Study of Lower Bound Functions for MAX-2-SAT*

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

Abstract

Recently, several lower bound functions are proposed for solving the MAX-2-SAT problem optimally in a branch-and-bound algorithm. These lower bounds improve significantly the performance of these algorithms. Based on the study of these lower bound functions, we propose a new, linear-time lower bound function. We show that the new lower bound function is admissible and it is consistently and substantially better than other known lower bound functions. The result of this study is a high-performance implementation of an exact algorithm for MAX-2-SAT which outperforms any implementation of the same class.

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 (Hansen, & Jaumard) and hardware circuit designs (Xu et al), efficient methods that can solve a large set of instances of MAX-SAT are eagerly sought. One important area of applications of MAX-2-SAT is that many NP-complete graph problems such as *maximum cut*, *independent set*, can be reduced to special instances of MAX-2-SAT (Cheriyan et al; Mahajan & Raman). Many of the proposed methods for MAX-SAT are based on approximation algorithms (Dantsin et al); some of them are based on branch-and-bound methods (Hansen, & Jaumard; Borchers & Furman; Bansal & Raman; Hirsch; Hirsch b; Gramm et al; Niedermeier & Rossmanith); and some of them are based on transforming MAX-SAT into SAT (Xu et al).

To the best of our knowledge, there are only six implementations of exact algorithms for MAX-SAT that are variants of the well-known Davis-Putnam-Logemann-Loveland (DPLL) procedure (Davis et al). One is due to (Wallace & Freuder) (implemented in Lisp); one is due to (Borchers & Furman) (implemented in C and publicly available); the next two are made available in 2003 by (Alsinet et al) and (Zhang et al), respectively. The last one is provided by (Zhao & Zhang) in 2004.

Since it is well-known that the tighter the bound the smaller the search tree in a typical branch-and-bound algorithm, it is not a surprise to see that Alsinet et al.'s implementation is better than Borchers and Furman's implementation because a better lower bound function is used in (Alsinet et al), and Shen and Zhang's implementation outperformed Alsinet et al.'s implementation because of another better lower bound function.

After studying these lower bound functions, we present in this paper a new lower bound function to further improve the performance of the branch-and-bound algorithm for MAX-2-SAT. We show that these new lower bound functions are admissible and provide tighter bounds than any known lower bound functions. In order to evaluate the new lower bound function, we present experimental results on many examples. We showed that the improved algorithm is consistently and substantially better than all other known algorithms (Borchers & Furman; Alsinet et al; Zhang et al).

Preliminary

Let F be a formula in CNF 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 in F satisfied by $\vec{X}$ and $K(F)$ be the minimal number of false clauses under any assignment. That is,

$$K(F) = m - \max\{S(\vec{X}, F) \mid \vec{X} \in \{0, 1\}^n\}.$$

If we use the right-hand side of the above equation for computing $K(F)$, we obtain a naive enumeration algorithm for MAX-SAT. $K(F)$ can also be computed by a simple recursive function as follows:

$$K(F) = \min(K(F_x), K(F_{\overline{x}}))$$

where F_x is the formula obtained from F by replacing x by 1 and $\overline{x}$ by 0. Algorithm based on the above equation

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

is also called Davis-Putanm-Logemann-Loveland procedure (Davis et al).

To speed up the computation, a lower bound is often introduced and the algorithm is called branch-and-bound algorithm. This paper introduces a new lower bound to improve the performance of the algorithm. To facilitate the discussion, we borrow some convention and the algorithm presented in (Zhang et al).

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 variable(x) does not appear in a partial assignment A, then we say literals x and $\overline{x}$ are *unassigned* in A. For an unassigned literal x, $\mu(x)$ is the number of the unit clause for literal x in the current clause set, including those generated during the search. Initially, $\mu(x)$ is the number of unit clauses for literal x in the input.

The algorithm taken from (Zhang et al) 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. By abuse of notations, we assume x_i is represented by i and $\overline{x_i}$ is represented by $\bar{i}$, where $1 \leq i \leq n$. Hence, we have $\mu(i) = \mu(x_i)$, $B(i) = B(x_i)$, $\mu(\bar{i}) = \mu(\overline{x_i})$ and $B(\bar{i}) = B(\overline{x_i})$..

The following result is provided in (Zhang et al).

Theorem 1 *Suppose F_0 is a set of binary clauses of n variables. Then max_2_sat(F_0, n) will return the minimum number of false clauses under any assignment. The time complexity of max_2_sat(F_0, n) is $O(n2^n)$ and the space complexity is $L/2 + O(n)$, where L is the size of the input.*

Lower Bounds

Since the algorithm is trying to find the minimum number of false clauses under any assignment, lower bounds which estimate the minimum number of false clauses at a given search node are useful for cutting out some search space. In line 3 of bb_max_2_sat in Figure 1, popular lower bound functions can be used to improve its performance. We may also compute upper bound functions to cut off some search space. In practice, we all adopted the two-phase approach used by Borchers and Furman by first calling a local search algorithm for computing an upper bound before the search starts. We found that this approach is much more effective than computing an upper bound in each internal node of the search tree.

However, for lower bounds, we have not found such an effective function which can be computed before the search. Many algorithms from linear programming can be used for computing lower bounds. However, since this lower bound function is called in every internal node of the search tree, it must be very efficient. In fact, the lower bounds proposed in (Alsinet et al), (Shen & Zhang), and our new lower bound function take only linear-time (in terms of the input size).

The following three lower bound functions are used in (Borchers & Furman; Alsinet et al; Shen & Zhang 2004):

Figure 1: An exact algorithm for MAX-2-SAT.

```
function max_2_sat ( F_0: clause set, n: integer)
    // initiation
    min_fcl := MAX_INT;
    for i := 1 to n do
        compute B(i) and B(ī) from F_0;
        // assuming no unit clauses in F_0
        μ(i) := μ(ī) := 0;
    end for
    bb_max_2_sat(1, n, 0);
    return min_fcl;
end function

function bb_max_2_sat( i, n: integer, k: integer )
1   if (i > n) // end of the search tree
2       if (k < min_fcl) min_fcl := k;
3   else if (lower_bound(i) + k < min_fcl)
4       if (μ(ī) + k < min_fcl) ∧ (μ(ī) < μ(i) + |B(i)|)
5           record_unit_clauses(x̄_i);
6           bb_max_2_sat(i + 1, n, k + μ(ī));
7           undo_record_unit_clauses(x̄_i);
8       end if
9       if (μ(i) + k < min_fcl) ∧ (μ(i) ≤ μ(ī) + |B(ī)|)
10          record_unit_clauses(x_i);
11          bb_max_2_sat(i + 1, n, k + μ(i));
12          undo_record_unit_clauses(x_i);
13      end if
14  end if
end function

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

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

Figure 2: The function for computing LB3.

```
function lower_bound3 ( i: integer ) return integer
1   LB3 = LB2;
2   for (x ∨ y) in S do
3       if (t(x) > 0) ∧ (t(y) > 0)
4           LB3 = LB3 + 1;
5           t(x) = t(x) − 1; t(y) = t(y) − 1;
6   end for
7   return LB3
end function
```

- Borchers and Furman's lower bound:
  LB1 = the number of conflicting (or empty) clauses by the current partial assignment;
- Alsinet, Manyà and Planes' lower bound:
  LB2 = LB1 + $\sum_{j=i}^{n} \min(\mu(\bar{j}), \mu(j))$;
- Shen and Zhang's lower bound:
  LB3 = lower_bound3(i).

Here $\mu(x)$ is the number of the unit clause for literal x under the current partial assignment and the function lower_bound3 is provided in Figure 2. 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 $\leq$ $K(F)$. In general, a lower bound LB is said to be *admissible* if LB $\leq$ $K(F)$. In other words, both LB1 and LB2 are admissible.

To compute LB3, for any literal x, let $t(x) = \mu(\overline{x}) - \mu(x)$ and S be the set of clauses of which both literals are unassigned under the current assignment.

The following theorem is given in (Shen & Zhang 2004):

Theorem 2 *LB1 $\leq$ LB2 $\leq$ LB3 $\leq$ $K(F)$.*

To prove the above theorem, we may use the following lemma:

Lemma 3 *If there is a clause $x \vee y$ in F such that $\mu(x) < \mu(\overline{x})$ and $\mu(y) < \mu(\overline{y})$, then LB2 $+ 1 \leq K(F)$.*

Proof Given $\mu(\overline{x}) > \mu(x)$ and $\mu(\overline{y}) > \mu(y)$, for a clause $x \vee y$, if we assign 0 to x, $\mu(y)$ will be increased by 1. So $\min(\mu(y), \mu(\overline{y}))$ in LB2 will be increased by 1. If we assign 1 to x, then $\min(\mu(x), \mu(\overline{x}))$ in LB2 will be increased because $\mu(\overline{x}) > \mu(x)$. In both cases, we can see that it is safe to increase LB2 by 1. []

A New Lower Bound

We design another lower bound, lower_bound4 in Figure 3, to fully explore the property revealed by Lemma 3.

It is easy to prove the following result.

Lemma 4 *The worst-case time complexity of lower_bound4(i) is $O((n-i)+\sum_{j=i}^{n} \max(|B(j)|, |B(\bar{j})|))$.*

Before we prove formally that LB4 is indeed an admissible lower bound for MAX-2-SAT, let us look at a simple example.

Figure 3: The function for computing LB4.

```
function lower_bound4 ( i: integer) return integer
1   LB4 = LB1;
2   while (i ≤ n) do
3       LB4 = LB4 + min(μ(ī), μ(i));
4       t = |μ(ī) − μ(i)|;
5       if μ(ī) > μ(i)  Y = B(i) else Y = B(ī)
6       for y ∈ Y if (t > 0) do
7           t = t − 1;
8           μ(y) = μ(y) + 1;
9       end for
10      i = i + 1
11  end while
12  return LB4
end function
```

Example 5 Suppose a 2CNF formula F is represented by μ and S, where $\mu(a) = \mu(b) = \mu(c) = \mu(\overline{c}) = 0$, $\mu(\overline{a}) = \mu(\overline{b}) = 1$ and $S = \{a \vee c, b \vee \overline{c}\}$. It is easy to see $K(F) = 1$. LB3 = 0 because there exists no clause $x \vee y$ in F such that $\mu(\overline{x}) > \mu(x)$ and $\mu(\overline{y}) > \mu(y)$.

Suppose $a < b < c$, then LB4 = 1 because $a \vee c$ will increase $\mu(c)$ by 1 and $b \vee \overline{c}$ will increase $\mu(\overline{c})$ by 1, making $\min(\mu(\overline{c}), \mu(c)) = 1$.

Let $F(i)$ be the remaining nontrivial (neither empty nor tautology) clauses when bb_max_2_sat(i, n, k) in Figure 1 is called. Recall that for any unassigned literal x, $\mu(x)$ is the number of the unit clause x and k in the procedure bb_max_2_sat(i, n, k) is the number of empty clauses created by the partial assignment on variables x_1 to x_{i-1}. Let $\mathbf{u}(x)$ be the multiset of $\mu(x)$ copies of the unit clause x and $xB(x)$ denote the set $\{(x \vee y) \mid y \in B(x)\}$. Then we have

$$F(i) = \mathbf{u}(x_i) \cup \mathbf{u}(\overline{x_i}) \cup x_i B(i) \cup \overline{x_i} B(\bar{i}) \cup F(i+1)$$

where $F(i+1)$ is the subset of $F(i)$ not containing the variable x_i (assuming $F(n+1) = \emptyset$).

Let F be $F(i)$ and x be x_i in

$$K(F) = \min(K(F_x), K(F_{\overline{x}}))$$

we have

$$\begin{aligned} K(F(i)_{x_i}) &= \mu(\bar{i}) + K(B(\bar{i}) \cup F(i+1)) && (1)\\ K(F(i)_{\overline{x_i}}) &= \mu(i) + K(B(i) \cup F(i+1)) && (2)\\ K(F(i)) &= \min(K(F(i)_{x_i}), K(F(i)_{\overline{x_i}})) && (3) \end{aligned}$$

Since $K(F(i+1)) \leq K(Y \cup F(i+1))$ for any clause set Y, we have $\min(\mu(\bar{i}), \mu(i)) + K(F(i+1)) \leq K(F(i))$, by discarding $B(i)$ and $B(\bar{i})$ from (1) and (2), respectively. This relation shows why LB2 is an admissible lower bound. To show that LB4 is admissible, we need the following lemma.

Lemma 6 *For any subset $X \subseteq B(i)$, if $|X| \leq \mu(\bar{i}) - \mu(i)$, then*

$$\min(\mu(\bar{i}), \mu(i)) + K(X \cup F(i+1)) \leq K(F(i)) \qquad (4)$$

Proof If $|X| = 0$, then (4) is trivially true. If $|X| > 0$, then $\mu(\bar{i}) > \mu(i)$ and (4) becomes

$$\mu(i) + K(X \cup F(i+1)) \leq K(F(i)) \quad (5)$$

According to (3), there are two cases:

Case 1: $K(F(i)_{x_i}) < K(F(i)_{\overline{x_i}})$. So $K(F(i)) = K(F(i)_{x_i}) = \mu(\bar{i}) + K(B(\bar{i}) \cup F(i+1))$ by (1). Because $\mu(\bar{i}) \geq \mu(i) + |X|$ and $K(B(\bar{i}) \cup F(i+1)) \geq K(F(i+1)) \geq K(X \cup F(i+1)) - |X|$, we have $K(F(i)) \geq \mu(i) + |X| + K(X \cup F(i+1)) - |X|$, or (5).

Case 2: $K(F(i)_{x_i}) \geq K(F(i)_{\overline{x_i}})$. So $K(F(i)) = K(F(i)_{\overline{x_i}}) = \mu(i) + K(B(i) \cup F(i+1))$ by (2). Since $X \subseteq B(i)$, $K(B(i) \cup F(i+1)) \geq K(X \cup F(i+1))$. So (5) holds. []

A mirror of the above lemma can be obtained by switching x_i and $\overline{x_i}$.

Theorem 7 *LB4* $\leq K(F)$.

Proof We will use Lemma 6 (or its mirror) at each iteration of i in lower_bound4. If $i = n$, then LB4 = LB3 = LB2 = $\min(\mu(i), \mu(\bar{i}))$. If $i < n$, suppose LB4′ is the value computed inside lower_bound4 for $X \cup F(i+1)$, where

$$F(i) = \mathbf{u}(x_i) \cup \mathbf{u}(\overline{x_i}) \cup X_i B(i) \cup \overline{x_i} B(\bar{i}) \cup F(i+1).$$

As an inductive hypothesis, we assume

$$\text{LB4}' \leq K(X \cup F(i+1)).$$

If $\mu(\bar{i}) > \mu(i)$ then $Y = B(i)$ and a subset X of Y is added at lines 6–9, where $|X| \leq t$ as computed at line 4. Since LB4 $= \min(\mu(\bar{i}), \mu(i)) +$ LB4′, by induction hypothesis, LB4 $\leq \min(\mu(\bar{i}), \mu(i)) + K(X \cup F(i+1)) \leq K(F(i))$. The case when $\mu(\bar{i}) \leq \mu(i)$ is similar. In both cases, LB4 $\leq K(F(i))$. []

Comparison of LB3 and LB4

In (Shen & Zhang 2004), it has been shown by experimental results that LB3 can improve the performance of MAX-2-SAT ten times faster than other lower bounds (LB1 and LB2). Since LB3 and LB4 have the same worst-case time complexity, we wish that LB4 is never smaller than LB3, as Example 5 already showed that LB4 can be greater than LB3.

Let us look at Example 5 again. We saw that LB4 = 1 when $a < b < c$. The same result can be obtained when $b < a < c$, $a < c < b$, or $b < c < a$. However, if $c < a < b$ (or $c < b < a$), then LB4 = 0 because neither $\mu(a)$ nor $\mu(b)$ can be increased by the two binary clauses.

This simple example shows that the value of LB4 depends on how the propositional variables are ordered in the implementation. In contrast, the value of LB3 does not depend on variable ordering. Variable ordering is an important performance issue of the algorithm in Figure 1. Several orderings have been compared in (Shen & Zhang 2004), including an ordering based on the implication graph of F.

The next example shows that the ordering for choosing clauses in the computation of LB3 and LB4 affect their values.

Figure 4: The function for computing LB4a.

```
function lower_bound4a ( i: variable) return integer
1   LB4 = LB1;
2   while (i ≤ n) do
3       LB4 = LB4 + min(μ(ī), μ(i));
4       t = |μ(ī) − μ(i)|;
5       if μ(ī) > μ(i)  Y = B(i) else Y = B(ī)
6       S = ∅
7       for j ∈ Y if (t > 0) do
8           if (μ(j) ≥ μ(j̄))  S = S ∪ {j} else
9               t = t − 1;
10              μ(j) = μ(j) + 1;
11      end for
12      for j ∈ S if (t > 0) do
13          t = t − 1;
14          μ(j) = μ(j) + 1;
15      end for
16      i = i + 1
17  end while
18  return LB4
end function
```

Example 8 Let F be represented by μ and S, where $\mu(a) = \mu(b) = \mu(c) = \mu(d) = 0$, $\mu(\bar{a}) = \mu(\bar{b}) = \mu(\bar{c}) = \mu(\bar{d}) = 1$, and $S = \{a \vee b, b \vee c, c \vee d\}$. Obviously, $K(F) = 2$. If $b \vee c$ is chosen first at line 2 of lower_bound3, then LB3 = 1; otherwise, LB3 = 2. Similarly, for computing LB4, if $b \vee c$ is chosen first at line 6 of lower_bound4, (either b or c must be the smallest in the variable ordering.) then LB4 = 1; otherwise LB4 = 2.

This example shows that LB3 may be 2 while LB4 may be 1. Thus LB3 and LB4 are different in general. Now we have the following question: If the same clause ordering is used for computing both LB3 and LB4, is it true that LB3 $\leq$ LB4? The answer is unfortunately no, as illustrated by the following example.

Example 9 Suppose F is represented by μ and S, where $\mu(a) = \mu(b) = \mu(c) = \mu(\bar{c}) = 0$, $\mu(\bar{a}) = \mu(\bar{b}) = 1$ and $S = \{a \vee c, a \vee b\}$. Then $K(F) = 1$ and LB3 = 1 because of $a \vee b$. However, if the clauses are chosen in the given order, then LB4 = 0 because $t = 1$ for a is used on c when $a \vee c$ is chosen.

To avoid the problem illustrated in the above example, at line 6 of lower_bound4, we may prefer those clauses in Y that will increase the value of LB4. This idea is implemented as function lower_bound4a in Figure 4. Those literals in Y that will not increase the value of LB4 immediately are stored in $S \subseteq Y$ and processed later at lines 13–16.

However, we do not have a proof that LB4 computed by lower_bound4a is always greater than or equal to LB3. To make sure LB4 $\geq$ LB3, let us say a clause ordering $<_c$ is LB3-*compatible* if for any two clauses $(a \vee b)$ and $(c \vee d)$, if $(a \vee b)$ satisfies $(t(a) > 0) \wedge (t(b) > 0)$ but $(c \vee d)$ does not, then $(a \vee b) <_c (c \vee d)$.

Theorem 10 *If we select clauses from Y at line 6 of lower_bound4(i) using a LB3-compatible clause ordering (from small to big), then LB3 $\leq$ LB4.*

In Example 9, any LB3-compatible ordering will place $a \vee b$ before $a \vee c$. Using this ordering for computing LB4, we have LB3 = LB4 = 1.

The only extra cost for using a LB3-compatible clause ordering in lower_bound4 is that we have to execute lines 6–9 twice: The first time for those $y \in Y$, $t(y) > 0$, and the second time for those such that $t(y) \leq 0$. In other words, using this ordering will not compromise the worst-case time complexity of lower_bound4. However, our implementation of LB4 using a LB3-compatible clause ordering is slightly slower than lower_bound4a, because of this second visit.

Experimental Results

We already implemented the algorithm presented in Figure 1 and the lower bound functions LB2 and LB3 (Shen & Zhang 2004). The implementation is written in C++ and the code is available to public. We have also implemented two lower bounds presented in this paper: lower_bound4 and lower_bound4a, and they are denoted by LB4 and LB4a, respectively. Note that the function lower_bound(i) in bb_max_2_sat (line 3) of Figure 1 returns a lower bound value minus LB1 in our experiments.

Table 1 shows the performance of five implementations on Borchers and Furman's benchmarks. The six implementations are BF (Borchers & Furman), AMP (Alsinet et al), ZZ (Zhao & Zhang), LB3 (Shen & Zhang 2004), LB4 and LB4a. The benchmarks contain randomly generated 2-CNFs of 50, 100, and 150 variables (#var), and are available from their web site. It is clear from that the table that LB4a provides the best result while LB3 and LB4 are ranked second. The harder the problem, the more significant the gain of LB4a over other implementations.

(Shen & Zhang 2004) have shown that LB3 is about ten times faster than other implementations on many random 2-CNFs. Table 2 provides a detailed comparison of LB3 and LB4a. The instances are random 2CNF with 100 or 200 variables and various number of clauses. The data show the average runs on 100 instances (of the same numbers of variables and clauses). It is clear from the table that LB4a explores only $1/4$ of the search space (represented by the number of branches in the search tree) of LB3 and LB4a is about 3 times faster than LB3 on average. Note that the number in the parentheses following the run time is the number of unfinished cases after two hours of running.

Conclusion

After studying several lower bound functions for the branch-and-bound algorithm of MAX-2-SAT, we have proposed a new lower bound function and proved that it is admissible and it is indeed a better lower bound function than LB2 and LB3. We have implemented this function and our experimental results showed that the lower bound function is consistently and substantially better than other lower bound functions. All the techniques presented in the paper can be applied to the weighted MAX-2-SAT problem where each

Table 1: Experimental results on Borchers and Furman's examples. The times (in seconds) are collected on a Pentium 4 linux machine, 2.4GHz, 1G memory; "–" means the job doesn't finish in one hour; $*$ means the output was erroneous.

Problem		#fcl	BF	AMP	ZZ	LB3	LB4	LB4a
#var	#cls							
50	100	4	0.02	0.03	0.31	0.02	0.02	0.01
	150	8	0.06	0.03	0.46	0.02	0.02	0.02
	200	16	4.18	0.38	0.55	0.03	0.03	0.03
	250	22	24	0.26	0.62	0.05	0.04	0.04
	300	32	350	4.88	1.05	0.09	0.09	0.07
	350	41	2556	10	1.58	0.09	0.17	0.11
	400	45	2308	4.65	1.23	0.05	0.08	0.06
	450	63	–	44	6.04	0.23	0.32	0.21
	500	66	–	17	4.15	0.12	0.17	0.15
100	200	5	0.14	0.16	0.41	0.03	0.03	0.04
	300	15	501	29	0.69	0.42	0.66	0.4
	400	29	–	1204	3.92	1.26	3.15	0.98
	500	44	–	–	968	192	180	62
	600	65	–	–	–	303	394	130
150	300	4	0.18	0.22	*	0.12	0.10	0.08
	450	22	–	–	*	64	19	7.11
	600	38	–	–	*	347	385	54

Table 2: Experimental results on random instances. The times (in seconds) are collected on a Pentium 4 linux machine, 2.4GHz, 1G memory.

Problem		LB3		LB4a	
#var	#cls	#branches	sec	#branches	sec
100	200	3.7e+03	0.035	1.63e+03	0.030
	250	6.22e+04	0.15	1.39e+04	0.082
	300	5.91e+05	1.08	1.07e+05	0.41
	350	2.28e+06	4.14	3.84e+05	1.5
	400	9.5e+06	17	1.09e+06	5.23
	450	1.77e+07	44	3.14e+06	14
	500	3.91e+07	81	7.32e+06	32
	550	7.32e+07	158	1.27e+07	55
	600	1.69e+08	373	3.38e+07	152
	650	3.15e+08	632	6.85e+07	337
	700	4.34e+08	1191	1.14e+08	512
200	400	1.61e+06	3.41	2.09e+05	0.88
	420	1.27e+07	28	1.3e+06	5.33
	440	5.79e+07	145	3.17e+06	15
	460	1.24e+08	330(3)	1.81e+07	90
	480	1.46e+08	405(10)	2.26e+07	117(1)
	500	2.69e+08	758(21)	4.31e+07	244(2)

clause has a weight. As future work, we will extend the lower bound function to other optimization problems such as the maximum cut and the maximum clique problems in graph theory.

References

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.

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.

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

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.

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.

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.

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.

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

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

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

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.

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

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

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

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.

H. Shen, H. Zhang, Improving Exact Algorithms for MAX-2-SAT, *Eighth International Symposium on Artificial Intelligence and Mathematics*, Fort Lauderdale, Florida, January 2004.

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.

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.

H. Zhang, H. Shen, F. Manyà: Exact algorithms for MAX-SAT. In *Proc. of International Workshop on First-order Theorem Proving* (FTP 2003).

X. Zhao, W. Zhang. An efficient algorithm for maximum boolean satisfiability based on unit propagation, linear programming, and dynamic weighting. Preprint, Department of Computer Science, Washington University. 2004.