New Inference Rules for Efficient Max-SAT Solving *

Federico Heras and Javier Larrosa
Universitat Politecnica de Catalunya
Jordi Girona 1,3
Barcelona, Spain
fheras@lsi.upc.edu, larrosa@lsi.upc.edu

Abstract

In this paper we augment the Max-SAT solver of (Larrosa & Heras 2005) with three new inference rules. The three of them are special cases of Max-SAT resolution with which better lower bounds and more value pruning is achieved. Our experimental results on several domains show that the resulting algorithm can be orders of magnitude faster than state-of-the-art Max-SAT solvers and the best Weighted CSP solver.

Introduction

Max-SAT is an optimization version of the SAT problem and it is known that many important problems can be naturally expressed as Max-SAT instances. They include academic problems such *Max-CUT* or *Max-CLIQUE*, as well as real problems in domains such as *routing* (H. Xu & Sakallah 2002), *bioinformatics* (D.M. Strickland & Sokol 2005), *scheduling* (Vasquez & Hao 2001) or *probabilistic reasoning* (Park 2002). In recent years, there has been a considerable effort in finding efficient solving techniques (de Givry *et al.* 2003; Shen & Zhang 2004; Xing & Zhang 2005; Chu Min Li & Planes 2005). In all these works the core algorithm is a simple depth-first branch-and-bound and their contributions are good quality lower bounds. Most authors describe lower bounds in a procedural way. The main drawback of such approach is that sometimes it is hard to see the *logic* that is behind. A notable exception is (Larrosa & Heras 2005), where a logical framework for Max-SAT is developed and SAT solving techniques are extended to Max-SAT. In that context, a fairly efficient solver that combines search and inference is presented. Such solver can be seen as Max-DPLL (namely, the extension of DPLL to Max-SAT) enhanced with neighborhood resolution at each visited node. Some of the ideas of this solver were borrowed from weighted constraint satisfaction (Cooper & Schiex 2004; Larrosa & Schiex 2004).

In this paper we improve the previous solver by adding new forms of inference beyond neighborhood resolution. In particular, we introduce three new inference rules, all of them expressable as particular cases of resolution. The first rule, that we call *directed resolution*, assumes an order among variables and is used to derive clauses involving small variables. The second and third rules are particular instances of *hyper-resolution* in which a small sequence of resolution steps derive a new clause smaller than any parent clause. We provide experimental results in different domains. The experiments indicate that our algorithm is orders of magnitud faster than any competitor. This is especially true as the ratio between the number of clauses and the number of variables increases. Note that these are the hardest instances for Max-SAT.

Preliminaries

In the sequel $X = \{x, y, z, ...\}$ is a set of boolean variables taking values over the set $\{\mathbf{t}, \mathbf{f}\}$, which stands for *true* and *false*, respectively. A *literal* is either a variable (e.g. x) or its negation (e.g. $\bar{x}$). We will use $l, h, q, ...$ to denote literals and $var(l)$ to denote the variable related to l (namely, $var(x) = var(\bar{x}) = x$). If variable x is instantiated to $\mathbf{t}$, literal x is satisfied and literal $\bar{x}$ is falsified. Similarly, if variable x is instantiated to $\mathbf{f}$, literal $\bar{x}$ is satisfied and literal x is falsified. An *assignment* is an instantiation of a subset of X. The assignment is *complete* if it instantiates all the variables (otherwise it is partial). A *clause* $C = l_1 \vee l_2 \vee \ldots \vee l_k$ is a disjunction of literals such that $\forall_{1 \leq i,j \leq k,\ i \neq j}\ var(l_i) \neq var(l_j)$. The size of a clause, noted $|C|$, is the number of literals that it has. $var(C)$ is the set of variables that appear in C (namely, $var(C) = \{var(l) | l \in C\}$). An assignment satisfies a clause iff it satisfies one or more of its literals. A formula in *conjunctive normal form* (CNF) is a conjuction of different clauses, normally expressed as a set. A satisfying complete assignment is called a *model*. Given a CNF formula, the SAT problem consists in determining whether there is any model for it or not.

The *empty clause*, noted $\square$, cannot be satisfied. Consequently, when a formula contains $\square$ it does not have any model and we say that it contains an *explicit contradiction*. Sometimes it is convenient to think of clause C as its equivalent $C \vee \square$.

The *instantiation* of a formula $\mathcal{F}$ by forcing the satisfaction of literal l, noted $\mathcal{F}[l]$, produces a new formula generated as follows: all clauses containing l are eliminated, and $\bar{l}$ is removed from all clauses where it appears.

A *weighted clause* is a pair (C, w) such that C is a classi-

*This research is supported by the MEC through project TIN2005-09312-C03-02.
Copyright © 2006, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

cal clause and w is the cost of its falsification. In this paper we assume costs being natural numbers. A *weighted* formula in conjunctive normal form (CNF) is a set of weighted clauses. The cost of an assignment is the sum of weights of all the clauses that it falsifies.

The negation of a weighted clause (C, w), noted $(\bar{C}, w)$, means that the *satisfaction* of C has cost w, while its negation is cost-free. Note that $(\bar{C}, w)$ is not CNF when $|C| > 1$. In classical SAT the *De Morgan* rule can be used to recover the CNF syntax, but such rule is not sound (namely, the equivalence among formulas is not preserved) with weighted clauses. Consider the expression $(A \vee \overline{l \vee C}, w)$ where $|A| >= 0$ and $|C| >= 1$. It can be transformed into clausal form with the following recursion:

$$(A \vee \overline{l \vee C}, w) \equiv \{(A \vee \bar{C}, w), (A \vee \bar{l} \vee C, w)\}$$

Example 1 *Find below the truth table of* $(x \vee \overline{y \vee z}, 1)$*, the truth table of* $\{(x \vee \bar{y}, 1), (x \vee \bar{z}, 1)\}$ *(wrong transformation using De Morgan) and the truth table of* $\{(x \vee \bar{z}, 1), (x \vee \bar{y} \vee z, 1)\}$ *(correct transformation using the previous recursive definition).*

$x\ y\ z$	$(x \vee \overline{y \vee z}, 1)$	$\{(x \vee \bar{y}, 1)$ $(x \vee \bar{z}, 1)\}$	$\{(x \vee \bar{z}, 1)$ $(x \vee \bar{y} \vee z, 1)\}$
f f f	0	$0 \oplus 0 = 0$	$0 \oplus 0 = 0$
f f t	1	$1 \oplus 0 = 1$	$1 \oplus 0 = 1$
f t f	1	$0 \oplus 1 = 1$	$0 \oplus 1 = 1$
f t t	1	$1 \oplus 1 = 2$	$1 \oplus 0 = 1$
t f f	0	$0 \oplus 0 = 0$	$0 \oplus 0 = 0$
t f t	0	$0 \oplus 0 = 0$	$0 \oplus 0 = 0$
t t f	0	$0 \oplus 0 = 0$	$0 \oplus 0 = 0$
t t t	0	$0 \oplus 0 = 0$	$0 \oplus 0 = 0$

Following (Larrosa & Heras 2005), we assume without loss of generality the existence of a known upper bound $\top$ of the optimal solution ($\top$ is a strictly positive natural number). A *model* is a complete assignment with cost less than $\top$. A Max-SAT instance is a pair $(\mathcal{F}, \top)$ and the task of interest is to find a model of minimum cost, if there is any.

Observe that any weight $w \geq \top$ indicates that the associated clause *must be necessarily satisfied*. Thus, we can replace w by $\top$ without changing the problem. Consequently, we can assume all costs in the interval $[0..\top]$. The sum of costs is defined as,

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

in order to keep the result within the interval $[0..\top]$. Let u and w two costs such that $u \geq w$. Their subtraction is defined as,

$$u \ominus w = \begin{cases} u - w & : \quad u \neq \top \\ \top & : \quad u = \top \end{cases}$$

Essentially, $\ominus$ behaves like the usual subtraction except in that $\top$ is an absorbing element. A clause with weight $\top$ is called *mandatory* (or *hard*).

Note that in Max-SAT, truth tables are tables with a cost associated to each truth assignments. Note as well that clauses with cost 0 do not have any effect in the formula and can be omitted.

A weighted CNF formula may contain $(\square, w)$. Since $\square$ cannot be satisfied, w is added to the cost of any assignment. Therefore, w is an explicit *lower bound* of the optimal model. When the lower bound and the upper bound have the same value (i.e., $(\square, \top) \in \mathcal{F}$) the formula does not have any model and we call this situation an *explicit contradiction*.

Max-DPLL

In (Larrosa & Heras 2005) the basic solving techniques from SAT have been extended to Max-SAT. Figure 1 (top) presents the pseudo-code of a simplification algorithm. It applies *aggregation* (line 4), *subsumption* (line 6), *hardening* (line 8) or *unit clause reduction* (line 9) until detecting a contradiction or quiescence. Function `Inference` (line 10) will be developed later. Assume for the moment that it returns unchanged the input formula along with the boolean value *true*. Figure 1 (bottom) presents the extension of the DPLL algorithm (Davis, Logemann, & Loveland 1962). Let $(\mathcal{F}, \top)$ be a Max-SAT instance. Max-DPLL($\mathcal{F}, \top$) returns the cost of the optimal model if there is any, else it returns $\top$. First, the input formula is simplified (line 1). If the simplified formula contains a contradiction, the algorithm returns $\top$ and backtracks (line 2). Else, if it does not contain any variable the trivial cost of the optimal model is returned (lines 3 and 4). Otherwise, a literal l is selected (line 5). The formula is instantiated with l and $\bar{l}$ and Max-DPLL is recursively called with each case (lines 6 and 7). Observe that the first recursive call is made with the $\top$ inherited from its parent, but the second call uses the output of the first call. This implements the typical upper bound updating of branch and bound. Finally, the best value of the two recursive calls is returned (line 8).

Max-RES

As shown by (Larrosa & Heras 2005), the notion of resolution can be extended to weighted formulas as follows,

$$\{(x \vee A, u), (\bar{x} \vee B, w)\} \equiv \left\{ \begin{array}{l} (A \vee B, m), \\ (x \vee A, u \ominus m), \\ (\bar{x} \vee B, w \ominus m), \\ (x \vee A \vee \bar{B}, m), \\ (\bar{x} \vee \bar{A} \vee B, m) \end{array} \right\}$$

where A and B are arbitrary disjunctions of literals and $m = \min\{u, w\}$.

$(x \vee A, u)$ and $(\bar{x} \vee B, w)$ are called the *prior clashing clauses*. $(A \vee B, m)$ is called the *resolvent*. $(x \vee A, u \ominus m)$ and $(\bar{x} \vee B, w \ominus m)$ are called the *posterior clashing clauses*. $(x \vee A \vee \bar{B}, m)$ and $(\bar{x} \vee \bar{A} \vee B, m)$ are called the *compensation clauses*. The effect of Max-RES, as in classical resolution, is to infer (namely, make explicit) a connection between A and B. However, there is an important difference between classical resolution and Max-RES. While the former yields the *addition* of a new clause, Max-RES is a transformation rule. Namely, it requires the *replacement* of the left-hand clauses by the right-hand clauses. The reason is that some cost of the prior clashing clauses must be substracted in order to *compensate* the new inferred information. Consequently, Max-RES is better understood as a *movement* of knowledge in the formula.

function `Simplify`($\mathcal{F}, \top$)
$stop := false$
do
  if $\{(C,u),(C,w)\} \subseteq \mathcal{F}$ **then**
    replace $\{(C,u),(C,w)\}$ **by** $\{(C, u \oplus w)\}$
  elseif $\{(A,\top),(A \vee B,u)\} \subseteq \mathcal{F}$ **then**
    remove $\{(A \vee B, u)\}$
  elseif $\{(\square,u),(C,w)\} \subseteq \mathcal{F} \wedge u \oplus w = \top$ **then**
    replace $\{(C,w)\}$ **by** $\{(C,\top)\}$
  elseif $(l,\top) \in \mathcal{F}$ **then apply** $\mathcal{F}[l]$
  else $(\mathcal{F}, stop) :=$ `Inference`($\mathcal{F}, \top$)
until $(((\square,\top) \in \mathcal{F}) \vee stop)$
return $(\mathcal{F})$
endfunction

function `Max-DPLL`($\mathcal{F}, \top$):nat
$\mathcal{F} :=$ `Simplify`($\mathcal{F}, \top$)
if $(\square,\top) \in \mathcal{F}$ **then return** $\top$
if $\mathcal{F} = \emptyset$ **then return** 0
if $\mathcal{F} = \{(\square,w)\}$ **then return** w
$l :=$`SelectLiteral`($\mathcal{F}$)
$v :=$`Max-DPLL`($\mathcal{F}[\,l\,], \top$)
$v :=$`Max-DPLL`($\mathcal{F}[\,\bar{l}\,], v$)
return v
endfunction

Figure 1: Max-DPLL. $(\mathcal{F}, \top)$ is a Max-SAT instance. If it has models, Max-DPLL returns the cost of the optimal one. Else Max-DPLL returns $\top$. The `Simplify`$(\mathcal{F}, \top)$ function converts the input formula into a simpler one.

Observe that in the $u = w < \top$ case, the posterior clashing clauses will have weight 0, hence they can be removed. In the $u \neq w$ case, one of the two posterior clashing clauses will have weight 0, hence it can be removed. In the $u = w = \top$ case, clashing clauses are not modified and compensation clauses are subsumed. Hence, the rule matches with classical resolution.

Max-DPLL with Inference

Similarly to what happens to DPLL, plain Max-DPLL does not seem to be very effective in practice. However, its performance can be improved dramatically if it is armed with more sophisticated solving techniques. One possibility is to let Max-DPLL perform a limited form of resolution at every search. Such process will presumably facilitate the posterior task of Max-DPLL. An example suggested in (Larrosa & Heras 2005) is Neighborhod Resolution (NRES), which is Max-RES restricted to the $A = B$ case,

$$\{(x \vee A, u), (\bar{x} \vee A, w)\} \equiv$$
$$\equiv \{(A, m), (x \vee A, u \ominus m), (\bar{x} \vee A, w \ominus m)\}$$

with $m = \min\{u, w\}$.

Example 2 *Consider the formula* $\{(\bar{y} \vee z, 1), (\bar{y} \vee \bar{z}, 1), (y, 1), (\bar{x}, 2)\}$ *with* $\top = 3$. *Max-DPLL is unable to simplify the formula. Nevertheless, NRES can be applied to the first and second clauses producing* $\{(\bar{y}, 1), (y, 1), (\bar{x}, 2)\}$. *Now NRES can be applied between the first and the second clauses producing* $\{(\square, 1), (\bar{x}, 2)\}$. *The second clause can be made mandatory* $\{(\square, 1), (\bar{x}, \top)\}$ *and subsequently unit clause reduction produces* $\{(\square, 1)\}$. *As can be observed Max-DPLL enhanced with NRES does not need to do any search whatsoever.*

function `Inference`($\mathcal{F}, \top$)
1. $stop := false$
2. **if** $\{(x \vee A, u), (\bar{x} \vee A, w)\} \subseteq \mathcal{F}$ **then**
3. **apply** *NRES*
4. **elseif** $var(l') < var(l)$ **and**
5. $\{(l \vee A, u), (\bar{l} \vee A \vee l', w)\} \subseteq \mathcal{F}$
6. **then apply** *DRES*
7. **elseif** $\{(x \vee y \vee A, u), (\bar{x} \vee z \vee A, v), (\bar{y} \vee z \vee A, w)\}$
8. $\subseteq \mathcal{F}$ **then apply** 2-*RES*
9. **elseif** $\{(x \vee y \vee A, u), (\bar{x} \vee z \vee A, v), (\bar{y} \vee A, w_1),$
10. $(\bar{z} \vee A, w_2)\} \subseteq \mathcal{F}$ **then apply** 3-*RES*
11. **else** $stop := true$
12. **return** $(\mathcal{F}, stop)$
endfunction

Figure 2: Inference.

The algorithm suggested in (Larrosa & Heras 2005) is obtained by activating function `Inference` (Figure 2) discarding lines 4-10.

New Inference Rules

Directed Resolution

Some times NRES cannot be applied in the current formula. However, it may be still possible to use resolution to *move* the knowledge where it can be used by NRES. *Directed resolution* (DRES) is one way to implement this idea. In the following, and without loss of generality, we will assume that the set of variables X is ordered.

DRES is the instantiation of Max-RES to the case in which the two clashing clauses only differ in that one of them has one additional literal h whose variable is smaller than the variable of the clashing literal (i.e., $var(h) < var(l)$, which yields,

$$(l \vee A \vee h, u), (\bar{l} \vee A, w) \equiv \left\{ \begin{array}{l} (A \vee h, m), \\ (l \vee A \vee h, u \ominus m), \\ (\bar{l} \vee A, w \ominus m), \\ (\bar{l} \vee \bar{h} \vee A, m) \end{array} \right\}$$

with $m = \min\{u, w\}$. The equivalence holds because the resolvent is $(A \vee h \vee A, m)$ which is equivalent to $(A \vee h, m)$. The first compensation clauses instantiates to $(l \vee A \vee h \vee \bar{A}, m)$. Clearly, it is trivially satisfied. Hence, it can be removed. The second compensation clauses instantiates to $(\bar{l} \vee \overline{A \vee h} \vee A, m)$, which is equivalent to $\{(\bar{l} \vee \bar{A} \vee \bar{h} \vee A, m), (\bar{l} \vee \bar{A} \vee h \vee A, m), (\bar{l} \vee A \vee \bar{h} \vee A, m)\}$. Clearly, the first and second clauses can be removed and the third clause is equivalent to $(\bar{l} \vee \bar{h} \vee A, m)$.

Example 3 *Consider the formula* $\{(x \vee y, 1), (x \vee z, 1)(\bar{y}, 1), (\bar{z}, 1), (\bar{x}, 2)\}$ *with* $\top = 3$ *and* $x < y < z$. *The formula cannot be simplified and NRES cannot be applied. However, we can apply DRES between the first and the third clause, and between the second and the fourth one. The result is* $\{(\bar{x} \vee \bar{y}, 1), (\bar{x} \vee \bar{z}, 1), (x, 2), (\bar{x}, 2)\}$. *NRES can be applied to the third and fourth clauses, producing* $\{(\bar{x} \vee \bar{y}, 1), (\bar{x} \vee \bar{z}, 1), (\square, 2)\}$. *Finally, the binary clauses can be made mandatory because* $2 \oplus 1 = \top$ *producing* $\{(\bar{x} \vee \bar{y}, \top), (\bar{x} \vee \bar{z}, \top), (\square, 2)\}$.

In the next example we show the importance of establishing an ordering between variables to avoid falling into cycles:

Example 4 *Consider the formula* $\{(x \vee y, 1), (\bar{x}, 1)\}$ *with* $\top = 3$. *If we apply DRES without ordering we obtain* $\{(\bar{x} \vee \bar{y}, 1), (x, 1)\}$. *Observe that we can apply again DRES without ordering and then we return to the initial formula.*

Hyper Resolution

In SAT, hyper resolution is a well known concept that refers to the compression of several resolution steps into one single step. We denote k-RES the compression of k resolution steps. NRES is beneficial because it derives smaller clauses. Pushing further this idea, we identify two situations where a small number of resolution steps derive smaller clauses. The first case corresponds to 2-RES. The initial situation is,

$$\{(l \vee h \vee A, u), (\bar{l} \vee q \vee A, v), (\bar{h} \vee q \vee A, w)\}$$

first, we apply Max-RES between the first and the second clauses obtaining,

$$\left\{ \begin{array}{l} (h \vee q \vee A, m_1), (l \vee h \vee A, u \ominus m_1), \\ (\bar{l} \vee q \vee A, v \ominus m_1), (l \vee h \vee \bar{q} \vee A, m_1), \\ (\bar{l} \vee \bar{h} \vee q \vee A, m_1), (\bar{h} \vee q \vee A, w) \end{array} \right\}$$

where $m_1 = min\{u, v\}$. Then, we apply NRES between the first and the last clauses,

$$\left\{ \begin{array}{l} (q \vee A, m), (h \vee q \vee A, m_1 \ominus m) \\ (\bar{h} \vee q \vee A, w \ominus m), (l \vee h \vee A, u \ominus m_1) \\ (\bar{l} \vee q \vee A, v \ominus m_1), (l \vee h \vee \bar{q} \vee A, m_1) \\ (\bar{l} \vee \bar{h} \vee q \vee A, m_1) \end{array} \right\}$$

where $m = min\{u, v, w\}$. Note that resolvent $(q \vee A, m)$ is smaller than any original clause.

Example 5 *Consider the formula* $\{(x \vee y, 1), (x \vee z, 1), (\bar{y} \vee \bar{z}, 1), (x, 2)\}$ *with* $\top = 3$ *and* $x < y < z$. *It cannot be simplified and NRES and DRES cannot be applied. However, we can apply the previous* 2*-RES transformation and we obtain* $\{(x, 3), (x \vee y \vee z, 1), (\bar{x} \vee \bar{y} \vee \bar{z}, 1)\}$. *The resulting formula can be simplified to* $\{(\bar{y} \vee \bar{z}, 1)\}$.

The second case corresponds to 3-RES. Given the initial situation,

$$\{(l \vee h \vee A, u), (\bar{h} \vee A, w_1), (\bar{l} \vee q \vee A, v), (\bar{q} \vee A, w_2)\}$$

we apply Max-RES to the first and second clauses (first step) and to the third and fourth clauses (second step) obtaining,

$$\left\{ \begin{array}{l} (l \vee A, m_1), (\bar{h} \vee A, w_1 \ominus m_1), (h \vee l \vee A, u \ominus m_1) \\ (\bar{h} \vee \bar{l} \vee A, m_1), (\bar{l} \vee A, m_2), (\bar{q} \vee A, w_2 \ominus m_2) \\ (q \vee \bar{l} \vee A, v \ominus m_2), (\bar{q} \vee l \vee A, m_2) \end{array} \right\}$$

where $m_1 = min\{u, w_1\}$ and $m_2 = min\{v, w_2\}$. Finally, we apply NRES (third step) to the first and fifth clauses,

$$\left\{ \begin{array}{l} (A, m), (\bar{h} \vee A, w_1 \ominus m_1), (h \vee l \vee A, u \ominus m_1) \\ (\bar{h} \vee \bar{l} \vee A, m_1), (\bar{q} \vee A, w_2 \ominus m_2) \\ (q \vee \bar{l} \vee A, v \ominus m_2), (\bar{q} \vee l \vee A, m_2) \\ (l \vee A, m_1 \ominus m), (\bar{l} \vee A, m_2 \ominus m) \end{array} \right\}$$

where $m = min\{m_1, m_2\}$. Observe that (A, m) is smaller than any original clause. Note that the two first resolution steps are similar to DRES, but regardless of the order between the variables.

Example 6 *Consider the formula* $\{(z \vee y, 1), (\bar{y}, 1), (\bar{z} \vee x, 1), (\bar{x}, 1)\}$ *with* $\top = 2$ *and* $x < y < z$. *It cannot be simplified and NRES, DRES and the previous* 2*-RES rule cannot be applied. However, we can apply the* 3*-RES rule. It produces* $\{(\square, 1), (\bar{z} \vee \bar{y}, 1), (z \vee \bar{x}, 1)\}$ *which is simplified to* $\{(\square, 1), (\bar{z} \vee \bar{y}, \top), (z \vee \bar{x}, \top)\}$.

The algorithm that incorporates the new inference rules is obtained by considering all the lines in the `Simplify` function.

Experimental Results

In this Section we evaluate the performance of Max-DPLL enhanced with the three new inference rules against some state-of-the-art Max-SAT solvers. Our C implementation as well as problem instances are freely available as part of the TOOLBAR software. [1] Although, our implementation is conceptually equivalent to the pseudo-code of Figures 1 and 2 it should be noted that such code aimed at clarity and simplicity. Thus, a direct translation into C is highly inefficient. The main source of inefficienty is the time that `Simplify` and `Inference` spend searching for clauses that match with the left-hand side of the simplification and inference rules. This overhead, which depends on the number of clauses, takes place at each iteration of the do-loops. One way to decrease it is to restrict the set of clauses under consideration. Our current implementation only takes into account clauses of arity less than or equal to two (the number of such clauses is $O(n^2)$ where n is the number of variables). Another way to decrease such overhead is to idenfity those events that may raise the applicability of the transformations. For instance, a clause may be made mandatory (line 5 of Figure 1) only when its weight or the weight of the empty clause increases. Similarly, a unit clause can be reduced (line 9 of Figure 1) only when a unit clause is made mandatory (by a cost increment) or a binary mandatory clause becomes unary (by a previous unit clause reduction). Then, our implementation reacts to these events

[1] `http://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/SoftCSP`

and triggers the corresponding rules. Such approach is well-known in the contraint satisfaction field and it is usually implemented with streams of pending events (Bessière 1994; de Givry *et al.* 2005).

We compared our implementation, that we call MAX-DPLL, with the following solvers:

- MAXSOLVER (Xing & Zhang 2005). It is restricted to instances with less than 200 variables and 1000 clauses. Unlike the others, a local search pre-process computes an initial good quality upper bound.
- LB4A (Shen & Zhang 2004). It is restricted to Max-2-SAT problems with unit weights and without repeated clauses.
- UP (Chu Min Li & Planes 2005). It is restricted to problems with unit weights.
- MEDAC (de Givry *et al.* 2005). Solves Max-SAT instances by transforming them to Weighted CSPs. Then, it performs a depth-first search and maintains a local consistency property called EDAC. There is a very close relation between MEDAC and Max-DPLL enhanced with NRES, DRES and the 3-RES rule introduced in this paper.

In (de Givry *et al.* 2003) it was shown that solving Max-SAT instances as Weighted CSPs was superior to pseudo-boolean solvers (OPBDP (Barth 1995) and PBS (F. Aloul & Sakallah 2002)) or MIP solvers (CPLEX). Given that MEDAC was shown to be much more efficient than the algorithm used in (de Givry *et al.* 2003), we have discarded those solvers in our comparison. In (Larrosa & Heras 2005) it was also shown that MEDAC was vastly superior to MAX-DPLL with NRES. Thus, we also omit it in the comparison. We have considered the following classes of problems:

- Random Max-2-SAT instances of 80 variables and Max-3-SAT instances of 40 variables with varying number of clauses generated with *Cnfgen*. [2], a random k-SAT generator that allows repeated clauses.
- Random Max-2-SAT instances of 100 variables with varying number of clauses using the generator of (Shen & Zhang 2004) that does not allow repeated clauses.
- Max-Cut instances from random graphs of 60 nodes with varying number of edges. Instances were formulated as Max-2-SAT without repeated clauses problems as proposed in (Shen & Zhang 2004).
- Max-Clique instances from the DIMACS challenge [3].

In all the random cases, samples had 30 instances. The number of clauses varied from 400 to 1000 because instances with less than 400 clauses were solved almost instantly (less than 1 second) with all the solvers while larger problems could not be solved by MaxSolver because of its own size limitations. Executions were made on a 3.2 Ghz Pentium 4 computer with Linux.

[2] `ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/contributed/UCSC/instances`

[3] `ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/clique`

Figure 3 presents the results on Max-2-SAT with and without repeated clauses. Plots report the mean cpu time required by each solver for the different number of clauses. It can be seen that, as the number of clauses increases, problems become harder and harder. MAX-DPLL is the only algorithm that can solve the problems within a reasonable time. In problems with repetitions, it can be up to 35 times faster than MEDAC the second best option, and more than 70 times faster than the rest. In problems without repetitions the differences are smaller, but still MAX-DPLL is more than 7 times faster than the rest.

Figure 4 presents the results on Max-3-SAT and Max-CUT. As before, plots report the mean cpu time required by each solver for the different number of clauses. It can be seen again that, as the number of clauses increases, problems become harder. Regarding Max-3-SAT, we observe that MEDAC and MAX-DPLL are one order of magnitude faster than the rest. MAX-DPLL is slightly faster than MEDAC. Regarding Max-CUT, MAX-DPLL is again the fastest algorithm, nearly 20 times faster than its follower LB4A.

The DIMACS Max-Clique benchmark includes 66 instances that we tried to solve with a time limit of 2 hours. MEDAC solved 32 instances. Other Max-SAT solvers could not be used because of their limitations. MAX-DPLL solved 37 instances. For comparison purposes, note that Max-Clique dedicated algorithms (Ostergard 2002) and (Régin 2003) solved 36 and 54 instances, respectively. Although they used different computers and implementations, our time limit was set in order to make a fair comparison.

Conclusions and Future Work

Max-SAT is an important problem with wide practical applicability and many recent works have been devoted to solving it efficiently. In the work of (Larrosa & Heras 2005), DPLL and resolution were extended from SAT to Max-SAT. It was also shown that Max-DPLL enhanced with a certain form of resolution, called NRES, provided a simple, yet fairly efficient algorithm. In this paper we have introduced three additional inference rules. The three of them are particular cases of resolution. Our experiments show that Max-DPLL augmented with these rules yields, probably, the best current Max-SAT solver. The performance of DPLL has been dramatically improved in the last years by incorporating features such as clause learning or restarts. Since Max-DPLL is so close to DPLL, we plan to investigate in the future the extension of these techniques to the Max-SAT context.

References

Barth, P. 1995. A davis-putnam based enumeration algorithm for linear pseudo-boolean optimization. In *Tech. Rep. MPI-I-95-2-003, Max-Plank Institut Fr Informatik*, 346–353.

Bessière, C. 1994. Arc-consistency and arc-consistency again. *Artificial Intelligence* 65(1):179–190.

Chu Min Li, F. M., and Planes, J. 2005. Exploiting unit propagation to compute lower bounds in branch and bound max-sat solvers. In *Proc. of the 11th CP*.

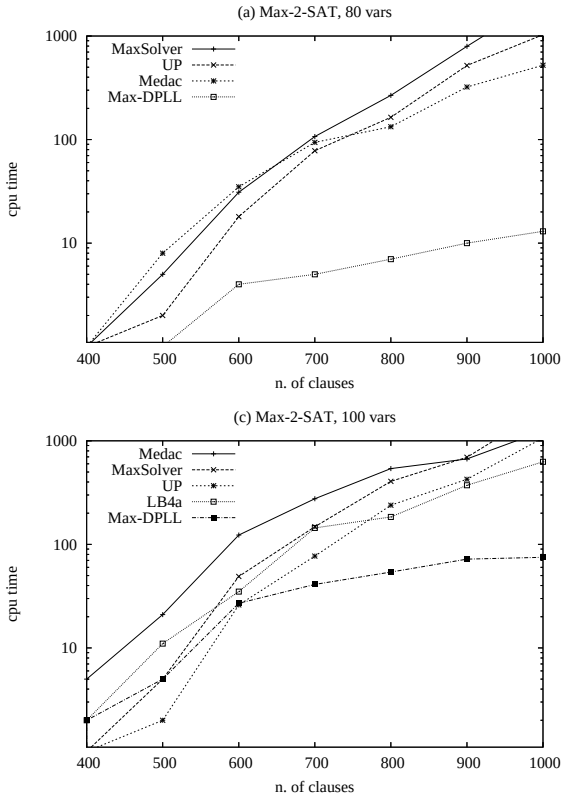

Figure 3: Results on Max-2-SAT with (top) and without (bottom) repeated clauses. Note the logarithmic scale.

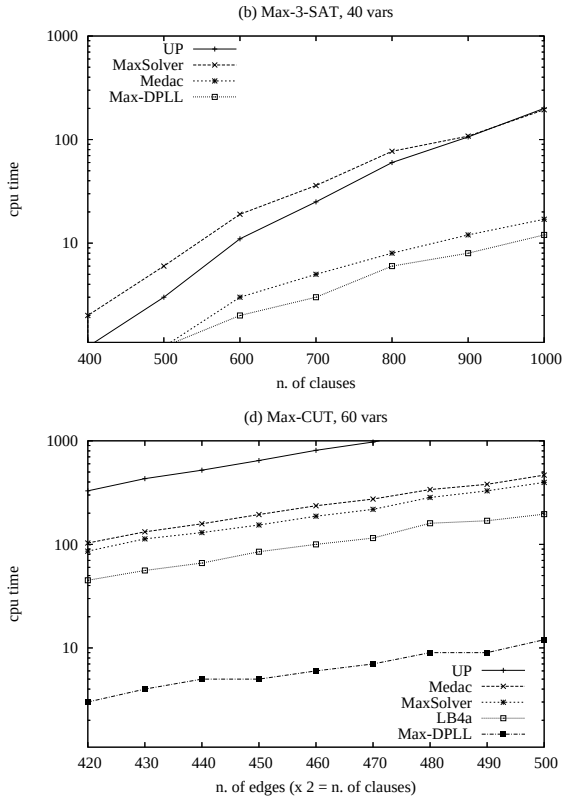

Figure 4: Results on Max-3-SAT (top) and Max-CUT (bottom). Note the logarithmic scale.

Cooper, M., and Schiex, T. 2004. Arc consistency for soft constraints. *Artificial Intelligence* 154(1-2):199–227.

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

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

de Givry, S.; Heras, F.; Larrosa, J.; and Zytnicki, M. 2005. Existential arc consistency: getting closer to full arc consistency in weighted csps. In *Proc. of the 19th IJCAI*.

D.M. Strickland, E. B., and Sokol, J. 2005. Optimal protein structure alignment using maximum cliques. *Operations Research* 53:389–402.

F. Aloul, A. Ramani, I. M., and Sakallah, K. 2002. Pbs: A backtrack-search pseudo boolean solver and optimizer. In *Proc. of Symp. on the Theory and Applications of Satisfiability Testing (SAT)*, 346–353.

H. Xu, R. R., and Sakallah, K. 2002. sub-sat: A formulation for relaxed boolean satisfiability with applications in rounting. In *Proc. Int. Symp. on Physical Design*.

Larrosa, J., and Heras, F. 2005. Resolution in max-sat and its relation to local consistency for weighted csps. In *Proc. of the 19th IJCAI*.

Larrosa, J., and Schiex, T. 2004. Solving weighted csp by maintaining arc-consistency. *Artificial Intelligence* 159(1-2):1–26.

Ostergard, P. R. J. 2002. A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics* 120:197–207.

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

Régin, J.-C. 2003. Using constraint programming to solve the maximum clique problem. In *Proc. of the 9th CP*, 634–648. Kinsale, Ireland: LNCS 2833. Springer Verlag.

Shen, H., and Zhang, H. 2004. Improving exact algorithms for max-2-sat. In *Proc. of the 8th International Symposium on Artificial Intelligence and Mathematics*.

Vasquez, M., and Hao, J. 2001. 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).

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