A two–phase exact algorithm for MAX–SAT and weighted MAX–SAT problems *

Brian Borchers† Judith Furman‡

March 31, 1997

Abstract

We describe a two phase algorithm for MAX–SAT and weighted MAX–SAT problems. In the first phase, we use the GSAT heuristic to find a good solution to the problem. In the second phase, we use an enumeration procedure based on the Davis–Putnam–Loveland algorithm, to find a provably optimal solution. The first heuristic stage improves the performance of the algorithm by obtaining an upper bound on the minimum number of unsatisfied clauses that can be used in pruning branches of the search tree.

We compare our algorithm with an integer programming branch and cut algorithm. Our implementation of the two phase algorithm is faster

*Research partially supported by ONR Grant number N00014–94–1–0391.
†Mathematics Department, New Mexico Tech, Socorro, NM 87801.
‡Department of Mathematical Sciences, Clemson University, Clemson, SC 29634

than the integer programming approach on many problems. However, the integer programming approach is more effective than the two phase algorithm on some classes of problems, including MAX–2–SAT problems.

Introduction

A propositional logic formula F can always be written in conjunctive normal form as the conjunction of m clauses, where each clause is the disjunction of a set of literals and each literal is either a variable or the negation of a variable. The satisfiability problem (SAT) can be stated as

> Given a formula F in conjunctive normal form, is there a truth assignment of the logical variables that makes F true?

The satisfiability problem is known to be NP-Complete, even if we restrict the clauses to three literals each (Garey and Johnson, 1979).

Exact algorithms for the satisfiability problem include the Davis–Putnam–Loveland algorithm (Davis and Putnam, 1960; Loveland, 1978), resolution (Robinson, 1965) and integer programming approaches (Blair et al., 1986; Harche and Thompson, 1990; Hooker, 1988; Hooker and Fedjki, 1990; Jeroslow and Wang, 1990). Heuristics for SAT include Selman, Levesque, and Mitchell's GSAT heuristic (Selman et al., 1992; Selman and Kautz, 1993) and Resende and Feo's GRASP heuristic (Resende and Feo, 1996). These algorithms search for a sat-

isfying truth assignment. If the heuristic discovers such a truth assignment then we have proven that the problem is satisfiable. However, if no satisfying truth assignment is discovered, then we have not shown that the problem is unsatisfiable.

This paper concerns itself with the related maximum satisfiability problem (MAX–SAT) and a weighted version of the maximum satisfiability problem. The MAX–SAT problem can be stated as

> Given a collection C of m clauses, $C_1, \ldots C_m$ involving n logical variables, find a truth assignment that maximizes the number of satisfied clauses in C.

The weighted MAX–SAT problem can be stated as

> Given a collection C of m clauses, $C_1, \ldots C_m$ involving n logical variables, with clause weights w_i, find a truth assignment that maximizes the total weight of the satisfied clauses in C.

A number of applications of the MAX–SAT problem are discussed by Hansen and Jaumard (Hansen and Jaumard, 1990).

The MAX–SAT and weighted MAX–SAT problems are NP-hard. Furthermore, there is no polynomial time approximation scheme for MAX–SAT unless P=NP (Papadimitriou and Yannakakis, 1991). However, Goemans and Williamson (1995) have shown that it is possible to approximate MAX–SAT within a factor of 0.758 in polynomial time (Goemans and Williamson, 1995).

Since most heuristics for the satisfiability problem work by constructing truth assignments that satisfy a large number of clauses, these heuristics are automatically applicable to the MAX–SAT problem. Hansen and Jaumard (Hansen and Jaumard, 1990) give a summary of several heuristics and approximation algorithms for MAX–SAT. The GSAT heuristic of Selman, Levesque, and Mitchell has also been extended to weighted maximum satisfiability problems (Jiang et al., 1995).

Integer programming approaches can also be used to solve the MAX–SAT and weighted MAX–SAT problems. Cheryian et al. (Cheriyan et al., 1996) describe a cutting plane algorithm for the MAX–2–SAT problem. In a companion paper, Borchers, Joy, and Mitchell (1995) describe a branch and cut algorithm for the general MAX–SAT problem.

In this paper we describe a new algorithm for the exact solution of MAX–SAT and weighted MAX–SAT problems. We then report on the results of computational tests in which our algorithm and an integer programming approach were used to solve a number of test problems.

It should be noted that after the submission of this paper, the authors became aware of another paper (Wallace and Freuder, 1996) which discusses a somewhat similar algorithm for MAX–SAT problems.

A Two–Phase Algorithm for MAX–SAT

Our algorithm for the MAX–SAT problem begins by using the GSAT heuristic to find a "good" solution to the problem. In this phase of the algorithm, we obtain an upper bound ub on the number of clauses unsatisfied in an optimal solution. A high quality heuristic solution helps to improve the performance of the second phase of the algorithm.

In the second phase of the algorithm, we use a backtracking algorithm based on the Davis–Putnam–Loveland procedure to implicitly enumerate all possible truth assignments. In this procedure, we have a partially specified truth assignment in which values of true or false have been assigned to the variables $x_{j_1}, \ldots, x_{j_l}$. We update the upper bound, ub, as new record solutions are found. We also keep track of $unsat$, the number of clauses that are left unsatisfiable by the current solution.

If $unsat \geq ub$, then the current partial solution can not be extended to yield a solution better than the current incumbent solution. We discard the current partial solution and backtrack to another partial solution. To backtrack, we find the variable that was most recently fixed at false during branching. We set this variable to true and continue processing.

If $unsat = ub - 1$, then we can perform unit clause tracking. A unit clause is a clause in which all but one of the literals are fixed at false. Any literals left in unit clauses can be fixed at true, because otherwise $unsat$ would increase to a value above ub. After performing unit clause tracking, we update the value of

unsat.

If we have examined the current solution and $unsat < ub$ then we must add another logical variable to the current partial solution. Since this variable must be tried at both true and false, this creates two new subproblems that must be considered. We refer to this step as "branching." To branch, our algorithm first selects the clauses with the smallest number of unfixed literals. We then pick an unfixed variable $x_{j_{l+1}}$ that appears in the largest number of these clauses. We fix $x_{j_{l+1}}$ at false. We then go on to the next iteration of the algorithm.

The algorithm stops when we have implicitly enumerated all possible solutions. Our algorithm for the MAX–SAT problem is summarized in algorithm 1.

Algorithm 1 (Two Phase Algorithm for MAX–SAT)

1. *Use the GSAT heuristic to obtain a heuristic solution. Let ub be the number of unsatisfied clauses in this solution.*
2. *Let* $l = 0$.
3. *The backtracking search.*
   (a) *Update unsat, the number of clauses left unsatisfiable by the current solution.*
   (b) *If* $unsat = ub - 1$, *perform unit clause tracking.*
   (c) *If* $unsat \geq ub$, *then goto step 4.*
   (d) *If all variables are fixed at true or false, then we have a new best solution. Let* $ub = unsat$. *Goto step 4.*

(e) *Otherwise, branch. Pick a variable* $x_{j_{l+1}}$ *and fix it at false. Let* $l = l + 1$.

4. *Backtrack. Find the largest* k *such that* x_{j_k} *has been fixed at false by branching. If no such* k *exists, then stop. Fix* x_{j_k} *at true. Let* $l = k$. *Goto step 3.*

It is relatively simple to modify this algorithm to work with weighted maximum satisfiability problems. Modifications to the GSAT heuristic for weighted problems are discussed in (Jiang et al., 1995). In the backtracking search, instead of counting the number of unsatisfied clauses in ub and $unsat$, we count the total weight of unsatisfied clauses. Unit clause tracking is performed whenever a unit clause has weight greater than or equal to $ub - unsat$.

It is also quite simple to modify this algorithm to solve SAT problems. In step 1, we skip the GSAT procedure and simply set $ub = 1$. With this modification, the algorithm will backtrack whenever the current solution has any unsatisfied clauses.

Computational Results

The algorithm described in the previous section was implemented in C. [1] The code for unweighted problems is referred to as "maxsat". The code for weighted

[1] This code and the test problems are available at http://www.nmt.edu/~borchers/maxsat.html.

problems is referred to as "wmaxsat". We compared the performance of our codes with an integer programming code for MAX–SAT and weighted MAX–SAT problems (Joy et al., 1996). This code is based on MINTO, a software package for the development of branch and cut algorithms (Nemhauser et al., 1994). All codes were tested on an IBM RS/6000-590 with 256 megabytes of main memory. The codes were compiled with IBM's xlc compiler using the –O3 flag for full code optimization.

In testing the codes, we had several objectives. Our first objective was to determine what size problems could reasonably be solved by our implementation of the two phase algorithm. Our second objective was to determine whether or not the use of the GSAT heuristic improved the overall performance of the two phase algorithm. Our third objective was to determine whether or not our algorithm was competitive with the integer programming approach.

We first tested both codes on a variety of randomly generated MAX–SAT problems. These problems had between fifty and one hundred and fifty variables, with between 100 and 750 clauses. Problem sizes were picked so that some problems would be close to satisfiability, while other problems would have a large number of unsatisfied clauses in the optimal solution.

Computational results are shown in tables 1 and 2. As the number of variables and clauses increases, these problems become very difficult for either code to solve.

For the MAX–SAT problems with two variables per clause, minto is substan-

tially faster in most cases. The difference in performance becomes more extreme as the number of clauses increases. However, for the problems with three variables per clause, maxsat is consistently faster than minto. As the number of clauses increases, minto becomes more competitive, but even with ten times as many clauses as variables, maxsat is faster than minto by a factor of five.

The good performance of minto on the problems with two variables per clause can be explained by the observation that minto processed very small branch and cut trees in solving these problems. On the problems with three variables per clause, minto had to process substantially larger branch and cut trees. Thus it appears that the lower bounds obtained by solving the linear programming problems and by adding cutting planes were more effective on two variable per clause problems than on problems with three variables per clause.

Next, we tested the codes on a number of randomly generated weighted MAX–SAT problems. These problems were randomly generated with the same sizes as the MAX–SAT test problems. Each clause was given an integer weight uniformly distributed between one and ten. Computational results are shown in table 3 and 4. Again, minto was faster than our wmaxsat code on the weighted MAX–2–SAT problems, while our wmaxsat code was faster than minto on the MAX–3–SAT problems. In most cases, the weighted problems were easier to solve than the corresponding unweighted problems.

In order to determine how effective the GSAT heuristic was, we also ran maxsat on a number of the test problems with the GSAT heuristic turned off.

These computational results are shown in table 5. For relatively small problems, the GSAT heuristic is not worthwhile. However, for the larger problems, using the GSAT heuristic has a dramatic effect on the CPU time required to solve a problem. The maxsat code with GSAT is as much as seven times faster than the code without the heuristic.

Summary and Conclusions

We have described a two–phase algorithm for the solution of MAX–SAT and weighted MAX–SAT problems. This algorithm uses the GSAT heuristic in the first phase to find a good heuristic solution. It then uses a modified Davis–Putnam–Loveland procedure to obtain a provably optimal solution. We have implemented the algorithm and compared it with an integer programming branch and cut algorithm.

Our implementation of the two–phase algorithm is capable of solving randomly generated MAX–2–SAT and MAX–3–SAT problems with as many as 150 variables and 675 clauses in a reasonable amount of time on a fast workstation. However, these problems are much harder to solve than simple satisfiability problems of similar size.

Our implementation of the two–phase algorithm was considerably faster than our integer programming approach for MAX–3–SAT problems, while the integer programming approach was faster than the two–phase algorithm for MAX–2–

			maxsat		minto	
Vars	Clauses	Opt Soln	Time	Backtracks	Time	Nodes
50	100	4	0.4	627	3.8	1
50	150	8	1.5	11368	5.3	5
50	200	16	116.2	1353283	19.8	37
50	250	22	652.4	6627660	37.6	41
50	300	32	8763.6	83792967	202.4	85
50	350	41	>12 hours	\/A	687.9	157
50	400	45	>12 hours	\/A	456.3	99
50	450	63	>12 hours	\/A	3716.5	503
50	500	66	>12 hours	\/A	2552.4	323
100	200	5	3.2	13592	8.8	3
100	300	15	13770.2	104553113	25.7	25
100	400	29	>12 hours	\/A	777.2	261
100	500	44	>12 hours	\/A	27592.8	2161
100	600	N/A	>12 hours	\/A	>12 hours	N/A
150	300	4	4.1	10006	14.2	3
150	450	22	>12 hours	\/A	708.8	251
150	600	38	>12 hours	\/A	4881.7	631

Table 1: Computational Results for the MAX–2–SAT test problems.

SAT problems. Neither algorithm was dominant on all problems, and the choice of an appropriate algorithm depends on the characteristics of the problem to be solved.

			maxsat		minto	
Vars	Clauses	Opt Soln	Time	Backtracks	Time	Nodes
50	250	2	0.9	314	26.8	77
50	300	4	3.2	11073	218.8	449
50	350	8	134.8	915731	3118.9	3025
50	400	11	883.6	6196183	10679.9	6443
50	450	15	4970.9	34246168	29030.2	10741
50	500	15	3154.9	19442201	14506.5	4405
100	500	4	191.5	514100	16557.3	16477
100	550	5	943.1	2734570	> 12 hours	N/A
100	600	N/A	> 12 hours	N/A	> 12 hours	N/A
150	675	2	99.2	88967	> 12 hours	N/A
150	750	N/A	> 12 hours	N/A	> 12 hours	N/A

Table 2: Computational Results for the MAX–3–SAT test problems.

			maxsat		minto	
Vars	Clauses	Opt Soln	Time	Backtracks	Time	Nodes
50	100	16	0.4	138	3.8	3
50	150	34	0.6	1307	4.8	5
50	200	69	6.9	46161	9.9	19
50	250	96	69.1	430704	50.2	57
50	300	132	257.0	1436778	43.7	35
50	350	211	11848.5	65333759	633.8	215
50	400	211	5950.9	28209843	170.2	69
50	450	257	19217.3	81357439	417.6	129
50	500	318	>12 hours	N/A	927.3	209
100	200	7	> 12 hours	N/A	7.8	1
100	300	67	> 12 hours	N/A	60.8	81
100	400	119	> 12 hours	N/A	817.8	358
100	500	241	> 12 hours	N/A	23749.3	2239
100	600	266	>12 hours	N/A	19347.0	1193
150	300	24	5.2	10564	16.2	9
150	450	79	>12 hours	N/A	226.0	165
150	600	N/A	>12 hours	N/A	>12 hours	N/A

Table 3: Computational Results for the weighted MAX–2–SAT test problems.

			maxsat		minto	
Vars	Clauses	Opt Soln	Time	Backtracks	Time	Nodes
50	250	1	0.8	55	11.6	31
50	300	13	1.4	1832	136.7	307
50	350	25	8.7	27798	851.3	1181
50	400	33	21.0	72165	1788.2	1565
50	450	35	19.3	58439	844.2	633
50	500	77	1911.1	7433808	>12 hours	N/A
100	500	6	8.4	9327	2451.9	2586
100	600	26	3875.0	7640263	>12 hours	N/A
150	675	2	12.1	6348	5636.0	4029
150	750	5	272.0	221057	>12 hours	N/A

Table 4: Computational Results for the weighted MAX–3–SAT test problems.

Problem			maxsat	
Vars	Clauses	Vars/Clause	time	backtracks
50	100	2	0.1	1427
50	150	2	3.0	35178
50	200	2	356.4	4417245
50	250	2	5451.1	60129879
50	300	2	27250.1	272440756
50	250	3	5.4	36809
50	300	3	44.4	383676
50	350	3	585.7	4377886
50	400	3	4190.0	32063459
50	450	3	18943.1	144850529
50	500	3	27849.2	195096258

Table 5: Computational Results for MAX–SAT problems without GSAT.

References

Blair, C. E., Jeroslow, R. G., and Lowe, J. K. (1986). Some results and experiments in programming techniques for propositional logic. *Computers and Operations Research*, 13(5):633–645.

Cheriyan, J., Cunningham, W. H., Tunçel, L., and Wang, Y. (1996). A linear programming and rounding approach to max 2-sat. In Johnson, D. S. and Trick, M. A., editors, *Cliques, Coloring, and Satisfiability*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 395–414. AMS.

Davis, M. and Putnam, H. (1960). A computing procedure for quantification theory. *Journal of The Association for Computing Machinery*, 7:201–215.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York.

Goemans, M. X. and Williamson, D. P. (1995). Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42:1115–1145.

Hansen, P. and Jaumard, B. (1990). Algorithms for the maximum satisfiability problem. *Computing*, 44:279–303.

Harche, F. and Thompson, G. L. (1990). The column subtraction algorithm, an exact method for solving weighted set covering packing and partitioning problems. *Computers and Operations Research*, 21:689–705.

Hooker, J. N. (1988). Resolution vs. cutting plane solution of inference problems: some computational experience. *Operations Research Letters*, 7(1):1–7.

Hooker, J. N. and Fedjki, C. (1990). Branch-and-cut solution of inference problems in propositional logic. *Annals of Mathematics and Artificial Intelligence*, 1:123–139.

Jeroslow, R. E. and Wang, J. (1990). Solving propositional satisfiability problems. *Annals of Mathematics and AI*, 1:167–187.

Jiang, Y., Kautz, H., and Selman, B. (1995). Solving problems with hard and soft constraints using a stochastic algorithm for MAX–SAT. Presented at the 1st International Joint Workshop on Artificial Intelligence and Operations Research, Timberline OR.

Joy, S., Mitchell, J. E., and Borchers, B. (1996). A branch-and-cut algorithm for MAX-SAT and weighted MAX-SAT. To appear in *Proceedings of the DIMACS Workshop on Satisfiability: Theory and Applications.*

Loveland, D. (1978). *Automated Theorem–Proving: A Logical Basis.* North Holland, New York.

Nemhauser, G. L., Savelsbergh, M. W. P., and Sigismondi, G. C. (1994). MINTO, a Mixed INTeger Optimizer. *Operations Research Letters*, 15(1):47–58.

Papadimitriou, C. H. and Yannakakis, M. (1991). Optimization, approximation, and complexity classes. *Journal of Computers and System Sciences*, 43:425–440.

Resende, M. G. C. and Feo, T. A. (1996). A GRASP for satisfiability. In Johnson, D. S. and Trick, M. A., editors, *Cliques, Coloring, and Satisfiability*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 499–520. AMS.

Robinson, J. A. (1965). A machine–oriented logic based on the resolution principle. *Journal of The Association for Computing Machinery*, 12(1):23–41.

Selman, B. and Kautz, H. A. (1993). An empirical study of greedy local search for satisfiability testing. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93), Washington, DC*, pages 46–51.

Selman, B., Levesque, H., and Mitchell, D. (1992). A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92), San Jose, CA*, pages 440–446.

Wallace, R. J. and Freuder, E. C. (1996). Comparitive studies of constraint satisfaction and Davis–Putnam algorithms for maximum satisfiability problems. In Johnson, D. S. and Trick, M. A., editors, *Cliques, Coloring, and*

Satisfiability, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 587–615. AMS.