Improving Exact Algorithms for MAX-2-SAT*

Haiou Shen, Hantao Zhang
Computer Science Department
The University of Iowa
Iowa City, IA 52242
{*hshen, hzhang*}*@cs.uiowa.edu*

November 17, 2003

Abstract

We study three new techniques which will speed up the branch-and-bound algorithm for the MAX-2-SAT problem: The first technique is a new lower bound function for the algorithm and we show that the new lower bound function is consistently better than other lower bound functions. The other two techniques are based on the strongly connected components of the implication graph of a 2CNF formula: One uses the graph to simplify the formula and the other uses the graph to design a new variable ordering. The experiments show that the simplification can reduce the size of the input substantially when used in preprocessing and that the new variable ordering performs much better when the clause-to-variable ratio is less than 2. The result of this research is a high-performance implementation of an exact algorithm for MAX-2-SAT which outperforms any implementation we know about in the same category. It also shows that our MAX-2-SAT implementation is a feasible and effective tool to solve large instances of the Max-Cut problem in graph theory.

1 Introduction

In recent years, there has been considerable interest in the maximum satisfiability problem (MAX-SAT) of propositional logic, which, given a set of propositional clauses, asks to find a truth assignment that satisfies the maximum number of clauses. The decision version of MAX-SAT is NP-complete, even if the clauses have at most two literals (so called the MAX-2-SAT problem). Because the MAX-SAT problem is fundamental to many practical problems in computer science [12] and computer engineering [19], efficient methods that can solve a large set of instances of MAX-SAT are eagerly sought. One important application of MAX-2-SAT is that NP-complete graph problems such as *maximum cut*, *independent set*, can be reduced to special instances of MAX-2-SAT [7, 15]. Many of the proposed methods for MAX-SAT are based on approximation algorithms [8]; some of them are based on branch-and-bound methods [12, 6, 4, 14, 13, 11, 16]; and some of them are based on transforming MAX-SAT into SAT [19].

To the best of our knowledge, there are only four implementations of exact algorithms for MAX-SAT that are variants of the well-known Davis-Putnam-Logemann-Loveland (DPLL) procedure [9]. One is due to Wallace and Freuder (implemented in Lisp) [18]; one is due to Borchers and Furman [6] (implemented in C and publicly available); the last two are made

*Partially supported by the National Science Foundation under Grant CCR-0098093.

available in 2003 by Alsinet, Manyà and Planes [1] (a substantial improvement over Borchers and Furman's implementation) and Zhang, Shen, and Manyà [20], respectively.

In this paper we will discuss three novel techniques intended to improve the performance of the branch-and-bound algorithm proposed in [20]. It is well-known that the tighter the bound the smaller the search tree in a typical branch-and-bound algorithm. We introduce a new lower bound function which can reduce the search tree substantially. The other techniques are based on the use of the strongly connected components (SCC) in the implication graph of a 2CNF instance. It is well-known that the satisfiability of 2CNF formula can be decided in linear time by computing SCC [3]. For MAX-2-SAT, we found that computing SCC can help us to (a) simplify the input greatly, and (b) design a new variable ordering for the branch-and-bound algorithm in [20].

In order to evaluate the new techniques, we present experimental results on thousands of MAX-2-SAT instances. We showed that the improved algorithm is consistently and substantially better than all the known algorithms [6, 1, 20]. We also show the performance of the algorithm on a large set of random MAX-CUT problems.

2 Preliminary

Let F be a formula in 2CNF with n variables $V = \{x_1, ..., x_n\}$ and m clauses. An *assignment* is a mapping from V to $\{0, 1\}$ and may be represented by a vector $\vec{X} \in \{0, 1\}^n$, where 0 means false and 1 means true. Let $S(\vec{X}, F)$ be the number of clauses satisfied by $\vec{X}$ and $K(F)$ be the number of minimal false clauses under any assignment.

For every literal x, we use variable(x) to denote the variable appearing in x. That is, variable(x) $= x$ for positive literal x and variable($\overline{x}$) $= x$ for negative literal $\overline{x}$. If y is a literal, we use $\overline{y}$ to denote x if $y = \overline{x}$.

A partial (complete) assignment can be represented by a set of literals (or unit clauses) in which each variable appears at most (exactly) once and each literal is meant to be true in the assignment. If a variable x does not appear in a partial assignment A, then we say literals x and $\overline{x}$ are *unassigned* in A. Let $u(x)$ record the number of unit clauses x generated during the search. If there are no unit clauses in the input, $u(x)$ is initialized to zero.

The algorithm we use is presented in Figure 1. In the algorithm, $B(x) = \{y \mid (x \vee y) \in F, \text{variable}(x) < \text{variable}(y)\}$ for each literal x.

The following result is provided in [20].

Theorem 2.1 *Suppose F is a set of binary clauses on n variables. Then* max_2_sat2(F, n, g_0) *returns true if and only if there exists an assignment under which at most g_0 clauses in F are false. The time complexity of* dec_max_sat(F, n, g_0) *is $O(n2^n)$ and the space complexity is $L/2 + O(n)$, where L is the size of the input.*

3 Lower Bounds

In line 2 of dec_max_2_sat in Figure 1, popular lower bound functions can be used to improve its performance. The following two lower bound functions are used in [1, 2, 20]:

- LB1 = the number of conflicting (i.e., empty) clauses by the current partial assignment.
- LB2 = LB1 + $\sum_{j=i}^{n} \min(u(\overline{j}), u(j))$.

Figure 1: A decision algorithm for MAX-2-SAT.

```
function max_2_sat2 ( F: clause set, n: variable, g_0: int) return boolean
        // initiation
        for i := 1 to n do
                compute B(i) and B(ī) from F;
                u(i) := u(ī) := 0; // assuming no unit clauses in F
        end for
        return dec_max_2_sat(1, g_0);
end function

function dec_max_2_sat( i: variable, g: integer ) return boolean
1       if (i > n) return true; // end of the search tree
2       if (lower_bound(i) > g) return false
3       // decide if we want to set variable i to true
4       if (u(ī) ≤ g) ∧ (u(ī) < u(i) + |B(i)|) then
5               record_unit_clauses(ī);
6               if (dec_max_2_sat(i + 1, g − u(ī))) return true;
7               undo_record_unit_clauses(ī);
8       end if
9       // decide if we want to set variable i to false
10      if (u(i) ≤ g) ∧ (u(i) ≤ u(ī) + |B(ī)|) then
11              record_unit_clauses(i);
12              if (dec_max_2_sat(i + 1, g − u(i))) return true;
13              undo_record_unit_clauses(i);
14      end if
15      return false;
end function

procedure record_unit_clauses ( x: literal )
        for y ∈ B(x) do u(y) := u(y) + 1 end for;
end procedure

procedure undo_record_unit_clauses ( x: literal )
        for y ∈ B(x) do u(y) := u(y) − 1 end for;
end procedure
```

where $u(x)$ is the number of unit clauses x under the current partial assignment. Using LB2 instead LB1 contributes greatly to the improved performance of Alsinet, Manyà and Planes' implementation over Borchers and Furman's. It is easy to see that LB1 $\leq$ LB2 $\leq$ $K(F)$ when $g_0 \leq K(F)$.

Lemma 3.1 *If there is a clause $x \vee y$ in F such that $u(x) < u(\overline{x})$ and $u(y) < u(\overline{y})$, then LB2 + 1 $\leq K(F)$.*

The above lemma allows us to design an enhanced lower bound function as follows. For any literal x, let $c(x) = u(\overline{x}) - u(x)$ and S be the set of clauses of which both literals are unassigned under the current assignment. Then the new lower bound can be computed by the following procedure:

LB3 := LB2;
for every clause $(x \vee y) \in S$ **do**
  if $(c(x) > 0) \wedge (c(y) > 0)$ **then**
    LB3 := LB3 + 1; $c(x) := c(x) - 1$; $c(y) := c(y) - 1$;
  end if
end for

Because $c(x) > 0$ and $c(y) > 0$ imply $u(\overline{x}) > u(x)$ and $u(\overline{y}) > u(y)$, for a clause $x \vee y$, if we assign 0 to x, $u(y)$ will be increased by 1. So $\min(u(y), u(\overline{y}))$ will be increased by 1. The same reason applies to y. We can see that it is safe to increase the lower bound by 1 in this case.

Theorem 3.2 *If $g_0 \leq K(F)$, then LB2 $\leq$ LB3 $\leq$ $K(F)$.*

Note that the function lower_bound(i) in dec_max_2_sat (line 2) of Figure 1 returns LB2 − LB1 for our old implementation and returns LB3 − LB1 for NB (New Bound) in our experiments.

4 Using SCC

Given a 2CNF formula F, the implication graph, G_F, of F is a directed graph where the vertices are the set of the literals whose variables appear in F and there is an edge from x to y iff $\overline{x} \vee y$. It is proved in [3] that F is unsatisfiable iff G_F has a strongly connected component (SCC) which contains both x and $\overline{x}$ for some literal x. For MAX-2-SAT, we may use SCC to simplify the original problem:

- If a SCC does not contain any conflicting literals, delete the literals in this SCC from the original 2CNF formula.
- If there are more than one SCC, divide the original 2CNF formula according the SCCs and run MAX-2-SAT program against each component separately.

The idea of "divide-and-conquer" is not new in SAT. For instance, Bayardo and Pehoushek [5] have used this idea for counting models of SAT. However, it is a new approach to use this idea based on SCC for SAT. While this idea is appealing, in the study of random MAX-2-SAT, we tested thousands of instances but found that very few of them contain more than one SCC with conflict (i.e., with conflicting literals). This should not be a surprise because

in the study of random graphs, Erdös and Rnyi [10] showed that for a simply graph with n nodes, when the number m of edges grows from 0 to $n(n-1)/2$, the first cycle appears when $\epsilon n < m < (1/2 - \epsilon)n$ and then the graph consists of trees and a few unicyclic components until $m \approx n/2$. When $m = (1+\epsilon)n/2$ there is a unique giant component and all other components are trees or unicyclic; after that, all other components will merge into the giant component. The application of this result to MAX-2-SAT implication graphs is that only the giant component can contain conflicting literals. That is, it is not likely there is more than one SCC with conflict.

Despite of the fact that we have only one SCC with conflict in most cases, we can combine , the above idea with some known pre-processing methods [12, 6, 4, 14, 13, 11, 16] to simplify a 2CNF formula before running the decision algorithm. We found that the following sequence of operations is very effective for most 2CNF formulas:

- Complementary Unit Rule: If $F = \{x \vee y\} \wedge \{x \vee \overline{y}\} \wedge \{\overline{x} \vee z\} \wedge \{\overline{x} \vee \overline{z}\} \wedge F'$, then $K(F) = K(F') + 1$.
- Convert F to the implication graph G_F and locate SCCs in G_F. If a SCC does not contains conflicting literals, we delete this component.
- Resolution Rule: If $F = \{x \vee y\} \wedge \{\overline{x} \vee z\} \wedge F'$ and F' does not contain x and $\overline{x}$, $K(F) = K(F' \wedge (y \vee z))$.
- Export each SCC as a new 2CNF formula: For each edge $(x \rightarrow y)$ in the component, add $\overline{x} \vee y$ in the formula.
- Solve each new 2CNF formula by the MAX-2-SAT algorithm.

As shown later in the paper, this preprocessing procedure reduces the size of random instance greatly and improves considerably the performance of our algorithm for random MAX-2-SAT instances.

Another new usage of SCCs is to design a variable ordering for the algorithm in Figure 1. This algorithm uses a fixed ordering, i.e, from 1 to n, to assign truth value to variables. It is found useful in [20] to sort the variables according their occurrences in the input in non-increasing order and then assign variables in that order. Using SCC, we design a new weight function for variables and we then sort the variables by this weight in non-increasing order.

The weight function is computed using the following procedure: At first each variable has a weight equal to 0. Then we update the weight by finding the shortest path between every pair of $(x, \overline{x})$ in the SCC. For any node y in the path, we increase the weight of y by 1.

The intuition behind this ordering is that we want to derive conflicting clauses as early as possible so that the lower bound checking at line 2 of the algorithm in Figure 1 can be more effective. If a clause appears in the shortest path connecting two conflicting literals in a SCC, we give the literals in this clause higher weights so that the truth value of this clause can be decided early. The experimental results in the next section show that the ordering by the new weight function performs better than the occurrence ordering when either $c = m/n$ or $K(F)$ is small.

5 Experimental Results

We have implemented all the three techniques presented in this paper to support the algorithm presented in Figure 1. The implementation is written in C++ and preliminary experiment results seem promising. We ran our program with four different configurations: sorting variables

Table 1: Experimental results on Borchers and Furman's examples. (in seconds)

Problem		false clauses	BF	AMP	OLD	SONB
#vars	#clauses					
50	100	4	0.02	0.03	0.01	0.02
	150	8	0.06	0.03	0.01	0.03
	200	16	4.18	0.38	0.06	0.04
	250	22	24	0.26	0.03	0.08
	300	32	350	4.88	0.71	0.4
	350	41	2556	10	1.26	0.49
	400	45	2308	4.65	0.23	0.24
	450	63	–	44	4.42	2.98
	500	66	–	17	1.04	0.48
100	200	5	0.14	0.16	0.05	0.07
	300	15	501	29	4.0	0.64
	400	29	–	1204	124	8.3
150	350	4	0.18	0.22	1.71	0.24
	450	22	–	–	–	235

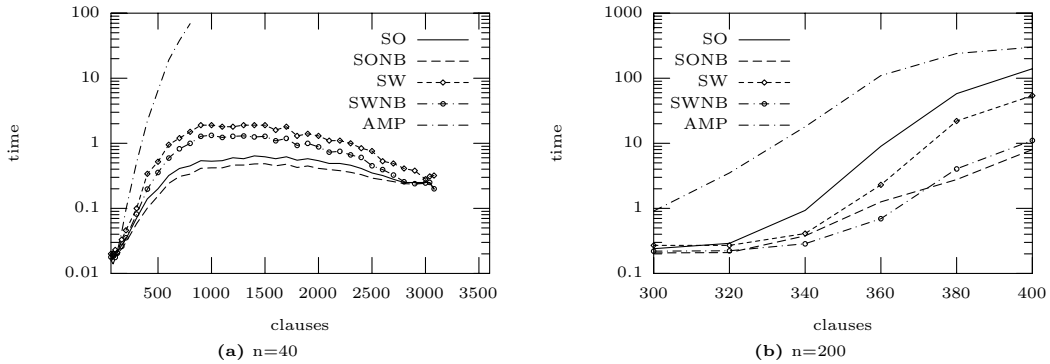
by occurrence (SO), sorting variables by occurrence with new lower bound (SONB), sorting variables by weight (SW) and sorting variables by weight with new lower bound (SWNB). All data are collected on a cluster of Pentium 4 2.4GHz linux machines each with 1GB memory.

Table 1 shows some results of Borchers and Furman's program (BF) [6], Alsinet et al.'s (AMP, the option LB2-I+JW is used), our old implementation (OLD) and our new implementation (SONB) on the random problems distributed by Borchers and Furman. In the table, "–" indicates an incomplete run after running for two hours. Note that the upbound of $K(F)$ in our algorithm is at first set to the number found by the first phase of Borchers and Furman's local search procedure and then decreased by one until the optimal value is decided. It is clear that our algorithm runs consistently faster than both Borchers and Furman's program and Alsinet et al.'s modification. For simple and small size problems, the old implementation sometimes is faster than the new implementation, because our new program uses a more complex preprocessing procedure. For hard problems, our new implementation is definitely faster than the old one.

Figure 2 compares Alsinet et al.'s program and our implementation on the random problems of 40 variables and 200 variables. We considered the following cases: $n = 40$ variables with $m = 60, 80, 100, 120, 160, 200, 300, ..., 3000, 3040, 3080$ clauses and $n = 200$ variables with $m = 300, 320, 340, 360, 380, 400$ clauses. For each case, we generated 100 random problems (excluding satisfiable ones). It is clear that all four configurations of our algorithm run consistently faster than Alsinet et al.'s program. From the figure, we can see that SO is better than SW on the instances of $n = 40$ variables, but SW is better than SO in many instances of the $n = 200$ variables. The new lower bound (LB3) is consistently better than LB2. Note that in the figure of $n = 40$ variables, the running time of our program is decreasing while $c = m/n$ is increasing when $c = m/n$ is large enough. The reason for this is that the preprocessing can reduce a lot of clauses when $c = m/n$ is very large. For $n = 200$, Alsinet et al.'s program could not finish one job when $m = 340$, 4 jobs when $m = 360$, 14 jobs when $m = 380$ and 35 jobs when $m = 400$. When $m = 400$, both SO and SW have one job unfinished (in two hours).

More detailed results comparing the four configurations, i.e., SO, SONB, SW, and SWNB, can be found in the full version of this paper (see also the appendix). In the following, we turn

Figure 2: Running time for SO, SONB, SW, SWNB and Alsinet et al.'s algorithm.



(a) n=40 **(b)** n=200

our attention to the Max-Cut problem.

Given an undirected simple graph $G = (V, E)$, where $V = \{x_1, ..., x_n\}$ is the set of vertices and E is the set of edges, let weight w_{x_i,x_j} be associated with each edge $(x_i, x_j) \in E$. The Max-Cut problem is to find a subset S of V such that

$$W(S, \overline{S}) = \sum_{x_i \in S, x_j \in \overline{S}} w_{x_i,x_j}$$

is maximized, where $\overline{S} = V - S$. In this paper, we let weight $w_{x_i,x_j} = 1$ for all edges. The following theorem shows how to reduce the Max-Cut problem to the MAX-2-SAT problem.

Theorem 5.1 [7, 15] *Given $G = (V, E)$, where $|V| = n$, $|E| = m$, we assume weight $w(x_i, x_j) = 1$ for each $(x_i, x_j) \in E$. We construct an instance of MAX-2-SAT as follows: Let V be the set of propositional variables and for each edge $(x_i, x_j) \in E$, we create exactly two binary clauses: $(x_i \vee x_j)$ and $(\overline{x_i} \vee \overline{x_j})$. Let F be the collection of such binary clauses, then the Max-Cut problem has a cut of weight k iff the MAX-2-SAT problem has an assignment under which $m + k$ clauses are true.*

We have run thousands instances of the random Max-Cut problem. The performance of our MAX-2-SAT solver is much better than a special-purpose program for Max-Cut. Some results are shown in the full version of the paper (see also the appendix). For the instances where the number of variables $n = 30$, 50, and 100, SO is better than SW. Those instances have a relatively large $c = m/n > 2$ in most cases. For the instances with $n = 1000$, SW is often more than 10 times faster than SO when $c = m/n < 2$.

6 Conclusion

We have given a new lower bound function for the branch-and-bound algorithm of MAX-2-SAT and showed that the new lower bound function is consistently better than other lower bound functions. We have also used SCCs of the implication graph of a 2CNF formula to simplify the formula and to design a new weight for variables. The experiments showed that sorting variables by weight performs much better when the clause to variable ratio $c = m/n$ is less than 2. The proposed preprocessing technique can reduce the size of the input greatly. The result of this research is a high-performance implementation of an exact algorithm for MAX-2-SAT, which outperforms any implementation we know about in the same category. We applied the MAX-2-SAT algorithm to solve large instances of Max-Cut problem, and the experimental results showed this approach is feasible and effective. All the techniques presented in the paper

can be applied to the weighted MAX-2-SAT problem where each clause has a weight. The Max-Cut problem with arbitrary weights can be easily converted into an instance of the weighted MAX-2-SAT. As future work, we will specialize and improve the MAX-2-SAT algorithm for the Max-Cut problem and will solve some real world Max-Cut problems. We will extend the techniques presented in the paper to solve general MAX-SAT problems.

References

[1] T. Alsinet, F. Manyà, J. Planes, Improved branch and bound algorithms for Max-SAT. *Proc. of 6th International Conference on the Theory and Applications of Satisfiability Testing*, SAT2003, pages 408-415.

[2] T. Alsinet, F. Manyà, J. Planes, Improved branch and bound algorithms for Max-2-SAT and Weighted Max-2-SAT. *Catalonian Conference on Artificial Intelligence*, 2003.

[3] B. Aspvall, M. F. Plass and R. E. Tarjan, A linear-time algorithm for testing the truth of certain quantified Boolean formulas, *Information Processing Letters*, 1979, 8(3):121-123

[4] N. Bansal, V. Raman, Upper bounds for MaxSat: Further improved. In Aggarwal and Rangan (eds.): *Proceedings of 10th Annual conference on Algorithms and Computation*, ISSAC'99, volume 1741 of Lecture Notes in Computer Science, pages 247-258, Springer-Verlag, 1999.

[5] R.J. Bayardo and J.D. Pehoushek. Counting models using connected components. In 7th Nat'l Conf. on Artificial Intelligence (AAAI), pages 157–162, 2000.

[6] B. Borchers, J. Furman, A two-phase exact algorithm for MAX-SAT and weighted MAX-SAT problems. *Journal of Combinatorial Optimization*, 2(4):299-306, 1999.

[7] J. Cheriyan, W.H. Cunningnham, L. Tuncel, Y. Wang. A linear programming and rounding approach to Max 2-Sat. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 26:395–414, 1996.

[8] E. Dantsin, A. Goerdt, E.A. Hirsch, R. Kannan, J. Kleinberg, C. Papadimitriou, P. Raghavan, U. Schöning. A deterministic $(2 - 2/(k + 1))^n$ algorithm for k–SAT based on local search. *Theoretical Computer Science*, 2002.

[9] M. Davis, G. Logemann, D. Loveland, A machine program for theorem-proving. *Communications of the Association for Computing Machinery*, 7 (July 1962), 394–397.

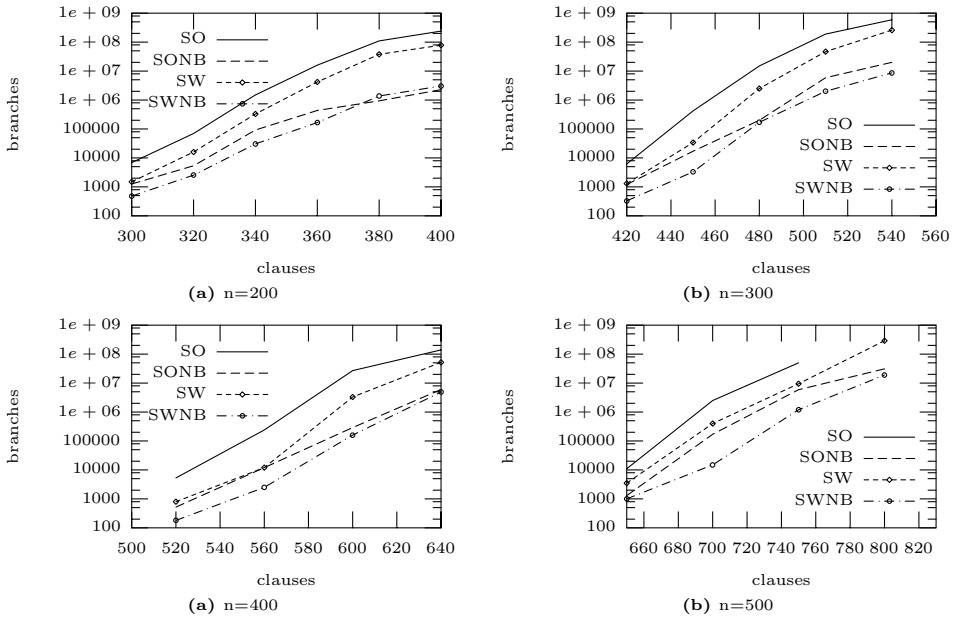
[10] P. Erdös and A. Rnyi, On the evolution of random graphs. *Mat. Kutato Int.* Kozl 5 (1960), 17-61.

[11] J. Gramm, E.A. Hirsch, R. Niedermeier, P. Rossmanith: New worst-case upper bounds for MAX-2-SAT with application to MAX-CUT. Preprint, submitted to Elsevier, May, 2001

[12] P. Hansen, B. Jaumard. Algorithms for the maximum satisfiability problem. *Computing*, 44:279-303, 1990.

[13] E.A. Hirsch. A new algorithm for MAX-2-SAT. In *Proceedings of 17th International Symposium on Theoretical Aspects of Computer Science*, STACS 2000, vol. 1770, Lecture Notes in Computer Science, pages 65-73. Springer-Verlag.

Figure 3: Performance comparison for SO, SONB, SW and SWNB



[14] E.A. Hirsch. New worst-case upper bounds for SAT. *Journal of Automated Reasoning*, 24(4):397-420, 2000.

[15] M. Mahajan, V. Raman. Parameterizing above guaranteed values: MaxSat and MaxCut. *Journal of Algorithms*, 31:335–354, 1999.

[16] R. Niedermeier, P. Rossmanith. New upper bounds for maximum satisfiability. *Journal of Algorithms*, 36:63-88, 2000.

[17] H. Shen, H. Zhang, An empirical study of max-2-sat phase transitions, *Proc. of LICS'03 Workshop on Typical Case Complexity and Phase Transitions*, Ottawa, CA, June 2003.

[18] R. Wallace, E. Freuder. Comparative studies of constraint satisfaction and Davis-Putnam algorithms for maximum satisfiability problems. In D. Johnson and M. Trick (eds.) *Cliques, Coloring and Satisfiability*, volume 26, pages 587-615, 1996.

[19] H. Xu, R.A. Rutenbar, K. Sakallah, sub-SAT: A formulation for related boolean satisfiability with applications in routing. ISPD'02, April, 2002, San Diego, CA.

[20] H. Zhang, H. Shen, F. Manyà: Exact algorithms for MAX-SAT. In *Proc. of International Workshop on First-order Theorem Proving* (FTP 2003). http://www.elsevier.com/gejng/31/29/23/135/23/show/Products/notes/index.htt

Appendix

Figure 3 and Table 2 show the results of four different configurations: SO, SONB, SW and SWNB on random MAX-2-SAT problems. Each case has 100 random instances. We can see that the preprocessing procedure can greatly reduce the problem size. We can also see that SW is faster than SO and SWNB is faster than SONB when $c = m/n$ is relatively small. Our new lower bound function can always prune more branches.

Table 2: Experimental results on random MAX-2-SAT instances. (Reduced is the problem after preprocessing)

Problem		Reduced		SO	SONB	SW	SWNB
#vars	#clauses	#vars	#clauses	branches	branches	branches	branches
200	300	53.4	80.8	6.88e+03	1.26e+03	1.46e+03	480
200	320	66	106	6.97e+04	5.37e+03	1.6e+04	2.54e+03
200	340	81	140	1.46e+06	9.2e+04	3.27e+05	3.04e+04
200	360	95.5	177	1.57e+07	4.37e+05	4.16e+06	1.67e+05
200	380	105	204	1.09e+08	9.38e+05	3.81e+07	1.38e+06
200	400	116	237	2.39e+08	2.28e+06	7.81e+07	3.06e+06
300	420	56.6	79.6	6.24e+03	1.19e+03	1.3e+03	326
300	450	77.4	115	4.16e+05	1.66e+04	3.42e+04	3.32e+03
300	480	102	166	1.48e+07	2.01e+05	2.51e+06	1.73e+05
300	510	122	210	1.89e+08	5.87e+06	4.71e+07	1.99e+06
300	540	139	253	5.93e+08	2.01e+07	2.63e+08	8.66e+06
400	520	49.2	64.1	5.28e+03	521	798	176
400	560	73.3	103	2.42e+05	1.25e+04	1.15e+04	2.51e+03
400	600	107	162	2.67e+07	2.88e+05	3.3e+06	1.6e+05
400	640	132	210	1.39e+08	5.97e+06	5.23e+07	4.91e+06
500	650	59	76	1.1e+04	1.3e+03	3.5e+03	1e+03
500	700	89	120	2.5e+06	1.7e+05	4e+05	1.5e+04
500	750	132	197	5e+07	5.9e+06	9.5e+06	1.2e+06
500	800	171	277	–	3.1e+07	2.9e+08	1.9e+07

Figure 4: Performance of SONB and SWNB for Max-Cut Problems