of the randomness of WSAT. However the conclusion is clear and we may say that the parallel approach is much better than the sequential approach with respect to the time usage and our approach is appropriate for utilizing parallel computing power and unused networked workstations for checking the satisfiability of SAT instances.

Test 3: To investigate how the number of subproblems affects the satisfiability test, we solved several hardest instances with different values for *MaxSubP*. The experiments were carried out on a Dawning 2000-II supercomputer with 82 nodes (80 nodes for computing and 2 nodes as the server). Each node has two 333MHz PowerPC CPU. The results are summarized in Table 4. As previously, the running times are given in seconds.

Table 4. Result of Test 3

Problem Instance	Max Depth	NumOf Jobs	Init Time	DP Time	Wsat Time
Logistics.d	2	4	0.0440	4.2341	12.1859
Logistics.d	3	8	0.4691	8.5456	6.2693
Logistics.d	4	16	0.8818	14.2175	10.3217
Logistics.d	5	32	0.5465	26.1273	12.9071
Ais12.cnf	2	4	0.2080	1.5362	2.2571
Ais12.cnf	3	8	0.2173	2.5335	1.9988
Ais12.cnf	4	16	0.9606	5.1357	1.8886
Ais12.cnf	5	32	1.3835	10.1181	1.9326
Qg511.cnf	2	4	0.1216	4.7137	86.5016
Qg511.cnf	3	4	0.1232	7.0734	29.0169
Qg511.cnf	4	3	0.3157	6.8622	29.6516
Qg511.cnf	5	3	0.3937	11.4745	41.5507
Qg511.cnf	6	2	0.2204	10.6388	23.1538

Here, column 2 gives the value of *MaxDep* (i.e., depth of the DP search tree), and column 3 gives the number of subproblems that have not been finished by DP. The other columns show the running times of PSAT at various stages.

It can be seen that, on the AIS problem and the quasigroup problem, there is a notable reduction of stochastic search time as *MaxDep* is increased to a certain value. On the LOGISTICS problem, the timing difference is not significant. Note also that the time for DP increases as *MaxDep* becomes larger. This is due to the fact that the subproblems have to be written into files.

4 Related Works

There exist several parallel and networked implementations of reasoning algorithms. For instance, efficient parallel and networked DP provers are reported in [1,10]. The purposes of these implementations are to divide and distribute a SAT instance in order to utilize the parallel computing power and networked workstations. Appropriate search space partitioning and workload balancing are the essential aspects in such implementations. The purpose of our implementation is different. We try to improve stochastic SAT solvers by first using an implementation of the DP procedure to partition and reduce SAT instances. We may achieve speed up by a factor much higher than the number of available networked workstations. This has been illustrated by the experimental results.

Other ways of combining complete and incomplete search have been proposed in the literature [3,8]. For example, one may use incomplete methods first and then use complete search [11]. This can be considered as an improvement of the backtracking procedure, while our approach is essentially an enhancement of stochastic search. We do not know of any parallel implementations of other hybrid methods. It is natural to parallelize our method because, after the first stage of (backtracking) search, we obtain some subproblems whose search spaces are disjoint.

5 Conclusion

We have combined the DP-procedure with WSAT for the satisfiability test. This approach is more efficient than using WSAT alone. We have also implemented the approach as a distributed system and have achieved additional efficiency by utilizing parallel computing power and unused networked workstations. As the instances of the test suites are concerned, the speed up factor of the sequential approach is significant for medium hard instances and is higher for the more difficult instances. Additional speed-up has been achieved by using the parallel approach. Generally, the speed up depends on the maximum number of subproblems and the number of available workstations.

The advantages of partitioning a problem into subproblems compared to using WSAT alone is that each subproblem is much smaller than the original problem. The implications of this are that the time needed for each trial of such a subproblem with WSAT is much shorter; and a solution of such a subproblem is expected to be found with much less time, if this subproblem indeed has a solution. For hard SAT instances, the speed up with our approaches is significant. Our approach is therefore an important step forward in SAT research and has scaled up the applicability of SAT tools.

As our approach is an enhancement of incomplete search, in this paper, we are mainly concerned with the performance gains over the sequential WSAT. In the future, we plan to compare our tool with DP provers. It is also necessary to improve our current implementation, to carry out more experiments and to further investigate several issues (e.g., the impact of the number of subproblems).

Acknowledgements. We would like to thank the anonymous reviewers for helpful comments. The Dawning supercomputer was made available to us through the Computer Network Information Center, Chinese Academy of Sciences. Yunquan Zhang also helped us with some of our experiments.

References

1. Max Böhm and Ewald Speckenmeyer. A fast parallel SAT-solver - efficient workload balancing. Annals of Mathematics and Artificial Intelligence 17(3-4):381–400. 1996.
2. Holger H. Hoos. Stochastic Local Search - Methods, Models, Applications. PhD thesis, CS Department, TU Darmstadt, 1998.
3. N. Jussien and O. Lhomme. Local search with constraint propagation and conflict-based heuristics. Proc. of the 17th National Conf. on Artificial Intelligence (AAAI), 169–174, 2000.
4. Henry Kautz, David McAllester and Bart Selman. Encoding Plans in Propositional Logic. Proc. KR'96, 374–384. 1996.
5. Henry Kautz and Bart Selman. Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search. Proc. of the 13th National Conference on Artificial Intelligence (AAAI'96), 1194–1201, 1996.
6. Chu Min Li and Anbulagan. Heuristics Based on Unit Propagation for Satisfiability Problems. Proc. of the 15th Int'l Joint Conf. on Artificial Intelligence (IJCAI-97), 366–371, 1997.
7. Bart Selman, Henry A. Kautz and Bram Cohen. Noise Strategies for Improving Local Search. Proc. of the 12th National Conf. on Artificial Intelligence (AAAI), Vol.1, 337–343, 1994.
8. A. Schaerf, Combining local search and look-ahead for scheduling and constraint satisfaction problems, *Proc. Int'l Joint Conf. on Artificial Intelligence* (*IJCAI-97*), Vol.2, 1254–1259, 1997.
9. Hantao Zhang and Mark E. Stickel. Implementing the Davis-Putnam Method. Journal of Automated Reasoning 24(1/2):277–296 (2000).
10. Hantao Zhang, Maria Paola Bonacina and Jieh Hsiang. PSATO: a distributed propositional prover and its application to quasigroup problems. J. of Symbolic Computation 21(4):543–560. 1996.
11. Jian Zhang and Hantao Zhang. Combining Local Search and Backtracking Techniques for Constraint Satisfaction, Proc. of the 13th National Conference on Artificial Intelligence (AAAI-96), Vol.1, 369–374, 1996.