Exploiting Inference Rules to Compute Lower Bounds for MAX-SAT Solving*

Han Lin[1] and **Kaile Su** [1,2] †
[1]Department of Computer Science, Sun Yat-sen University, Guangzhou 510275, P. R. China
[2]IIIS, Griffith University, Brisbane, Qld 4111, Australia
[1]underchange@yahoo.com, [2]k.su@griffith.edu.au

Abstract

In this paper we present a general logical framework for (weighted) MAX-SAT problem, and study properties of inference rules for branch and bound MAX-SAT solver. Several rules, which are not equivalent but Λ-equivalent, are proposed, and we show that Λ-equivalent rules are also sound. As an example, we show how to exploit inference rules to achieve a new lower bound function for a MAX-2-SAT solver. Our new function is admissible and consistently better than the well-known lower bound function. Based on the study of inference rules, we implement an efficient solver and the experimental results demonstrate that our solver outperforms the most efficient solver that has been implemented very recently [Heras and Larrosa, 2006], especially for large instances.

1 Introduction

In the past few years MAX-SAT solving techniques have drawn increasing attention [Gramm *et al.*, 2003; de Givry *et al.*, 2003; Alsinet *et al.*, 2003; Shen and Zhang, 2004; Li *et al.*, 2005; 2006; Larrosa and Heras, 2005; Xing and Zhang, 2005; Heras and Larrosa, 2006] in various fields of artificial intelligence, because many problems can be handled by transforming an original problem instance into a MAX-SAT instance and solving it with a MAX-SAT solver. These problems include *MAX-CUT*, *MAX-CLIQUE*, as well as problems in domains such as *routing* [Xu *et al.*, 2003], *bioinformatics* [Strickland *et al.*, 2005], *scheduling* [Vasquez and Hao, 2001], and *probabilistic reasoning* [Park, 2002].

MAX-SAT is an optimization version of the SAT problem. In the last decade significant progress has been made in the performance of SAT solvers. A modern SAT solver can handle instances containing up to a million variables within reasonable time [Kautz and Selman, 2005]. Almost every *complete* SAT solver is based on the *DPLL* algorithm [Davis *et al.*, 1962], and almost every DPLL-based solver employs transformation rules, such as unit propagation and resolution, to achieve high efficiency. Such rules improve SAT solvers in two main aspects. First, it may be used to simplify a formula. Second, at each internal node of the search tree, the solver may attempt to use such rules to prove unsatisfiability of the formula instantiated by a partial assignment, and if the formula is really unsatisfiable, the solver will not extend the partial assignment any more. Although such rules are of great use in SAT solving, it can not be applied directly to MAX-SAT solvers for several reasons. First, using these rules to simplify a formula may not preserve the optimal value. Second, since the goal of a MAX-SAT solver is to find maximum number of satisfied clauses, the proof of unsatisfiability is meaningless. This is why there is a big gap between SAT and MAX-SAT solving. Fortunately, transformation rules can be used to compute lower bounds during the MAX-SAT searching process. In this paper, we further study the properties of transformation rules for (weighted) MAX-SAT and propose several new rules to improve the lower bounds.

Due to the differences between the logic behind the MAX-SAT problem and that behind SAT, [Larrosa and Heras, 2005] developed a logical framework to extend the SAT solving techniques to MAX-SAT. However, because of the simplicity of this framework, we can not study the properties of the transformation rules intensively. In order to avoid this limitation, we extend the framework by introducing several new fundamental concepts.

This paper is organized as follows. In Section 2, we describe our logical framework for MAX-SAT, and intensively study the properties of inference rules. In Section 3, we propose several new inference rules to improve MAX-SAT solvers. In Section 4, as a case study, we show how to exploit inference rules to improve the lower bound function of MAX-2-SAT. Section 5 shows experimental results. Finally we conclude the paper in Section 6.

2 A Logical Framework for Weighted MAX-SAT

In this section, we propose a logical framework for weighted MAX-SAT, which is more general and suitable for study-

*This work has been supported by the Australian Research Council grant DP0452628, National Basic Research 973 Program of China under grant 2005CB321902, National Natural Science Foundation of China grants 60496327, 10410638 and 60473004, and Guangdong Provincial Natural Science Foundation grants 04205407 and 06023195.

†Corresponding author

ing the MAX-SAT solving techniques than the one proposed in [Larrosa and Heras, 2005; Heras and Larrosa, 2006].

2.1 Basic Notation and Definitions

For convenience we introduce some notation and definitions. In propositional logic, a variable x may take values 1 (for $true$) or 0 (for $false$). A *literal* is either a variable or its negation. The variable related to literal l is denoted by $var(l)$. The negation of a variable (or literal) x is denoted by $\bar{x}$. A *clause* $C = l_1 \vee l_2 \vee \ldots \vee l_m$ is a disjunction of literals in which no variable occurs more than once. A *weighted clause* is a pair (C, w) where C is a clause and w is the *weight* of this clause. Without loss of generality, we assume w is a natural number in this paper. A *CNF formula* is a conjunction of clauses. A *weighted CNF formula* is a set of weighted clause. A *bounded formula* is a pair (F, k) where F is a weighted CNF formula and k, a natural number, is the *upper bound* of this formula. For a formula F over a variable set V, an *assignment* for F is a mapping from V' to $\{0, 1\}$ and is denoted by a vector $\vec{X} \in \{0,1\}^n$, where n is the number of variables in V'. The assignment is complete if $V' \supseteq V$; otherwise it is partial. An assignment satisfies a clause if it satisfies at least one literal in the clause. An assignment satisfies a CNF formula if it satisfies every clause in the formula. The *instantiation* of a formula F by forcing literal l to be $true$, denoted by $F[l]$, produces a new formula as follows: all clauses containing literal l are eliminated, and $\bar{l}$ is deleted from each clause where it occurs. We use $\Box$ to denote *empty clause*, which can not be satisfied by any assignment.

Given a weighted CNF formula F and a complete assignment $\vec{X}$ for F, we set

$$COST(F, \vec{X}) = \sum_{(C,w)\in F \text{ and } \vec{X} \text{ does not satisfy } C} w$$

which indicates the cost of $\vec{X}$ in F.

Given a weighted formula F, weighted MAX-SAT problem is to find an assignment $\vec{X}$ such that $COST(F, \vec{X})$ is minimum. We denote such minimal value by $OPT(F)$. Since unweighted MAX-SAT problem is a particular case in which $w = 1$ for every weighted clause (C, w) in F, we only consider weighted MAX-SAT problem in this paper.

2.2 The Logical Framework

Definition 1 *A complete assignment $\vec{X}$ is called a model of a bounded formula (F, k), if $COST(F, \vec{X}) < k$. A bounded formula is unsatisfiable if it has not any model.*

Obviously, $(F, 0)$ is unsatisfiable for any weighted formula F and if $(\Box, k) \in F$, then (F, k) is also unsatisfiable.

Note that in the above definition, if $k = 1$, then a model should satisfy every clause in the formula, which is the same as the case in the context of SAT problem. So SAT solving can be considered as a special case in our logical framework.

According to Definition 1, we can naturally generalize *semantic entailment* relation and *equivalence* relation from classical positional logic to our logical framework.

Definition 2 *Given two bounded formulas (F_1, k_1) and (F_2, k_2),*

$$(F_1, k_1) \models (F_2, k_2)$$

holds[1] *if every model of (F_1, k_1) is also a model of (F_2, k_2). (F_1, k_1) and (F_2, k_2) are equivalent, denoted by $(F_1, k_1) \equiv (F_2, k_2)$, if $(F_1, k_1) \models (F_2, k_2)$ and $(F_2, k_2) \models (F_1, k_1)$.*

Definition 3 *Given two weighted CNF formulas F_1 and F_2,*

$$F_1 \models F_2$$

holds, if $(F_1, k) \models (F_2, k)$ for any k. Two weighted CNF formulas F_1 and F_2 are equivalent, denoted by $F_1 \equiv F_2$, if $F_1 \models F_2$ and $F_2 \models F_1$.

The following proposition and corollary reveal the relation between our definitions and the weighted MAX-SAT problem.

Lemma 1 *Let F_1 and F_2 be weighted CNF formulas, $F_1 \models F_2$, if and only if for every complete assignment $\vec{X}$ of F_1 and F_2, we have*

$$COST(F_1, \vec{X}) \geq COST(F_2, \vec{X})$$

Proposition 1 *Given two weighted CNF formulas F_1 and F_2, if $F_1 \models F_2$, then $OPT(F_1) \geq OPT(F_2)$.*

Corollary 1 *Given two weighted CNF formulas F_1 and F_2, if $F_1 \equiv F_2$, then $OPT(F_1) = OPT(F_2)$.*

Now we describe the procedure MAX-DPLL, which is slightly different from [Larrosa and Heras, 2005], for solving weighted MAX-SAT as follows.

function MAX-DPLL((F, k):bounded formula):interger;
1. F:=ApplyEquivalentRules((F, k));
2. F':=ApplyNonequivalentRules((F, k));
3. **if** $(\Box, k) \in F'$ **then return** k;
4. **if** $F = \emptyset$ **then return** 0;
5. **if** $F = \{(\Box, w)\}$ **then return** w;
6. l:=SelectLiteral(F);
7. v:=MAX-DPLL($(F[l], k)$);
8. v:=MAX-DPLL($(F[\bar{l}], v)$);
9. **return** v;

end function

Procedure 1: The procedure for solving weighted MAX-SAT.

The MAX-DPLL procedure is much like the DPLL, and its correctness is derived from the recursive equation

$$OPT(F) = min\{OPT(F[l]), OPT(F[\bar{l}])\}$$

The sub-procedures ApplyEquivalentRules and ApplyNonequivalentRules in lines 1 and 2 will be discussed later. For the moment, they can be considered as procedures just returning the input formula F. The initial value of variable k,

[1] Of course we can generalize this definition by replacing (F_1, k_1) with more than one bounded formula, but it is meaningless in the context of MAX-SAT solving.

upper bound of formula F, can be obtained by running a local search procedure, such as [Borchers and Furman, 1999].

Now let's pay attention to inference rules for weighted MAX-SAT. An inference rule is in this form:

$$P \Rightarrow Q$$

where P and Q are weighted CNF formulas. The procedure for applying an inference rule to a formula can be described as follows (WCF is short for weighted CNF formula):

function ApplyRule($P \Rightarrow Q$:inference rule, F:WCF):WCF;

1. **if** $P \subseteq F$ **then** $F := (F - P) \cup Q$;
2. **return** F;

end function

Procedure 2: The procedure for applying an inference rule to a weighted CNF formula.

Definition 4 *An inference rule $P \Rightarrow Q$ is sound if for any weighted CNF formula R, we have $OPT(P \cup R) = OPT(Q \cup R)$.*

An inference rule $P \Rightarrow Q$ is an equivalent rule if $P \equiv Q$. The following proposition shows that applying equivalent rules to a formula can preserve its optimal value.

Proposition 2 *All equivalent rules are sound.*

For a bounded formula (F, k), [Larrosa and Heras, 2005; Heras and Larrosa, 2006] incorporated the following inference rules into their weighted MAX-SAT solvers:

- BR1: $\{(A, k), (A \vee B, w)\} \Rightarrow \{(A, k)\}$
- BR2: $\{(A, w), (A, u)\} \Rightarrow \{(A, w \oplus u)\}$
- BR3: If $w \oplus u = k$ then $\{(A, w), (A \vee B, u)\} \Rightarrow \{(A, w), (A \vee B, k)\}$
- BR4: $\{(A, 0)\} \Rightarrow \{\ \}$
- RES: $\{(x \vee A, u), (\bar{x} \vee B, w)\} \Rightarrow \{(A \vee B, m), (x \vee A, u \ominus m), (\bar{x} \vee B, w \ominus m), (x \vee A \vee \bar{B}, m), (\bar{x} \vee \bar{A} \vee B, m)\}$

where x is any literal, A and B are arbitrary disjunction of literals, $m = min\{u, w\}$ in the RES rule, and operators $\oplus$ and $\ominus$ are defined as

$$a \oplus b = min\{a + b, k\}$$

$$a \ominus b = \begin{cases} a - b & a \neq k \\ k & a = k \end{cases}$$

Note that [Larrosa and Heras, 2005; Heras and Larrosa, 2006] incorporated some special cases of RES rather than the general RES rule.

By exploiting the above rules, [Heras and Larrosa, 2006] implemented a weighted MAX-SAT solver that outperforms other state-of-art solvers. However, we can observe that all the above rules are equivalent rules. It is no doubt that equivalent rules are sound, that is, preserve the model and the optimal value, so it is safe to exploit equivalent rules to produce a new formula. But if we only consider such rules, very few rules can be exploited.

An alternative way is to choose rules in this form $P \Rightarrow Q$ such that $OPT(P) = OPT(Q)$. Unfortunately, applying such kind of rules is not safe, because $OPT(P) = OPT(Q)$ does not ensure the validity of $OPT(P \cup R) = OPT(Q \cup R)$ where P, Q, and R are weighted CNF formulas. A counter example is given as follows.

Example 1 $OPT(\{(x, 1)\}) = OPT(\{(x \vee y, 1)\}) = 0$, *but* $OPT(\{(x, 1), (\bar{x}, 1)\}) = 1$ *while* $OPT(\{(x \vee y, 1), (\bar{x}, 1)\}) = 0$.

Thus, if we attempt to incorporate more inference rules to weighted MAX-SAT solvers, we should find rules that are not necessarily equivalent but can preserve the optimal value of the original formula. For this purpose, we introduce the following definition.

Definition 5 *Let Λ be a non-empty set of variables, two weighted CNF formulas F_1 and F_2 are said to be Λ-equivalent, denoted by $F_1 \approx_\Lambda F_2$, if for every weighted CNF formula P over Λ, $F_1 \models P$ if and only if $F_2 \models P$. An inference rule $P \Rightarrow Q$ is a Λ-equivalent rule if $P \approx_\Lambda Q$.*

For Λ-equivalent rules, we have the following proposition and theorem.

Proposition 3 *All equivalent rules are Λ-equivalent for any Λ.*

Theorem 1 *If $P \Rightarrow Q$ is a Λ-equivalent rule, and R is a weighted CNF formula over Λ, then $OPT(P \cup R) = OPT(Q \cup R)$.*

Theorem 1 gives the condition under which we can apply Λ-equivalent rules to the formula. And Proposition 3 shows that Λ-equivalent is a weaker notation than equivalent, that is, the set of equivalent rules is the subset of the set of Λ-equivalent rules. So, theoretically, considering Λ-equivalent rules will provide more choices than considering equivalent rules only. Thus, it is promising to employ Λ-equivalent rules in weighted MAX-SAT solvers. In Section 3, we will propose several Λ-equivalent rules.

Now let us pay attention to nonequivalent rules which are neither equivalent nor Λ-equivalent. We note that every inference rule $P \Rightarrow Q$ considered in this paper should satisfy $P \models Q$. So although applying a nonequivalent rule to a formula may not preserve its optimal value, we may exploit it to compute a lower bound. From Proposition 1, we can conclude that lower bounds that are computed in this way are admissible, namely the lower bound is always less than or equal to the optimal value. In fact, sometimes it is easier to use nonequivalent rules to obtain conflict clauses. For example, [Shen and Zhang, 2004; Li *et al.*, 2005] apply inference rules like $\{(\bar{x}, 1), (x \vee y, 1), (\bar{y}, 1)\} \Rightarrow \{(\square, 1)\}$. However, since such rules are not equivalent, they can not be used to achieve a tight lower bound usually. In Section 4, we will discuss this problem.

3 Some Λ-Equivalent Inference Rules

In this section we present two Λ-Equivalent Rules and show in which situation they can be applied.

The first Λ-equivalent rule we propose is

1-RES: $\{(x \vee A, u), (\bar{x} \vee B, w)\} \Rightarrow \{(A \vee B, min\{u, w\})\}$[2]

where x is a literal, and A and B are arbitrary clauses (may be empty).

Theorem 2 *Let Λ be a non-empty set of variables. If $var(x) \notin \Lambda$, then 1-RES is a Λ-equivalent rule.*

Proof Without loss of generality, we may consider A and B as variables. It is easy to verify that $\{(x \vee A, u), (\bar{x} \vee B, w)\} \models \{(A \vee B, min\{u, w\})\}$, so for any weighted formula P, whenever $\{(A \vee B, min\{u, w\})\} \models P$, we also have $\{(x \vee A, u), (\bar{x} \vee B, w)\} \models P$. Now suppose $\{(x \vee A, u), (\bar{x} \vee B, w)\} \models P$, where P is a weighted formula not containing x. There are four cases.

- Case 1: $A = 0$ and $B = 0$, if $x = 0$, the cost is u, otherwise the cost is v, from Lemma 1, the cost of P in this case should be less than $min\{u, v\}$, which is not greater than the formula $\{(A \vee B, min\{u, w\})\}$ in this case.
- Case 2: $A = 0$ and $B = 1$, if $x = 0$ the cost is u, otherwise is 0, from Lemma 1, the cost of P should be 0, which is also not greater than $\{(A \vee B, min\{u, w\})\}$.
- Case 3: $A = 1$ and $B = 0$, analysis is similar with above cases.
- Case 4: $A = 1$ and $B = 1$, analysis is similar with above cases.

To sum up, we can conclude that no matter in which case, the cost of P is less than or equal to $\{(A \vee B, min\{u, w\})\}$, so we also have $\{(A \vee B, min\{u, w\})\} \models P$. □

According to Theorem 1 and Theorem 2, if a formula F contains $(x \vee A, u)$ and $(\bar{x} \vee B, w)$, and $var(x)$ does not occur in other clauses in F, we can apply 1-RES to F.

Example 2 *Let $F = \{(x \vee y, 2), (\bar{x} \vee z, 3)\}$. If we apply the RES rule, we will get $\{(y \vee z, 2), (\bar{x} \vee z, 1), (x \vee y \vee \bar{z}, 2), (\bar{x} \vee \bar{y} \vee z, 2)\}$, which is too complicated. But if we apply the 1-RES rule, we will get a much simpler formula $\{(y \vee z, 2)\}$.*

The second Λ-equivalent rule we propose is

2-RES: $\{(x, u), (x \vee y, v), (\bar{x} \vee A, w)\} \Rightarrow$
$\{(y \vee A, min\{u \oplus v, w\}), (\bar{y} \vee A, min\{u, w\})\}$

where x and y are literals, and A is the conjunction of arbitrary literals (may be empty).

Note that we can apply the RES rule to the right side of the 2-RES rule to get simpler formula further.

Theorem 3 *Let Λ be a non-empty set of variables. If $var(x) \notin \Lambda$, then 2-RES is a Λ-equivalent rule.*

Proof The proof is similar with Theorem 1, so we omit it here. □

According to Theorem 1 and Theorem 3, if a formula F contains $\{(x, u), (x \vee y, v)$, and $(\bar{x} \vee A, w)$, and $var(x)$ does not occur in other clauses in F, we can apply 2-RES to F.

[2]This rule can be considered as an extension of the elimination rule [Bansal and Raman, 1999] to weighted MAX-SAT.

Example 3 *Let $F = \{(x, 3), (x \vee y, 2), (\bar{x} \vee z, 1), (\bar{z}, 1)\}$. The 2-RES rule transforms F into $\{(y \vee z, 1), (\bar{y} \vee z, 1), (\bar{z}, 1)\}$. And the RES rule transform the resulting formula into $\{(z, 1), (\bar{z}, 1)\}$, which can be simplified to $\{\Box, 1\}$.*

4 A New Lower Bound Function for MAX-2-SAT

MAX-2-SAT is a special case of weighted MAX-SAT, in which each clause has weight 1 and at most two literals. [Shen and Zhang, 2004] studied several lower bound functions for MAX-2-SAT, and proposed a lower bound function lower_bound4a, which is proved to be admissible and consistently better than others. The procedure for calculating this function is as follows (LB1 is the number of conflicting (or empty) clauses by the current partial assignment).

function lower_bound4a(i:integer):integer;
1. LB4:=LB1;
2. **while** ($i \leq n$) do
3. LB4:=LB4+min($\mu(\bar{i})$,$\mu(i)$);
4. $t := |\mu(\bar{i}) - \mu(i)|$;
5. **if** $\mu(\bar{i}) > \mu(i)$ **then** $Y := B(i)$ **else** $Y := B(\bar{i})$;
6. $S := \phi$;
7. **for** $j \in Y$ **if** $(t > 0)$ **do**
8. **if** $(\mu(j) > \mu(\bar{j}))$ **then** $S := S \cup \{j\}$ **else**
9. $t := t - 1$; $\mu(j) := \mu(j) + 1$;
10. **end if**
11. **end for**
12. **for** $j \in S$ **if** $(t > 0)$ **do**
13. $t := t - 1$; $\mu(j) := \mu(j) + 1$;
14. **end for**
15. $i := i + 1$;
16. **end while**
17. **return** LB4;

end function

Procedure 3: The procedure for computing LB4a.

The main idea of this procedure is to exploit the inference rule $\{(\bar{x}, 1), (x \vee y, 1), (\bar{y}, 1)\} \Rightarrow \{(\Box, 1)\}$ to obtain a lower bound (Due to limited space, for more details, please refer to the original paper). However, this rule is not equivalent, so the tight may be not tight enough. According to the RES rule, $\{(\bar{x}, 1), (x \vee y, 1), (\bar{y}, 1)\} \equiv \{(\bar{x} \vee \bar{y}), (\Box, 1)\}$, so we exploit the rule $\{(\bar{x}, 1), (x \vee y, 1), (\bar{y}, 1)\} \Rightarrow \{(\bar{x} \vee \bar{y}, 1), (\Box, 1)\}$ to get a better lower bound. Procedure 4 illustrates our algorithm. Line 12 and 17 change the formula by deleting $x \vee y$ and adding the clause $\bar{x} \vee \bar{y}$. k can be considered as a constant number, in our code, we choose $k = 3$. So the time complexity of lower_bound5 is the same as lower_bound4a. In order to compare with lower_bound4a, we write the pseudo code in the same style as procedure 3. Note that the real code is far from this pseudo code.

function lower_bound5(i, k:integer):integer;

1. LB5:=LB1;
2. $i_1 := i$;
3. **while** $(k > 0)$ **do**
4. **while** $(i \leq n)$ **do**
5. LB5:=LB5+min($\mu(\bar{i})$,$\mu(i)$);
6. $t := |\mu(\bar{i}) - \mu(i)|$;
7. **if** $\mu(\bar{i}) > \mu(i)$ **then** $Y := B(i)$ **else** $Y := B(\bar{i})$;
8. $S := \phi$;
9. **for** $j \in Y$ **if** $(t > 0)$ **do**
10. **if** $(\mu(j) > \mu(\bar{j}))$ **then** $S := S \cup \{j\}$ **else**
11. $t := t - 1$; $\mu(j) := \mu(j) + 1$;
12. $B(i) := B(i) - \{j\}$; $B(\bar{i}) := B(\bar{i}) \cup \{\bar{j}\}$;
13. **end if**
14. **end for**
15. **for** $j \in S$ **if** $(t > 0)$ **do**
16. $t := t - 1$; $\mu(j) := \mu(j) + 1$;
17. $B(i) := B(i) - \{j\}$; $B(\bar{i}) := B(\bar{i}) \cup \{\bar{j}\}$;
18. **end for**
19. **if** $\mu(\bar{i}) > \mu(i)$ **then** $\mu(\bar{i}) := t$ **else** $\mu(i) := t$;
20. $i := i + 1$;
21. **end while**
22. $i := i_1$; $k := k - 1$;
23. **end while**
24. **return** LB5;

end function

Procedure 4: The procedure for computing LB5.

The following theorem shows that LB5 is admissible and consistently better than LB4a.

Theorem 4 $LB4a \leq LB5 \leq OPT(F)$.

5 Experimental Results

The experimental results are presented in this section. Our solver, called Λ-SAT, is compared with the solver *Toolbar* (http://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/SoftCSP), which implemented by [Larrosa and Heras, 2005; Heras and Larrosa, 2006]. Other solvers are not freely available and [Heras and Larrosa, 2006] showed that Toolbar outperforms others. So we only compare our solver with Toolbar.

Our solver implements the variable selection strategy known as Jeroslow-Wang rule [Jeroslow and Wang, 1990], which is widely employed in state-of-art SAT solvers. For inference rules, we choose the same as [Heras and Larrosa, 2006], additionally we also employ the new rules proposed in this paper. Our tool is implemented in C.

We run two solvers with random MAX-2-SAT (experimental results are illustrated in Table 1, #var stands for the number of variables and #cls stands for the number of clauses, #node stands for the mean number of nodes of the whole search tree, time stands for the mean running time over 30 instances) and MAX-3-SAT instances (experimental results are illustrated in Table 2). The instances are generated by the well-know generator *Cnfgen*. Executions are run on a 2.4 Ghz Pentium 4 computer with Linux.

Problem		Toolbar		Λ-SAT	
#var	#cls	#node	time(s)	#node	time(s)
50	100	383	0.00	6	0.03
	150	541	0.01	37	0.04
	200	748	0.02	150	0.05
	250	1167	0.04	289	0.06
	300	1803	0.07	685	0.09
	350	2771	0.13	997	0.11
	400	4213	0.22	1887	0.17
	450	5214	0.29	2815	0.24
	500	7264	0.42	3674	0.30
	550	8761	0.54	4530	0.38
	600	13140	0.82	7365	0.58
	650	18160	1.14	10523	0.81
	700	24973	1.61	14378	1.09
100	200	1877	0.09	155	0.09
	250	5135	0.42	967	0.18
	300	10405	1.05	3217	0.46
	350	28698	3.39	9759	1.31
	400	60210	7.48	26931	3.74
	450	115093	15.10	54202	7.39
	500	244641	33.26	116368	16.01
150	300	13608	2.28	3961	0.94
	350	62872	13.14	20706	4.57
	400	216731	47.83	89632	19.99
	450	575106	128.77	202468	47.44
	500	1358349	320.63	527558	127.43

Table 1: The experimental results on random MAX-2-SAT.

Problem		Toolbar		Λ-SAT	
#var	#cls	#node	time(s)	#node	time(s)
50	150	209	0.00	0	0.00
	200	657	0.01	51	0.00
	250	4087	0.12	1320	0.10
	300	9889	0.35	5051	0.25
	350	26836	1.09	13329	0.63
	400	51133	2.28	28257	1.37
	450	77010	3.74	41410	2.13
	500	141339	7.18	72781	4.03
	550	208578	11.32	129703	7.34
	600	383773	22.31	188048	11.07
	650	434565	25.71	276834	16.91
	700	673423	40.80	368897	23.42
80	300	2283	0.08	0	0.00
	350	21204	1.09	2566	0.18
	400	115956	7.07	29980	1.90
	450	552097	37.65	218965	15.18
	500	1412277	104.62	604266	45.93
120	450	39008	3.23	0	0.00
	500	386564	39.04	48689	4.85

Table 2: The experimental results on random MAX-3-SAT.

The results demonstrate that for small instances, the performance of the our solver is nearly the same as Toolbar, while

for large instances the former outperforms the latter, and especially for some larger instances, our solver can be orders of magnitude faster than Toolbar.

6 Conclusions and Future Work

The main contributions of this paper are in two aspects. First, we develop a general logical framework that is suitable for studying weighted MAX-SAT problem. This is because many concepts in SAT are easy to be extended in our framework. This may provide more chances to extend techniques taken by SAT solvers to MAX-SAT solvers. Second, we propose several new inference rules, which can be exploited to get lower bound of better quality. The experimental results support our conclusion.

For future work, we will further study how to extend SAT techniques to MAX-SAT solver, such as *clause learning* and *restarts*. Moreover, for unweighted MAX-SAT, we notice that several lower bound computation techniques [Li *et al.*, 2006], which focus on searching for disjoint inconsistent subformulas, have been proposed. Based on our logical framework, we will attempt to generalize such techniques to compute lower bounds for weighted MAX-SAT efficiently. Recently, a simple model have been presented for generating random weighted MAX-2-SAT instances (http://www.nlsde.buaa.edu.cn/ kexu/benchmarks/max-sat-benchmarks.htm). Such instances are transformed from forced satisfiable SAT instances of Model RB [Xu and Li, 2006; Xu *et al.*, 2005] and are, unfortunately, too hard for both toolbar and our solver. We will try to improve our solver to handle these benchmarks within reasonable time.

References

[Alsinet *et al.*, 2003] T. Alsinet, F. Manyà, and J. Planes. Improved branch and bound algorithms for max-sat. In *Proceedings of the 6th International Conference on the Theory and Applications of Satisfiability Testing*, 2003.

[Bansal and Raman, 1999] N. Bansal and V. Raman. Upper bounds for maxsat: further improved. In *Proceedings of ISAAC 1999*, pages 247–258, 1999.

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

[Davis *et al.*, 1962] M. Davis, G. Logemann, and G. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5:394–397, 1962.

[de Givry *et al.*, 2003] S. de Givry, J. Larrosa, P. Meseguer, and T. Schiex. Solving max-sat as weighted csp. In *Proc. of the 9th CP*, pages 363–376, Kinsale, Ireland, 2003. Springer LNCS 2833.

[Gramm *et al.*, 2003] J. Gramm, E.A. Hirsch, R. Niedermeier, and P. Rossmanith. Worst-case upper bounds for max-2-sat with application to max-cut. *Discrete Applied Mathematics*, 130(2):139–155, 2003.

[Heras and Larrosa, 2006] Federico Heras and Javier Larrosa. New inference rules for efficient max-sat solving. In *Proceedings of the 21st AAAI*, Boston, Massachusetts, July 2006.

[Jeroslow and Wang, 1990] R.G. Jeroslow and J. Wang. Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, 1:167–187, 1990.

[Kautz and Selman, 2005] Henry Kautz and Bart Selman. The state of sat. *Discrete and Applied Math*, 2005.

[Larrosa and Heras, 2005] Javier Larrosa and Federico Heras. Resolution in max-sat and its relation to local consistency in weighted csps. In *Proceedings of the 19th IJCAI*, Edinburgh, Scotland, 2005.

[Li *et al.*, 2005] Chu Min Li, Felip Manyà, and Jordi Planes. Exploiting unit propagation to compute lower bounds in branch and bound max-sat solvers. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming*, pages 403–414, Sitges, Spain, 2005. Springer LNCS 3709.

[Li *et al.*, 2006] Chu-Min Li, Felip Manyà, and Jordi Planes. Detecting disjoint inconsistent subformulas for computing lower bounds for max-sat. In *Proceedings of Twenty-First National Conference on Artificial Intelligence (AAAI 2006)*, July 2006.

[Park, 2002] J. D. Park. Using weighted max-sat engines to solve mpe. In *Proceedings of 18th AAAI*, pages 682–687, 2002.

[Shen and Zhang, 2004] H. Shen and H. Zhang. Study of lower bound functions for max-2-sat. In *Proceedings of AAAI-2004*, pages 185–190, 2004.

[Strickland *et al.*, 2005] Dawn M. Strickland, Earl Barnes, and Joel S. Sokol. Optimal protein structure alignment using maximum cliques. *Operations Research*, 53(3):389–402, May–June 2005.

[Vasquez and Hao, 2001] M. Vasquez and J. Hao. A logic-constrained knapsack formulation and a tabu algorithm for the daily photograph scheduling of an earth observation satellite. *Journal of Computational Optimization and Applications*, 20(2), 2001.

[Xing and Zhang, 2005] Z. Xing and W. Zhang. Maxsolver: An efficient exact algorithm for (weighted) maximum satisfiability. *Artificial Intelligence*, 164(1-2):47–80, 2005.

[Xu and Li, 2006] Ke Xu and Wei Li. Many hard examples in exact phase transitions. *Theoretical Computer Science*, 355:291–302, 2006.

[Xu *et al.*, 2003] H. Xu, R. Rutenbar, and K. Sakallah. sub-sat: a formulation for relaxed boolean satisfiability with applications in routing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(6):814–820, June 2003.

[Xu *et al.*, 2005] Ke Xu, Frederic Boussemart, Fred Hemery, and Christophe Lecoutre. A simple model to generate hard satisfiable instances. In *Proceedings of 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)*, pages 337–342, Edinburgh, Scotland, 2005.