Max-SAT Algorithms For Real World Instances -
Extended Abstract

Pedro Filipe Medeiros da Silva

November 23, 2010

Abstract

The Maximum Satisfiability (Max-SAT) problem is an optimization version of the Boolean Satisfiability problem (SAT) that has gained recognition in the last years. Still, most Max-SAT solvers lag behind the efficiency of SAT solvers for most problems, especially in real world problems. There are currently two different approaches to solve Max-SAT, branch-and-bound algorithms and translation based algorithms. The first performs best on solving homogeneous instances, whereby the second is better for heterogeneous instances, which usually result from real-world problems. We propose combining these two different approaches, by using a pre-processor with inference rules from branch-and-bound solvers, and then solve the processed instance with a translation based solver.

1 Introduction

NP-Complete (NP-C) [1] is a complexity class of problems that are NP and NP-hard. For NP problem it is possible to verify if a solution is correct in polynomial time. P problems are a subset of problems of NP that are known to be solved in polynomial time. NP-hard problems are at least as hard as the hardest problem in NP and there is a polynomial reduction function that translates any NP problem into a NP-hard problem. Until now, there is no known algorithm for a NP-hard problem that can be solved in polynomial time; if there was such an algorithm, the NP class would be equal to the P class.

Although NP-C and NP-hard problems can not be solved efficiently in modern computers, there are many real world problems that are NP-hard and need to be solved. Today, there are so many problems that are NP-C [2], that the best way to solve them, is to reduce the problem to a well know target NP-C problem and solve it as an instance of the target problem. The original NP-C problem is the Boolean Satisfiability problem (SAT) [1]. The development of SAT as a target problem to solve decision-based problems explains why there is such a large SAT community.

Maximum satisfiability (Max-SAT) is an optimization version of the SAT problem that is NP-hard and APX-complete [3]; this means that Max-SAT is at least as hard as SAT, and that it does not accept a polynomial-time approximation algorithm. At first a glance, it seems that there is no reason to use Max-SAT to solve problems, due to its difficulty. Actually, there are many problems that can be more easily translated to Max-SAT than to SAT. Max-SAT
is optimization-based, meaning that it finds the best solution from all feasible solutions. This contrasts to SAT, that is a decision based problem, meaning that it decides whether an assignment is a solution or not. Many real world problems are optimization based, and so, are more suitable to be translated to Max-SAT.

1.1 Max-SAT problem

The Maximum Satisfiability problem (Max-SAT) is an optimization version of the SAT problem, whereby one must find an assignment to the formula's variables, such that there is a maximum number of satisfied clauses (or the equivalent of finding the minimum number of unsatisfied clauses).

Example 1.1 With a CNF formula $\phi = \{x_1, \overline{x}_1 \lor x_2, \overline{x}_1 \lor \overline{x}_2, x_1 \lor \overline{x}_3, \overline{x}_1 \lor \overline{x}_3\}$, the maximum number of satisfiable clauses is four or alternatively the minimum number of unsatisfiable clauses is one. The solution is to assign $x_1$ to FALSE.

Besides Max-SAT, there are variants that enable the expression of more real world problems. We take into account three of these extensions: Weighted Max-SAT problem, where each clause has a weight and the problem is to find an assignment that maximizes the sum of weights of satisfied clauses; Partial Max-SAT problem, where each clause in the formula can be either soft or hard and the hard clauses must be obligatorily satisfied, while the number of satisfied soft clauses must be maximized; The weighted partial Max-SAT problem is the combination of the partial Max-SAT problem, where clauses are hard or soft, and the weighted Max-SAT problem, where soft clauses have a weight value. One must find an assignment that satisfies the hard clauses and maximizes the sum of the weight of satisfied soft clauses.

2 Max-SAT Algorithms

The most popular technique for exact Max-SAT solvers is the branch-and-bound (BnB) technique, that will be further explained in the next subsection. Another approach, which consists in efficiently translating Max-SAT instances to other problem instances, such as SAT, where they can be solved with state-of-the-art solvers designed for the SAT problem, has been found to be effective on solving real-world problems.

2.1 Branch and Bound Algorithms

Branch and Bound (BnB) is an algorithm paradigm used for solving combinatorial optimization problems, first introduced in [4] for linear programming. Throughout this section, we will tackle the Max-SAT problem as the minimization of unsatisfiable clauses, for the sake of simplicity.

BnB for Max-SAT works as follows: BnB explores the search space as a binary tree, where the root is the Max-SAT instance $\phi$, the left child node is $\phi$ with the variable $x$ set to $\text{FALSE}$ (or $\text{TRUE}$) and the right child node is $\phi$ with $x$ set to $\text{TRUE}$ or $\text{FALSE}$. BnB explores the search space in a depth-first way and at each node compares the best solution found so far, the upper bound (UB), with the number of unsatisfied clauses plus an underestimation
of the number of unsatisfiable clauses, the lower bound (LB). If LB ≥ UB, it
means that this current assignment is worst than the current best, and so it
can be pruned and the algorithm backtracks in the search tree. If otherwise
LB < UB, BnB recursively applies the algorithm, assigning a Boolean value to
a variable in the left branch and its negation to the right branch and so on.
The algorithm stops when the whole valid search tree is explored, returning the
minimum number of unsatisfiable clauses of the original input formula, which
is in the UB variable.

One approach current Max-SAT solvers use to enhance the BnB is to apply
inference rules before creating a new formula. These formulas produce a new
equivalently formula, that often reduces the number of clauses and variable in
the formula and so, reduces the search space. Some of these inference rules are
presented next.

2.1.1 Inference

A complete resolution for Max-SAT is presented in [5]. Although it can not
be used in efficient solvers, it is possible to infer some more useful rules when
branching in BnB solvers. These rules are also called transformation rules, as
they replace the premises of the rule with the conclusion. If they were to just add
the conclusions, like in SAT inference, it could increase the number of falsified
clauses, making the inference not sound.

The basis for using inference is to transform a Max-SAT instance into an
equivalent but simpler one. This transformation can induce the discovery of
empty clauses and by doing it, increasing the lower bound. The advantage of
inference over underestimations, is that all computation may be stored. So all
the empty clauses discovered using inference do not have to be recomputed.
This makes inference more incremental than an underestimations.

Overall, most inference rules in Max-SAT can be reduced to three categories:
single resolution, variable elimination and hyper-resolution. We will overview
most of the inference rules used in modern Max-SAT solvers.

Single resolution The unit clause reduction[6], is a direct extension of the
unit propagation used in SAT. If there is a hard clause that is unit, (l, ⊤),
then all occurrences of l in other clauses can be removed. It is simple to see
that the unit clause must be satisfied and so l must also be satisfied. Formally
\{(l, ⊤), (l ∨ C, w)\} ≡ \{(l, ⊤), (C, w)\}, where C is a disjunction of literals, u and
w are clause weights and ⊤ is the hard clause weight.

Example 2.1 Given the partial weighted Max-SAT instance \(φ = \{(x_1 ∨ x_2, 1),
(x_2, 1), (x_1, ⊤)\}\), the unit clause reduction rule would transform the first clause
and would result in the formula \(φ' = \{(x_2, 1), (x_1, ⊤)\}\).

If a hard clause is a subset of other clause, then the other clause is removed,
given that the hard clause must be satisfied and doing so satisfies any clause
with the same or more literals. For example, \{(C, ⊤), (C ∨ D, w)\} ≡ \{(C, ⊤)\},
where D is a disjunction of literals.

Example 2.2 Given the partial weighted Max-SAT instance \(φ = \{(x_1 ∨ x_2, 1),
(x_2, 1), (x_1, ⊤)\}\), absorption would remove the first clause and result in the for-
mula \(φ' = \{(x_2, 1), (x_1, ⊤)\}\).
Neighbour resolution[7] is a particular case of Max-SAT resolution where both clauses share a set of literals. Formally \( \{(l \lor C, u), (\bar{l} \lor C, w)\} \equiv \{(C, w), (\bar{C}, u)\} \), where \( \oplus \) is the operation between two clauses weights from which results a new weight that is the minimum of the two.

**Example 2.3** Given the partial weighted Max-SAT instance \( \phi = \{(x_1 \lor x_2, \top), (\bar{x}_2 \lor x_1, 1), (\bar{x}_1, 1)\} \), neighbour resolution can be applied to the first two clauses of \( \phi \) with respect to literal \( x_2 \). Applying it would result in the formula \( \phi' = \{ (x_1 \lor x_2, \top), (x_1, 1), (\bar{x}_1, 1)\} \).

The dominating unit clause rule[8], can be seen as the unit propagation rule for Max-SAT that is sound. If the weight of a unit clause where the literal \( l \) appears in \( \phi \), is the same or higher than the number of clauses where \( \bar{l} \) appears, then \( l \) can be safely assigned.

**Example 2.4** Given the partial weighted Max-SAT instance \( \phi = \{(x_1, 2), (\bar{x}_1 \lor x_2, 1), (x_1 \lor x_2, 1)\} \). \( \phi \) can be safely assigned the partial assignment \( A : \{x_1 = \text{TRUE}\} \) due to the dominating unit clause rule.

**Hyper-resolution** Hyper resolution in the context of SAT is a resolution-based rule concepts in which various resolution steps are compressed into one. Next we present some of the uses of hyper-resolution in Max-SAT.

Chain resolution is an original rule from [9] that captures a chain between two unit clauses using a subset of binary clauses. This rule allows to derive an empty clause. Formally,

\[
\left\{ \begin{array}{l}
(l_1, u_1), \\
(l_i \lor l_{i+1}, u_{i+1})_{1 \leq i < k}, \\
(l_k, u_k)
\end{array} \right\} \equiv \left\{ \begin{array}{l}
(l_i, m_i \odot m_{i+1})_{1 \leq i \leq k}, \\
(l_i \lor l_{i+1}, u_{i+1} \oplus m_{i+1})_{1 \leq i < k}, \\
(l_j \lor l_{i+1}, m_{i+1})_{1 \leq j < k}, \\
(l_k, u_k \oplus m_{k+1}), \\
(\Box, m_{k+1})
\end{array} \right\},
\]

where \( m_i = \min\{u_1, u_2, \cdots, u_i\} \) and \( \forall 1 \leq i < j \leq k, l_i \neq l_j \).

**Example 2.5** Given the partial weighted Max-SAT instance \( \phi = \{(x_1, 1), (\bar{x}_3, 1), (\bar{x}_1 \lor x_2, \top), (x_2 \lor x_3, 1)\} \), we can apply chain resolution. The resulting formula derives a new empty clause: \( \phi = \{(\bar{x}_3 \lor x_2, \top), (x_1 \lor x_2, \top), (x_2 \lor \bar{x}_3, \top), (\Box, 1)\} \).

Chain resolution is equal to simple neighbour resolution when only two clauses are used in the chain resolution rule.

Cycle resolution is also an original rule from [9] that identifies a cycle of binary clauses with the starting and ending literal of the cycle from a single clause. The rule allows to derive only a unit clause, but it can be used with other rules that use unit clauses to derive empty clauses. Formally,

\[
\left\{ \begin{array}{l}
(l_i \lor l_{i+1}, u_i)_{1 \leq i < k}, \\
(l_i \lor l_k, u_k)
\end{array} \right\} \equiv \left\{ \begin{array}{l}
(l_i \lor l_i, m_{i-1} \odot m_i)_{2 \leq i \leq k}, \\
(l_i \lor l_{i+1}, u_i \odot m_i)_{2 \leq i < k}, \\
(l_i \lor l_i \lor l_{i+1}, m_i)_{2 \leq i < k}, \\
(l_i \lor l_i \lor l_{i+1}, m_i)_{2 \leq i < k}, \\
(l_i \lor l_{i+1}, u_k \odot m_k), \\
(l_i, m_k)
\end{array} \right\},
\]

where \( m_i = \min\{u_1, u_2, \cdots, u_i\} \) and \( \forall 1 \leq i < k, l_i \neq l_j \).
Example 2.6  Given the partial weighted Max-SAT instance $\phi = \{(\overline{x}_1 \vee x_3, 1), (\overline{x}_1 \vee x_2, \top), (x_1, 1)\}$, a new unit clause can be derived using cycle resolution: $\phi' = \{(\overline{x}_1, 1), (\overline{x}_1 \vee x_2, \top), (\overline{x}_1 \vee x_3 \vee \overline{x}_3, 1), (x_1 \vee \overline{x}_2 \vee x_3, 1), (x_1, 1)\}$. The new unit clause can be used to find a new empty clause with neighbour resolution.

Cycle resolution can also degenerate to neighbour resolution with $k = 2$.

2.2 Translation based algorithms

As previously mentioned, many SAT based techniques cannot be efficiently used in Max-SAT solvers. The main reason is the soundness requirement in the Max-SAT problem. Due to the mature nature of SAT solvers, an efficient translation from Max-SAT problems to SAT may be useful to solve certain types of Max-SAT instances.

2.2.1 SAT translation

SAT4Jmaxsat[10] is a Max-SAT solver that translates a Max-SAT instance into a SAT problem. It does so by, given the Max-SAT instance $\phi$, adding a blocking variable $b_i$ to each clause in $\phi$, where $i = |\phi|$. Then it uses SAT4J, a SAT solver from the same author, that supports cardinality constraints, to solve the new instance as a SAT problem. Each time the SAT solver finds an assignment $A$ that satisfies the formula, it adds a cardinality constraint $\sum_{i=1}^{n} b_i < |$ blocking variable with value TRUE $. |$. When the formula can no longer be satisfied, the number of blocking variables assigned with the value TRUE in the last iteration is the solution for the original Max-SAT problem.

An approach to translate a Max-SAT instance problem to a SAT instance problem while reducing the number of blocking variable, is to add a blocking variable only to the unsatisfiable subformulas (or unsatisfiable cores) of the Max-SAT instance. Most SAT solvers can extract these unsatisfiable cores as a resolution refutation of the SAT problem. This approach was first introduced by [11] and was further developed in [12], where it describes the algorithm from [11] and adapts it to Max-SAT naming it MSU1 (originally it was only meant to Partial Max-SAT).

MSU1 was later improved in [12] to use a different encoding for the cardinality constraint instead of the pairwise encoding. It also imposes an AtMost1 constraint to the number of blocking variables in each clause. The cardinality constraint encoding used in this new version was based on Binary Decision Diagrams (BDDs), being is linear in the number of variables in the constraint[13].

3 Combining approaches

Although BnB is the most popular technique to solve Max-SAT instances, branching on each literal in the formula quickly explodes the search space. On the other hand, the translation technique is useful when there are many unsatisfiable subformulas.

It can be said that BnB is more suitable for random, homogeneous instances, whereby algorithms such as msumcore[12] are best with industrial instances,
where there are many more variables, but are much less random and are heterogeneous.

The focus of this section is to extend the approach used in the msuncore algorithm, with some of the techniques used in BuB, more specifically inference rules as a pre-processor of the search phase.

At the time of writing this thesis, msuncore focused in partial Max-SAT instances and so we will be focusing also on these instances. We will also be using the notation introduced in 2.1.1, this time only for partial Max-SAT, instead of weighted partial Max-SAT. The only difference is the weight of soft clauses always being one.

In this section, we will adapt some of the inference rules present before to partial Max-SAT and to efficient implementations. In the next section, we will analyse the impact of the different inference techniques and combinations to partial Max-SAT instances to verify if they are suitable as pre-processor of instances before using algorithms such as msuncore.

3.1 Efficient Inference

As stated previously, we will focus on efficient inference rules for partial Max-SAT. The limitation of using only partial Max-SAT gives us more strict rules. Also, at the time of the development of this solution the major translation based algorithm, msuncore, only supported partial Max-SAT instances.

Single resolution and variable elimination, presented in section 2.1.1, are simple enough, so it will not be the focus of this simplification.

In hyper resolution, the inference rules are more complex and may use several clauses and literals. Hence we will limit the scope of application not only to make it usable as a pre-processing, but also to simplify the implementation. The latter also affects the first.

3.1.1 Chain resolution

This rule was already present in section 2.1.1. The usefulness of this rule lies in the fact that it uses soft unit and binary clauses, something that is difficult to use in most inference rules. Besides that, it generates an empty clause, cutting the search space.

Although this rule generates a linear number of clauses, this number can be more than three times the original number of clauses of the inference rule. We have limited this rule so that both unit clauses are soft. If any unit clause were hard, it could be applied the dominating unit clause rule. Therefore, using chain resolution when a unit clause is hard is not helpful.

Next, we present a simplified chain resolution rule, where the unit clauses can be only soft clauses.

\[
\begin{align*}
\begin{cases}
(l_1, u_1), \\
(\overline{l_i} \lor l_{i+1}, u_{i+1})_{1 \leq i < k}, \\
(l_k, u_k)
\end{cases}
\end{align*}
\]

\[
\equiv
\begin{cases}
(l_i \lor l_{i+1}, u_{i+1} \ominus 1)_{1 \leq i < k}, \\
(l_i \lor \overline{l_{i+1}}, 1)_{1 \leq i < k}, \\
(\square, 1)
\end{cases}
\]

where \( \forall 1 \leq i < j \leq k, l_i \neq l_j \) and \( u_1 = 1, u_k = 1 \).

This simplification allows us to limit the number of clauses generated with chain resolution to at most the double of the number of original clauses. If all clauses are soft clauses, then the number of clauses actually decreases.
The example from section 2.5 already applies this rule.

3.1.2 Cycle resolution

Cycle resolution is another hyper resolution rule from the same authors of chain resolution. The main objective for using this rule is to derive a new unit clause so that it can be used with other rules, such as chain resolution.

This rule also generates a linear number of clauses, but it can be up to four times the original number of clauses and it does not have the same impact of space search reduction. These factors limit the usefulness of this rule.

We will be using cycle resolution with $k \leq 3$ and the first clause being soft. These restrictions allow a good trade-off between the number of non-unit clauses generated and the unit clause gained from this trade.

When $k = 2$, cycle resolution is equal to neighbour resolution with binary clauses. We present the full inference rule with 3 clauses that we will be using:

$$\left\{ \begin{array}{l}
(l_1 \lor l_2, u_1), \\
(l_2 \lor l_3, u_2), \\
(l_1 \lor l_3, u_3)
\end{array} \right\} \equiv \left\{ \begin{array}{l}
(l_2 \lor l_3, u_2 \ominus 1), \\
(l_1 \lor l_2 \lor l_3, 1), \\
(l_1 \lor l_2 \lor l_3, 1), \\
(l_1 \lor l_3, u_3 \ominus 1), \\
(l_1, 1)
\end{array} \right\}$$

where $\forall 1 \leq i < j \leq k \ l_i \neq l_j$ and $u_1 = 1$.

4 Testing

In this sections we the define the strategies used in order to verify if pre-processing partial Max-SAT instance with inference techniques is useful for real world instances in translation based approaches.

First we will describe the techniques used for pre-processing as well the configuration used for testing. Next there is an analyses of the results obtained from the tests made.

4.1 Test Setup

The techniques used for pre-processing the partial Max-SAT instances were already described in section 3.1. We will use abbreviations for each technique in regard to the pre-processing made to the instances: none, the original instance without pre-processing; neigh, neighbour resolution; unit, dominating unit clause; cycle, cycle resolution; all, all the rules mention applied.

The version of the msuncore solver used is 1.2. The machine setup consists of Intel® Xeon® @5160 dual processor at 3.0 GHz with 2GiB for each solver run.

The instances used for these tests, were the same used at the Max-SAT evaluation 2008[14]: Randomly generated instances, 150 total; Instances crafted from some particular problems, 298 total; Industrial instances, converted from most real world problems, 1950 total.

Each instance had an upper bound time limit of 1200 seconds, for both pre-processing and solving of each instance.
4.2 Test Analyses

First in Table 1, we analyse how many instances were pre-processed, i.e., how many instances were successfully transformed with the applied inference rule(s) within the given time-out.

Then, in table 2, we compare how many instances were solved within each of the categories: random, crafted and industrial. For each instance, there is a column that displays the number of instances that were solved with msuncore given the time limit. Also in the same column, in parenthesis, is displayed the ratio between the total number of instances with the number of instances solved with the time limit.

Pre-processing The first step is to pre-process all instances, the pre-processor was used with each technique for each instance. Because some techniques may impact the number of unsatisfiable clauses, there is a comment line in each processed instance with the number of unsatisfiable clauses eliminated.

The number of processed instances for each technique is reflected in table 1. For those instances that were unable to be pre-processed, the instance without pre-processing was used instead. Unit inference was the only inference rule that did not pre-process all the instances. This made all the inference rules applied also not able to process all the instances. These results make the unit pre-processing and all rules applied less reliable, as most of the instances will not be pre-processed.

Instances Table 2 lists the number of solved instances per technique used and per type of instance.

As expected, the use of the pre-processor did not change the overall result of the number of random instances solved. The number of solved instances for the random instances was practically the same as after use any technique before. The crafted instances has minor in gain or losses by the use of pre-processing. However the variation was slightly bigger than in the random instances.

As expected the industrial instances were the ones where msuncore was most effective. Both chain and neighbour resolution pre-processed instances enabled msuncore to solve more 4 and 11 instances, respectively. The overall conclusions are discussed in section 5.

<table>
<thead>
<tr>
<th>Pre-processing</th>
<th>Nr. processed (out of 2398)</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>2398</td>
<td>100%</td>
</tr>
<tr>
<td>neigh</td>
<td>2398</td>
<td>100%</td>
</tr>
<tr>
<td>unit</td>
<td>848</td>
<td>35.36%</td>
</tr>
<tr>
<td>cycle</td>
<td>2398</td>
<td>100%</td>
</tr>
<tr>
<td>chain</td>
<td>2398</td>
<td>100%</td>
</tr>
<tr>
<td>all</td>
<td>688</td>
<td>28.69%</td>
</tr>
</tbody>
</table>

Table 1: Pre-processed instances
<table>
<thead>
<tr>
<th>Solved instances</th>
<th>Random</th>
<th>Crafted</th>
<th>Industrial</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>27 (18.00%)</td>
<td>49 (16.44%)</td>
<td>1544 (79.18%)</td>
</tr>
<tr>
<td>neigh</td>
<td>27 (18.00%)</td>
<td>45 (15.10%)</td>
<td>1548 (79.38%)</td>
</tr>
<tr>
<td>unit</td>
<td>27 (18.00%)</td>
<td>45 (15.10%)</td>
<td>1538 (78.87%)</td>
</tr>
<tr>
<td>cycle</td>
<td>27 (18.00%)</td>
<td>45 (15.10%)</td>
<td>1531 (78.51%)</td>
</tr>
<tr>
<td>chain</td>
<td>28 (18.67%)</td>
<td>46 (15.44%)</td>
<td>1555 (79.74%)</td>
</tr>
<tr>
<td>all</td>
<td>28 (18.67%)</td>
<td>49 (16.44%)</td>
<td>1540 (78.97%)</td>
</tr>
</tbody>
</table>

Table 2: Solved instances

5 Conclusions and Future Work

Max-SAT and their variants are used to solve real world problems that can not be solved with ease in their original codification. The technique that has shown to be the most efficient to solve these problems is translation-based. Our work tried to bring some of the techniques used in the well known branch-and-bound method.

We adopted techniques from related work in SAT such as the pure literal rule and the equivalent unit propagation for Max-SAT. We also used some more recent original techniques known as hyper resolution from the works of [15].

The expectation that using inference techniques as pre-processing for translation based solvers would improve results significantly was gored. The results that can be seen in section 4.2 clearly show that there were no improvements in most cases, with the exception of neighbour resolution and chain resolution, where there were 11 more instances solved. Dominating unit clause and cycle resolution produce the worse results, which made the combination of all techniques overall worse as well.

As a post-mortem analysis, the study of the pre-processed instances should reveal more information for the reason of these results. Only some of the inference rules were helpful, while other were actually worse than not using them.

As future work, the techniques used for pre-processing that did help in solving industrial instances, such as neighbour resolution and chain resolution, could be incorporated inside the translation based approach to trim the solution after each unsuccessful iteration. Also, more techniques could also be used, and a combination for the more efficient ones would be used as a final version of the pre-processor.

Finally, the pre-processor could be further improved to give intermediary results. Future versions should incorporate this feature.

References


