Towards Solving Many-Valued MaxSAT*

Josep Argelich
Universitat de Lleida, Spain
jargelich@diei.udl.es

Xavier Domingo
GPS MicroSAT, Spain
xdomingo@gpsmicrosat.com

Chu-Min Li
Université de Picardie, France
cli@laria.u-picardie.fr

Felip Manyà
IIIA-CSIC, Spain
felip@iiia.csic.es

Jordi Planes
Universitat de Lleida, Spain
jplanes@diei.udl.es

Abstract

We define the MaxSAT problem for many-valued CNF formulas, called many-valued MaxSAT, and establish its complexity class. We then describe a basic branch and bound algorithm for solving many-valued MaxSAT, and an exact many-valued MaxSAT solver we have implemented. Finally, we report the experimental investigation we have performed to compare our solver with Boolean MaxSAT solvers on graph coloring instances. The results obtained indicate that many-valued CNF formulas can become a competitive formalism for representing and solving combinatorial optimization problems.

1 Introduction

In recent years there has been an increasing interest in developing efficient SAT solvers and SAT encodings aimed at solving computationally difficult NP-complete decision problems [15]. Given the good results achieved on a variety of combinatorial decision problems, the SAT community has recently devoted considerable attention to design and implement efficient MaxSAT solvers with the aim of solving NP-hard combinatorial optimization problems [1, 12, 22, 21, 2, 16]. Nowadays, Boolean MaxSAT solvers are able to solve random 2-SAT and random 3-SAT instances with 100 variables up to three orders of magnitude faster than just five years ago.

The main objective of our work is to start exploring the role that many-valued CNF formulas can play on solving NP-hard combinatorial optimization problems via their reduction to many-valued MaxSAT. On the one hand,

many-valued CNF formulas offer a good formalism for representing and solving NP-complete decision problems. On the other hand, to the best of our knowledge, the MaxSAT problem for many-valued CNF formulas (many-valued MaxSAT) has not been investigated so far, and no exact solver is available for that problem.

In this paper we define many-valued MaxSAT and establish its complexity class. We then describe a basic branch and bound algorithm for solving many-valued MaxSAT, and an exact many-valued MaxSAT solver we have implemented. Finally, we report the experimental investigation we have performed to compare our solver with Boolean MaxSAT solvers on graph coloring instances. The results obtained indicate that many-valued CNF formulas can become a competitive formalism for representing and solving NP-hard combinatorial optimization problems.

This work is part of a research program about many-valued satisfiability we have carried out during the last decade [3, 7, 8, 9, 10, 11, 19, 4, 5, 6]). Our aim is bridging the gap between Boolean SAT/MaxSAT encodings and constraint satisfaction formalisms. The challenge is to combine the inherent efficiencies of Boolean SAT/MaxSAT solvers operating on uniform encodings with the much more compact and natural representations, and more sophisticated propagation techniques of CSP/MaxCSP formalisms.

The paper is structured as follows. In Section 2, we define the many-valued clausal forms, many-valued MaxSAT and establish its complexity class. In Section 3 we describe a basic branch and bound algorithm for many-valued MaxSAT, and analyze the differences between Boolean MaxSAT solvers and many-valued MaxSAT solvers. In Section 4 we describe the many-valued MaxSAT solver we implemented. In Section 5, we report the empirical investigation.

*Research partially supported by projects TIC2003-00950 and TIN2004-07933-C03-03 funded by the *Ministerio de Educación y Ciencia*. The fourth author is supported by a grant *Ramón y Cajal*.

2 Definitions and Complexity

We first formally define the syntax and semantics of signed CNF formulas, and then of monosigned CNF formulas. Next, we define the MaxSAT problem for monosigned CNF formulas, and prove its NP-hardness.

We focus our attention on monosigned CNF formulas in the present paper. Our experience on encoding combinatorial problems as many-valued SAT instances indicates that most of the problems can be naturally represented with that kind of formulas.

Definition 1. A *truth value set*, or *domain*, N is a non-empty finite set $\{i_1, i_2, \dots, i_n\}$ where $n \in \mathbb{N}$. The cardinality of N is denoted by $|N|$. A total order $\leq$ is associated with N .

Definition 2. A *sign* is a set $S \subseteq N$ of truth values. A *signed literal* is an expression of the form $S : p$, where S is a sign and p is a propositional variable. The *complement* of a signed literal $S : p$, denoted by $\overline{S} : p$, is $(N \setminus S) : p$. A *signed clause* is a disjunction of signed literals. A *signed CNF formula* is a *multiset* of signed clauses.

Definition 3. An *assignment* is a mapping that assigns to every propositional variable an element of the truth value set. An assignment I *satisfies* a signed literal $S : p$ iff $I(p) \in S$, *satisfies* a signed clause C iff it satisfies at least one of the signed literals in C , and *satisfies* a signed CNF formula Γ iff it satisfies all clauses in Γ . A signed CNF formula is *satisfiable* iff it is satisfied by at least one assignment; otherwise it is *unsatisfiable*.

Definition 4. A sign S is *monosigned* if it either is a singleton (i.e. it contains exactly one truth value) or the complement of a singleton. A signed literal $S : p$ is a *monosigned literal* if its sign S is monosigned. A monosigned literal is *positive* if it is identical to $\{i\} : p$, and is *negative* if it is identical to $\{\bar{i}\} : p$, for some $i \in N$. A signed clause (a signed CNF formula) is a *monosigned clause* (a *monosigned CNF formula*) if all its literals are monosigned.

In the following, we represent positive monosigned literals of the form $\{i\} : p$ by $p = i$, and negative monosigned literals of the form $\{\bar{i}\} : p$ by $p \neq i$.

Definition 5. The MaxSAT problem for a monosigned CNF formula ϕ , or many-valued MaxSAT, is the problem of finding an assignment that minimizes the number of unsatisfied clauses (or equivalently, that maximizes the number of satisfied clauses).

Example 1. We want to solve the problem of coloring a graph with two colors in such a way that the minimum number of adjacent vertices are colored with the same color. If we consider the graph with vertices $\{v_1, v_2, v_3\}$ and with edges $\{(v_1, v_2), (v_1, v_3), (v_2, v_3)\}$, that problem is encoded as a many-valued MaxSAT instance as follows: the set of truth values is $\{1, 2\}$ and the set of propositional variables is $\{v_1, v_2, v_3\}$, and (ii) there are two binary monosigned clauses for every edge:

$$\begin{aligned} &\{(v_1 \neq 1 \vee v_2 \neq 1), (v_1 \neq 2 \vee v_2 \neq 2), \\ &(v_1 \neq 1 \vee v_3 \neq 1), (v_1 \neq 2 \vee v_3 \neq 2), \\ &(v_2 \neq 1 \vee v_3 \neq 1), (v_2 \neq 2 \vee v_3 \neq 2)\} \end{aligned}$$

An assignment that minimizes the number of unsatisfied clauses provides a coloring that minimizes the number of adjacent vertices with the same color.

Proposition 6. *Many-valued MaxSAT is NP-hard.*

Proof. Since the SAT problem for monosigned CNF formulas is NP-complete [5], it follows that the MaxSAT problem for monosigned CNF formulas is NP-hard. For a monosigned CNF formula ϕ with m clauses, the SAT problem is equivalent to prove that there exists an assignment that satisfies at least m clauses of ϕ .

3 A Basic Branch and Bound Algorithm

In this section we describe a basic branch and bound algorithm that we have designed for solving many-valued MaxSAT. Based on that description, we describe the solver we have implemented in the next section.

The space of all possible assignments for a monosigned CNF formula ϕ can be represented as a search tree, where internal nodes represent partial assignments and leaf nodes represent complete assignments. Our branch and bound algorithm for many-valued MaxSAT 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 ($unsat$) plus an underestimation of the number of clauses that will become unsatisfied if we extend the current partial assignment into a complete assignment (*underestimation*). The sum $unsat + underestimation$ is called lower bound (LB). Obviously, if $UB \leq LB$, a better assignment 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 $UB > LB$, it extends the current partial assignment by instantiating one more variable, say p ; which leads to the

Input: $mv\text{-maxsat}(\phi, UB)$: A CNF formula ϕ and an upper bound UB

```

1: if  $\phi = \emptyset$  or  $\phi$  only contains empty clauses then
2:   return  $\#empty\text{-clauses}(\phi)$ 
3: end if
4: if  $LB(\phi) \geq UB$  then
5:   return  $\infty$ 
6: end if
7:  $p \leftarrow select\text{-variable}(\phi)$ 
8: for all  $i \in N_p$  do
9:    $UB \leftarrow \min(UB, mv\text{-maxsat}(\phi_{p=i}, UB))$ 
10: end for
11: return  $UB$ 

```

Output: The minimal number of unsatisfied clauses of ϕ

Figure 1. Basic branch and bound algorithm for many-valued MaxSAT

creation of $|N|$ branches from the current branch: the left branch corresponds to assigning the first truth value to p , the next branch corresponds to assigning the second truth value to p , and so on. In that case, the formula associated with a branch is obtained from the formula of the current node by applying the monosigned one-literal rule [17] using the monosigned literal $p = i$, where i is the truth value assigned to p in that branch. The solution to MaxSAT is the value that UB takes after exploring the entire search tree.

Given a monosigned CNF formula ϕ , applying the monosigned one-literal rule using the monosigned literal $p = i$ amounts to delete all the clauses of ϕ that contain the literal $p = i$ and all the clauses of ϕ that contain a literal of the form $p \neq k$, where $k \neq i$. Besides, all the literals $p \neq i$ and all the literals of the form $p = k$, where $k \neq i$, are removed from ϕ . Notice that, in contrast to monosigned SAT solvers, we cannot apply the one-literal rule to all the unit clauses of the formula under consideration (i.e., we cannot apply what is known as unit propagation) because we cannot guarantee that the solution obtained is optimal. For instance, given the monosigned CNF formula $p = 1 \wedge p \neq 1 \wedge (p \neq 1 \vee p = 3) \wedge (p \neq 1 \vee p \neq 3)$ we get two empty clauses if we start applying unit propagation using the unit clause $p = 1$, but we get one empty clause if we start applying unit propagation using the unit clause $p \neq 1$.

Figure 1 shows the pseudo-code of our basic branch and bound algorithms for many-valued MaxSAT. We use the following notation:

- $\#empty\text{-clauses}(\phi)$ is a function that returns the number of empty clauses of ϕ .
- $LB(\phi)$ is a lower bound for ϕ .

- UB is an upper bound of the number of unsatisfied clauses in an optimal solution. We assume that the initial value is ∞ .
- $select\text{-variable}(\phi)$ is a function that returns a variable of ϕ following an heuristic.
- N_p is the current domain of variable p .
- $\phi_{p=i}$ is the formula obtained by applying the one-literal rule to ϕ using the monosigned literal $p = i$.

An implementation of that basic algorithm should be augmented with preprocessing techniques, the computation of an initial upper bound by a local search algorithm, clever variable selection heuristics, powerful inference techniques, lower bounds of good quality, and efficient data structures.

We close this section by pointing out some differences between many-valued MaxSAT solvers and Boolean MaxSAT solvers:

- **Branching:** Boolean solvers traverse binary trees, while many-valued solvers traverse $|N|$ -ary trees. We show in the experimental investigation that $|N|$ -ary trees lead to best performance profiles.
- **Lower bounds:** Most of the existing Boolean MaxSAT solvers use the following lower bound [20]:

$$LB(\phi) = unsat + \sum_{p \text{ occurs in } \phi} min(ic(p), ic(\neg p)),$$

where ϕ is the formula associated with the current partial assignment, $unsat$ is the number of clauses unsatisfied by the current partial assignment, and $ic(p)$ ($ic(\neg p)$) —inconsistency count of p ($\neg p$)— is the number of clauses that become unsatisfied if the current partial assignment is extended by fixing p to true (false); in other words, $ic(p)$ ($ic(\neg p)$) coincides with the number of unit clauses of ϕ that contain $\neg p$ (p). In the lower bound that we define below for many-valued MaxSAT, we only compute the inconsistency count of positive literals.

- **Variable selection heuristics:** In Boolean encodings, the domain size of the many-valued variables of the original problem is hidden in the encoding. This structural information is not exploited in variable selection heuristics. For example, the heuristic that consists of considering first variables of minimum domain size can be used in many-valued solvers, but not in Boolean solvers.
- The fact that any many-valued assignment assigns exactly one value of the domain to each many-valued variable is implicit in many-valued MaxSAT. In

Boolean MaxSAT, a many-valued variable with a domain N is represented by $|N|$ Boolean variables plus a set of clauses (usually called ALO and AMO) that ensure that any satisfying Boolean assignment assigns exactly one truth value to the many-valued variable. In contrast to many-valued MaxSAT solvers, Boolean MaxSAT solvers explore assignments in which more than one value of the domain is assigned to a certain many-valued variable. Suppose Boolean variables p_1, p_2 , and p_3 encode a many-valued variable p with domain $\{1, 2, 3\}$, a Boolean MaxSAT solver explores, for example, assignments containing $p_1 = 1, p_2 = 0$, and $p_3 = 1$. This type of assignments, in which more than one Boolean variable is true, are not considered by many-valued MaxSAT solvers.

4 Description of Mv-MaxSAT

Our solver, called Mv-MaxSAT, implements the previous basic branch and bound algorithm augmented with a number of improvements that we explain below.

4.1 Computation of the lower bound

We incorporated into Mv-MaxSAT the following original lower bound:

$$LB(\phi) = unsat + \sum_p \text{occurs in } \phi \min(ic(p=1), ic(p=2), \dots, ic(p=n))$$

where $\{1, 2, \dots, n\}$ is the current domain of variable p , ϕ is the monosigned CNF formula associated with the current partial assignment, $unsat$ is the number of clauses unsatisfied by the current partial assignment, and $ic(p=i)$ — inconsistency count of $p=i$ — is the number of clauses that become unsatisfied if the current partial assignment is extended by fixing p to i ; in other words, $ic(p=i)$ coincides with the number of unit clauses of ϕ that contain the literal $p \neq i$ plus the unit clauses that contain literals of the form $p = k$, where $k \neq i$. Observe that we do not need to compute the inconsistency count of negative literals.

4.2 Variable selection heuristic

Mv-MaxSAT incorporates a variable selection heuristic that takes into account the domain size of the variables: It instantiates first the variables with minimum domain size that appear most often. The heuristic of Mv-MaxSAT is static in order to take advantage of the very simple lazy data structures we have defined (see next section). To compute the domain size of a variable we only take into account the truth values of that variable that occur in the input formula.

4.3 Lazy data structures

Clauses are represented by ordered lists, and we have a pointer to the penultimate literal and to the last literal of the clause. The ordering used to represent clauses is the ordering provided by the static variable selection heuristic. When a monosigned literal L is fixed to true, the clauses whose penultimate literal has the same variable as L are evaluated. If any of the instantiated literals in the clause is true, the clause becomes satisfied; otherwise, we derive a unit clause whose only literal is the last literal of the clause. This approach has two advantages: the cost of maintaining that data structure when the solvers backtracks is constant (we do not have to undo pointers like in adjacency lists) and, at each step, we evaluate a minimum number of clauses (we do not evaluate all the clauses that contain the variable we are instantiating, we only evaluate the clauses in which the penultimate literal contains that variable). In addition, we also maintain an array that contains, for each literal, the unit clauses in which that literal appears. This array is used to derive empty clauses and to compute lower bounds. This array is the only data structure that the algorithm maintains when it backtracks.

5 Experimental Investigation

We conducted an experimental investigation —on a 2GHz Pentium IV with 512 Mb of RAM under Linux— in order to compare the performance of Mv-MaxSAT with two state-of-the-art Boolean MaxSAT solvers:

- GLMS [14]: a MaxSAT solver that encodes the input instance as a constraint network and solves that network with a state-of-the-art Max-CSP solver with a sophisticated and good performing lower bound. It was developed by Givry, Larrosa, Meseguer & Schiex and presented at CP-2003.
- AMP [2]: a branch and bound MaxSAT solver with lazy data structures and a static variable selection heuristic. It was developed by Alsinet, Manyà & Planes and presented at SAT-2005.

As benchmarks we used graph coloring instances, and solved the problem of minimizing the number of adjacent vertices with the same color (cf. Example 1). We considered randomly generated graph coloring instances, which were generated with the generator of Culberson [13].

In the first experiment we created a sample formed by 4 sets of 100 instances; the number of vertices was fixed

to 12, the minimum number of colors to ensure that all adjacent vertices have different colors was 8, and we generated unsatisfiable many-valued MaxSAT instances where the number of colors ranged from 2 to 5. We incorporated different kinds of branching into Mv-MaxSAT: binary, n-ary and regular¹, and compared their performance profile on the same instances. The results obtained are shown in Figure 2, where along the vertical axis we show the mean time needed to solve an instance of a set of 100 instances. We observe that the n-ary branching is superior to the rest of branchings, and we decided to fix that branching in Mv-MaxSAT. It is important to notice that this branching can only be applied when we encode the problem using many-valued variables. In Boolean encodings, that information is hidden in the encoding.

Figure 2. Comparison of different kinds of branchings in Mv-MaxSAT

In the second experiment we compared Mv-MaxSAT with two state-of-the-art MaxSAT solvers: GLMS and AMP [2]. In this case, we created a sample formed by 7 sets of 100 instances; the number of vertices was fixed to 15, the minimum number of colors to ensure that all adjacent vertices have different colors was 10, and we generated unsatisfiable MaxSAT instances where the number of colors ranged from 2 to 8. The results obtained are shown in Figure 3, where along the vertical axis we show the mean time needed to solve an instance of a set of 100 instances. We observe that Mv-MaxSAT has a good performance profile; it is up to 2 orders of magnitude faster than GLMS. Despite the fact that our results are preliminary, we believe

¹For all $i \in N$, let $\uparrow i$ denote the sign $\{j \in N \mid j \geq i\}$, where $\leq$ is the total order associated with N , and let $\overline{\uparrow i}$ denote the complement of $\uparrow i$. A sign S is *regular* if it either is identical to $\uparrow i$ (positive) or to $\overline{\uparrow i}$ (negative) for some $i \in N$. A signed literal $S : p$ is a *regular literal* if its sign S is regular. By a regular branching we mean a binary branching in which instead of having a branch for $p = i$ and another branch for $p \neq i$, we have a branch for $\uparrow i : p$ and another branch for $\overline{\uparrow i} : p$.

that Mv-MaxSAT can become a competitive formalism for representing and solving NP-hard combinatorial optimization problems. We should also take into account that the current version of Mv-MaxSAT could be improved considerably. For example, AMP uses a local search solver to get the initial upper bound while we take as initial upper bound ∞ . With just that improvement we can speed up our solver. Other improvements consists of adding preprocessing techniques and more inference at each node of the proof tree, as well as incorporating a lower bound of better quality.

Figure 3. Comparison between Mv-MaxSAT and the Boolean MaxSAT solvers AMP and GLMS

6 Concluding Remarks

We have explored a new research avenue: the use of many-valued MaxSAT solvers and encodings for representing and solving NP-hard combinatorial optimization problems. Here, we have presented the first contributions: the definition of the problem, its complexity class, a general branch and bound scheme for designing many-valued MaxSAT solvers, the first implementation of an exact many-valued MaxSAT solver, and an empirical investigation that provides experimental evidence that it is worth to pursue the work we have started in this paper.

Our plan is to continue working in several directions: (i) improving the existing implementation of Mv-MaxSAT; (ii) improving the quality of the lower bound; (iii) implementing a solver with dynamic variable selection heuristics; (iv) implementing a parallel version of the solver; (v) creating a testbed with a number of representative combinatorial optimization problems; and (vi) extending the results obtained to arbitrary signed CNF formulas.

References

- [1] J. Alber, J. Gramm, and R. Niedermeier. Faster exact algorithms for hard problems: A parameterized point of view. In *25th Conf. on Current Trends in Theory and Practice of Informatics*, Springer LNCS, pages 168–185, 1998.
- [2] T. Alsinet, F. Manyà, and J. Planes. Improved exact solver for weighted Max-SAT. In *Proceedings of SAT-2005, St. Andrews, Scotland*, pages 371–377. Springer LNCS 3569, 2005.
- [3] C. Ansótegui, R. Béjar, A. Cabiscol, C. M. Li, and F. Manyà. Resolution methods for many-valued CNF formulas. In *Proceedings of SAT-2002, Cincinnati, USA*, pages 156–163, 2002.
- [4] C. Ansótegui, J. Larrubia, and F. Manyà. Boosting Chaff’s performance by incorporating CSP heuristics. In *Proceedings of CP-2003, Kinsale, Ireland*, pages 96–107. Springer LNCS 2833, 2003.
- [5] C. Ansótegui and F. Manyà. New logical and complexity results for Signed-SAT. In *Proceedings of ISMVL-2003, Tokyo, Japan*, pages 181–187, 2003.
- [6] C. Ansótegui and F. Manyà. Mapping problems with finite-domain variables into problems with Boolean variables. In *Proceedings of SAT-2004, Vancouver, Canada*, pages 1–15. Springer LNCS 3542, 2004.
- [7] C. Ansótegui, F. Manyà, R. Béjar, and C. Gomes. Solving many-valued SAT encodings with local search. In *Proceedings of the Workshop on Probabilistic Approaches in Search, AAAI-2002, Edmonton, Canada*, 2002.
- [8] B. Beckert, R. Hähnle, and F. Manyà. Transformations between signed and classical clause logic. In *Proceedings of ISMVL-1999, Freiburg, Germany*, pages 248–255, 1999.
- [9] B. Beckert, R. Hähnle, and F. Manyà. The SAT problem of signed CNF formulas. In D. Basin, M. D’Agostino, D. Gabbay, S. Matthews, and L. Viganò, editors, *Labelled Deduction*, volume 17 of *Applied Logic Series*, pages 61–82. Kluwer, Dordrecht, 2000.
- [10] R. Béjar, R. Hähnle, and F. Manyà. A modular reduction of regular logic to classical logic. In *Proceedings of ISMVL-2001, Warsaw, Poland*, pages 221–226, 2001.
- [11] R. Béjar and F. Manyà. A comparison of systematic and local search algorithms for regular CNF formulas. In *Proceedings of ECSQARU-1999, London, England*, pages 22–31. Springer LNAI 1638, 1999.
- [12] B. Borchers and J. Furman. A two-phase exact algorithm for MAX-SAT and weighted MAX-SAT problems. *Journal of Combinatorial Optimization*, 2:299–306, 1999.
- [13] J. Culberson. Graph coloring page: The flat graph generator. See <http://web.cs.ualberta.ca/~joe/Coloring/Generators/>, 1995.
- [14] S. de Givry, J. Larrosa, P. Meseguer, and T. Schiex. Solving Max-SAT as weighted CSP. In *Proceedings of CP-2003, Kinsale, Ireland*, pages 363–376. Springer LNCS 2833, 2003.
- [15] H. A. Kautz and B. Selman. Ten challenges redux: Recent progress in propositional reasoning and search. In *Proceedings of CP-2003, Kinsale, Ireland*, pages 1–18. Springer LNCS 2833, 2003.
- [16] C. M. Li, F. Manyà, and J. Planes. Exploiting unit propagation to compute lower bounds in branch and bound Max-SAT solvers. In *Proceedings of CP-2005, Sitges, Spain*, pages 403–414. Springer LNCS 3709, 2005.
- [17] F. Manyà. *Proof Procedures for Multiple-Valued Propositional Logics*. PhD thesis, Universitat Autònoma de Barcelona, 1996. Published in [18].
- [18] F. Manyà. *Proof Procedures for Multiple-Valued Propositional Logics*. Number 9 in Monografies de l’Institut d’Investigació en Intel·ligència Artificial. IIIA-CSIC, Bellaterra (Barcelona), 1999.
- [19] F. Manyà, R. Béjar, and G. Escalada-Imaz. The satisfiability problem in regular CNF-formulas. *Soft Computing: A Fusion of Foundations, Methodologies and Applications*, 2(3):116–123, 1998.
- [20] R. Wallace and E. Freuder. Comparative studies of constraint satisfaction and Davis-Putnam algorithms for maximum satisfiability problems. In *Cliques, Coloring and Satisfiability*, volume 26, pages 587–615. American Mathematical Society, 1996.
- [21] Z. Xing and W. Zhang. Efficient strategies for (weighted) maximum satisfiability. In *Proceedings of CP-2004*, pages 690–705, 2004.
- [22] H. Zhang, H. Shen, and F. Manyà. Exact algorithms for MAX-SAT. In *Electronic Notes on Theoretical Computer science*, 86(1), 2003.