STAIRS 2014
U. Endriss and J. Leite (Eds.)
© 2014 The Authors and IOS Press.
This article is published online with Open Access by IOS Press and distributed under the terms of the Creative Commons Attribution Non-Commercial License.
doi:10.3233/978-1-61499-421-3-1

On the Extension of Learning for Max-SAT

André ABRAMÉ and Djamal HABET
Aix Marseille Université, CNRS, ENSAM, Université de Toulon, LSIS UMR 7296, 13397, Marseille, France.
emails: {andre.abrame,djamal.habet}@lsis.org

Abstract. One of the most critical components of Branch & Bound (BnB) solvers for Max-SAT is the estimation of the lower bound. At each node of the search tree, they detect inconsistent subsets (IS) of the formula by unit propagation based methods and apply a treatment to them. The currently best performing Max-SAT BnB solvers perform a very little amount of memorization, thus the same IS may be detected and treated several times during the exploration of the search tree. We address in this paper the problem of increasing the learning performed by BnB solvers. We present new sets of clause patterns which produce unit resolvent clauses when they are transformed by max-resolution. We study experimentally the impact of these transformation' memorization in our solver AHMAXSAT and we discuss their effects on the solver behavior.

1. Introduction

The Max-SAT problem consists in finding, for a given CNF formula, a Boolean assignment of the variables of this problem which maximizes (minimizes) the number of satisfied (falsified) clauses. In the weighted version of Max-SAT, a positive weight is associated to each clause and the goal is to find an assignment which maximizes (minimizes) the sum of the weights of the satisfied (falsified) clauses. For clarity reasons, we use in this paper unweighted notations and examples. Nevertheless, all the presented results can easily be extended to weighted Max-SAT.

Among the complete methods for solving Max-SAT, Branch and Bound (BnB) algorithms (e.g. WMAXSATZ [10,8,9]) have shown their efficiency, especially on random and crafted instances. BnB solvers explore the whole search space and compare, at each node of the search tree, the current number of falsified clauses plus an (under-)estimation of the ones which will become falsified (the lower bound, LB) to the best solution found so far (the upper bound, UB). If LB $\geq$ UB, then no better solution can be found by extending the current branch and they perform a backtrack. The estimation of the remaining inconsistencies is a critical component of BnB solvers: it is one of the most time-consuming components of the solvers and its quality determines the number of explored nodes.

Efficient BnB Max-SAT solvers estimate the number of clauses which will become falsified by counting the disjoint inconsistent subsets (IS) of the formula. They use unit propagation (UP) based methods to detect inconsistencies and analyze the propagation steps which have led to them to build inconsistent subsets. Each detected IS must be treated to ensure it will be counted only once. Two treatments are actually used by BnB solvers. If an IS matches (completely or partially) some patterns, then solvers transform it (completely or partially) by applying several max-resolution steps [1,3,4,9,5] on its clauses, and they keep the modifications in the lower nodes of the search tree. This treatment acts as a (restricted) learning or memorizing mechanism. Otherwise, they simply remove the IS's clauses from the formula or apply the max-resolution based treatment and, in both cases, they keep the modifications only at the current node of the search tree (i.e. modifications are undone before the next decision).

We propose in this paper to increase the amount of learning performed by BnB solvers. We focus on the subsets of clauses which produce, once transformed by max-resolution, unit resolvent clauses. The benefits of memorizing such transformations are double. They reduce the number of redundant propagations and max-resolution transformations. Moreover, the produced unit clauses may empower the detection of IS by unit propagation. We define new patterns corresponding to these subsets. We study experimentally the impact of their transformation's memorization on the behavior of our solver AHMAXSAT by varying the sizes of the patterns and the sizes of their clauses. The results obtained show the interest of our approach and give interesting clues on the impact of the max-resolution transformations on the solver's behavior.

2. Preliminaries

We give in this section the definitions and notations used in this paper and we present the max-resolution inference rule, which is the keystone of the learning procedure for Max-SAT.

2.1. Definitions and Notations

A formula Φ in conjunctive normal form (CNF) defined on a set of propositional variables $X = \{x_1, \ldots, x_n\}$ is a conjunction of clauses. A clause c_j is a disjunction of literals and a literal l is a variable x_i or its negation $\overline{x}_i$. Alternatively, a formula can be represented as a multiset of clauses $\Phi = \{c_1, \ldots, c_m\}$ and a clause as a set of literals $c_j = \{l_{j_1}, \ldots, l_{j_k}\}$. An assignment can be represented as a set I of literals which cannot contain both a literal and its negation. If x_i is assigned to $true$ (resp. $false$) then $x_i \in I$ (resp. $\overline{x}_i \in I$). I is a complete assignment if $|I| = n$ and it is partial otherwise. A literal l is said to be satisfied by an assignment I if $l \in I$ and falsified if $\overline{l} \in I$. A variable which does not appear either positively or negatively in I is unassigned. A clause is satisfied by I if at least one of its literals is satisfied, and it is falsified if all its literals are falsified. By convention, an empty clause (denoted by $\square$) is always falsified. Eventually, solving the Max-SAT problem consists in finding a complete assignment which maximizes (minimizes) the number of satisfied (falsified) clauses of Φ.

2.2. max-resolution rule

The max-resolution inference rule [1,3,4] is the Max-SAT version of the SAT resolution. It is defined as follows:

$$\frac{c_i = \{x, y_1, \dots, y_s\},\ c_j = \{\bar{x}, z_1, \dots, z_t\}}{cr = \{y_1, \dots, y_s, z_1, \dots, z_t\},\ cc_1,\ \dots,\ cc_t,\ cc_{t+1},\ \dots,\ cc_{t+s}}$$

with: $cc_1 = \{x, y_1, \dots, y_s, \bar{z}_1, z_2, \dots, z_t\}$, $cc_2 = \{x, y_1, \dots, y_s, \bar{z}_2, z_3, \dots, z_t\}$, $\dots$, $cc_t = \{x, y_1, \dots, y_s, \bar{z}_t\}$, $cc_{t+1} = \{\bar{x}, z_1, \dots, z_t, \bar{y}_1, y_2, \dots, y_s\}$, $cc_{t+2} = \{\bar{x}, z_1, \dots, z_t, \bar{y}_2, y_3, \dots, y_s\}$, $\dots$, $cc_{t+s} = \{\bar{x}, z_1, \dots, z_t, \bar{y}_s\}$. The premises of the rule are the original clauses c_i and c_j which are removed from the formula, while the conclusions are the resolvent clause cr and the compensation clauses $cc_1, \dots, cc_{t+s}$ added to keep formula's equivalency.

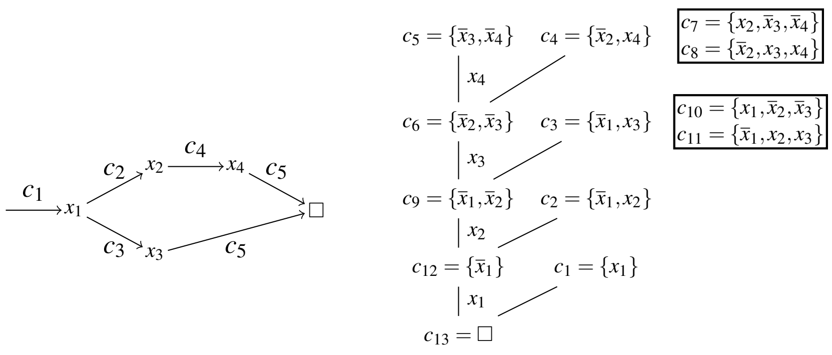
3. Learning in State of the Art BnB Solvers

At each node of the search tree, BnB solvers calculate the lower bound (LB) by estimating the number of disjoint inconsistent subsets (IS) remaining in the current formula. In this section, we first present the main techniques of IS detection and transformation. Then, we recall the learning scheme used by the best performing BnB solvers.

Recent BnB Max-SAT solvers apply unit propagation (UP) based methods to detect inconsistent subsets (more precisely simulated unit propagation (SUP) [7] and failed literals (FL) [8]). For each unit clause $\{l\}$, they remove all the occurrences of $\bar{l}$ from the clauses and all the clauses containing l. This process is repeated until an empty clause (a conflict) is found or no more unit clause remains. When an empty clause is found by UP, an inconsistent subset (IS) of the formula can be built by analyzing the propagation steps which have led to the conflict. The propagation steps made by SUP or FL can be represented by an implication graph [11], where the nodes are the assigned variables and the arrows connect the falsified literals of the unit clauses to the variables they propagate.

Once detected, an IS can be transformed by applying max-resolution operations between its clauses. The original clauses of the IS are removed from the formula, while a resolvent clause and compensation clauses are added. The resolvent is falsified by the current decisions. Thus, SUP or FL are not needed anymore to detect again the transformed IS. Consequently, by keeping the formula's transformations solvers memorize the IS and avoid its redundant detection and treatment. The following example illustrates the transformation of an IS by max-resolution.

Example 1 *Lets consider a formula $\Phi = \{c_1, \dots, c_5\}$ with $c_1 = \{x_1\}$, $c_2 = \{\bar{x}_1, x_2\}$, $c_3 = \{\bar{x}_1, x_3\}$, $c_4 = \{\bar{x}_2, x_4\}$ and $c_5 = \{\bar{x}_3, \bar{x}_4\}$. The application of SUP leads to the propagation sequence $< x_1@c_1, x_2@c_2, x_3@c_3, x_4@c_4 >$ (meaning that x_1 is propagated by c_1, then x_2 by c_2, etc.). The clause c_5 is empty and the corresponding implication graph is shown on Fig. 1(a). Hence, Φ is an inconsistent subset. Its transformation by the max-resolution rule is done as follows. Max-resolution is first applied between c_5 and c_4 on the variable x_4. The intermediary resolvent $c_6 = \{\bar{x}_2, \bar{x}_3\}$ is produced as well as the compensation clauses $c_7 = \{x_2, \bar{x}_3, \bar{x}_4\}$ and $c_8 = \{\bar{x}_2, x_3, x_4\}$. The original clauses c_4 and c_5 are removed from the formula. Then, the max-resolution is applied between the intermediary resolvent c_6 and the next original clause c_3 on the variable x_3 and so forth. Fig. 1(b) shows the max-resolution steps with in boxes the compensation clauses (note that this treatment is close to the conflict analysis procedure of modern SAT solvers [11]). After complete transformation, we obtain the formula $\Phi' = \{\square, c_7, c_8, c_{10}, c_{11}\}$ with $c_{10} = \{x_1, \bar{x}_2, \bar{x}_3\}$ and $c_{11} = \{\bar{x}_1, x_2, x_3\}$.*



c_1
x_1
c_2
x_2
c_4
x_4
c_5
c_3
x_3
c_5
$\square$
$c_5 = \{\bar{x}_3, \bar{x}_4\}$
$c_4 = \{\bar{x}_2, x_4\}$
$c_7 = \{x_2, \bar{x}_3, \bar{x}_4\}$
$c_8 = \{\bar{x}_2, x_3, x_4\}$
x_4
$c_6 = \{\bar{x}_2, \bar{x}_3\}$
$c_3 = \{\bar{x}_1, x_3\}$
$c_{10} = \{x_1, \bar{x}_2, \bar{x}_3\}$
$c_{11} = \{\bar{x}_1, x_2, x_3\}$
x_3
$c_9 = \{\bar{x}_1, \bar{x}_2\}$
$c_2 = \{\bar{x}_1, x_2\}$
x_2
$c_{12} = \{\bar{x}_1\}$
$c_1 = \{x_1\}$
x_1
$c_{13} = \square$

(a) Implication graph (b) Max-SAT resolution steps

Figure 1. Implication graph and Max-SAT resolution steps applied on the formula Φ from Example 1.

However, the Max-SAT clause learning scheme has two drawbacks which prevent its generalization. Firstly, the added resolvents and compensation clauses can increase quickly the size of the formula if learning is frequently used. Secondly, it may reduce the quality of the LB estimation and thus the number of explored nodes of the search tree may increase.

For the reasons cited above, the current best performing BnB solvers [6] keep the transformations made by the max-resolution rule only in the sub-part of the search tree (i.e. changes are undone when backtracking). Also, they restrict the application of this rule to particular sets of clauses corresponding to one (or a combination) of the following patterns (with on top the clauses of the original formula and on bottom the clauses obtained after transformation):

$$(1)\frac{\{\{x_1,x_2\},\{x_1,\bar{x}_2\}\}}{\{\{x_1\}\}},\ (2)\frac{\{\{x_1,x_2\},\{x_1,x_3\},\{\bar{x}_2,\bar{x}_3\}\}}{\{\{x_1\},\{x_1,x_2,x_3\},\{\bar{x}_1,\bar{x}_2,\bar{x}_3\}\}},\ (3)\frac{\{\{x_1\},\{\bar{x}_1,x_2\},\{\bar{x}_2,x_3\},\ldots,\{\bar{x}_{k-1},x_k\},\{\bar{x}_k\}\}}{\{\square,\{x_1,\bar{x}_2\},\{x_2,\bar{x}_3\},\ldots,\{x_{k-1},\bar{x}_k\}\}}$$

These patterns do not necessarily cover the IS entirely. Especially, the application of the max-resolution based transformation on patterns (1) or (2) produces a unit resolvent clause $\{x_1\}$. Memorizing such transformations reduce the number of redundant propagations and max-resolution steps. Moreover, the produced unit clauses can be used in the sub-part of the search tree to make further unit propagation steps, and thus it may improve the number of detected IS.

4. Extended Learning

The lower bound computation, based on the estimation of the number of disjoint inconsistent subsets, is a critical part of BnB Max-SAT solvers. Indeed, it is one of their most time-consuming components and the estimation quality guides the backtrack and thus determines the number of explored nodes in the search tree. Thus, a balance must be struck between the time spent computing this estimation and its quality. In this regard, learning seems a natural way to limit the time spent on the LB computation by making the IS detection and transformation more incremental.

We present in this section a generalization of the patterns producing unit resolvent clauses, which we name *Unit Clause Subsets*. We motivate this generalization and show that the corresponding patterns can be detected efficiently. Also, we give statistics on their occurrences in the instances of the benchmark used in Section 5.

4.1. Unit Clause Subset

We have seen in the previous section that two of the patterns used for memorization by the current best performing BnB solvers produce after transformation a unit resolvent clause. The goal of memorizing such transformations is double. On the one hand, it reduces the number of redundant propagations and on the other hand, it may empower the detection of inconsistencies in lower nodes of the search tree since more unit clauses are available for applying unit propagation.

We propose in this paper to extend the amount of learning performed by BnB solvers by considering more patterns which produce unit resolvent clauses when transformed by max-resolution. These patterns can be formally defined as follows.

Definition 1 (Unit Clause Subset (UCS)) *Let Φ be a CNF formula. A unit clause subset (UCS) is a set $\{c_{i_1},\ldots,c_{i_k}\} \subset \Phi$ with $\forall j \in \{1,\ldots,k\}$, $|c_{i_j}| > 1$ such that there exists an order of application of $k-1$ max-resolution steps on $c_{i_1},\ldots,c_{i_k}$ which produces a unit resolvent clause. We denote the set of the UCS's patterns of size k by k-UCS.*

Example 2 *Below the patterns of the 3-UCS set:*

$$(2)\ \frac{\{\{x_1,x_2\},\{x_1,x_3\},\{\bar{x}_2,\bar{x}_3\}\}}{\{\{x_1\},\{x_1,x_2,x_3\},\{\bar{x}_1,\bar{x}_2,\bar{x}_3\}\}},\quad (4)\ \frac{\{\{x_1,x_2\},\{\bar{x}_2,x_3\},\{x_1,\bar{x}_2,\bar{x}_3\}\}}{\{\{x_1\},\{\bar{x}_1,\bar{x}_2,x_3\}\}},$$

$$(5)\ \frac{\{\{x_1,x_2\},\{x_1,x_3\},\{x_1,\bar{x}_2,\bar{x}_3\}\}}{\{\{x_1\},\{x_1,x_2,x_3\}\}},\quad (6)\ \frac{\{\{x_1,x_2\},\{x_1,\bar{x}_2,x_3\},\{x_1,\bar{x}_2,\bar{x}_3\}\}}{\{\{x_1\}\}}$$

Since one of the goals of memorizing transformations which produce unit resolvent clauses is to increase the number of assignments made by unit propagation (SUP or FL), we do not consider the subsets of clauses which contain unit clauses. In the best case (if they contain only one unit clause), the transformation of such subsets lets the number of unit clauses of the formula unchanged. In the worst case (if they contain more than one unit clause), the transformed formula contains less unit clauses than the original one. Thus the number of assignments made by unit propagation and consequently the number of detected IS may be reduced. Eventually, we make a distinction between the patterns of the k-UCS sets depending on the size of their clauses. We denote k^b-UCS the subset of k-UCS composed of the patterns which contain only binary clauses and k^t-UCS the subset composed of the patterns which contain at least one ternary clause. It should be noted that the patterns (1) and (2) presented in Section 3 belong respectively to the sets 2-UCS and 3^b-UCS.

4.2. Detecting k-UCS

The k-UCS patterns are easily detectable by analyzing the implication graph. Indeed, the clauses which are between the conflict and the *first unit implication point* (FUIP) [11] produce a unit resolvent clause when transformed by max-resolution. Thus, solvers simply have to count the number and the sizes of clauses between the conflict and the FUIP to know if they are in presence of a valid UCS. This does not change the complexity of the conflict analysis procedure and the computational overhead is negligible.

4.3. Occurrences of k-UCS

We have measured the rate of occurrences of the k-UCS patterns in the inconsistent subsets detected by our solver AHMAXSAT. Tab. 1 show the results obtained on the benchmark presented in Section 5. It is interesting to observe that the patterns which are currently used for learning by state of the art solvers (2-UCS and 3^b-UCS) occur in less than 3.5% of the IS. On average, UCS are detected in 35% of the ISs. This value however varies considerably from one instance class to another.

Table 1. Percentage of occurrences of the k-UCS's patterns.

	Instances classes	2-UCS	3^b-UCS	3^t-UCS	4^b-UCS	4^t-UCS	5^b-UCS	5^t-UCS	k-UCS, $k > 5$
unweighted	crafted/bipartite	0	0.41	0	0	0.01	38.45	0.14	21.47
	crafted/maxcut	0	11.58	0	3.17	5.48	3.81	3.6	8.64
	random/highgirth	0.06	0.05	0.01	0.03	0.02	0.02	0.03	0.53
	random/max2sat	0	1.91	0	15.97	0.11	8.41	1.01	17.5
	random/max3sat	0.38	1.75	0.98	2.29	2.54	0.91	3.52	12.35
	random/min2sat	0	1.91	0	13.01	0.01	9.13	0.06	21.4
weighted	crafted/frb	0	5.36	0	0	8.05	1.17	0	5.08
	crafted/ramsey	0	1.06	0	0	0.19	0.12	0.21	0.85
	crafted/wmaxcut	0	12.99	0	0.66	8.25	3.87	5.39	10.37
	random/wmax2sat	0	2.13	0	17.69	0.09	9.56	1.21	14.58
	random/wmax3sat	0.17	1.02	0.52	1.53	1.55	0.66	2.36	8.77
	Total	0.06	3.33	0.15	6.72	1.87	8.99	1.61	13.14

5. Empirical Evaluation of UCS Learning

We present in this section an empirical evaluation of the impact of the k-UCS transformation's memorization on all the random and crafted instances from the unweighted and weighted categories of the Max-SAT Competition 2013[1]. Note that we do not include any (weighted) Partial Max-SAT instances nor industrial ones in our experiments. Even if the results presented in this paper can naturally be extended to these instance categories, our solver AHMAXSAT does not handle them efficiently. A performing BnB solver for (weighted) Partial Max-SAT must consider both the soft and the hard parts of the instances. Thus, it must include SAT mechanisms such as nogood learning, activity-based branching heuristic or backjumping and our solver currently does not [2,11]. For the industrial instances, solvers must have a very efficient memory management. None of the best performing BnB solvers (including ours) handles huge industrial instances efficiently[2].

We have implemented the UCS learning scheme in our BnB solver AHMAXSAT[3]. The experiments are performed on machines equipped with Intel Xeon 2.4 Ghz proces-

[1] Available from http://maxsat.ia.udl.cat:81/13/benchmarks.

[2] See for instance http://maxsat.ia.udl.cat:81/13/results/index.html

[3] An early version of AHMAXSAT has been submitted to the Max-SAT Competition 2013. It was the version 1.16. Since that competition, we have made numerous optimizations. The version presented in this paper is numbered 1.52.

sors and 24 Gb of RAM and running under a GNU/Linux operating system. The cutoff time is fixed to 1800 seconds per instance, as in the Max-SAT Competitions.

In the rest of this section, we present and discuss the results obtained by increasing the learning performed by AHMAXSAT. Starting from a variant using the patterns used by state of the art solvers ($\{2,3^b\}$-UCS), we add the memorization of the 3^t-UCS transformations, then the 4-UCS ones and the 5-UCS ones. Preliminary results (not reported in this paper) suggest that memorizing k-UCS transformations with $k > 5$ has a negative impact on the solver performances. The obtained results are presented in the Tables 2, 3 and 4. For each AHMAXSAT variant, we present the number of solved instances (columns S) and the averages of: numbers of decisions (columns D), solving time (columns T), number of propagations per decision (columns P/D) and number of IS detected per decision (columns □/D). The main goal of these two last indicators is to show the reduction of the redundant propagations and IS detections. They may also indicate a loss in the quality of the LB estimation, which is one of the known drawbacks of the memorization.

$\{2,3^t\}$-UCS vs. $\{2,3\}$-UCS As one can have expected, memorizing the transformations of the 3^t-UCS in addition to the ones of the patterns used by state of the art BnB solvers (i.e. $\{2,3^b\}$-UCS) does not change much the behavior of AHMAXSAT since these patterns occur very rarely in the benchmark's instances. On the instances with a non-null rate of occurrences (random/max3sat and random/wmax3sat), we can observe a slight reduction of the average number of decisions (column D) and of the average solving time (column T). On the overall benchmark, the average solving time is reduced by 2%.

$\{2,3\}$-UCS vs. $\{2,3,4\}$-UCS If we add the memorization of the transformations of the 4-UCS, both the averages of the number of propagations per decision (column P/D) and of the number of inconsistencies detected per decision (column □/D) decrease significantly (respectively -13% and -26%). Since the amount of memorization increases, one can expect a slight reduction of these two values, but not in such proportion. An important reduction probably indicates a loss in the quality of the LB estimation. This explanation is confirmed by the increase of the average number of decisions (+8% on average). Consequently, 4 less instances are solved and the solving time is on average 36% higher.

It is interesting to notice that the impact of memorizing the 4-UCS transformations vary from one instance's class to another. The loss in performances occurs on the classes with a high percentage of 4^b-UCS occurrences (random/max2sat and random/wmax2sat where the average solving time increases of 244% and 171% respectively). Conversely, on the instance's classes with a low percentage of occurrences of 4^b-UCS (crafted/frb and crafted/wmaxcut), the average solving time is reduced (-35% and -34% respectively).

$\{2,3\}$-UCS vs. $\{2,3,4^t\}$-UCS If we ignore the 4^b-UCS and memorize only the transformations of the $\{2,3,4^t\}$-UCS, both the average number of decisions and the average solving time are equal or reduced on all the instance classes (respectively -14% and -7% on average) and one more instance is solved. One can observe that the average numbers of propagations per decision and of IS detected per decision decrease slightly (respectively -0.5% and -2.5% on average).

$\{2,3,4^t\}$-UCS vs. $\{2,3,4^t,5\}$-UCS The same behavior as for the 4-UCS can be observed if we add the memorization of the transformations of the 5-UCS. Both the average number of propagations per decision and the average number of inconsistencies detected per decision decrease significantly (-20% and -35%). Consequently, the average number of decisions increases (+42%), 7 instances less are solved and the average solving time increases of 90%. As for the 4-UCS, the loss is particularly high on instance classes with a high percentage of 5^b-UCS occurrences (crafted/bipartite, random/max2sat, random/min2sat and random/wmax2sat where the average solving time increase of respectively 235%, 322%, 188% and 261%).

Table 2. Detailed results of the variants AHMAXSAT $\{2,3^b\}$-UCS and AHMAXSAT $\{2,3\}$-UCS. The two first columns give respectively the instances classes and the number of instances per class.

	Instances		AHMAXSAT $\{2,3^b\}$-UCS					AHMAXSAT $\{2,3\}$-UCS				
		#	S	D	T	P/D	□/D	S	D	T	P/D	□/D
unweighted	crafted/bipartite	100	100	35392	96.9	2267	148	100	35416	97.2	2267	148
	crafted/maxcut	67	56	235202	58.9	274	42	56	235202	59.5	274	42
	random/highgirth	82	7	4953454	1194.1	87	5	7	4991152	1199.3	87	5
	random/max2sat	100	100	40410	84.9	1860	99	100	40408	84.7	1859	99
	random/max3sat	100	98	425744	332.6	432	46	98	407917	315.6	429	45
	random/min2sat	96	96	1025	2.5	1937	64	96	1019	2.5	1937	64
weighted	crafted/frb	34	14	494542	77	79	12	14	494542	76.7	79	12
	crafted/ramsey	15	4	158350	55.4	26	10	4	158350	55.3	26	10
	crafted/wmaxcut	67	62	40645	60.6	409	169	62	40649	60.6	409	169
	random/wmax2sat	120	120	4337	54.4	4957	534	120	4307	53.7	4955	535
	random/wmax3sat	40	40	49771	138	1142	202	40	48515	134.8	1140	201
	Global	821	697	157583	114.5	1902	173	697	155380	111.9	1901	173

Table 3. Detailed results of the variants AHMAXSAT $\{2,3,4\}$-UCS and AHMAXSAT $\{2,3,4^t\}$-UCS.

	Instances		AHMAXSAT $\{2,3,4\}$-UCS					AHMAXSAT $\{2,3,4^t\}$-UCS				
		#	S	D	T	P/D	□/D	S	D	T	P/D	□/D
unweighted	crafted/bipartite	100	100	35456	96.9	2266	148	100	35478	97.3	2266	148
	crafted/maxcut	67	56	153539	37.5	207	30	56	153326	42.3	267	39
	random/highgirth	82	7	4960255	1186.5	88	5	6	4603981	1091.2	87	5
	random/max2sat	100	94	175425	291.8	1375	62	100	38878	81.4	1854	98
	random/max3sat	100	100	443640	319.5	401	41	100	413013	308.4	414	43
	random/min2sat	96	96	1198	2.3	1433	34	96	1025	2.5	1936	64
weighted	crafted/frb	34	14	289467	49.3	71	9	14	288458	48.6	71	9
	crafted/ramsey	15	4	158040	56.6	26	10	4	157902	55.4	26	10
	crafted/wmaxcut	67	62	21703	35.1	395	143	62	21757	35.2	397	142
	random/wmax2sat	120	120	13386	145.8	4434	353	120	4250	52.9	4957	531
	random/wmax3sat	40	40	46640	126.3	1114	192	40	45956	126.6	1126	196
	Global	821	693	169317	152	1656	128	698	133829	103.7	1892	169

Table 4. Detailed results of the variants AHMAXSAT $\{2,3,4^t,5\}$-UCS and AHMAXSAT $\{2,3,4^t,5^t\}$-UCS.

	Instances		AHMAXSAT $\{2,3,4^t,5\}$-UCS					AHMAXSAT $\{2,3,4^t,5^t\}$-UCS				
		#	S	D	T	P/D	□/D	S	D	T	P/D	□/D
unweighted	crafted/bipartite	100	99	185669	326,1	1365	60	100	34909	94,7	2258	147
	crafted/maxcut	67	56	188088	46,7	240	33	56	177964	43,8	251	35
	random/highgirth	82	6	4607524	1101,1	87	5	6	4597721	1102,3	87	5
	random/max2sat	100	94	214125	344,2	1327	59	100	38841	77,8	1793	91
	random/max3sat	100	100	450338	309,3	380	38	100	426143	293,5	385	39
	random/min2sat	96	96	3856	7,2	1484	37	96	979	2,4	1931	64
weighted	crafted/frb	34	14	247556	42,4	66	8	14	210245	37,2	70	8
	crafted/ramsey	15	4	157332	56,0	26	10	4	157478	55,5	26	10
	crafted/wmaxcut	67	62	21521	33,4	347	117	62	19993	30,8	346	116
	random/wmax2sat	120	120	16751	191,0	4322	333	120	3898	48,2	4844	504
	random/wmax3sat	40	40	45874	123,4	1093	187	40	44804	120,3	1100	188
	Global	821	691	190109	197,2	1504	109	698	135686	99,1	1850	159

$\{2,3,4^t\}$-UCS *vs.* $\{2,3,4^t,5^t\}$-UCS As previously, if we ignore the 5^b-UCS and memorize only the transformation of the $\{2,3,4^t,5^t\}$-UCS there is no significant loss in solving time on any instance classes and the average solving time is reduced by 4,5%.

To summarize. by memorizing the transformations of the patterns of the sets $\{3^t,4^t,5^t\}$-UCS in addition to the sets $\{2,3^b\}$-UCS used by state of the art solvers, our solver AHMAXSAT solves one instance more. The average number of decisions and the average solving time are both reduced by 14%. It should be noted that the increase of the formula's size is limited and does not affect the solver efficiency.

Discussion It is commonly admitted that memorizing the transformations made by the max-resolution rule may in some cases reduce the quality of the LB estimation. However, to the best of our knowledge, the reasons of this behavior have never been properly described. The empirical study we have performed shows that the transformations of some specific patterns (i.e. the 4^b-UCS and the 5^b-UCS) seem particularly affected by this phenomenon. Moreover, the detailed statistics obtained show a correlation between a decrease of the number of propagations, a decrease of the number of detected IS and an increase of the number of decisions. Indeed, if the number of propagated variables is reduced, then less IS will be detected and the quality of the LB estimation will be reduced. Consequently, the backtracks will occur below in the search tree and more decisions will be needed to solve instances. We illustrate in the example below how the transformation of a 4^b-UCS decreases the number of propagated variables.

Example 3 *Lets consider the subset $\Phi'' = \{c_2,c_3,c_4,c_5\}$ of the formula Φ from Example 1. If we add the clause $c_{14} = \{\bar{x}_1,x_4\}$ to Φ'', there are two possible UCS: $\psi_1 = \{c_3,c_5,c_{14}\}$ and $\psi_2 = \{c_2,c_3,c_4,c_5\}$ which are respectively a 3^b-UCS and a 4^b-UCS. If ψ_1 is transformed by max-resolution, we obtain the formula $\Phi^{(3)} = \{c_2,c_4,c_{15},c_{16},c_{17}\}$ with $c_{15} = \{\bar{x}_1\}$, $c_{16} = \{\bar{x}_1,x_3,x_4\}$ and $c_{17} = \{x_1,\bar{x}_3,\bar{x}_4\}$. The assignment $x_1 = true$ leads to the following propagation sequence in $\Phi^{(3)}$: $< x_2@c_2, x_4@c_4 >$. The clause c_{15} is falsified while c_{16} and c_{17} are satisfied. If ψ_2 is transformed by max-resolution, we obtain the formula $\Phi^{(4)} = \{c_{12},c_7,c_8,c_{10},c_{11},c_{14}\}$ (this transformation is described in Example 1). The assignment $x_1 = true$ in $\Phi^{(4)}$ leads to the propagation sequence $< x_4@c_{14} >$. The clause c_{12} is falsified, c_8 and c_{10} are satisfied and c_7 and c_{11} are reduced. There is no more unit clauses, and x_2 cannot be propagated. It is interesting to notice that the two remaining reduced clauses $c_7 = \{x_2,\bar{x}_3\}$ and $c_{11} = \{x_2,x_3\}$ may lead to the propagation of x_2 if we apply the max-resolution between them. But the sole unit propagation mechanism is not sufficient to propagate x_2.*

Even if we have described here via an example how transformations may reduce the number of propagated variables and its impact on the quality of the LB estimation, the specific features of the transformed subsets of clauses concerned by this phenomenon are unclear. A thorough study of these characteristics would be of great interest to improve the BnB solvers learning procedure.

Finally, we have tested (using the protocol described previously) the two best performing BnB solvers of the Max-SAT Competition 2013: WMAXSATZ2009 and WMAXSATZ2013. The results (Tab. 5) shows that AHMAXSAT solves more instances than the two other solvers (respectively 41 and 7 more, columns S) and its average solving time is respectively 59% and 32% lower (columns T).

Table 5. Detailed comparison of AHMAXSAT $\{2,3,4^t,5^t\}$-UCS, WMAXSATZ2009 and WMAXSATZ2013.

Instances		#	WMAXSATZ2009			WMAXSATZ2013			AHMAXSAT		
			S	D	T	S	D	T	S	D	T
unweighted	crafted/bipartite	100	99	527295	268.7	99	796983	282.3	**100**	34909	**94.7**
	crafted/maxcut	67	55	850803	97.4	55	755340	54.7	**56**	177964	**43.8**
	random/highgirth	82	0	-	-	0	-	-	**6**	4597721	1102.3
	random/max2sat	100	96	666713	288.1	100	523266	169.8	100	38841	**77.8**
	random/max3sat	100	97	2211487	381.7	100	1476192	**242.9**	100	426143	293.5
	random/min2sat	96	77	648900	185.5	96	22402	9.4	96	979	**2.4**
weighted	crafted/frb	34	9	1379041	12.2	14	1537566	62.8	14	210245	**37.2**
	crafted/ramsey	15	4	876667	93.4	4	549137	**52.6**	4	157478	55.5
	crafted/wmaxcut	67	61	75186	80.8	**63**	126254	73.5	62	19993	30.8
	random/wmax2sat	120	119	82064	288.9	120	81440	134.2	120	3898	**48.2**
	random/wmax3sat	40	40	328504	177.1	40	257175	130.7	40	44804	**120.3**
	Total	821	657	716729	240.2	691	541647	145	**698**	135686	**99.1**

6. Conclusion

We have presented in this paper new sets of patterns which produce, when transformed by max-resolution, unit resolvent clauses. The experimental study shows that the transformation' memorization of some pattern sets (namely the $\{2,3,4^t,5^t\}$-UCS) reduces significantly both the number of decisions made by our solver AHMAXSAT and its solving time. These experiments show that the transformation's memorization of the $\{4^b,5^b\}$-UCS reduces the solver's capability to detect inconsistencies and thus its performances. We have described this phenomenon, which had never been done before to the best of our knowledge. As future work, we will make a thorough study of this phenomenon to draw up a general learning framework for BnB Max-SAT solvers.

References

[1] M. L. Bonet, J. Levy, and F. Manyà, 'Resolution for max-sat', *Artificial Intelligence*, **171**(8-9), 606–618, (2007).

[2] N. Eén and N. Sorensson, 'An extensible sat-solver', in *SAT'03*, pp. 502–518. Springer, (2003).

[3] F. Heras and J. Larrosa, 'New inference rules for efficient max-sat solving', in *AAAI'06*, volume 1, pp. 68–73. AAAI Press, (2006).

[4] J. Larrosa and F. Heras, 'Resolution in max-sat and its relation to local consistency in weighted csps', in *IJCAI'05*, pp. 193–198. Morgan Kaufmann Publishers Inc., (2005).

[5] J. Larrosa, F. Heras, and S. de Givry, 'A logical approach to efficient max-sat solving', *Artificial Intelligence*, **172**(23), 204–233, (2008).

[6] C. M. Li, F. Manyà, N. Mohamedou, and J. Planes, 'Exploiting cycle structures in max-sat', in *SAT'09*, pp. 467–480, Springer Berlin / Heidelberg, (2009).

[7] C. M. Li, F. Manyà, and J. Planes, 'Exploiting unit propagation to compute lower bounds in branch and bound max-sat solvers', in *CP'05*, pp. 403–414, Springer Berlin / Heidelberg, (2005).

[8] C. M. Li, F. Manyà, and J. Planes, 'Detecting disjoint inconsistent subformulas for computing lower bounds for max-sat', in *AAAI'06*, pp. 86–91. AAAI Press, (2006).

[9] C. M. Li, F. Manyà, and J. Planes, 'New inference rules for max-sat', *Journal of Artificial Intelligence Research*, **30**, 321–359, (2007).

[10] C. M. Li, F. Many, N. Mohamedou, and J. Planes, 'Resolution-based lower bounds in maxsat', *Constraints*, **15**(4), pp. 456–484, (2010).

[11] J. P. Marques-Silva and K. A. Sakallah, 'Grasp: A search algorithm for propositional satisfiability', *IEEE Transactions on Computers*, **48**(5), pp. 506–521, (August 1999).