Journal of Artificial Intelligence Research 30 (2007) 321–359 Submitted 11/06; published 10/07

New Inference Rules for Max-SAT

Chu Min Li CHU-MIN.LI@U-PICARDIE.FR
LaRIA, Université de Picardie Jules Verne
33 Rue St. Leu, 80039 Amiens Cedex 01, France

Felip Manyà FELIP@IIIA.CSIC.ES
IIIA, Artificial Intelligence Research Institute
CSIC, Spanish National Research Council
Campus UAB, 08193 Bellaterra, Spain

Jordi Planes JPLANES@DIEI.UDL.ES
Computer Science Department, Universitat de Lleida
Jaume II, 69, 25001 Lleida, Spain

Abstract

Exact Max-SAT solvers, compared with SAT solvers, apply little inference at each node of the proof tree. Commonly used SAT inference rules like unit propagation produce a simplified formula that preserves satisfiability but, unfortunately, solving the Max-SAT problem for the simplified formula is not equivalent to solving it for the original formula. In this paper, we define a number of original inference rules that, besides being applied efficiently, transform Max-SAT instances into equivalent Max-SAT instances which are easier to solve. The soundness of the rules, that can be seen as refinements of unit resolution adapted to Max-SAT, are proved in a novel and simple way via an integer programming transformation. With the aim of finding out how powerful the inference rules are in practice, we have developed a new Max-SAT solver, called MaxSatz, which incorporates those rules, and performed an experimental investigation. The results provide empirical evidence that MaxSatz is very competitive, at least, on random Max-2SAT, random Max-3SAT, Max-Cut, and Graph 3-coloring instances, as well as on the benchmarks from the Max-SAT Evaluation 2006.

1. Introduction

In recent years there has been a growing interest in developing fast exact Max-SAT solvers (Alber, Gramm, & Niedermeier, 2001; Alsinet, Manyà, & Planes, 2003b, 2005; de Givry, Larrosa, Meseguer, & Schiex, 2003; Li, Manyà, & Planes, 2005; Xing & Zhang, 2004; Zhang, Shen, & Manyà, 2003) due to their potential to solve over-constrained NP-hard problems encoded in the formalism of Boolean CNF formulas. Nowadays, Max-SAT solvers are able to solve a lot of instances that are beyond the reach of the solvers developed just five years ago. Nevertheless, there is yet a considerable gap between the difficulty of the instances solved with current SAT solvers and the instances solved with the best performing Max-SAT solvers.

The motivation behind our work is to bridge that gap between complete SAT solvers and exact Max-SAT solvers by investigating how the technology previously developed for SAT (Goldberg & Novikov, 2001; Li, 1999; Marques-Silva & Sakallah, 1999; Zhang, 1997; Zhang, Madigan, Moskewicz, & Malik, 2001) can be extended and incorporated into Max-

SAT. More precisely, we focus the attention on branch and bound Max-SAT solvers based on the Davis-Putnam-Logemann-Loveland (DPLL) procedure (Davis, Logemann, & Loveland, 1962; Davis & Putnam, 1960).

One of the main differences between SAT solvers and Max-SAT solvers is that the former make an intensive use of unit propagation at each node of the proof tree. Unit propagation, which is a highly powerful inference rule, transforms a SAT instance ϕ into a satisfiability equivalent SAT instance ϕ' which is easier to solve. Unfortunately, solving the Max-SAT problem for ϕ is, in general, not *equivalent* to solving it for ϕ'; i.e., the number of unsatisfied clauses in ϕ and ϕ' is not the same for every truth assignment. For example, if we apply unit propagation to the CNF formula $\phi = \{x_1, \bar{x}_1 \vee x_2, \bar{x}_1 \vee \neg x_2, \bar{x}_1 \vee x_3, \bar{x}_1 \vee \neg x_3\}$, we obtain $\phi' = \{\Box, \Box\}$, but ϕ and ϕ' are not equivalent because any interpretation satisfying $\neg x_1$ unsatisfies one clause of ϕ and two clauses of ϕ'. Therefore, if we want to compute an optimal solution, we cannot apply unit propagation as in SAT solvers.

We proposed in a previous work (Li et al., 2005) to use unit propagation to compute lower bounds in branch and bound Max-SAT solvers instead of using unit propagation to simplify CNF formulas. In our approach, we detect disjoint inconsistent subsets of clauses via unit propagation. It turns out that the number of disjoint inconsistent subsets detected is an underestimation of the number of clauses that will become unsatisfied when the current partial assignment is extended to a complete assignment. That underestimation plus the number of clauses unsatisfied by the current partial assignment provides a good performing lower bound, which captures the lower bounds based on inconsistency counts that most of the state-of-the-art Max-SAT solvers implement (Alsinet, Manyà, & Planes, 2003a; Alsinet et al., 2003b; Borchers & Furman, 1999; Wallace & Freuder, 1996; Zhang et al., 2003), as well as other improved lower bounds (Alsinet, Manyà, & Planes, 2004; Alsinet et al., 2005; Xing & Zhang, 2004, 2005).

On the one hand, the number of disjoint inconsistent subsets detected is just a conservative underestimation for the lower bound, since every inconsistent subset ϕ increases the lower bound by one independently of the number of clauses of ϕ unsatisfied by an optimal assignment. However, an optimal assignment can violate more than one clause of an inconsistent subset. Therefore, we should be able to improve the lower bound based on counting the number of disjoint inconsistent subsets of clauses.

On the other hand, despite the fact that good quality lower bounds prune large parts of the search space and accelerate dramatically the search for an optimal solution, whenever the lower bound does not reach the best solution found so far (upper bound), the solver continues exploring the search space below the current node. During that search, solvers often redetect the same inconsistencies when computing the lower bound at different nodes. Basically, the problem with lower bound computation methods is that they do not simplify the CNF formula in such a way that the unsatisfied clauses become explicit. Lower bounds are just a pruning technique.

To overcome the above two problems, we define a set of *sound* inference rules that transform a Max-SAT instance ϕ into a Max-SAT instance ϕ' which is easier to solve. In Max-SAT, an inference rule is sound whenever ϕ and ϕ' are equivalent.

Let us see an example of inference rule: Given a Max-SAT instance ϕ that contains three clauses of the form $l_1, l_2, \bar{l}_1 \vee \bar{l}_2$, where l_1, l_2 are literals, we replace ϕ with the CNF formula

$$\phi' = (\phi - \{l_1, l_2, \bar{l}_1 \vee \bar{l}_2\}) \cup \{\square, l_1 \vee l_2\}.$$

Note that the rule detects a contradiction from $l_1, l_2, \bar{l}_1 \vee \bar{l}_2$ and, therefore, replaces these clauses with an empty clause. In addition, the rule adds the clause $l_1 \vee l_2$ to ensure the equivalence between ϕ and ϕ'. For any assignment containing either $l_1 = 0, l_2 = 1$, or $l_1 = 1, l_2 = 0$, or $l_1 = 1, l_2 = 1$, the number of unsatisfied clauses in $\{l_1, l_2, \bar{l}_1 \vee \bar{l}_2\}$ is 1, but for any assignment containing $l_1 = 0, l_2 = 0$, the number of unsatisfied clauses is 2. Note that even when any assignment containing $l_1 = 0, l_2 = 0$ is not the best assignment for the subset $\{l_1, l_2, \bar{l}_1 \vee \bar{l}_2\}$, it can be the best for the whole formula. By adding $l_1 \vee l_2$, the rule ensures that the number of unsatisfied clauses in ϕ and ϕ' is also the same when $l_1 = 0, l_2 = 0$.

That inference rule adds the new clause $l_1 \vee l_2$, which may contribute to another contradiction detectable via unit propagation. In this case, the rule allows to increase the lower bound by 2 instead of 1. Moreover, the rule makes explicit a contradiction among $l_1, l_2, \bar{l}_1 \vee \bar{l}_2$, so that the contradiction does not need to be redetected below the current node.

Some of the inference rules defined in the paper are already known in the literature (Bansal & Raman, 1999; Niedermeier & Rossmanith, 2000), others are original for Max-SAT. The new rules were inspired by different unit resolution refinements applied in SAT, and were selected because they could be applied in a natural and efficient way. In a sense, we can summarize our work telling that we have defined the Max-SAT counterpart of SAT unit propagation.

With the aim of finding out how powerful the inference rules are in practice, we have designed and implemented a new Max-SAT solver, called MaxSatz, which incorporates those rules, as well as the lower bound defined in a previous work (Li et al., 2005), and performed an experimental investigation. The results provide empirical evidence that MaxSatz is very competitive, at least, on random Max-2SAT, random Max-3SAT, Max-Cut, and Graph 3-coloring instances, as well as on the benchmarks from the Max-SAT Evaluation 2006[1].

The structure of the paper is as follows. In Section 2, we give some preliminary definitions. In Section 3, we describe a basic branch and bound Max-SAT solver. In Section 4, we define the inference rules and prove their soundness in a novel and simple way via an integer programming transformation. We also give examples to illustrate that the inference rules may produce better quality lower bounds. In Section 5, we present the implementation of the inference rules in MaxSatz. In Section 6, we describe the main features of MaxSatz. In Section 7, we report on the experimental investigation. In Section 8, we present the related work. In Section 9, we present the conclusions and future work.

2. Preliminaries

In propositional logic a variable x_i may take values 0 (for false) or 1 (for true). A literal l_i is a variable x_i or its negation $\bar{x}_i$. A clause is a disjunction of literals, and a CNF formula ϕ is a conjunction of clauses. The length of a clause is the number of its literals. The size of ϕ, denoted by $|\phi|$, is the sum of the length of all its clauses.

1. http://www.iiia.csic.es/~maxsat06

An assignment of truth values to the propositional variables satisfies a literal x_i if x_i takes the value 1 and satisfies a literal $\bar{x}_i$ if x_i takes the value 0, satisfies a clause if it satisfies at least one literal of the clause, and satisfies a CNF formula if it satisfies all the clauses of the formula. An empty clause, denoted by $\Box$, contains no literals and cannot be satisfied. An assignment for a CNF formula ϕ is complete if all the variables occurring in ϕ have been assigned; otherwise, it is partial.

The Max-SAT problem for a CNF formula ϕ is the problem of finding an assignment of values to propositional variables that minimizes the number of unsatisfied clauses (or equivalently, that maximizes the number of satisfied clauses). Max-SAT is called Max-kSAT when all the clauses have k literals per clause. In the following, we represent a CNF formula as a multiset of clauses, since duplicated clauses are allowed in a Max-SAT instance.

CNF formulas ϕ_1 and ϕ_2 are equivalent if ϕ_1 and ϕ_2 have the same number of unsatisfied clauses for every complete assignment of ϕ_1 and ϕ_2.

3. A Basic Max-SAT Solver

The space of all possible assignments for a CNF formula ϕ can be represented as a search tree, where internal nodes represent partial assignments and leaf nodes represent complete assignments. A basic branch and bound algorithm for Max-SAT explores the search tree in a depth-first manner. At every node, the algorithm compares the number of clauses unsatisfied by the best complete assignment found so far —called upper bound (UB)— with the number of clauses unsatisfied by the current partial assignment ($\#emptyClauses$) plus an underestimation of the minimum number of non-empty clauses that will become unsatisfied if we extend the current partial assignment into a complete assignment ($underestimation$).

The sum $\#emptyClauses + underestimation$ is a lower bound (LB) of the minimum number of clauses unsatisfied by any complete assignment extended from the current partial assignment. Obviously, if $LB \geq UB$, a better solution cannot be found from this point in search. In that case, the algorithm prunes the subtree below the current node and backtracks to a higher level in the search tree.

If $LB < UB$, the algorithm tries to find a possible better solution by extending the current partial assignment by instantiating one more variable; which leads to the creation of two branches from the current branch: the left branch corresponds to assigning the new variable to false, and the right branch corresponds to assigning the new variable to true. In that case, the formula associated with the left (right) branch is obtained from the formula of the current node by deleting all the clauses containing the literal $\bar{x}$ (x) and removing all the occurrences of the literal x ($\bar{x}$); i.e., the algorithm applies the *one-literal rule*.

The solution to Max-SAT is the value that UB takes after exploring the entire search tree.

Figure 1 shows the pseudo-code of a basic solver for Max-SAT. We use the following notations:

- *simplifyFormula*(ϕ) is a procedure that simplifies ϕ by applying sound inference rules.
- *#emptyClauses*(ϕ) is a function that returns the number of empty clauses in ϕ.

Input: *max-sat*(ϕ, UB) : A CNF formula ϕ and an upper bound UB

```
φ ← simplifyFormula(φ);
if φ = ∅ or φ only contains empty clauses then
   return #emptyClauses(φ);
end if
LB ← #emptyClauses(φ) + underestimation(φ, UB);
if  LB ≥ UB then
   return UB;
end if
x ← selectVariable(φ);
UB ← min(UB, max-sat(φ_x̄, UB));
return min(UB, max-sat(φ_x, UB));
```

Output: The minimal number of unsatisfied clauses of ϕ

Figure 1: A basic branch and bound algorithm for Max-SAT

- LB is a lower bound of the minimum number of unsatisfied clauses in ϕ if the current partial assignment is extended to a complete assignment. We assume that its initial value is 0.
- *underestimation*(ϕ, UB) is a function that returns an underestimation of the minimum number of non-empty clauses in ϕ that will become unsatisfied if the current partial assignment is extended to a complete assignment.
- UB is an upper bound of the number of unsatisfied clauses in an optimal solution. We assume that its initial value is the total number of clauses in the input formula.
- *selectVariable*(ϕ) is a function that returns a variable of ϕ following an heuristic.
- ϕ_x ($\phi_{\bar{x}}$) is the formula obtained by applying the one-literal rule to ϕ using the literal x ($\bar{x}$).

State-of-the-art Max-SAT solvers implement the basic algorithm augmented with powerful inference techniques, good quality lower bounds, clever variable selection heuristics, and efficient data structures.

We have recently defined (Li et al., 2005) a lower bound computation method in which the underestimation of the lower bound is the number of disjoint inconsistent subsets that can be detected using unit propagation. The pseudo-code is shown in Figure 2.

Example 1 *Let ϕ be the following CNF formula:*

$$\{x_1, x_2, x_3, x_4, \bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3, \bar{x}_4, x_5, \bar{x}_5 \vee \bar{x}_2, \bar{x}_5 \vee x_2\}.$$

With our approach we are able to establish that the number of disjoint inconsistent subsets of clauses in ϕ is at least 3. Therefore, the underestimation of the lower bound is 3. The steps performed are the following ones:

Input: *underestimation*(ϕ, UB) : A CNF formula ϕ and an upper bound UB
1: *underestimation* $\leftarrow 0$;
2: apply the one-literal rule to the unit clauses of ϕ (unit propagation) until an empty clause is derived;
3: **if** no empty clause can be derived **then**
4: return *underestimation*;
5: **end if**
6: $\phi \leftarrow \phi$ without the clauses that have been used to derive the empty clause;
7: *underestimation* := *underestimation* + 1;
8: **if** *underestimation*+#*emptyClauses*$(\phi) \geq UB$ **then**
9: return *underestimation*;
10: **end if**
11: go to 2;
Output: the underestimation of the lower bound for ϕ

Figure 2: Computation of the underestimation using unit propagation

1. *$\phi = \{x_4, \bar{x}_4, x_5, \bar{x}_5 \vee \bar{x}_2, \bar{x}_5 \vee x_2\}$, the first inconsistent subset detected using unit propagation is $\{x_1, x_2, x_3, \bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3\}$, and underestimation $= 1$.*

2. *$\phi = \{x_5, \bar{x}_5 \vee \bar{x}_2, \bar{x}_5 \vee x_2\}$, the second inconsistent subset detected using unit propagation is $\{x_4, \bar{x}_4\}$, and underestimation $= 2$.*

3. *$\phi = \emptyset$, the third inconsistent subset detected using unit propagation is $\{x_5, \bar{x}_5 \vee \bar{x}_2, \bar{x}_5 \vee x_2\}$, and underestimation $= 3$. Since ϕ is empty, the algorithm stops.*

4. Inference Rules

We define the set of inference rules considered in the paper. They were inspired by different unit resolution refinements applied in SAT, and were selected because they could be applied in a natural and efficient way. Some of them are already known in the literature (Bansal & Raman, 1999; Niedermeier & Rossmanith, 2000), others are original for Max-SAT.

Before presenting the rules, we define an integer programming transformation of a CNF formula used to establish the soundness of the rules. The method of proving soundness is novel in Max-SAT, and provides clear and short proofs.

4.1 Integer Programming Transformation of a CNF Formula

Assume that $\phi = \{c_1, ..., c_m\}$ is a CNF formula with m clauses over the variables $x_1, ..., x_n$. Let c_i $(1 \leq i \leq m)$ be $x_{i_1} \vee ... \vee x_{i_k} \vee \bar{x}_{i_{k+1}} \vee ... \vee \bar{x}_{i_{k+r}}$. Note that we put all positive literals in c_i before the negative ones.

We consider all the variables in c_i as integer variables taking values 0 or 1, and define the integer transformation of c_i as

$$\mathcal{E}_i(x_{i_1}, ..., x_{i_k}, x_{i_{k+1}}, ..., x_{i_{k+r}}) = (1 - x_{i_1})...(1 - x_{i_k})x_{i_{k+1}}...x_{i_{k+r}}$$

Obviously, $\mathcal{E}_i$ has value 0 iff at least one of the variables x_{i_j}'s ($1 \leq j \leq k$) is instantiated to 1 or at least one of the variables x_{i_s}'s ($k+1 \leq s \leq k+r$) is instantiated to 0. In other words, $\mathcal{E}_i$=0 iff c_i is satisfied. Otherwise $\mathcal{E}_i$=1.

A literal l corresponds to an integer denoted by l itself for our convenience. The intention of the correspondence is that the literal l is satisfied if the integer l is 1 and is unsatisfied if the integer l is 0. So if l is a positive literal x, the corresponding integer l is x, $\bar{l}$ is 1-x=1-l, and if l is a negative literal $\bar{x}$, l is 1-x and $\bar{l}$ is x=1-(1-x)=1-l. Consequently, $\bar{l}$=1-l in any case.

We now generically write c_i as $l_1 \vee l_2 \vee ... \vee l_{k+r}$. Its integer programming transformation is

$$\mathcal{E}_i = (1 - l_1)(1 - l_2)...(1 - l_{k+r}).$$

The integer programming transformation of a CNF formula $\phi = \{c_1, ..., c_m\}$ over the variables $x_1, ..., x_n$ is defined as

$$\mathcal{E}(x_1, ..., x_n) = \sum_{i=1}^{m} \mathcal{E}_i \tag{1}$$

That integer programming transformation was used (Huang & Jin, 1997; Li & Huang, 2005) to design a local search procedure, and is called pseudo-Boolean formulation by Boros and Hammer (2002). Here, we extend it to empty clauses: if c_i is empty, then $\mathcal{E}_i$=1.

Given an assignment A over the variables $x_1, ..., x_n$, the value of $\mathcal{E}$ is the number of unsatisfied clauses in ϕ. If A satisfies all clauses in ϕ, then $\mathcal{E} = 0$. Obviously, the minimum number of unsatisfied clauses of ϕ is the minimum value of $\mathcal{E}$.

Let ϕ_1 and ϕ_2 be two CNF formulas, and let $\mathcal{E}_1$ and $\mathcal{E}_2$ be their integer programming transformations. It is clear that ϕ_1 and ϕ_2 are equivalent if, and only if, $\mathcal{E}_1$=$\mathcal{E}_2$ for every complete assignment for ϕ_1 and ϕ_2.

4.2 Inference Rules

We next define the inference rules and prove their soundness using the previous integer programming transformation. In the rest of the section, ϕ_1, ϕ_2 and ϕ' denote CNF formulas, and $\mathcal{E}_1$, $\mathcal{E}_2$, and $\mathcal{E}'$ their integer programming transformations. To prove that ϕ_1 and ϕ_2 are equivalent, we prove that $\mathcal{E}_1 = \mathcal{E}_2$.

Rule 1 *(Bansal & Raman, 1999) If* ϕ_1=$\{l_1 \vee l_2 \vee ... \vee l_k,\ \bar{l}_1 \vee l_2 \vee ... \vee l_k\} \cup \phi'$, *then* ϕ_2=$\{l_2 \vee ... \vee l_k\} \cup \phi'$ *is equivalent to* ϕ_1.

Proof 1

$$\begin{aligned}
\mathcal{E}_1 &= (1 - l_1)(1 - l_2)...(1 - l_k) + l_1(1 - l_2)...(1 - l_k) + \mathcal{E}' \\
&= (1 - l_2)...(1 - l_k) + \mathcal{E}' \\
&= \mathcal{E}_2 \quad \blacksquare
\end{aligned}$$

General case resolution does not work in Max-SAT (Bansal & Raman, 1999). Rule 1 establishes that resolution works when two clauses give a strictly shorter resolvent.

Rule 1 is known in the literature as *replacement of almost common clauses*. We pay special attention to the case k=2, where the resolvent is a unit clause, and to the case k=1, where the resolvent is the empty clause. We describe this latter case in the following rule:

Rule 2 *(Niedermeier & Rossmanith, 2000) If* ϕ_1=$\{l,\ \bar{l}\} \cup \phi'$*, then* ϕ_2=$\{\Box\} \cup \phi'$ *is equivalent to* ϕ_1*.*

Proof 2 $\mathcal{E}_1$=1-l+ l+$\mathcal{E}'$=1+ $\mathcal{E}'$=$\mathcal{E}_2$ ■

Rule 2, which is known as *complementary unit clause rule*, can be used to replace two complementary unit clauses with an empty clause. The new empty clause contributes to the lower bounds of the search space below the current node by incrementing the number of unsatisfied clauses, but not by incrementing the underestimation, which means that this contradiction does not have to be redetected again. In practice, that simple rule gives rise to considerable gains.

The following rule is a more complicated case:

Rule 3 *If* ϕ_1=$\{l_1,\ \bar{l}_1 \vee \bar{l}_2,\ l_2\} \cup \phi'$*, then* ϕ_2=$\{\Box,\ l_1 \vee l_2\} \cup \phi'$ *is equivalent to* ϕ_1*.*

Proof 3

$$
\begin{aligned}
\mathcal{E}_1 &= 1 - l_1 + l_1 l_2 + 1 - l_2 + \mathcal{E}' \\
&= 1 + 1 - l_1 + l_2(l_1 - 1) + \mathcal{E}' \\
&= 1 + 1 - l_1 - l_2(1 - l_1) + \mathcal{E}' \\
&= 1 + (1 - l_1)(1 - l_2) + \mathcal{E}' \\
&= \mathcal{E}_2 \quad \blacksquare
\end{aligned}
$$

Rule 3 replaces three clauses with an empty clause, and adds a new binary clause to keep the equivalence between ϕ_1 and ϕ_2.

Pattern ϕ_1 was considered to compute underestimations by Alsinet et al. (2004) and Shen and Zhang (2004); and is also captured by our method of computing underestimations based on unit propagation (Li et al., 2005). Larrosa and Heras mentioned (2005) that existential directional arc consistency (de Givry, Zytnicki, Heras, & Larrosa, 2005) can capture this rule. Note that underestimation computation methods by Alsinet et al. and Shen and Zhang do not add any additional clause as in our approach, they just detect contradictions.

Let us define a rule that generalizes Rule 2 and Rule 3. Before presenting the rule, we define a lemma needed to prove its soundness.

Lemma 1 *If* ϕ_1=$\{l_1,\ \bar{l}_1 \vee l_2\} \cup \phi'$ *and* ϕ_2=$\{l_2,\ \bar{l}_2 \vee l_1\} \cup \phi'$*, then* ϕ_1 *and* ϕ_2 *are equivalent.*

Proof 4

$$
\begin{aligned}
\mathcal{E}_1 &= 1 - l_1 + l_1(1 - l_2) + \mathcal{E}' \\
&= 1 - l_1 + l_1 - l_1 l_2 + \mathcal{E}' \\
&= 1 - l_2 + l_2 - l_1 l_2 + \mathcal{E}' \\
&= 1 - l_2 + (1 - l_1) l_2 + \mathcal{E}' \\
&= \mathcal{E}_2 \quad \blacksquare
\end{aligned}
$$

Rule 4 *If* ϕ_1=$\{l_1,\ \bar{l}_1 \vee l_2,\ \bar{l}_2 \vee l_3,\ ...,\ \bar{l}_k \vee l_{k+1},\ \bar{l}_{k+1}\} \cup \phi'$, *then* ϕ_2=$\{\Box,\ l_1 \vee \bar{l}_2,\ l_2 \vee \bar{l}_3,\ ...,\ l_k \vee \bar{l}_{k+1}\} \cup \phi'$ *is equivalent to* ϕ_1.

Proof 5 *We prove the soundness of the rule by induction on k. When k=1,* $\phi_1 = \{l_1,\ \bar{l}_1 \vee l_2,\ \bar{l}_2\} \cup \phi'$. *By applying Rule 3, we get* $\{\Box,\ l_1 \vee \bar{l}_2\} \cup \phi'$*, which is* ϕ_2 *when* $k = 1$*. Therefore,* ϕ_1 *and* ϕ_2 *are equivalent.*

Assume that Rule 4 is sound for $k = n$*. Let us prove that it is sound for* $k = n + 1$*. In that case:*

$$\phi_1 = \{l_1,\ \bar{l}_1 \vee l_2,\ \bar{l}_2 \vee l_3,\ ...,\ \bar{l}_n \vee l_{n+1},\ \bar{l}_{n+1} \vee l_{n+2},\ \bar{l}_{n+2}\} \cup \phi'.$$

By applying Lemma 1 to the last two clauses of ϕ_1 *(before* ϕ'*), we get*

$$\{l_1,\ \bar{l}_1 \vee l_2,\ \bar{l}_2 \vee l_3,\ ...,\ \bar{l}_n \vee l_{n+1},\ \bar{l}_{n+1},\ l_{n+1} \vee \bar{l}_{n+2}\} \cup \phi'.$$

By applying the induction hypothesis to the first $n+1$ *clauses of the previous CNF formula, we get*

$$\{\Box,\ l_1 \vee \bar{l}_2,\ l_2 \vee \bar{l}_3,\ ...,\ l_n \vee \bar{l}_{n+1},\ l_{n+1} \vee \bar{l}_{n+2}\} \cup \phi',$$

which is ϕ_2 *when* $k = n + 1$*. Therefore,* ϕ_1 *and* ϕ_2 *are equivalent and the rule is sound.* ■

Rule 4 is an original inference rule. It captures linear unit resolution refutations in which clauses and resolvents are used exactly once. The rule simply adds an empty clause, eliminates two unit clauses and the binary clauses used in the refutation, and adds new binary clauses that are obtained by negating the literals of the eliminated binary clauses. So, all the operations involved can be performed efficiently.

Rule 3 and Rule 4 make explicit a contradiction, which does not need to be redetected in the current subtree. So, the lower bound computation becomes more incremental. Moreover, the binary clauses added by Rule 3 and Rule 4 may contribute to compute better quality lower bounds either by acting as premises of an inference rule or by being part of an inconsistent subset of clauses, as is illustrated in the following example.

Example 2 *Let* ϕ=$\{x_1, \bar{x}_1 \vee \bar{x}_2, x_3, \bar{x}_3 \vee x_2, x_4, \bar{x}_1 \vee \bar{x}_4, \bar{x}_3 \vee \bar{x}_4\}$*. Depending on the ordering in which unit clauses are propagated, unit propagation detects one of the following three inconsistent subsets of clauses:* $\{x_1, \bar{x}_1 \vee \bar{x}_2, x_3, \bar{x}_3 \vee x_2\}$*,* $\{x_1, x_4, \bar{x}_1 \vee \bar{x}_4\}$*, or* $\{x_3, x_4, \bar{x}_3 \vee \bar{x}_4\}$*. Once an inconsistent subset is detected and removed, the remaining set of clauses is satisfiable. Without applying Rule 3 and Rule 4, the lower bound computed is 1, because the underestimation computed using unit propagation is 1.*

Note that Rule 4 can be applied to the first inconsistent subset $\{x_1, \bar{x}_1 \vee \bar{x}_2, x_3, \bar{x}_3 \vee x_2\}$*. If Rule 4 is applied, a contradiction is made explicit and the clauses* $x_1 \vee x_2$ *and* $x_3 \vee \bar{x}_2$ *are added. So,* ϕ *becomes* $\{\Box, x_1 \vee x_2, x_3 \vee \bar{x}_2, x_4, \bar{x}_1 \vee \bar{x}_4, \bar{x}_3 \vee \bar{x}_4\}$*. It turns out that* $\phi - \{\Box\}$ *is an inconsistent set of clauses detectable by unit propagation. Therefore, the lower bound computed is 2.*

If the inconsistent subset $\{x_1, x_4, \bar{x}_1 \vee \bar{x}_4\}$ *is detected, Rule 3 can be applied. Then, a contradiction is made explicit and the clause* $x_1 \vee x_4$ *is added. So,* ϕ *becomes* $\{\Box, x_1 \vee x_4, \bar{x}_1 \vee \bar{x}_2, x_3, \bar{x}_3 \vee x_2, \bar{x}_3 \vee \bar{x}_4\}$*. It turns out that* $\phi - \{\Box\}$ *is an inconsistent set of clauses detectable by unit propagation. Therefore, the lower bound computed is 2.*

Similarly, if the inconsistent subset $\{x_3, x_4, \bar{x}_3 \vee \bar{x}_4\}$ *is detected and Rule 3 is applied, the lower bound computed is 2.*

We observe that, in this example, Rule 3 and Rule 4 not only make explicit a contradiction, but also allow to improve the lower bound.

Unit propagation needs at least one unit clause to detect a contradiction. A drawback of Rule 3 and Rule 4 is that they consume two unit clauses for deriving just one contradiction. A possible situation is that, after branching, those two unit clauses could allow unit propagation to derive two disjoint inconsistent subsets of clauses, as we show in the following example.

Example 3 *Let* $\phi=\{x_1, \bar{x}_1 \vee x_2, \bar{x}_1 \vee x_3, \bar{x}_2 \vee \bar{x}_3 \vee x_4, x_5, \bar{x}_5 \vee x_6, \bar{x}_5 \vee x_7, \bar{x}_6 \vee \bar{x}_7 \vee x_4, \bar{x}_1 \vee \bar{x}_5\}$*. Rule 3 replaces* x_1*,* x_5*, and* $\bar{x}_1 \vee \bar{x}_5$ *with an empty clause and* $x_1 \vee x_5$*. After that, if* x_4 *is selected as the next branching variable and is assigned 0, there is no unit clause in* ϕ *and no contradiction can be detected via unit propagation. The lower bound is 1 in this situation. However, if Rule 3 was not applied before branching,* ϕ *has two unit clauses after branching. In this case, the propagation of* x_1 *allows to detect the inconsistent subset* $\{x_1, \bar{x}_1 \vee x_2, \bar{x}_1 \vee x_3, \bar{x}_2 \vee \bar{x}_3\}$*, and the propagation of* x_5 *allows to detect the inconsistent subset* $\{x_5, \bar{x}_5 \vee x_6, \bar{x}_5 \vee x_7, \bar{x}_6 \vee \bar{x}_7\}$*. So, the lower bound computed after branching is 2.*

On the one hand, Rule 3 and Rule 4 add clauses that can contribute to detect additional conflicts. On the other hand, each application of Rule 3 and Rule 4 consumes two unit clauses, which cannot be used again to detect further conflicts. The final effect of these two factors will be empirically analyzed in Section 7.

Finally, we present two new rules that capture unit resolution refutations in which (i) exactly one unit clause is consumed, and (ii) the unit clause is used twice in the linear derivation of the empty clause.

Rule 5 *If* $\phi_1=\{l_1,\ \bar{l}_1 \vee l_2,\ \bar{l}_1 \vee l_3,\ \bar{l}_2 \vee \bar{l}_3\} \cup \phi'$*, then* $\phi_2=\{\Box,\ l_1 \vee \bar{l}_2 \vee \bar{l}_3,\ \bar{l}_1 \vee l_2 \vee l_3\} \cup \phi'$ *is equivalent to* ϕ_1*.*

Proof 6

$$\begin{aligned}
\mathcal{E}_1 &= 1 - l_1 + l_1(1 - l_2) + l_1(1 - l_3) + l_2 l_3 + \mathcal{E}' \\
&= 1 - l_1 + l_1 - l_1 l_2 + l_1 - l_1 l_3 + l_2 l_3 + \mathcal{E}' \\
&= 1 + l_2 l_3 - l_1 l_2 l_3 + l_1 - l_1 l_2 - l_1 l_3 + l_1 l_2 l_3 + \mathcal{E}' \\
&= 1 + (1 - l_1) l_2 l_3 + l_1(1 - l_2 - l_3 + l_2 l_3) + \mathcal{E}' \\
&= 1 + (1 - l_1) l_2 l_3 + l_1(1 - l_2)(1 - l_3) + \mathcal{E}' \\
&= \mathcal{E}_2 \quad \blacksquare
\end{aligned}$$

We can combine a linear derivation with Rule 5 to obtain Rule 6:

Rule 6 *If* $\phi_1=\{l_1,\ \bar{l}_1 \vee l_2,\ \bar{l}_2 \vee l_3,\ \ldots,\ \bar{l}_k \vee l_{k+1},\ \bar{l}_{k+1} \vee l_{k+2},\ \bar{l}_{k+1} \vee l_{k+3},\ \bar{l}_{k+2} \vee \bar{l}_{k+3}\} \cup \phi'$*, then* $\phi_2=\{\Box,\ l_1 \vee \bar{l}_2,\ l_2 \vee \bar{l}_3,\ \ldots,\ l_k \vee \bar{l}_{k+1},\ l_{k+1} \vee \bar{l}_{k+2} \vee \bar{l}_{k+3},\ \bar{l}_{k+1} \vee l_{k+2} \vee l_{k+3}\} \cup \phi'$ *is equivalent to* ϕ_1*.*

Proof 7 *We prove the soundness of the rule by induction on k. When k=1,*

$$\phi_1 = \{l_1,\ \bar{l}_1 \vee l_2,\ \bar{l}_2 \vee l_3,\ \bar{l}_2 \vee l_4,\ \bar{l}_3 \vee \bar{l}_4\} \cup \phi'.$$

By Lemma 1, we get

$$\{l_1 \vee \bar{l}_2,\ l_2,\ \bar{l}_2 \vee l_3,\ \bar{l}_2 \vee l_4,\ \bar{l}_3 \vee \bar{l}_4\} \cup \phi'.$$

By Rule 5, we get

$$\{l_1 \vee \bar{l}_2,\ \Box,\ l_2 \vee \bar{l}_3 \vee \bar{l}_4,\ \bar{l}_2 \vee l_3 \vee l_4\} \cup \phi',$$

which is ϕ_2 when $k = 1$. Therefore, ϕ_1 and ϕ_2 are equivalent.

Assume that Rule 6 is sound for $k = n$. Let us prove that it is sound for $k = n + 1$. In that case:

$$\phi_1 = \{l_1,\ \bar{l}_1 \vee l_2,\ \bar{l}_2 \vee l_3,\ ...,\ \bar{l}_{n+1} \vee l_{n+2},\ \bar{l}_{n+2} \vee l_{n+3},\ \bar{l}_{n+2} \vee l_{n+4},\ \bar{l}_{n+3} \vee \bar{l}_{n+4}\} \cup \phi'.$$

By Lemma 1, we get

$$\{l_1 \vee \bar{l}_2,\ l_2,\ \bar{l}_2 \vee l_3,\ ...,\ \bar{l}_{n+1} \vee l_{n+2},\ \bar{l}_{n+2} \vee l_{n+3},\ \bar{l}_{n+2} \vee l_{n+4},\ \bar{l}_{n+3} \vee \bar{l}_{n+4}\} \cup \phi'.$$

By applying the induction hypothesis, we get

$$\{l_1 \vee \bar{l}_2,\ \Box,\ l_2 \vee \bar{l}_3,\ ...,\ l_{n+1} \vee \bar{l}_{n+2},\ l_{n+2} \vee \bar{l}_{n+3} \vee \bar{l}_{n+4},\ \bar{l}_{n+2} \vee l_{n+3} \vee l_{n+4}\} \cup \phi',$$

which is ϕ_2 when $k = n + 1$. Therefore, ϕ_1 and ϕ_2 are equivalent and the rule is sound. ■

Similarly to Rule 3 and Rule 4, Rule 5 and Rule 6 make explicit a contradiction, which does not need to be redetected in subsequent search. Therefore, the lower bound computation becomes more incremental. Moreover, they also add clauses which can improve the quality of the lower bound, as illustrated in the following example.

Example 4 *Let ϕ=$\{x_1, \bar{x}_1 \vee x_2, \bar{x}_1 \vee x_3, \bar{x}_2 \vee \bar{x}_3, x_4, x_1 \vee \bar{x}_4, \bar{x}_2 \vee \bar{x}_4, \bar{x}_3 \vee \bar{x}_4\}$. Depending on the ordering in which unit clauses are propagated, unit propagation can detect one of the following inconsistent subsets: $\{x_1, \bar{x}_1 \vee x_2, \bar{x}_1 \vee x_3, \bar{x}_2 \vee \bar{x}_3\}$, $\{x_4, x_1 \vee \bar{x}_4, \bar{x}_2 \vee \bar{x}_4, \bar{x}_1 \vee x_2\}$, $\{x_4, x_1 \vee \bar{x}_4, \bar{x}_3 \vee \bar{x}_4, \bar{x}_1 \vee x_3\}$, in which Rule 5 is applicable. If Rule 5 is not applied, the lower bound computed using the underestimation function of Figure 2 is 1, since the remaining clauses of ϕ are satisfiable once the inconsistent subset of clauses is removed. Rule 5 allows to add two ternary clauses contributing to another contradiction. For example, Rule 5 applied to $\{x_1, \bar{x}_1 \vee x_2, \bar{x}_1 \vee x_3, \bar{x}_2 \vee \bar{x}_3\}$ adds to ϕ clauses $x_1 \vee \bar{x}_2 \vee \bar{x}_3$ and $\bar{x}_1 \vee x_2 \vee x_3$, which, with the remaining clauses of ϕ ($\{x_4, x_1 \vee \bar{x}_4, \bar{x}_2 \vee \bar{x}_4, \bar{x}_3 \vee \bar{x}_4\}$), give the second contradiction detectable via unit propagation. So the lower bound computed using Rule 5 is 2.*

In contrast to Rule 3 and Rule 4, Rule 5 and Rule 6 consume exactly one unit clause for deriving an empty clause. Since a unit clause can be used at most once to derive a conflict via unit propagation, Rule 5 and Rule 6 do not limit the detection of further conflicts via unit propagation.

5. Implementation of Inference Rules

In this section, we describe the implementation of all the inference rules presented in Section 4. We suppose that the CNF formula is loaded and, for every literal ℓ, a list of clauses containing ℓ is constructed. The application of a rule means that some clauses in ϕ_1 are removed from the CNF formula, new clauses in ϕ_2 are inserted into the formula, and the lower bound is increased by 1. Note that in all the inference rules selected in our approach, ϕ_2 contains fewer literals and fewer clauses than ϕ_1, so that new clauses of ϕ_2 can be inserted in the place of the removed clauses of ϕ_1 when an inference rule is applied. Therefore, we do not need dynamic memory management and the implementation can be faster.

Rule 1 for k=2 and Rule 2 can be applied using a matching algorithm (see, e.g., Cormen, Leiserson, Rivest, & Stein, 2001, for an efficient implementation) over the lists of clauses. The first has a time complexity of $O(m)$, where m is the number of clauses in the CNF formula. The second has a time complexity of $O(u)$, where u is the number of unit clauses in the CNF formula. These rules are applied at every node, before any lower bound computation or application of other inference rules. Rule 1 (k=2) is applied as many times as possible to derive unit clauses before applying Rule 2.

The implementation of Rule 3, Rule 4, Rule 5, and Rule 6 is entirely based on unit propagation. Given a CNF formula ϕ, unit propagation constructs an implication graph G (see, e.g., Beame, Kautz, & Sabharwal, 2003), from which the applicability of inference rules is detected. In this section, we first describe the construction of the implication graph, and then describe how to determine the applicability of Rule 3, Rule 4, Rule 5, and Rule 6. Then, we analyze the complexity, termination and (in)completeness of the application of the rules. Finally we discuss the extension of the inference rules to weighted Max-SAT and their implementation.

5.1 Implication Graph

Given a CNF formula ϕ, Figure 3 shows how unit propagation constructs an implication graph whose nodes are literals.

Note that every node in G corresponds to a different literal, where ℓ and $\bar{\ell}$ are considered as different literals. When the CNF formula contains several copies of a unit clause ℓ, the algorithm adds just one node with label ℓ.

Example 5 *Let* ϕ=$\{x_1, x_1, \bar{x}_1 \vee x_2, \bar{x}_1 \vee x_3, \bar{x}_2 \vee \bar{x}_3 \vee x_4, x_5, \bar{x}_5 \vee x_6, \bar{x}_5 \vee x_7, \bar{x}_6 \vee \bar{x}_7 \vee \bar{x}_4, \bar{x}_5 \vee x_8\}$. *UnitPropagation constructs the implication graph of Figure 4, in which we add a special node $\Box$ to highlight the contradiction.*

G is always acyclic because every added edge connects a new node. It is well known that the time complexity of unit propagation is $O(|\phi|)$, where $|\phi|$ is the size of ϕ (see, e.g., Freeman, 1995).

We associate clause c=$\bar{\ell}_1 \vee \bar{\ell}_2 \vee ... \vee \bar{\ell}_{k-1} \vee \ell_k$ with node ℓ_k if node ℓ_k is added into G because of c. Note that node ℓ_k does not have any incoming edge if and only if c is unit (k=1), and the node has only one incoming edge if and only if c is binary (k=2). Once G is constructed, if G contains both ℓ and $\bar{\ell}$ for some literal ℓ (i.e., unit propagation deduces a contradiction), it is easy to identify all nodes from which there exists a path to ℓ or $\bar{\ell}$ in G; i.e., the clauses

Input: $UnitPropagation(\phi)$: ϕ is a CNF formula not containing the complementary unit clauses ℓ and $\bar{\ell}$ for any literal ℓ
initialize G as the empty graph
add a node labeled with ℓ for every literal ℓ in a unit clause c of ϕ
repeat
if $\ell_1, \ell_2, ..., \ell_{k-1}$ are nodes of G, $c = \bar{\ell}_1 \vee \bar{\ell}_2 \vee ... \vee \bar{\ell}_{k-1} \vee \ell_k$ is a clause of ϕ, and ℓ_k is not a node of G, **then**
add into G a node labeled with ℓ_k
add into G a directed edge from node ℓ_i to ℓ_k for every i $(1 \leq i < k)$
end if
until no more nodes can be added or there is a literal ℓ such that both ℓ and $\bar{\ell}$ are nodes of G
Return G
Output: Implication graph G of ϕ

Figure 3: Unit propagation for constructing implication graphs

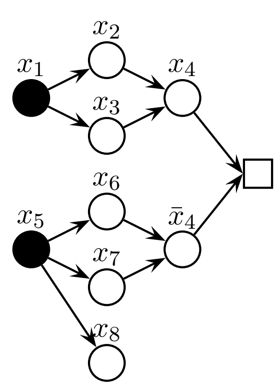

Figure 4: Example of implication graph

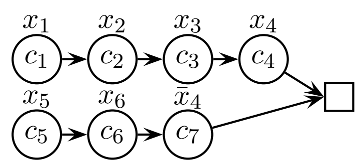

Figure 5: Example of implication graph

implying ℓ or $\bar{\ell}$. All these clauses constitute an inconsistent subset S of ϕ. In the above example, clauses $x_1, \bar{x}_1 \vee x_2, \bar{x}_1 \vee x_3$ and $\bar{x}_2 \vee \bar{x}_3 \vee x_4$ imply x_4, and clauses $x_5, \bar{x}_5 \vee x_6, \bar{x}_5 \vee x_7$ and $\bar{x}_6 \vee \bar{x}_7 \vee \bar{x}_4$ imply $\bar{x}_4$. Clause $\bar{x}_5 \vee x_8$ does not contribute to the contradiction. The inconsistent subset S is $\{x_1, \bar{x}_1 \vee x_2, \bar{x}_1 \vee x_3, \bar{x}_2 \vee \bar{x}_3 \vee x_4, x_5, \bar{x}_5 \vee x_6, \bar{x}_5 \vee x_7, \bar{x}_6 \vee \bar{x}_7 \vee \bar{x}_4\}$.

5.2 Applicability of Rule 3, Rule 4, Rule 5, and Rule 6

We assume that unit propagation deduces a contradiction and, therefore, the implication graph G contains both ℓ and $\bar{\ell}$ for some literal ℓ. Let S_ℓ be the set of all nodes from which there exists a path to ℓ, let $S_{\bar{\ell}}$ be the set of all nodes from which there exists a path to $\bar{\ell}$, and let $S{=}S_\ell \cup S_{\bar{\ell}}$. As a clause is associated with each node in G, we also use S, S_ℓ, and $S_{\bar{\ell}}$ to denote the set of clauses associated with the nodes in S, S_ℓ, and $S_{\bar{\ell}}$, respectively. Lemma 2 and Lemma 3 are used to detect the applicability of Rule 3, Rule 4, Rule 5, and Rule 6.

Lemma 2 *Rule 3 and Rule 4 are applicable if*

1. *in S_ℓ (resp. $S_{\bar{\ell}}$), there is one unit clause and all the other clauses are binary,*
2. *nodes in S_ℓ (resp. $S_{\bar{\ell}}$) form an implication chain starting at the unit clause and ending at ℓ (resp. $\bar{\ell}$),*
3. *$S_\ell \cap S_{\bar{\ell}}$ is empty.*

Proof 8 *Starting from the node corresponding to the unit clause in S_ℓ (resp. $S_{\bar{\ell}}$), and following in parallel the two implication chains, we have ϕ_1 in Rule 3 or Rule 4 by writing down the clause corresponding to each node.* ■

Example 6 *Let ϕ be the following CNF formula containing clauses c_1 to c_7: $\{c_1 : x_1,\ \ c_2 : \bar{x}_1 \vee x_2,\ \ c_3 : \bar{x}_2 \vee x_3,\ \ c_4 : \bar{x}_3 \vee x_4,\ \ c_5 : x_5,\ \ c_6 : \bar{x}_5 \vee x_6,\ \ c_7 : \bar{x}_6 \vee \bar{x}_4\}$. Unit propagation constructs the implication graph shown in Figure 5, which contains the complementary literals x_4 and $\bar{x}_4$.*

Rule 4 is applicable because $\ell{=}x_4$, $S_\ell{=}\{x_1(c_1),\ \ x_2(c_2),\ \ x_3(c_3),\ \ x_4(c_4)\}$, and $S_{\bar{\ell}}{=}\{x_5(c_5),\ \ x_6(c_6),\ \ \bar{x}_4(c_7)\}$. It is easy to verify that the three conditions of Lemma 2 are satisfied.

Remark: ϕ can be rewritten as $\{c_1 : x_1,\ \ c_2 : \bar{x}_1 \vee x_2,\ \ c_3 : \bar{x}_2 \vee x_3,\ \ c_4 : \bar{x}_3 \vee x_4,\ \ c_7 : \bar{x}_4 \vee \bar{x}_6,\ \ c_6 : x_6 \vee \bar{x}_5,\ \ c_5 : x_5\}$ to be compared with ϕ_1 in Rule 4.

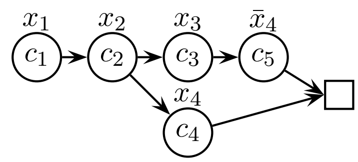

Figure 6: Example of implication graph

The application of Rule 3 and Rule 4 consists of replacing every binary clause c in S with a binary clause obtained by negating every literal of c, removing the two unit clauses of S from ϕ, and incrementing $\#emptyClauses(\phi)$ by 1.

Lemma 3 *Rule 5 and Rule 6 are applicable if*

1. *in* $S{=}S_\ell \cup S_{\bar{\ell}}$*, there is one unit clause and all the other clauses are binary; i.e., all nodes in* S*, except for the node corresponding to the unit clause, have exactly one incoming edge in* G*.*

2. $S_\ell \cap S_{\bar{\ell}}$ *is non-empty and contains* k $(k>0)$ *nodes forming an implication chain of the form* $\ell_1 \rightarrow \ell_2 \rightarrow \ \cdots \ \rightarrow \ell_k$*, where* ℓ_k *is the last node of the chain.*

3. $(S_\ell \cup S_{\bar{\ell}})$*-*$(S_\ell \cap S_{\bar{\ell}})$ *contains exactly three nodes :* ℓ*,* $\bar{\ell}$*, and a third one. Let* ℓ_{k+1} *be the third literal,*

   if $\ell_{k+1} \in S_\ell$*, then* G *contains the following implications*

   $$\ell_k \rightarrow \ell_{k+1} \rightarrow \ell$$
   $$\ell_k \rightarrow \bar{\ell}$$

   if $\ell_{k+1} \in S_{\bar{\ell}}$*, then* G *contains the following implications*

   $$\ell_k \rightarrow \ell$$
   $$\ell_k \rightarrow \ell_{k+1} \rightarrow \bar{\ell}$$

Proof 9 *Assume, without loss of generality, that* $\ell_{k+1} \in S_\ell$*; the case* $\ell_{k+1} \in S_{\bar{\ell}}$ *is symmetric. The implication chain formed by the nodes of* $S_\ell \cap S_{\bar{\ell}}$ *corresponds to the clauses* $\{\ell_1, \bar{\ell}_1 \vee \ell_2, \ldots, \bar{\ell}_{k-1} \vee \ell_k\}$*, which, together with the three clauses* $\{\bar{\ell}_k \vee \ell_{k+1}, \bar{\ell}_{k+1} \vee \ell, \bar{\ell}_k \vee \bar{\ell}\}$ *corresponding to* $\ell_k \rightarrow \ell_{k+1} \rightarrow \ell$ *and* $\ell_k \rightarrow \bar{\ell}$*, give* ϕ_1 *in Rule 5 or Rule 6.* ■

Example 7 *Let* ϕ *be the following CNF formula containing clauses* c_1 *to* c_5*:* $\{c_1 : x_1,\ \ c_2 : \bar{x}_1 \vee x_2,\ \ c_3 : \bar{x}_2 \vee x_3,\ \ c_4 : \bar{x}_2 \vee x_4,\ \ c_5 : \bar{x}_3 \vee \bar{x}_4\}$*. Unit propagation constructs the implication graph shown in Figure 6, which contains the complementary literals* x_4 *and* $\bar{x}_4$*.*

We have $S_{x_4}{=}\{x_1(c_1),\ x_2(c_2),\ x_4(c_4)\}$ *and* $S_{\bar{x}_4}{=}\{x_1(c_1),\ x_2(c_2),\ x_3(c_3),\ \bar{x}_4(c_5)\}$*. The nodes in* $S_{x_4} \cap S_{\bar{x}_4}$ *obviously form an implication chain:* $x_1 \rightarrow x_2$*.* $(S_{x_4} \cup S_{\bar{x}_4})$*-*$(S_{x_4} \cap S_{\bar{x}_4}){=}\{x_3(c_3),\ x_4(c_4),\ \bar{x}_4(c_5)\}$*.* G *contains* $x_2 \rightarrow x_3 \rightarrow \bar{x}_4$ *and* $x_2 \rightarrow x_4$*. Rule 6 is applicable.*

The application of Rule 5 and Rule 6 consists of removing the unit clause of $S_\ell \cup S_{\bar{\ell}}$ from ϕ, replacing each binary clause c in $S_\ell \cap S_{\bar{\ell}}$ with a binary clause obtained from c by negating the two literals of c, replacing the three binary clauses in $(S_\ell \cup S_{\bar{\ell}})$-$(S_\ell \cap S_{\bar{\ell}})$ with two ternary clauses, and incrementing $\#emptyClauses(\phi)$ by 1.

5.3 Complexity, Termination, and (In)Completeness of Rule Applications

In our branch and bound algorithm for Max-SAT, we combine the application of the inference rules and the computation of the underestimation of the lower bound. Given a CNF formula ϕ, function *underestimation* uses unit propagation to construct an implication graph G. Once G contains two nodes ℓ and $\bar{\ell}$ for some literal ℓ, G is analyzed to determine whether some inference rule is applicable. If some rule is applicable, it is applied and ϕ is transformed into an equivalent Max-SAT instance. Otherwise, all clauses contributing to the contradiction are removed from ϕ, and the underestimation is incremented by 1. This procedure is repeated until unit propagation cannot derive more contradictions. Finally, all removed clauses, except those removed or replaced due to inference rule applications, are reinserted into ϕ. The underestimation, together with the new ϕ, is returned.

It is well known that unit propagation can be implemented with a time complexity linear in the size of ϕ (see, e.g., Freeman, 1995). The complexity of determining the applicability of the inference rules using Lemma 2 and Lemma 3 is linear in the size of G, bounded by the number of literals in ϕ, if we assume that the graph is represented by doubly-linked lists. The application of an inference rule is obviously linear in the size of G. So, the whole time complexity of function *underestimation* with inference rule applications is $O(d * |\phi|)$, where d is the number of contradictions that function *underestimation* is able to detect using unit propagation. Observe that the factor d is needed because the application of the rules inserts new clauses in the place of the removed clauses.

Since every inference rule application reduces the size of ϕ, function *underestimation* with inference rule applications has linear space complexity, and it always terminates. Recall that new clauses added by the inference rules can be stored in the place of the old ones. The data structures for loading ϕ can be statically and efficiently maintained.

We have proved that the inference rules are sound. The following example shows that the application of the rules is not necessarily complete in our implementation, in the sense that not all possible applications of the inference rules are necessarily done.

Example 8 *Let ϕ=$\{x_1, x_3, x_4, \bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_4, \bar{x}_1 \vee \bar{x}_2, x_2\}$. Unit propagation may discover the inconsistent subset S=$\{x_1, x_3, x_4, \bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_4\}$. In this case, no inference rule is applicable to S. Then, the underestimation of the lower bound is incremented by 1, and ϕ becomes $\{\bar{x}_1 \vee \bar{x}_2, x_2\}$. Unit propagation cannot detect more contradictions in ϕ, and function underestimation stops after reinserting $\{x_1, x_3, x_4, \bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_4\}$ into ϕ. The value 1 is returned, together with the unchanged ϕ. Note that Rule 3 is applicable to the subset $\{x_1, \bar{x}_1 \vee \bar{x}_2, x_2\}$ of ϕ, but is not applied.*

Actually, function underestimation only applies Rule 3 if unit propagation detects the inconsistent subset $\{x_1, \bar{x}_1 \vee \bar{x}_2, x_2\}$ instead of $\{x_1, x_3, x_4, \bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_4\}$. The detection of an inconsistent subset depends on the ordering in which unit clauses are propagated in unit propagation. In this example, the inconsistent subset $\{x_1, \bar{x}_1 \vee \bar{x}_2, x_2\}$ is discovered if unit clause x_2 is propagated before x_3 and x_4. Further study is needed to define orderings for unit clauses that maximize the application of inference rules.

Observe that our algorithm is deterministic, and always computes the same lower bound if the order of clauses is not changed.

5.4 Inference Rules for Weighted Max-SAT

The inference rules presented in this paper can be naturally extended to weighted Max-SAT. In weighted Max-SAT, every clause is associated with a weight and the problem consists of finding a truth assignment for which the sum of the weights of unsatisfied clauses is minimum. For example, the weighted version of Rule 3 could be

Rule 7 *If* $\phi_1 = \{(l_1, w_1),\ (\bar{l}_1 \vee \bar{l}_2, w_2),\ (l_2, w_3)\} \cup \phi'$, *then* $\phi_2 = \{(\square, w),\ (l_1 \vee l_2, w),\ (l_1, w_1 - w),\ (\bar{l}_1 \vee \bar{l}_2, w_2 - w),\ (l_2, w_3 - w)\} \cup \phi'$ *is equivalent to* ϕ_1

where w_1, w_2 and w_3 are positive integers representing the clause weight, and w=min(w_1, w_2, w_3). Mandatory clauses, that have to be satisfied in any optimal solution, are specified with the weight ∞. Note that if $w \neq \infty$, ∞-w=∞ and if w=∞, no optimal solution can be found and the solver should backtrack. Clauses with weight 0 are removed. Observe that ϕ_1 can be rewritten as $\phi_{11} \cup \phi_{12}$, where $\phi_{11} = \{(l_1, w),\ (\bar{l}_1 \vee \bar{l}_2, w),\ (l_2, w)\}$, and $\phi_{12} = \{(l_1, w_1 - w),\ (\bar{l}_1 \vee \bar{l}_2, w_2 - w),\ (l_2, w_3 - w)\} \cup \phi'$. Then, the weighted inference rule is equivalent to the unweighted version applied w times to the (unweighted) clauses of ϕ_{11}.

Similarly, the weighted version of Rule 4 could be

Rule 8 *If* $\phi_1 = \{(l_1, w_1)\ (\bar{l}_1 \vee l_2, w_2),\ (\bar{l}_2 \vee l_3, w_3),\ \ldots,\ (\bar{l}_k \vee l_{k+1}, w_{k+1}),\ (\bar{l}_{k+1}, w_{k+2})\} \cup \phi'$, *then* $\phi_2 = \{(\square, w),\ (l_1 \vee \bar{l}_2, w),\ (l_2 \vee \bar{l}_3, w),\ \ldots,\ (l_k \vee \bar{l}_{k+1}, w),\ (l_1, w_1 - w),\ (\bar{l}_1 \vee l_2, w_2 - w),\ (\bar{l}_2 \vee l_3, w_3 - w),\ \ldots,\ (\bar{l}_k \vee l_{k+1}, w_{k+1} - w),\ (\bar{l}_{k+1}, w_{k+2} - w)\} \cup \phi'$ *is equivalent to* ϕ_1

where w=min(w_1, w_2, …, w_{k+2}). Observe that ϕ_1 can also be rewritten as $\phi_{11} \cup \phi_{12}$, with $\phi_{11} = \{(l_1, w)\ (\bar{l}_1 \vee l_2, w),\ (\bar{l}_2 \vee l_3, w),\ \ldots,\ (\bar{l}_k \vee l_{k+1}, w),\ (\bar{l}_{k+1}, w)\}$, The weighted version of Rule 4 is equivalent to the unweighted Rule 4 applied w times to the (unweighted) clauses of ϕ_{11}.

The current implementation of the inference rules can be naturally extended to weighted inference rules. If an inconsistent subformula is detected and a rule is applicable (clause weights are not considered in the detection of the inconsistent subformula and of the applicability of the rule, provided that clauses with weight 0 have been discarded), then ϕ_{11} and ϕ_{12} are separated after computing the minimal weight w of all clauses in the detected inconsistent subformula, and the rule is applied to ϕ_{11}. The derived clauses and clauses in ϕ_{12} can be used in subsequent reasoning.

6. MaxSatz: a New Max-SAT Solver

We have implemented a new Max-SAT solver, called MaxSatz, that incorporates the lower bound computation method based on unit propagation defined in Section 3, and applies the inference rules defined in Section 4. The name of MaxSatz comes from the fact that the implementation of our algorithm incorporates most of the technology that was developed for the SAT solver Satz (Li & Anbulagan, 1997b, 1997a).

MaxSatz incorporates the lower bound based on unit propagation, and applies Rule 1, Rule 2, Rule 3, Rule 4, Rule 5, and Rule 6. In addition, MaxSatz applies the following techniques:

- Pure literal rule: If a literal only appears with either positive polarity or negative polarity, we delete the clauses containing that literal.

- Empty-Unit clause rule (Alsinet et al., 2003a): Let $neg1(x)$ ($pos1(x)$) be the number of unit clauses in which x is negative (positive). If $\#emptyClauses(\phi) + neg1(x) \geq UB$, then we assign x to false. If $\#emptyClauses(\phi) + pos1(x) \geq UB$, then we assign x to true.

- Dominating Unit Clause (DUC) rule (Niedermeier & Rossmanith, 2000): If the number of clauses in which a literal x ($\bar{x}$) appears is not greater than $neg1(x)$ ($pos1(x)$), then we assign x to false (true).

- Variable selection heuristic: Let $neg2(x)$ ($pos2(x)$) be the number of binary clauses in which x is negative (positive), and let $neg3(x)$ ($pos3(x)$) be the number of clauses containing three or more literals in which x is negative (positive). We select the variable x such that $(neg1(x)+4*neg2(x)+neg3(x))$*$(pos1(x)+4*pos2(x)+pos3(x))$ is the largest. The fact that binary clauses are counted four times more than other clauses was determined empirically.

- Value selection heuristic: Let x be the selected branching variable. If $neg1(x) + 4 * neg2(x) + neg3(x) < pos1(x) + 4 * pos2(x) + pos3(x)$, set x to true. Otherwise set x to false. This heuristics was also determined empirically.

In this paper, in order to compare the inference rules defined, we have used three simplified versions of MaxSatz:

- MaxSat0: does not apply any inference rule defined in Section 4.

- MaxSat12: applies rules 1 and 2, but not rules 3, 4, 5 and 6.

- MaxSat1234: applies rules 1, 2, 3 and 4, but not rules 5 and 6.

Actually, MaxSatz corresponds to MaxSat123456 in our terminology. MaxSat12 corresponds to an improved version of the solver UP (Li et al., 2005), using a special ordering for propagating unit clauses in unit propagation. MaxSat12 maintains two queues during unit propagation: Q_1 and Q_2. When MaxSat12 starts the search for an inconsistent subformula via unit propagation, Q_1 contains all the unit clauses of the CNF formula under consideration (more recently derived unit clauses are at the end of Q_1), and Q_2 is empty. The unit clauses derived during the application of unit propagation are stored in Q_2, and unit propagation does not use any unit clause from Q_1 unless Q_2 is empty. Intuitively, this ordering prefers unit clauses which were non-unit clauses before starting the application of unit propagation. This way, the derived inconsistent subset contains, in general, less unit clauses. The unit clauses which have not been consumed will contribute to detect further inconsistent subsets. Our experimental results (Li, Manyà, & Planes, 2006) show that the search tree size of MaxSat12 is substantially smaller than that of UP, and MaxSat12 is substantially faster than UP. MaxSat0, Maxsat1234, and MaxSatz use the same ordering as MaxSat12 for propagating unit clauses in unit propagation.

The source code of MaxSat0, MaxSat12, MaxSat1234, and MaxSatz can be found at http://web.udl.es/usuaris/m4372594/jair-maxsatz-solvers.zip, and at http://www.laria.u-picardie.fr/˜cli/maxsatz.tar.gz.

7. Experimental Results

We report on the experimental investigation performed for unweighted Max-SAT in order to evaluate the inference rules defined in Section 4, and to compare MaxSatz with the best performing state-of-the-art solvers that were publicly available when this paper was submitted. The experiments were performed on a Linux Cluster with processors 2 GHz AMD Opteron with 1 Gb of RAM.

The structure of this section is as follows. We first describe the solvers and benchmarks that we have considered. Then, we present the experimental evaluation of the inference rules. Finally, we show the experimental comparison of MaxSatz with other solvers.

7.1 Solvers and Benchmarks

MaxSatz was compared with the following Max-SAT solvers:

- `BF`[2] (Borchers & Furman, 1999): a branch and bound Max-SAT solver which uses MOMS as dynamic variable selection heuristic and does not consider underestimations in the computation of the lower bound. It was developed by Borchers and Furman in 1999.

- `AGN`[3] (Alber et al., 2001): a branch and bound Max-2SAT solver. It was developed by Alber, Gramm and Niedermeier in 1998.

- `AMP`[4] (Alsinet et al., 2003b): a branch and bound Max-SAT solver based on BF that incorporates a lower bound of better quality and the Jeroslow-Wang variable selection heuristic (Jeroslow & Wang, 1990). It was developed by Alsinet, Manyà and Planes and presented at SAT-2003.

- `toolbar`[5] (de Givry et al., 2003; Larrosa & Heras, 2005): a Max-SAT solver whose inference was inspired in soft arc consistency properties implemented in weighted CSP solvers. It was developed by de Givry, Larrosa, Meseguer and Schiex and was first presented at CP-2003. We used version 2.2 with default parameters.

- `MaxSolver`[6] (Xing & Zhang, 2004): a branch and bound Max-SAT solver that applies a number of efficient inference rules. It was developed by Xing and Zhang and presented at CP-2004. We used the second release of this solver.

- `Lazy`[7] (Alsinet et al., 2005): a branch and bound Max-SAT solver with lazy data structures and a static variable selection heuristic. It was developed by Alsinet, Manyà and Planes and presented at SAT-2005.

2. Downloaded in October 2004 from http://infohost.nmt.edu/~borchers/satcodes.tar.gz
3. Downloaded in October 2005 from http://www-fs.informatik.uni-tuebingen.de/~gramm/
4. Available at http://web.udl.es/usuaris/m4372594/software.html
5. Downloaded in October 2005 from http://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/ToolBarIntro
6. Downloaded in October 2005 from http://cic.cs.wustl.edu/maxsolver/
7. Available at http://web.udl.es/usuaris/m4372594/software.html

- UP[8] (Li et al., 2005): a branch and bound Max-SAT solver with the lower bound computation method based on unit propagation (cf. Section 3). It was developed by Li, Manyà and Planes and presented at CP-2005.

We used as benchmarks randomly generated Max-2SAT instances and Max-3SAT instances, graph 3-coloring instances[9], as well as Max-Cut instances[10]. We also considered the unweighted Max-SAT benchmarks submitted to the Max-SAT Evaluation 2006, including Max-Cut, Max-Ones, Ramsey numbers, and random Max-2SAT and Max-3SAT instances.

We generated Max-2SAT instances and Max-3SAT instances using the generator `mwff.c` developed by Bart Selman, which allows for duplicated clauses. For Max-Cut, we first generated a random graph of m edges in which every edge is randomly selected among the set of all possible edges. If the graph is not connected, it is discarded. If the graph is connected, we used the encoding of Shen and Zhang (2005) to encode the Max-Cut instance into a CNF: we created, for each edge (x_i, x_j), exactly two binary clauses $(x_i \vee x_j)$ and $(\bar{x}_i \vee \bar{x}_j)$. If ϕ is the collection of such binary clauses, then the Max-Cut instance has a cut of weight k iff the Max-SAT instance has an assignment under which $m + k$ clauses are satisfied.

For graph 3-coloring, we first used Culberson's generator to generate a random k-colorable graph of type IID (independent random edge assignment, variability=0) with k vertices and a fixed edge density. We then used Culberson's converter to SAT with standard conversion and three colors to generate a Max-SAT instance: for each vertex x_i and for each color $j \in \{1, 2, 3\}$, a propositional variable x_{ij} is defined meaning that vertex i is colored with color j. For each vertex x_i, four clauses are added to encode that the vertex is colored with exactly one color: $x_{i1} \vee x_{i2} \vee x_{i3}$, $\bar{x}_{i1} \vee \bar{x}_{i2}$, $\bar{x}_{i1} \vee \bar{x}_{i3}$, and $\bar{x}_{i2} \vee \bar{x}_{i3}$; and, for each edge (x_i, x_j), three clauses are added to encode that vertex x_i and vertex x_j do not have the same color: $\bar{x}_{i1} \vee \bar{x}_{j1}$, $\bar{x}_{i2} \vee \bar{x}_{j2}$, and $\bar{x}_{i3} \vee \bar{x}_{j3}$.

In random Max-2SAT and Max-3SAT instances, clauses are entirely independent to each other and do not have structure. In the graph 3-coloring instances and Max-Cut instances used in this paper, clauses are not independent and have structure. For example, in a Max-Cut instance, every time we have a clause $x_i \vee x_j$, we also have the clause $\bar{x}_i \vee \bar{x}_j$; the satisfaction of these two clauses means that the corresponding edge is in the cut. In a graph 3-coloring instance, every time we have a ternary clause $x_{i1} \vee x_{i2} \vee x_{i3}$ encoding that vertex i is colored with at least a color, we also have three binary clauses $\bar{x}_{i1} \vee \bar{x}_{i2}$, $\bar{x}_{i1} \vee \bar{x}_{i3}$, and $\bar{x}_{i2} \vee \bar{x}_{i3}$ encoding that vertex i cannot be colored with two or more colors. Max-Cut instances only contain binary clauses, but graph 3-coloring instances contain a ternary clause for every vertex in the graph. While we can derive an optimal cut from an optimal assignment of a Max-SAT encoding of any Max-Cut instance, an optimal assignment of a Max-SAT encoding of a 3-coloring instance may assign more than one color to some vertices.

8. Available at http://web.udl.es/usuaris/m4372594/software.html
9. Given an undirected graph $G = (V, E)$, where $V = \{x_1, \ldots, x_n\}$ is the set of vertices and E is the set of edges, and a set of three colors, the graph 3-coloring problem is the problem of coloring every vertex with one of the three colors in such a way that, for each edge $(x_i, x_j) \in E$, vertex x_i and vertex x_j do not have the same color.
10. Given an undirected graph $G = (V, E)$, let w_{x_i,x_j} be the weight associated with each edge $(x_i, x_j) \in E$. The weighted 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 set weight $w_{x_i,x_j} = 1$ for all edges.

The Max-Cut and Ramsey numbers instances from the Max-SAT Evaluation 2006 contain different structures. For example, the underlying graphs in the Max-Cut instances have different origins such as fault diagnosis problems, coding theory problems, and graph clique problems. Max-2SAT and Max-3SAT instances from the evaluation do not contain duplicated clauses.

We computed an initial upper bound with a local search solver for each instance. We did not provide any parameter to any solver except the instance to be solved and the initial upper bound. In other words, we used the default values for all the parameters. The instances from the Max-SAT Evaluation 2006 were solved in the same conditions as in the evaluation; i.e., no initial upper bound was provided to the solvers, and the maximum time allowed to solve an instance was 30 minutes.

7.2 Evaluation of the Inference Rules

In the first experiment performed to evaluate the impact of the inference rules of Section 4, we solved sets of 100 random Max-2SAT instances with 50 and 100 variables; the number of clauses ranged from 400 to 4500 for 50 variables, and from 400 to 1000 for 100 variables. The results obtained are shown in Figure 7. Along the horizontal axis is the number of clauses, and along the vertical axis is the mean time (left plot), in seconds, needed to solve an instance of a set, and the mean number of branches of the proof tree (right plot). Notice that we use a log scale to represent both run-time and branches.

We observe that the rules are very powerful for Max-2SAT and the gain increases as the number of variables and the number of clauses increase. For 50 variables and 1000 clauses (the clause to variable ratio is 20), MaxSatz is 7.6 times faster than MaxSat1234; and for 100 variables and 1000 clauses (the clause to variable ratio is 10), MaxSatz is 9.2 times faster than MaxSat1234. The search tree of MaxSatz is also substantially smaller than that of MaxSat1234. Rule 5 and Rule 6 are more powerful than Rule 3 and Rule 4 for Max-2SAT. The intuitive explanation is that MaxSatz and MaxSat1234 detect many more inconsistent subsets of clauses containing one unit clause than subsets containing two unit clauses, so that Rule 5 and Rule 6 can be applied many more times than Rule 3 and Rule 4 in MaxSatz.

Recall that, on the one hand, every application of Rule 3 and Rule 4 consumes two unit clauses but only produces one empty clause, limiting unit propagation in detecting more conflicts in subsequent search. On the other hand, Rule 3 and Rule 4 add clauses which may contribute to detect further conflicts. Depending on the number of clauses (or more precisely, the clause to variable ratio) in a formula, these two factors have different importance. When there are relatively few clauses, unit propagation relatively does not easily derive a contradiction from a unit clause, and the binary clauses added by Rule 3 and Rule 4 are relatively important for deriving additional conflicts and improving the lower bound, which explains why the search tree of MaxSat1234 is smaller than the search tree of MaxSat12 for instances with 100 variables and less than 600 clauses. On the contrary, when there are many clauses, unit propagation easily derives a contradiction from a unit clause, so that the two unit clauses consumed by Rule 3 and Rule 4 would probably allow to derive two disjoint inconsistent subsets of clauses. In addition, the binary clauses added by Rule 3 and Rule 4 are relatively less important for deriving additional conflicts, considering the large number of clauses in the formula. In this case, the search tree of MaxSat1234 is larger than the search tree of MaxSat12. However, in both cases, MaxSat1234 is faster that MaxSat12, meaning that the incremental lower bound computation due to Rule 3 and Rule 4 is very effective, since the redetection of many conflicts is avoided thanks to Rule 3 and Rule 4.

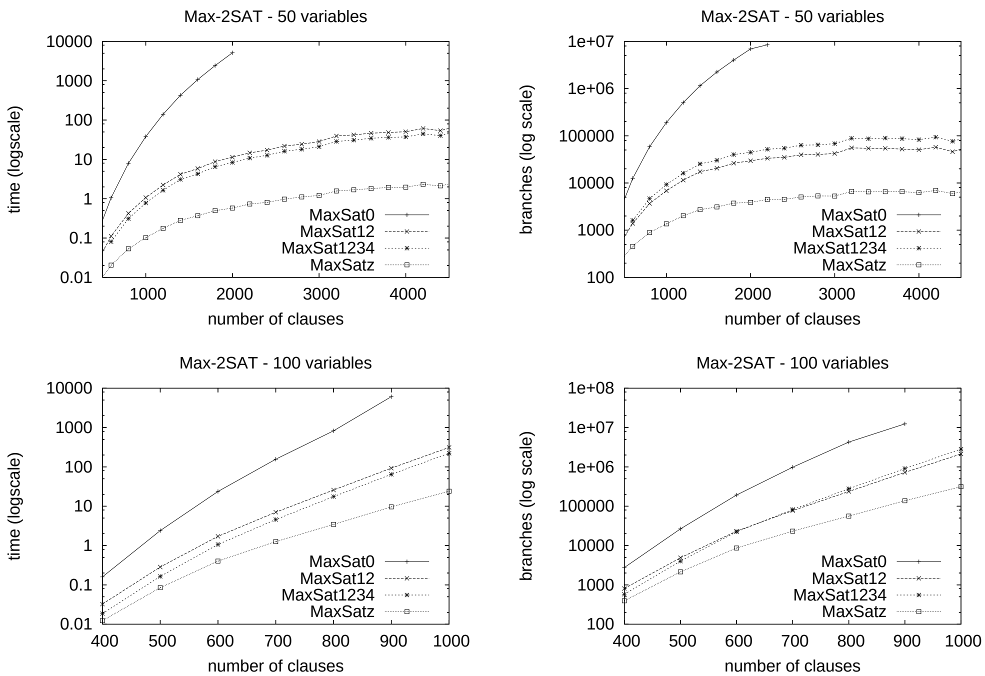

Figure 7: Comparison among MaxSat12, MaxSat1234 and MaxSatz on random Max-2SAT instances.

Rule 5 and Rule 6 do not limit unit propagation in detecting more conflicts, since their application produces one empty clause and consumes just one unit clause, which allows to derive at most one conflict in any case. The added ternary clauses allow to improve the lower bound, so that the search tree of MaxSatz is substantially smaller than the search tree of MaxSat1234. The incremental lower bound computation due to Rule 5 and Rule 6 also contributes to the time performance of MaxSatz. For example, while the search tree of MaxSatz for instances with 50 variables and 2000 clauses is about 11.5 times smaller than the search tree of MaxSat1234, MaxSatz is 14 times faster than MaxSat1234.

In the second experiment, we solved random Max-3SAT instances instead of random Max-2SAT instances. We solved instances with 50 and 70 variables; the number of clauses ranged from 400 to 1200 for 50 variables, and from 500 to 1000 for 70 variables. The results obtained are shown in Figure 8.

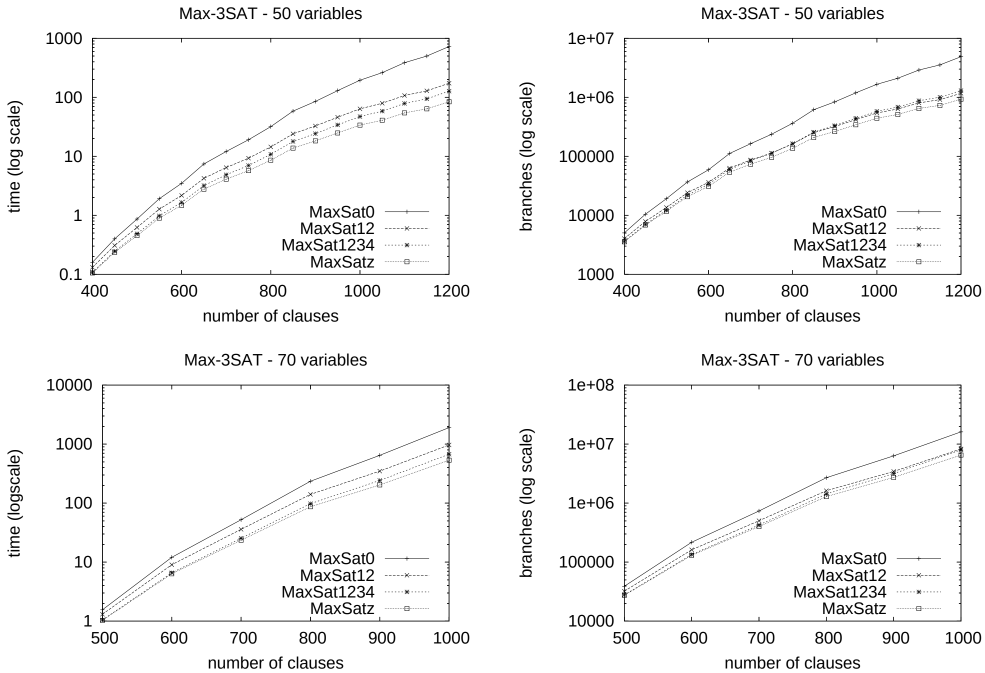

Figure 8: Comparison among MaxSat12, MaxSat1234 and MaxSatz on random Max-3SAT instances.

Although the rules do not involve ternary clauses, they are also powerful for Max-3SAT. Similarly to Max-2SAT, Rule 3 and Rule 4 slightly improve the lower bound when there are relatively few clauses, but do not improve the lower bound when the number of clauses increases. They improve the time performance thanks to the incremental lower bound computation they allowed. The gain increases as the number of clauses increases. For example, for problems with 70 variables, when the number of clauses is 600, MaxSat1234 is 36% faster than MaxSat12 and, when the number of clauses is 1000, the gain is 44%. Rule 5 and Rule 6 improve both the lower bound and the time performance of MaxSatz. The gain increases as the number of clauses increases.

In the third experiment we considered the Max-Cut problem for graphs with 50 vertices and a number of edges ranging from 200 to 800. Figure 9 shows the results of comparing the inference rules on Max-Cut instances. We observe that the rules allow us to solve the instances much faster. Similarly to random Max-2SAT, Rule 3 and Rule 4 do not improve the lower bound when there are many clauses, but improve the time performance due to the incremental lower bound computation they allowed. Rule 5 and Rule 6 are more powerful than Rule 3 and Rule 4 for these instances, which only contain binary clauses but have some structure. In addition, the reduction of the tree size due to Rule 5 and Rule 6 contributes to the time performance of MaxSatz more than the incrementality of the lower bound computation, as for random Max-2SAT. For example, the search tree of MaxSatz for instances with 800 edges is 40 times smaller than the search tree of MaxSat1234, and MaxSatz is 47 times faster.

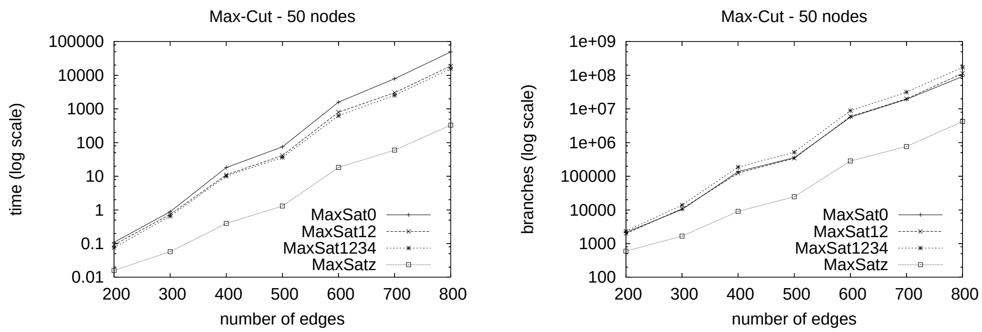

Figure 9: Experimental results for Max-Cut

In the fourth experiment we considered graph 3-coloring instances with 24 and 60 vertices, and with density of edges ranging from 20% to 90%. Figure 10 shows the results of comparing the inference rules on graph 3-coloring instances. We observe that Rule 1 and Rule 2 are not useful for these instances; the tree size of MaxSat0 and MaxSat12 is almost the same, and MaxSat12 is slower than MaxSat0. On the contrary, other rules are very useful for these instances, especially because they allow to reduce the search tree size by deriving better lower bounds.

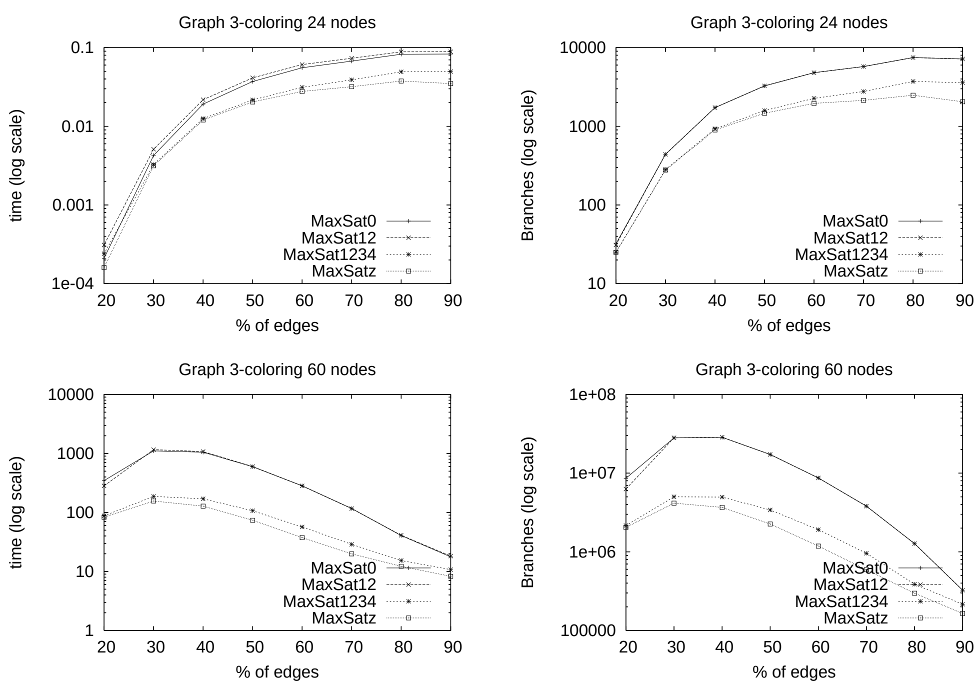

Figure 10: Experimental results for Graph 3-Coloring

Note that Rule 3 and Rule 4 have more impact than Rule 5 and Rule 6 on reducing the cost of solving the instances. This is probably due to the fact that two unit clauses are needed to detect a contradiction, so that Rule 3 and Rule 4 are applied many more times. Also note that the instances with 60 vertices become easier to solve when the density of the graph is high.

In the fifth experiment, we compared different inference rules on the benchmarks submitted to the Max-SAT Evaluation 2006. Solvers ran in the same conditions as in the evaluation. In Table 1, the first column is the name of the benchmark set, the second column is the number of instances in the set, and the rest of columns display the average time, in seconds, needed by each solver to solve an instance (the number of solved instances in brackets). The maximum time allowed to solve an instance was 30 minutes.

In is clear that MaxSat12 is better than MaxSat0, MaxSat1234 is better than MaxSat12, and MaxSatz is better than MaxSat1234. For example, MaxSatz solves three MAXCUT johnson instances within the time limit, while the other solvers only solve two instances. The average time for MaxSatz to solve one of these three instances is 44.46 seconds, the third instance needing more time to be solved than the other two instances.

Set Name	#Instances	`MaxSat0`	`MaxSat12`	`MaxSat1234`	`MaxSatz`
MAXCUT brock	12	471.01(10)	277.12(12)	225.11(12)	**14.01(12)**
MAXCUT c-fat	7	1.92 (5)	3.11 (5)	2.84 (5)	**0.07(5)**
MAXCUT hamming	6	39.42(2)	29.43(2)	29.48(2)	**171.30(3)**
MAXCUT johnson	4	14.91(2)	8.57 (2)	7.21 (2)	**44.46(3)**
MAXCUT keller	2	512.66(2)	213.64(2)	163.26(2)	**6.82(2)**
MAXCUT p hat	12	72.16(9)	286.09(12)	226.24(12)	**16.81(12)**
MAXCUT san	11	801.95(7)	305.75(7)	245.70(7)	**258.65(11)**
MAXCUT sanr	4	323.67(3)	134.74(3)	107.76(3)	**71.00(4)**
MAXCUT max cut	40	610.28(35)	481.48(40)	450.05(40)	**7.18(40)**
MAXCUT SPINGLASS	5	0.22 (2)	0.19 (2)	0.15 (2)	**0.14(2)**
MAXONE	45	**0.03 (45)**	**0.03 (45)**	**0.03 (45)**	**0.03(45)**
RAMSEY	48	8.93 (34)	8.42 (34)	7.80 (34)	**7.78(34)**
MAX2SAT 100VARS	50	95.01(50)	11.30(50)	8.14 (50)	**1.25(50)**
MAX2SAT 140VARS	50	153.28(49)	51.76(50)	34.14(50)	**6.94(50)**
MAX2SAT 60VARS	50	1.35 (50)	0.08 (50)	0.06 (50)	**0.02(50)**
MAX2SAT DISCARDED	180	126.98(162)	71.85(173)	68.97(175)	**22.72(180)**
MAX3SAT 40VARS	50	11.52(50)	3.33 (50)	2.52 (50)	**1.92(50)**
MAX3SAT 60VARS	50	167.17(50)	72.72(50)	52.14(50)	**40.27(50)**

Table 1: Evaluation of the rules with benchmarks from the MAX-SAT Evaluation 2006.

7.3 Comparison of MaxSatz with Other Solvers

In the first experiment, that we performed to compare MaxSatz with other state-of-the-art Max-SAT solvers, we solved sets of 100 random Max-2SAT instances with 50, 100 and 150 variables; the number of clauses ranged from 400 to 4500 for 50 variables, from 400 to 1000 for 100 variables, and from 300 to 650 for 150 variables. The results of solving such instances with BF, AGN, AMP, Lazy, toolbar, MaxSolver, UP and MaxSatz are shown in Figure 11. Along the horizontal axis is the number of clauses, and along the vertical axis is the mean time, in seconds, needed to solve an instance of a set. When a solver needed too much time to solve the instances at a point, it was stopped and the corresponding point is not shown in the figure. That is why for 50 variable instances, BF has only one point in the figure (for 400 clauses); and for 100 variable instances, BF and AMP also have only one point in the figure (for 400 clauses). The version of MaxSolver we used limits the number of clauses to 1000 in the instances to be solved. We ran it for instances up to 1000 clauses.

We see dramatic differences on performance between MaxSatz and the rest of solvers in Figure 11. For the hardest instances, MaxSatz is up to two orders of magnitude faster than the second best performing solvers (UP). For those instances, MaxSatz needs 1 second to solve an instance while solvers like MaxSolver and toolbar are not able to solve these instances after 10,000 seconds.

In the second experiment, we solved random Max-3SAT instances instead of random Max-2SAT instances. The results obtained are shown in Figure 12. We did not consider AGN because it can only solve Max-2SAT instances. We solved instances with 50, 70 and 100 variables; the number of clauses ranged from 500 to 1200 for 50 variables, from 500 to 1000 for 70 variables, and from 450 to 800 for 100 variables. For 70 variables, AMP has only one point in the figure (for 500 clauses) and BF is too slow. For 100 variables, we compared only the two best solvers. Once again, we observe dramatic differences on the performance profile of MaxSatz and the rest of solvers. Particularly remarkable are the differences between MaxSatz and toolbar (the second best performing solver on Max-3SAT), where we see that MaxSatz is up to 1,000 times faster than toolbar on the hardest instances.

In the third experiment, we considered the Max-Cut problem of graphs with 50 vertices and a number of edges ranging from 200 to 700. Figure 13 shows the results obtained. BF has only one point in the figure (for 200 edges). MaxSolver solved instances up to 500 edges (1000 clauses). We observe that MaxSatz is superior to the rest of solvers.

In the fourth experiment, we considered the 3-coloring problem of graphs with 24 and 60 vertices, and a density of edges ranging from 20% to 90%. AGN was not considered because it can only solve Max-2SAT instances. For 60 vertices, we only compared the three best solvers, of which MaxSolver is a different version not limiting the number of clauses of the instance to be solved. Figure 14 shows the comparative results for different solvers. MaxSatz is the best performing solver, and UP and MaxSolver are substantially better than the rest of solvers.

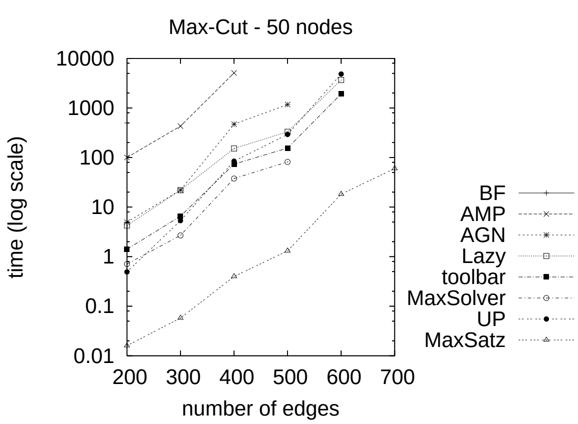

Figure 13: Experimental results for Max-Cut

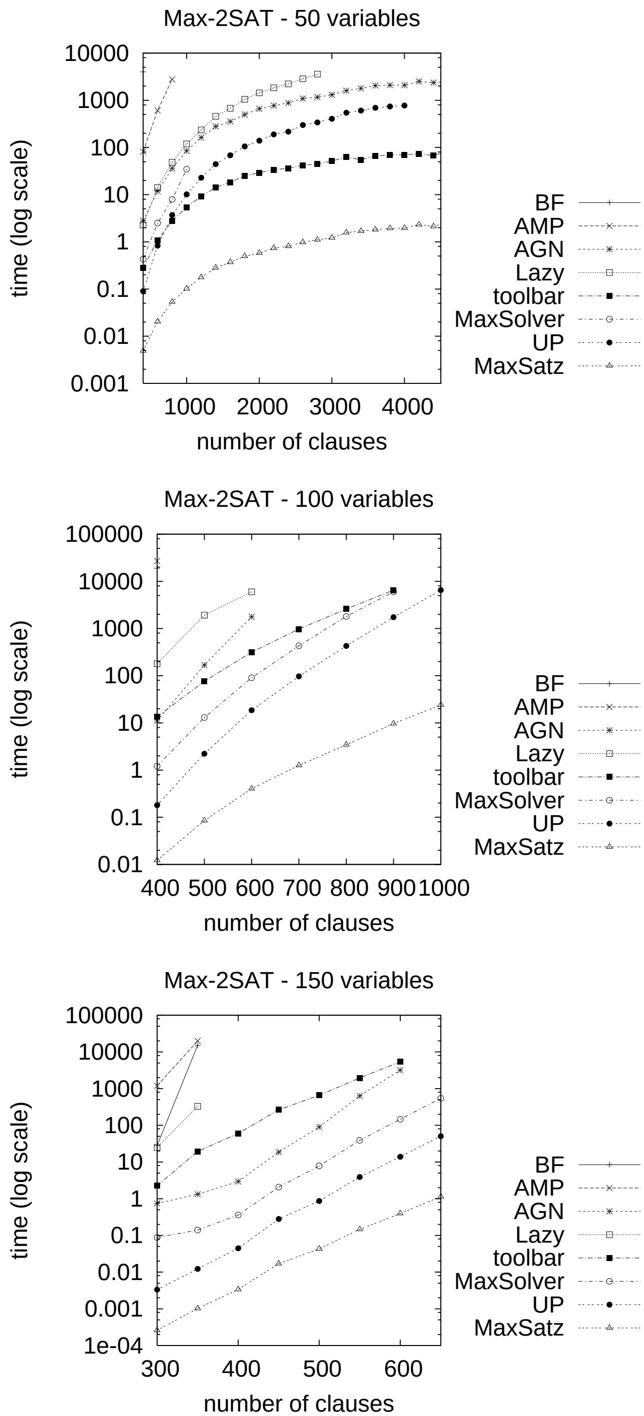

Figure 11: Experimental results for 50-variable, 100-variable and 150-variable random Max-2SAT instances.

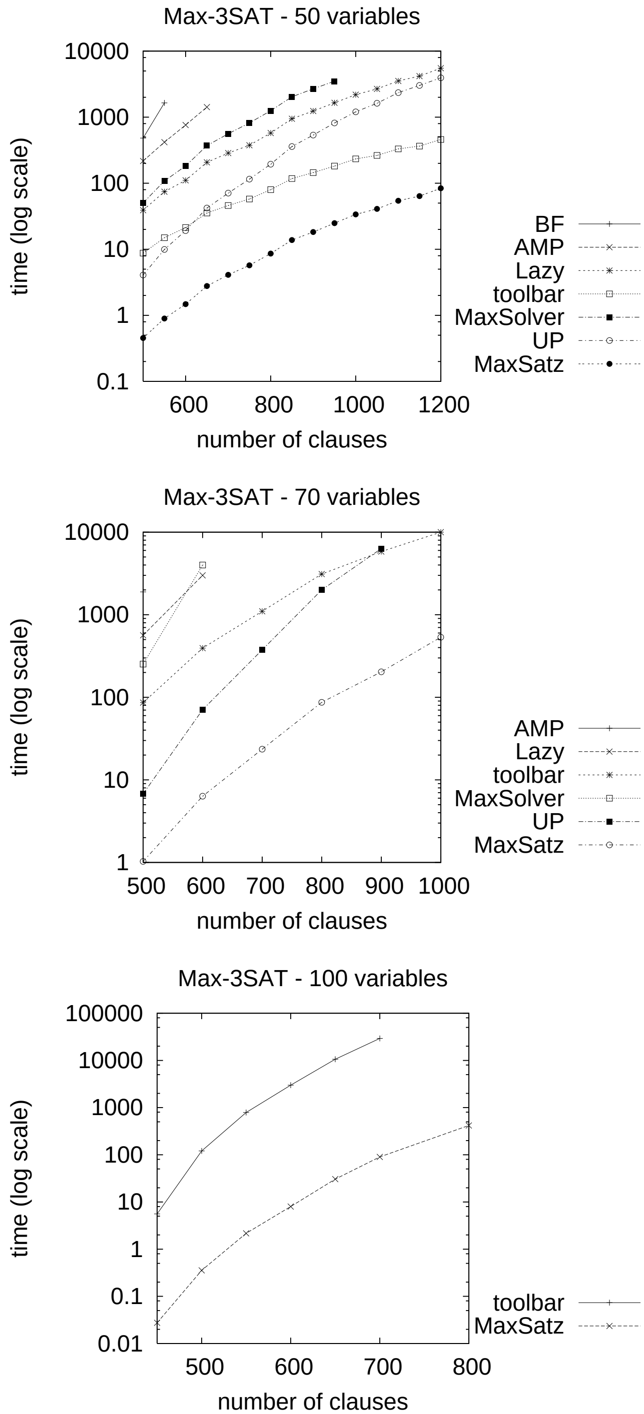

Figure 12: Experimental results for 50-variable, 70-variable and 100-variable random Max-3SAT instances.

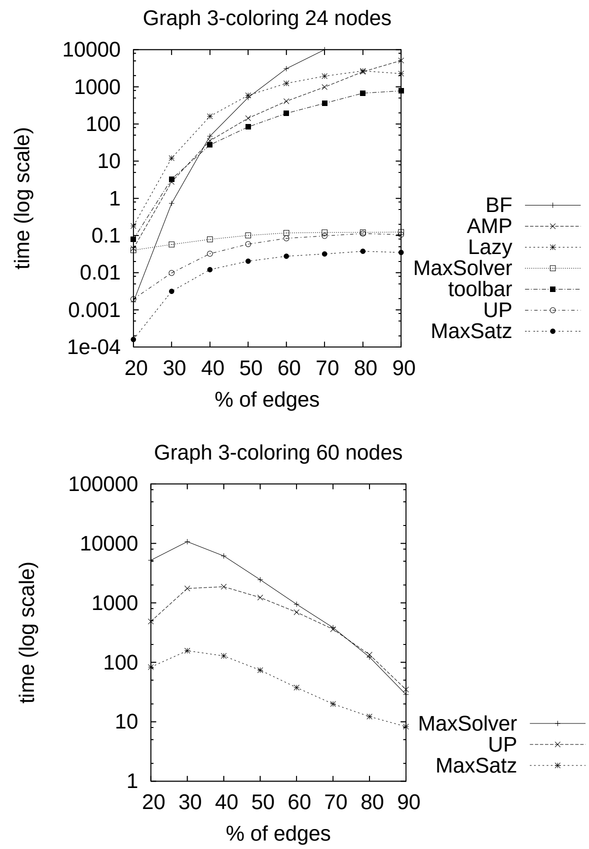

Figure 14: Experimental results for Graph 3-Coloring

In the fifth experiment, we compared the Max-SAT solvers on the benchmarks submitted to the Max-SAT Evaluation 2006. Solvers ran in the same conditions as in the evaluation. In Table 2, the first column is the name of the benchmark set, the second column is the number of instances of the set, and the rest of columns display the average time, in seconds, needed by each solver to solve an instance within a time limit of 30 minutes (the number of instances solved within the time limit in brackets). A dash means that the corresponding solver cannot solve the set of instances. It is clear that MaxSatz is the best performing solver for all the sets.

8. Related Work

The simplest method to compute a lower bound consists of just counting the number of clauses unsatisfied by the current partial assignment (Borchers & Furman, 1999). One step forward is to incorporate an underestimation of the number of clauses that will become unsatisfied if the current partial assignment is extended to a complete assignment. The most basic method was defined by Wallace and Freuder (1996):

Set Name	#Instances	BF	AMP	AGN	toolbar	Lazy	MaxSolver	UP	MaxSatz
MAXCUT brock	12	(0)	545.81(1)	856.65(8)	470.23(12)	159.28 (12)	380.09(2)	629.85(9)	**14.01(12)**
MAXCUT c-fat	7	6.06 (1)	1.95 (3)	32.70(5)	42.84(5)	13.23 (4)	41.58(3)	7.19 (5)	**0.07(5)**
MAXCUT hamming	6	(0)	636.04(1)	159.99(1)	145.84(2)	265.35 (2)	(0)	294.89(2)	**171.30(3)**
MAXCUT johnson	4	(0)	394.17(2)	92.90(2)	11.07(2)	13.50 (2)	1.34 (1)	29.42(2)	**44.46(3)**
MAXCUT keller	2	(0)	197.15(1)	39.36(1)	255.39(2)	348.75 (2)	(0)	615.54(2)	**6.82 (2)**
MAXCUT DIMACS p hat	12	605.44(2)	107.79(8)	16.11(8)	235.60(11)	259.33 (10)	14.00(8)	140.23(9)	**16.81(12)**
MAXCUT san	11	(0)	563.19(1)	72.35(2)	568.09(7)	956.54 (5)	283.34(2)	812.47(5)	**258.65(11)**
MAXCUT sanr	4	(0)	428.18(1)	909.32(3)	234.89(3)	410.53 (3)	138.32(1)	538.10(3)	**71.00(4)**
MAXCUT max cut	40	(0)	(0)	1742.79(3)	736.34(18)	1027.21 (7)	(0)	623.03(13)	**7.18(40)**
MAXCUT SPINGLASS	5	0.21 (1)	0.13 (1)	12.70(2)	5.72 (2)	0.05 (1)	570.68(2)	0.86 (2)	**0.14(2)**
MAXONE	45	0.02 (21)	**0.03 (45)**	-	35.35(44)	278.58 (26)	0.06 (45)	0.31 (45)	**0.03 (45)**
RAMSEY ram k	48	8.53 (30)	38.44(30)	-	4.14(27)	10.48 (25)	0.20 (20)	19.65(25)	**7.78 (34)**
MAX2SAT 100VARS	50	0.14 (10)	143.23(11)	185.69(30)	244.05(34)	273.44 (22)	532.47(16)	192.34(48)	**1.25 (50)**
MAX2SAT 140VARS	50	0.08 (10)	91.93(12)	126.34(28)	262.30(26)	217.12 (17)	168.42(18)	75.57(39)	**6.94 (50)**
MAX2SAT 60VARS	50	1.92 (3)	514.02(44)	6.34 (50)	2.01 (50)	26.44 (50)	81.82(50)	0.94 (50)	**0.02 (50)**
MAX2SAT DISCARDED	180	357.65(28)	439.54(76)	99.70(108)	178.23(116)	85.08 (87)	308.58(73)	166.29(149)	**22.72(180)**
MAX3SAT 40VARS	50	170.49(22)	202.18(50)	-	10.19 (50)	69.72 (50)	66.34(49)	60.50(50)	**1.92(50)**
MAX3SAT 60VARS	50	4.07 (16)	168.00(25)	-	361.95(43)	242.40 (28)	139.03(22)	166.76(37)	**40.27(50)**

Table 2: Experimental results for benchmarks from the MAX-SAT Evaluation 2006.

$$\texttt{LB}(\phi) = \#emptyClauses(\phi) + \sum_{x \text{ occurs in } \phi} min(ic(x), ic(\bar{x}))$$

where ϕ is the CNF formula associated with the current partial assignment, and $ic(x)$ ($ic(\bar{x})$) —inconsistency count of x ($\bar{x}$)— is the number of unit clauses of ϕ that contain $\bar{x}$ (x).

The underestimation of the lower bound can be improved by applying to binary clauses the Directional Arc Consistency (DAC) count defined by Wallace (1995) for Max-CSP. The DAC count of a value of the variable x in ϕ is the number of variables which are inconsistent with that value of x. For example, if ϕ contains clauses $x \vee y$, $x \vee \bar{y}$, and $\bar{x} \vee y$, the value 0 of x is inconsistent with y. Note that value 0 of y is also inconsistent with x. These two inconsistencies are not disjoint and cannot be summed. Wallace defined a direction from x to y, so that only the inconsistency for value 0 of x is counted. After defining a direction between every pair of variables sharing a constraint, one computes the DAC count for all values of x by checking all variables to which a direction from x is defined. The underestimation considering the DAC count of Wallace is as follows:

$$\sum_{x \text{ occurs in } \phi} (min(ic(x), ic(\bar{x})) + min(dac(x), dac(\bar{x}))$$

where $dac(x)$ ($dac(\bar{x})$) is the DAC count of the value 1(0) of x. Wallace statically defined all directions, so that $dac(x)$ and $dac(\bar{x})$ can be computed in a preprocessing step for every x and do not need to be recomputed during search. This is improved by Larrosa, Meseguer and Schiex (1999) by introducing reversible DAC, which searches for better directions to obtain a better lower bound at every node of the search tree. An improvement of DAC counts is the additional incorporation of inconsistencies contributed by disjoint subsets of variables, based on particular variable partitions (Larrosa & Meseguer, 2002).

Inconsistent and DAC counts deal with unit and binary clauses. Lower bounds dealing with longer clauses include star rule (Shen & Zhang, 2004; Alsinet et al., 2004) and UP (Li et al., 2005).

In the star rule, the underestimation of the lower bound is the number of disjoint inconsistent subformulas of the form $\{l_1, \ldots, l_k, \bar{l}_1 \vee \cdots \vee \bar{l}_k\}$. The star rule, when $k = 1$, is equivalent to the inconsistency counts of Wallace and Freuder.

UP subsumes the inconsistent count method based on unit clauses and the star rule. Its effectiveness for producing a good lower bound can be illustrated with the following example: let ϕ be a CNF formula containing the clauses $x_1, \bar{x}_1 \vee x_2, \bar{x}_1 \vee x_3, \bar{x}_2 \vee \bar{x}_3 \vee x_4, x_5, \bar{x}_5 \vee x_6, \bar{x}_5 \vee x_7, \bar{x}_6 \vee \bar{x}_7 \vee \bar{x}_4$. UP easily detects that inconsistent subset with 8 clauses and 7 variables, in time linear in the size of the formula. Note that this subset is not detected by any of the lower bounds described above, except for the variable partition based approach of Larrosa and Meseguer (2002) in the case that the 7 variables are in the same partition.

We mention two more lower bound computation methods. One is called LB4 and was defined by Shen and Zhang (2004). It is similar to UP but restricted to Max-2SAT instances and using a static variable ordering. Another is based on linear programming and was defined by Xing and Zhang (2005).

Regin et al. (2001) suggested to use arc consistency, instead of unit propagation, to detect disjoint inconsistent subsets of constraints in weighted constraint networks. However, to the best of our knowledge, this idea has not been incorporated in any lower bound computation method implemented by the Constraint Programming community.

A good lower bound computation method has a dramatic impact on the performance of a Max-SAT solver. Another approach to speed up a Max-SAT solver consists of applying inference rules to transform a Max-SAT instance ϕ into an equivalent but simpler Max-SAT instance ϕ'. Inference rules that have proven to be useful in practice include: (i) the pure literal rule (Alsinet et al., 2003b; Xing & Zhang, 2004; Li et al., 2005; Zhang et al., 2003); (ii) the dominating unit clause rule, first proposed by Niedermeier and Rossmanith (2000), and later applied in several solvers (Alsinet et al., 2004; Xing & Zhang, 2004; Li et al., 2005); (iii) the almost common clause rule, first proposed by Bansal and Raman (1999) and restated as Rule 1 in this paper. The rule was extended to weighted Max-SAT byAlsinet et al. (2004); was called neighborhood resolution by Larrosa and Heras (2005); and used as a preprocessing technique by Alsinet et al. (2004), Shen and Zhang (2005), and Li et al. (2005); (iv) the complementary unit clause rule (Niedermeier & Rossmanith, 2000), restated as Rule 2 in this paper; and (v) the coefficient-determining unit propagation rule (Xing & Zhang, 2005) based on integer programming.

The inference rules presented in this paper simplify a Max-SAT formula ϕ and allow to improve the lower bound computation, since they all transform a Max-SAT formula ϕ into a simpler and equivalent formula containing more empty clauses. Their soundness (i.e., the fact that they transform a formula into an equivalent one) can be proved in several ways, including (i) checking all possible variable assignments, (ii) using integer programming as done in Section 4, and (iii) using soft local consistency techniques defined for Weighted Constraint Networks (WCN); Max-SAT can be defined as a subcase of WCN where variables are Boolean and only unit costs are used.

Soft local consistency techniques for WCN are based on two basic equivalence preserving transformations called *projection* and *extension* (Schiex, 2000; Cooper & Schiex, 2004). Given a Max-SAT instance, projection replaces two binary clauses $x \vee y$ and $x \vee \bar{y}$ with the unit clause x, which is Rule 1 for k=2. Extension is the inverse operation of projection and replaces a unit clause x with two binary clauses $x \vee y$ and $x \vee \bar{y}$ for a selected variable y. If the projection operation is rather straightforward for a SAT or Max-SAT instance, the extension operation is very ingenious. To see this, note that Rule 3 can be proved or applied with an extension followed by a projection:

$$\begin{aligned} l_1,\ \bar{l}_1 \vee \bar{l}_2,\ l_2 &= l_1 \vee l_2, l_1 \vee \bar{l}_2,\ \bar{l}_1 \vee \bar{l}_2,\ l_2 \\ &= l_1 \vee l_2, \bar{l}_2, l_2 \\ &= l_1 \vee l_2, \square \end{aligned}$$

Lemma 1 can also be proved using an extension followed by a projection:

$$\begin{aligned} l_1,\ \bar{l}_1 \vee l_2 &= l_1 \vee \bar{l}_2,\ l_1 \vee l_2,\ \bar{l}_1 \vee l_2 \\ &= l_1 \vee \bar{l}_2, l_2 \end{aligned}$$

The extension operation cannot be used in an unguided way because it may cancel a previous projection. One way to guide its use is to define an ordering between variables to enforce directional arc consistency (Cooper, 2003; Cooper & Schiex, 2004). Directional arc consistency allows to concentrate weights on the same variables by shifting weights from earlier variables to later ones in a given ordering. For example if $x_1 < x_2$ in a given variable ordering, one can extend unit clause x_1 to $x_1 \vee x_2, x_1 \vee \bar{x}_2$, but cannot extend unit clause x_2 to $x_1 \vee x_2,\ \bar{x}_1 \vee x_2$, allowing unit clauses to be concentrated on variable x_2. Nevertheless, how to define the variable ordering to efficiently exploit as much as possible the power of soft arc consistency techniques in the lower bound computation remains an open problem.

The approach with inference rules for Max-SAT presented in this paper does not need any predefined ordering among variables, since rule applications combining several projection and extension operations are entirely guided by unit propagation.

The projection and extension operations can be extended to constraints involving more than two variables to achieve high-order consistency in WCN (Cooper, 2005). For a Max-SAT instance, the extended projection and extension operations can be stated using Rule 1 for k>2. For the two formulas ϕ_1 and ϕ_2 in Rule 1, replacing ϕ_1 with ϕ_2 is a projection and ϕ_2 with ϕ_1 is an extension. Given a unit clause x and three variables x, y, z, the extension of the unit clause x to the set of three variables can be done as follows : replacing x by $x \vee y$ and $x \vee \bar{y}$, and then $x \vee y$ and $x \vee \bar{y}$ by $x \vee y \vee z$, $x \vee y \vee \bar{z}$, $x \vee \bar{y} \vee z$ and $x \vee \bar{y} \vee \bar{z}$.

Rule 5 can be proved or applied by extending the four clauses of ϕ_1 to ternary clauses on the three variables of l_1, l_2 and l_3, and then applying the projection operation to obtain ϕ_2.

Larrosa et al. (2007), based on a logical approach, independently and in parallel with our work, defined and implemented a chain resolution rule and a cycle resolution rule for weighted Max-SAT. These two rules are extensions of Rules 2-RES and 3-RES presented, also independently and in parallel with our work (Heras & Larrosa, 2006).

The chain resolution could be stated as follows:

$$\left\{\begin{array}{l}(l_1, u_1),\\ (\bar{l}_i \vee l_{i+1}, u_{i+1})_{1\leq i<k},\\ (\bar{l}_k, u_{k+1})\end{array}\right\} = \left\{\begin{array}{l}(l_i, m_i - m_{i+1})_{1\leq i\leq k},\\ (\bar{l}_i \vee l_{i+1}, u_{i+1} - m_{i+1})_{1\leq i<k},\\ (l_i \vee \bar{l}_{i+1}, m_{i+1})_{1\leq i<k},\\ (\bar{l}_k, u_{k+1} - m_{k+1}),\\ (\Box, m_{k+1})\end{array}\right\}$$

where, for $1{\leq}i{\leq}k{+}1$, u_i is the weight of the corresponding clause, $m_i{=}\min(u_1, u_2, \ldots, u_i)$, and all variables in the literals are different. The weight of a mandatory clause is denoted by $\top$, and the subtraction $-$ is extended so that $\top - u_i{=}\top$. The chain resolution rule is equivalent to Rule 4 if it is applied to unweighted Max-SAT. The main difference between the chain resolution rule and the weighted version of Rule 4 presented in Section 5.4 is that the chain resolution shifts a part of the weight from unit clause $(l_1, m_1 - m_{k+1})$, that is derived in the weighted version of Rule 4, to create unit clauses $(l_i, m_i - m_{i+1})_{1<i\leq k}$, $(l_1, m_1 - m_{k+1})$ itself becoming $(l_1, m_1 - m_2)$.

The cycle resolution rule could be stated as follows:

$$\left\{ \begin{array}{l} (\bar{l}_i \vee l_{i+1}, u_i)_{1 \leq i < k}, \\ (\bar{l}_1 \vee \bar{l}_k, u_k) \end{array} \right\} = \left\{ \begin{array}{l} (\bar{l}_1 \vee l_i, m_{i-1} - m_i)_{2 \leq i \leq k}, \\ (\bar{l}_i \vee l_{i+1}, u_i - m_i)_{2 \leq i < k}, \\ (\bar{l}_1 \vee l_i \vee \bar{l}_{i+1}, m_i)_{2 \leq i < k}, \\ (l_1 \vee \bar{l}_i \vee l_{i+1}, m_i)_{2 \leq i < k}, \\ (\bar{l}_1 \vee \bar{l}_k, u_k - m_k), \\ (\bar{l}_1, m_k) \end{array} \right\}$$

When a subset of binary clauses have a cyclic structure, the cycle resolution rule allows to derive a unit clause. Note that the detection of the cyclic structure appears rather time-consuming if it is applied at every node of a search tree and that $2\times(k$-$2)$ new ternary clauses have to be inserted. So, Larrosa et al. apply the cycle resolution rule in practice only for the case k=3, which is similar to Rule 5, when applied to unweighted Max-SAT. The cycle resolution rule applied to unweighted Max-SAT for k=3 can replace Rule 5 and Rule 6 in MaxSatz, but with the following differences compared with Rule 5 and Rule 6:

- the application of Rule 5 and Rule 6 is entirely based on inconsistent subformulas detected by unit propagation. The detection of the applicability of Rule 5 and Rule 6 is easy and has very low overhead, since the inconsistent subformulas are always detected in MaxSatz to compute the lower bound (with or without Rule 5 and Rule 6). Every application of Rule 5 or Rule 6 allows to increment the lower bound by 1.

- the cycle resolution rule needs an extra detection of the cyclic structure, but allows to derive a unit clause from the cyclic structure. The derived unit clause could then be used in a unit propagation, and possibly could allow to detect an inconsistent subformula and increase the lower bound by 1.

It would be an interesting future research topic to implement the cycle resolution rule in MaxSat1234 (i.e., MaxSatz without Rule 5 and Rule 6) to evaluate the overhead of detecting the cyclic structure and the usefulness of the unit clauses and the ternary clauses derived using the cycle resolution rule, and to compare the implemented solver with MaxSatz. It would be also interesting to compare the chain resolution rule and the cycle resolution rule with the weighted inference rules presented in Section 5.4.

A more general Max-SAT resolution rule, where the conclusions were not in clausal form, was defined by Larrosa and Heras (2005). Independently, Bonet et al. (2006, 2007) and Heras and Larrosa (2006) defined a version of the rule with the conclusions in clausal form. Bonet et al. (2006, 2007) also proved that this rule is complete for Max-SAT. Recently, Ansótegui et al. (2007b, 2007a) have shown that Max-SAT resolution for many-valued CNF formulas provides a logical framework for the global and local consistency properties defined for WCN.

9. Conclusions and Future Work

One of the main drawbacks of state-of-the-art Max-SAT solvers is the lack of suitable inference techniques that allow to detect as much contradictions as possible and to simplify the formula at each node of the search tree. Existing approaches put the emphasis on computing underestimations of good quality, but the problem with underestimations is that the same contradictions are computed once and again. Furthermore, it turns out that UP, one of the currently best performing underestimations consisting of detecting disjoint inconsistent subsets of clauses in a CNF formula via unit propagation, is still too conservative. To make the computation of lowers more incremental and to improve the underestimation, we have defined a number of original inference rules for Max-SAT that, based on derived contradictions by unit propagation, transform a Max-SAT instance into an equivalent Max-SAT instance which is easier to solve. The rules were carefully selected taking into account that they should be applied efficiently. Since all these rules are based on contradiction detection, they should be particularly useful for hard Max-SAT instances containing many contradictions.

With the aim of finding out how powerful the inference rules are in practice, we have developed a new Max-SAT solver, called MaxSatz, which incorporates those rules, and performed an experimental investigation. The results of comparing MaxSatz with inference rules and MaxSatz without inference rules provide empirical evidence of the usefulness of these rules in making lower bound computation more incremental and in improving the quality of lower bounds. The results of comparing MaxSatz with a large selection of the solvers available at the time of submitting this paper provide empirical evidence that MaxSatz, at least for the instances solved, is faster than the other solvers. We observed gains of several orders of magnitude for the hardest instances. Interestingly, for the benchmarks used, the second best solver was generally different: UP for Max-2SAT, toolbar for Max-3SAT, MaxSolver for Max-Cut, and MaxSolver and UP for graph 3-coloring. So, MaxSatz is more robust than the rest of solvers. It is worth mentioning that MaxSatz, enhanced with a lower bound based on failed literal detection (Li et al., 2006), was the best performing solver for unweighted Max-SAT instances in the Max-SAT Evaluation 2006. The second and third best performing solvers were, respectively, improved versions of toolbar and Lazy[11].

As future work we plan to study the orderings of unit clauses in unit propagation to maximize the application of inference rules, and to define new inference rules for ternary clauses. We are extending the results of this paper to weighted Max-SAT, which is more suitable for modeling problems such as maximum clique, set covering and combinatorial auctions, as well as constraint satisfaction problems such as hard instances of Model RB (Xu, Boussemart, Hemery, & Lecoutre, 2005; Xu & Li, 2006). We are also adapting the results of this paper to the partial Max-SAT solvers developed by Argelich and Manyà (2005, 2006, 2007).

Acknowledgments

Research partially supported by projects TIN2004-07933-C03-03 and TIN2006-15662-C02-02 funded by the *Ministerio de Educación y Ciencia*. The first author was partially supported by National 973 Program of China under Grant No. 2005CB321900. The second author was supported by a grant *Ramón y Cajal*. Finally, we would like to thank the referees for their detailed comments and suggestions.

11. See http://www.iiia.csic.es/~maxsat06 for details. Note that the results of the Max-SAT Evaluation 2006 can be compared with the results of this paper because they were obtained with the same cluster under the same conditions.

References

Alber, J., Gramm, J., & Niedermeier, R. (2001). Faster exact algorithms for hard problems: A parameterized point of view. *Discrete Mathematics*, *229*(1–3), 3–27.

Alsinet, T., Manyà, F., & Planes, J. (2003a). Improved branch and bound algorithms for Max-2-SAT and weighted Max-2-SAT. In *Proceedings of the Catalonian Conference on Artificial Intelligence (CCIA-03), P. Mallorca, Spain*, Vol. 100 of *Frontiers in Artificial Intelligence and Applications*, pp. 435–442. IOS Press.

Alsinet, T., Manyà, F., & Planes, J. (2003b). Improved branch and bound algorithms for Max-SAT. In *Proceedings of the 6th International Conference on the Theory and Applications of Satisfiability Testing (SAT-03), Portofino, Italy*, pp. 408–415.

Alsinet, T., Manyà, F., & Planes, J. (2004). A Max-SAT solver with lazy data structures. In *Proceedings of the 9th Ibero-American Conference on Artificial Intelligence (IBERAMIA-04), Puebla, México*, LNCS 3315, pp. 334–342. Springer.

Alsinet, T., Manyà, F., & Planes, J. (2005). Improved exact solver for weighted Max-SAT. In *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT-05), St. Andrews, Scotland*, LNCS 3569, pp. 371–377. Springer.

Ansótegui, C., Bonet, M. L., Levy, J., & Manyà, F. (2007a). Inference rules for high-order consistency in weighted CSP. In *Proceedings of the 22nd National Conference on Artificial Intelligence (AAAI-07), Vancouver, Canada*, pp. 167–172. AAAI Press.

Ansótegui, C., Bonet, M. L., Levy, J., & Manyà, F. (2007b). The logic behind weighted CSP. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07), Hyderabad, India*, pp. 32–37. AAAI Press.

Argelich, J., & Manyà, F. (2005). Solving over-constrained problems with SAT technology. In *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT-05), St. Andrews, Scotland*, LNCS 3569, pp. 1–15. Springer.

Argelich, J., & Manyà, F. (2006). Exact Max-SAT solvers for over-constrained problems. *Journal of Heuristics*, *12*(4–5), 375–392.

Argelich, J., & Manyà, F. (2007). Partial Max-SAT solvers with clause learning. In *Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing (SAT-07), Lisbon, Portugal*, LNCS 4501, pp. 28–40. Springer.

Bansal, N., & Raman, V. (1999). Upper bounds for MaxSat: Further improved. In *Proceedings of 10th International Symposium on Algorithms and Computation (ISAAC-99), Chennai, India*, LNCS 1741, pp. 247–260. Springer.

Beame, P., Kautz, H., & Sabharwal, A. (2003). Understanding the power of clause learning. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03), Acapulco, México*, pp. 94–99. Morgan Kaufman.

Bonet, M. L., Levy, J., & Manyà, F. (2006). A complete calculus for Max-SAT. In *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing (SAT-06), Seattle, USA*, LNCS 4121, pp. 240–251. Springer.

Bonet, M. L., Levy, J., & Manyà, F. (2007). Resolution for Max-SAT. *Artificial Intelligence*, *171*, 606–618.

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

Boros, E., & Hammer, P. (2002). Pseudo-Boolean optimization. *Discrete Applied Mathematics*, *123*, 155–225.

Cooper, M. C. (2003). Reduction operations in fuzzy or valued constraint satisfaction. *Fuzzy Sets and Systems*, *134*, 311–342.

Cooper, M. C. (2005). High-order consistency in valued constraint satisfaction. *Constraints*, *10*, 283–305.

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

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2001). *Introduction to Algorithms* (second edition). MIT Press.

Davis, M., Logemann, G., & Loveland, D. (1962). A machine program for theorem-proving. *Communications of the ACM*, *5*, 394–397.

Davis, M., & Putnam, H. (1960). A computing procedure for quantification theory. *Journal of the ACM*, *7*(3), 201–215.

de Givry, S., Larrosa, J., Meseguer, P., & Schiex, T. (2003). Solving Max-SAT as weighted CSP. In *Proceedings of 9th International Conference on Principles and Practice of Constraint Programming (CP-03), Kinsale, Ireland*, LNCS 2833, pp. 363–376. Springer.

de Givry, S., Zytnicki, M., Heras, F., & Larrosa, J. (2005). Existential arc consistency: Getting closer to full arc consistency in weighted csps. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05), Edinburgh, Scotland*, pp. 84–89.

Freeman, J. W. (1995). *Improvements to Propositional Satisfiability Search Algorithms*. Ph.D. thesis, Department of Computer and Information Science, University of Pennsylvania, PA, USA.

Goldberg, E., & Novikov, Y. (2001). BerkMin: A fast and robust SAT solver. In *Proceedings of Design, Automation and Test in Europe (DATE-02), Paris, France*, pp. 142–149. IEEE Computer Society.

Heras, F., & Larrosa, J. (2006). New inference rules for efficient Max-SAT solving. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI-06), Boston, USA*. AAAI Press.

Huang, W. Q., & Jin, R. C. (1997). Solar: A learning from human algorithm for solving SAT. *Science in China (Series E)*, *27*(2), 179–186.

Jeroslow, R. G., & Wang, J. (1990). Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, *1*, 167–187.

Larrosa, J., & Heras, F. (2005). Resolution in Max-SAT and its relation to local consistency in weighted CSPs. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05), Edinburgh, Scotland*, pp. 193–198. Morgan Kaufmann.

Larrosa, J., Heras, F., & de Givry, S. (2007). A logical approach to efficient Max-SAT solving. *Artificial Intelligence*, *(in press)*.

Larrosa, J., & Meseguer, P. (2002). Partition-based lower bound for Max-CSP. *Constraints*, *7*(3–4), 407–419.

Larrosa, J., Meseguer, P., & Schiex, T. (1999). Maintaining reversible DAC for Max-CSP. *Artificial Intelligence*, *107*(1), 149–163.

Li, C. M. (1999). A constraint-based approach to narrow search trees for satisfiability. *Information Processing Letters*, *71*, 75–80.

Li, C. M., & Anbulagan (1997a). Heuristics based on unit propagation for satisfiability problems. In *Proceedings of 15th the International Joint Conference on Artificial Intelligence (IJCAI-97), Nagoya, Japan*, pp. 366–371. Morgan Kaufmann.

Li, C. M., & Anbulagan (1997b). Look-ahead versus look-back for satisfiability problems. In *Proceedings of the 3rd International Conference on Principles of Constraint Programming (CP-97), Linz, Austria*, LNCS 1330, pp. 341–355. Springer.

Li, C. M., & Huang, W. Q. (2005). Diversification and determinism in local search for satisfiability. In *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT-05), St. Andrews, Scotland*, LNCS 3569, pp. 158–172. Springer.

Li, C. M., Manyà, F., & Planes, J. (2005). 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 (CP-05), Sitges, Spain*, LNCS 3709, pp. 403–414. Springer.

Li, C. M., Manyà, F., & Planes, J. (2006). Detecting disjoint inconsistent subformulas for computing lower bounds for Max-SAT. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI-06), Boston, USA*, pp. 86–91. AAAI Press.

Marques-Silva, J. P., & Sakallah, K. A. (1999). GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, *48*(5), 506–521.

Niedermeier, R., & Rossmanith, P. (2000). New upper bounds for maximum satisfiability. *Journal of Algorithms*, *36*, 63–88.

Régin, J. C., Petit, T., Bessière, C., & Puget, J. F. (2001). New lower bounds of constraint violations for over-constrained problems. In *7th International Conference on Principles and Practice of Constraint Programming (CP-01), Paphos, Cyprus*, LNCS 2239, pp. 332–345. Springer.

Schiex, T. (2000). Arc consistency for soft constraints. In *Proceedings of the 6th International Conference on Principles of Constraint Programming (CP-00), Singapore*, LNCS 1894, pp. 411–424. Springer.

Shen, H., & Zhang, H. (2004). Study of lower bound functions for max-2-sat. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-04), San Jose, USA*, pp. 185–190. AAAI Press.

Shen, H., & Zhang, H. (2005). Improving exact algorithms for max-2-sat. *Annals of Mathematics and Artificial Intelligence*, *44*, 419–436.

Wallace, R. J. (1995). Directed arc consistency preprocessing. In *Constraint Processing, Selected Papers*, LNCS 923, pp. 121–137. Springer.

Wallace, R. J., & Freuder, E. (1996). Comparative studies of constraint satisfaction and Davis-Putnam algorithms for maximum satisfiability problems. In Johnson, D., & Trick, M. (Eds.), *Cliques, Coloring and Satisfiability*, Vol. 26, pp. 587–615. American Mathematical Society.

Xing, Z., & Zhang, W. (2004). Efficient strategies for (weighted) maximum satisfiability. In *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming (CP-04), Toronto, Canada*, LNCS 3258, pp. 690–705. Springer.

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

Xu, K., Boussemart, F., Hemery, F., & Lecoutre, C. (2005). A simple model to generate hard satisfiable instances. In *Proceedings of 19th International Joint Conference on Artificial Intelligence (IJCAI-05), Edinburgh, Scotland*, pp. 337–342.

Xu, K., & Li, W. (2006). Many hard examples in exact phase transitions. *Theoretical Computer Science*, *355*, 291–302.

Zhang, H. (1997). SATO: An efficient propositional prover. In *Proceedings in the Conference on Automated Deduction (CADE-97)*, pp. 272–275.

Zhang, H., Shen, H., & Manyà, F. (2003). Exact algorithms for MAX-SAT. *Electronic Notes in Theoretical Computer Science*, *86*(1).

Zhang, L., Madigan, C., Moskewicz, M., & Malik, S. (2001). Efficient conflict driven learning in a Boolean satisfiability solver. In *International Conference on Computer Aided Design (ICCAD-01), San Jose, USA*, pp. 279–285.