Transforming Inconsistent Subformulas in MaxSAT Lower Bound Computation*

Chu Min Li[1,2], Felip Manyà[3,4], Nouredine Ould Mohamedou[2], and Jordi Planes[4]

[1] Hunan Normal University, Changsha, China
[2] MIS, Université de Picardie Jules Verne, 5 Rue du Moulin Neuf 80000 Amiens, France
[3] Artificial Intelligence Research Institue (IIIA, CSIC), Campus UAB, 08193 Bellaterra, Spain
[4] Computer Science Department, Universitat de Lleida, Jaume II 69, 25001 Lleida, Spain

Abstract. We define a new heuristic that guides the application of cycle resolution (CR) in MaxSAT, and show that it produces better lower bounds than those obtained by applying CR exhaustively as in Max-DPLL, and by applying CR in a limited way when unit propagation detects a contradiction as in MaxSatz.

1 Introduction

The lower bound (LB) computation method implemented in modern branch and bound MaxSAT solvers has two components: (i) the *underestimation component*, which detects disjoint inconsistent subformulas (typically using unit propagation (UP) [1] or unit propagation enhanced with failed literal detection [2]), and takes the number of detected inconsistent subformulas as an underestimation of the LB; and (ii) the *inference component*, which applies cost preserving inference rules that, in the best case, make explicit a contradiction by deriving an empty clause which allows to increment the LB.

We analyze more deeply than before the impact of the *cycle resolution* (CR) inference rule on the performance of MaxSAT solvers. It is well-known that Max-DPLL [3] applies CR exhaustively and does not combine its application with the underestimation component, while MaxSatz [4] applies CR when the underestimation component detects an inconsistent subformula via unit propagation which includes one particular unit clause and the premises of CR. In this paper, we provide evidence that the exhaustive application of CR is not the best option in general, and that combining its application with an underestimation component incorporating failed literal detection may produce better quality LBs. To better exploit the power of CR in MaxSAT solvers, we define a new heuristic that guides the application of CR during failed literal detection. Experiments on a new version of MaxSatz implementing this heuristic, called MaxSatz$_c$, show that MaxSatz$_c$ substantially speeds up MaxSatz.

2 Inference Rules

MaxSatz [4] incorporates the following rules (also called Rule 1, Rule 2, Rule 3, and Rule 4 in this paper):

$$l_1,\ \bar{l}_1 \vee \bar{l}_2,\ l_2 \Longrightarrow \square,\ l_1 \vee l_2 \tag{1}$$

* This research was funded by MEC research projects TIN2006-15662-C02-02, TIN2007-68005-C04-04, and CONSOLIDER CSD2007-0022, INGENIO 2010.

P.J. Stuckey (Ed.): CP 2008, LNCS 5202, pp. 582–587, 2008.

$$l_1,\ \bar{l}_1 \vee l_2,\ \bar{l}_2 \vee l_3, \ldots,\ \bar{l}_k \vee l_{k+1},\ \bar{l}_{k+1} \Longrightarrow \square,\ l_1 \vee \bar{l}_2,\ l_2 \vee \bar{l}_3, \cdots,\ l_k \vee \bar{l}_{k+1} \tag{2}$$

$$l_1,\ \bar{l}_1 \vee l_2,\ \bar{l}_1 \vee l_3,\ \bar{l}_2 \vee \bar{l}_3 \Longrightarrow \square,\ l_1 \vee \bar{l}_2 \vee \bar{l}_3,\ \bar{l}_1 \vee l_2 \vee l_3 \tag{3}$$

$$\begin{array}{c} l_1,\ \bar{l}_1 \vee l_2,\ \bar{l}_2 \vee l_3, \cdots,\ \bar{l}_k \vee l_{k+1}, \\ \bar{l}_{k+1} \vee l_{k+2},\ \bar{l}_{k+1} \vee l_{k+3},\ \bar{l}_{k+2} \vee \bar{l}_{k+3} \end{array} \Longrightarrow \begin{array}{c} \square,\ l_1 \vee \bar{l}_2,\ l_2 \vee \bar{l}_3, \cdots,\ l_k \vee \bar{l}_{k+1}, \\ l_{k+1} \vee \bar{l}_{k+2} \vee \bar{l}_{k+3},\ \bar{l}_{k+1} \vee l_{k+2} \vee l_{k+3} \end{array} \tag{4}$$

Max-DPLL [3] incorporates several rules for weighted MaxSAT, including chain resolution (which is equivalent to Rule 2 in the unweighted case) and CR restricted to 3 variables, which is as follows in the unweighted case:

$$\begin{array}{l} \bar{l}_1 \vee l_2 \\ \bar{l}_1 \vee l_3 \\ \bar{l}_2 \vee \bar{l}_3 \end{array} \Longrightarrow \begin{array}{c} \bar{l}_1 \\ l_1 \vee \bar{l}_2 \vee \bar{l}_3 \\ \bar{l}_1 \vee l_2 \vee l_3 \end{array} \tag{5}$$

In the sequel, when we say CR we mean CR restricted to 3 variables. Rule 3 and Rule 4 in MaxSatz, and CR in Max-DPLL capture this special structure: $\bar{l}_1 \vee l_2, \bar{l}_1 \vee l_3, \bar{l}_2 \vee \bar{l}_3$, which we call *cycle structure*. Max-DPLL applies CR in an unlimited way: it replaces every subset of three binary clauses matching the cycle structure with one unit clause and two ternary clauses. MaxSatz applies CR in a limited way: it applies CR only when the underestimation component detects a contradiction containing the cycle structure by applying unit propagation.

3 CR and Failed Literal Detection

The next example shows that it is worth applying CR in scenarios not considered in the version of MaxSatz used in the 2007 MaxSAT Evaluation, called MaxSatz-07 in this paper. MaxSatz-07 computes underestimations using failed literal detection after no more contradictions can be derived using unit propagation, but does not check if a rule is applicable when a failed literal is detected. The applicability of rules is just checked when unit propagation (without failed literals) derives a contradiction. So, CR is applied only when unit propagation allows to apply Rule 3 or Rule 4. Assume that a MaxSAT instance ϕ contains

$$x_1 \vee x_2, \bar{x}_2 \vee x_3, \bar{x}_2 \vee x_4, \bar{x}_3 \vee \bar{x}_4, \bar{x}_1 \vee x_5,\ \ \bar{x}_5 \vee x_6,\ \ \bar{x}_1 \vee x_7,\ \bar{x}_6 \vee \bar{x}_7$$
$$x_8 \vee \bar{x}_2, x_8 \vee x_3, x_8 \vee x_4, \bar{x}_8 \vee x_9, \bar{x}_8 \vee x_{10}, \bar{x}_8 \vee x_{11}, \bar{x}_9 \vee \bar{x}_{10} \vee \bar{x}_{11}$$

Since there is no unit clause, Rule 3 and Rule 4 are not applied. Failed literal detection just detects the inconsistent subformula in the first line (branching on the variable x_1). However, if CR is applied to $\bar{x}_2 \vee x_3,\ \bar{x}_2 \vee x_4,\ \bar{x}_3 \vee \bar{x}_4$, the underestimation component detects 2 inconsistent subformulas (one branching on x_1 and the other on x_8). We would like to highlight two features of this example: (i) the inconsistent subformula detected when CR is applied after branching on x_1 is smaller than when CR is not applied: $x_1 \vee x_2,\ \bar{x}_2,\ \bar{x}_1 \vee x_5,\ \bar{x}_5 \vee x_6,\ \bar{x}_1 \vee x_7,\ \bar{x}_6 \vee \bar{x}_7$ instead of $x_1 \vee x_2,\ \bar{x}_2 \vee x_3,\ \bar{x}_2 \vee x_4,\ \bar{x}_3 \vee \bar{x}_4,\ \bar{x}_1 \vee x_5,\ \bar{x}_5 \vee x_6,\ \bar{x}_1 \vee x_7,\ \bar{x}_6 \vee \bar{x}_7$; and (ii) the added ternary clauses

($x_2 \vee \bar{x}_3 \vee \bar{x}_4$ and $\bar{x}_2 \vee x_3 \vee x_4$) may contribute to detect further inconsistent subformulas: When branching on x_8, it detects the inconsistent subformula $x_8 \vee \bar{x}_2,\ x_8 \vee x_3,\ x_8 \vee x_4,\ \bar{x}_8 \vee x_9,\ \bar{x}_8 \vee x_{10},\ \bar{x}_8 \vee x_{11},\ \bar{x}_9 \vee \bar{x}_{10} \vee \bar{x}_{11},\ x_2 \vee \bar{x}_3 \vee \bar{x}_4$, which contains one of the added ternary clauses.

Proposition 1. *Let l be a failed literal in ϕ (i.e., UP($\phi \wedge l$) derives an empty clause), and let S_l be the set of clauses used to derive the contradiction in UP($\phi \wedge l$). If S_l contains the cycle structure $\bar{l}_1 \vee l_2,\ \bar{l}_1 \vee l_3,\ \bar{l}_2 \vee \bar{l}_3$, and S'_l is S_l after applying CR to the cycle structure, then $S'_l - \{l_1 \vee \bar{l}_2 \vee \bar{l}_3, \bar{l}_1 \vee l_2 \vee l_3\}$ is inconsistent.*

Proposition 1 states that if an inconsistent subformula containing the cycle structure is detected using unit propagation enhanced with failed literal detection and CR is applied, then the inconsistent subformula is smaller than the inconsistent subformula that can be derived without applying CR and, moreover, CR adds two ternary clauses that can be used to detect other inconsistent subformulas. This is the rationale behind the heuristic, defined in the next section, for guiding the application of CR during the process of detecting failed literals in the underestimation component.

4 A New Heuristic for Guiding the Application of CR

We have seen that CR can improve the lower bound if its application allows to reduce the size of an inconsistent subformula and liberate two ternary clauses. We define now a heuristic that guides the application of CR during the process of detecting failed literals with the aim of capturing situations in which CR could be beneficial. This heuristic is implemented in Algorithm 1, where occ2(l) is the number of occurrences of literal l in binary clauses of ϕ, and S_l is an inconsistent subformula detected by applying unit propagation to $\phi \wedge l$.

Between the two literals of a variable x that are likely to be failed (since their satisfaction results in at least two new unit clauses), Algorithm 1 detects first the literal l with more occurrences in binary clauses. Note that l has less chances to be failed than $\bar{l}$ because its satisfaction produces fewer new unit clauses. If l is a failed literal and S_l contains a cycle structure, CR is applied in S_l before deciding whether $\bar{l}$ is failed. If $\bar{l}$ is also failed, the inconsistent subformula $S_l \cup S_{\bar{l}} - \{l, \bar{l}\}$ is transformed into a smaller inconsistent subformula by applying CR in S_l, and two ternary clauses are liberated. If $\bar{l}$ is not a failed literal in the current node, we do not have an inconsistent subformula that can be transformed using CR in S_l, but S_l is now smaller thanks to CR and will be easier to redetect it in the subtree. Note that $\bar{l}$ has higher chances to fail in the subtree and to produce an inconsistent subformula transformed using CR.

On the other hand, if l is not failed, Algorithm 1 does not detect an inconsistent subformula. It is not checked whether $\bar{l}$ is failed, and CR is not applied to $S_{\bar{l}}$ if $\bar{l}$ is failed, avoiding the application of CR that does not allow to transform an inconsistent subformula.

With the aim of evaluating the impact of Algorithm 1 in the performance of MaxSatz, we define the following solvers:

MaxSatz-07: Standard version of MaxSatz, implementing all MaxSatz inference rules, and failed literal detection, besides UP, in the underestimation component.

Algorithm 1. *flAndCycle*(ϕ, x), combining CR and failed literal detection

Input: A MaxSAT instance ϕ and a variable x such that occ2(x)$\geq$ 2 and occ2($\bar{x}$)$\geq$ 2
Output: ϕ in which CR is possibly applied, and an underestimation

```
1 begin
2     if occ2(x)>occ2(x̄) then l⟵x; else l⟵x̄;
      underestimation ⟵ 0;
      if UP(φ ∧ l) derives a contradiction then
3         apply CR in S_l if S_l contains a cycle structure;
          if UP(φ ∧ l̄) derives a contradiction then
4             apply CR in S_l̄ if S_l̄ contains a cycle structure;
              underestimation ⟵ 1;
5     return new φ and underestimation
6 end
```

MaxSatz: Optimized version of MaxSatz-07. The optimizations make MaxSatz substantially faster for Max-2SAT, but slightly slower for Max-3SAT when the clauses-to-variables ratio is small. All the following solvers are implemented on top of MaxSatz.

MaxSatz$_c$: The failed literal detection of MaxSatz is replaced by the following procedure: for every variable x such that occ2(x)$\geq$ 2 and occ2($\bar{x}$)$\geq$ 2, call Algorithm 1. Compared with MaxSatz, after detecting failed literals l and $\bar{l}$, and incrementing the underestimation by 1, the inconsistent subformula $S_l \cup S_{\bar{l}} - \{l, \bar{l}\}$ is transformed by applying CR, so that MaxSatz$_c$ has additional clauses for detecting other inconsistencies.

MaxSatz$_c^p$: MaxSatz$_c$ but applying CR exhaustively at the root node as a preprocessing. For instances without binary clauses, MaxSatz$_c^p$ is simply MaxSatz$_c$.

MaxSatz^p: MaxSatz but applying CR exhaustively at the root node as a preprocessing. For instances without binary clauses, MaxSatz^p is simply MaxSatz.

MaxSatz$_{c^*}$: MaxSatz but applying CR exhaustively at each node after applying UP and the inference rules (Rule 1, Rule 2, Rule 3, and Rule 4), and before applying failed literal detection.

The exhaustive applications of CR in MaxSatz$_{c^*}$ and in the preprocessing of MaxSatz$_c^p$ and MaxSatz^p are not combined with failed literal detection. We do not know a priori whether they allow to transform an inconsistent subformula. Their comparison with MaxSatz$_c$ will allow to see the effectiveness of CR combined with failed literal detection to transform inconsistent subformulas.

5 Experimental Results and Analysis

We compared the different versions of MaxSatz —on a Linux Cluster where the nodes have a 2GHz AMD Opteron processor with 1Gb of RAM— using sets of 100 random

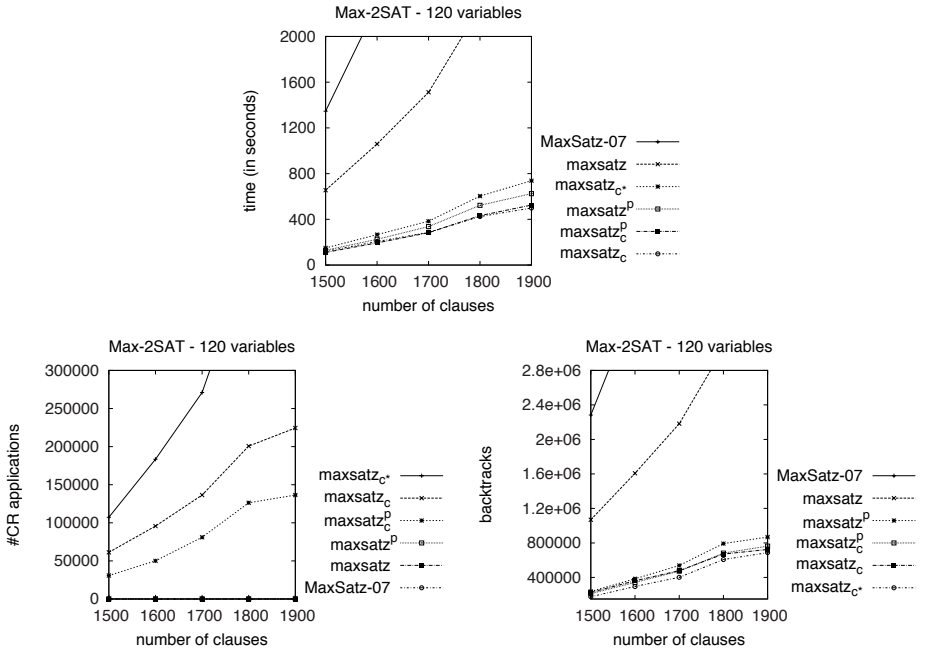

Fig. 1. Random Max-2SAT

Max-2SAT instances with 120 variables; the number of clauses ranged from 1500 to 1900. The results are shown in Figure 1: mean time needed to solve an instance of the set (top plot), mean number of CR applications without counting the applications of Rule 3 and Rule 4 (bottom left), and mean search tree size (bottom right).

It is quite easy to detect an inconsistent subformula in a Max-2SAT instance using unit propagation or failed literal detection, especially when the clauses-to-variables ratio is high. Since a cycle structure implies a failed literal, and its complementary literal easily fails during search, most cycle structures are likely contained in an inconsistent subformula detected using failed literal detection, explaining the behaviour of the exhaustive applications of CR in the preprocessing of MaxSatz$_c^p$ and MaxSatz^p for random Max-2SAT, since most applications of CR probably allow to transform an inconsistent subformula. Nevertheless, MaxSatz$_c$, with guided CR applications aiming at transforming inconsistent subformulas, is always the best solver in terms of runtime: It is 5.4 times faster than MaxSatz for the hardest instances (1900 clauses). Notice that MaxSatz is substantially faster than MaxSatz-07 on Max-2SAT.

Additional experiments on Max-3SAT and Max-CUT instances, and on the instances from the 2007 MaxSAT Evaluation (not included here for lack of space) also provide empirical evidence that, in general, MaxSatz$_c$ outperforms the rest of solvers, and applying CR exhaustively is the worst alternative, making MaxSatz$_{c*}$ slower than MaxSatz for Max-3SAT.

References

1. Li, C.M., Manyà, F., Planes, J.: Exploiting unit propagation to compute lower bounds in branch and bound Max-SAT solvers. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 403–414. Springer, Heidelberg (2005)
2. Li, C.M., Manyà, F., Planes, J.: Detecting disjoint inconsistent subformulas for computing lower bounds for Max-SAT. In: Proceedings AAAI 2006, pp. 86–91 (2006)
3. Larrosa, J., Heras, F., de Givry, S.: A logical approach to efficient Max-SAT solving. Artificial Intelligence 172(2–3), 204–233 (2008)
4. Li, C.M., Manyà, F., Planes, J.: New inference rules for Max-SAT. Journal of Artificial Intelligence Research 30, 321–359 (2007)