

ULRR

Lower bounds and upper bounds for MaxSAT

Item Type	Meetings and Proceedings
Authors	Heras, Federico;Morgado, Antonio;Marques-Silva, Joao
Citation	Learning and Intelligent OptimizatioN Conference LION 6. Lecture Notes in Computer Science;7219, pp. 402-407
Publisher	Springer
Download date	2026-03-16 21:47:51
Item License	https://creativecommons.org/licenses/by-nc-sa/1.0/
Link to Item	https://hdl.handle.net/10344/2768

Lower Bounds and Upper Bounds for MaxSAT ^{*}

Federico Heras, Antonio Morgado, and Joao Marques-Silva

CASL, University College Dublin, Ireland

Abstract. This paper presents several ways to compute *lower* and *upper bounds* for MaxSAT based on calling a complete SAT solver. Preliminary results indicate that (i) the bounds are of high quality, (ii) the bounds can boost the search of MaxSAT solvers on some benchmarks, and (iii) the upper bounds computed by a *Stochastic Local Search* procedure (*SLS*) can be substantially improved when its search is initialized with an assignment provided by a complete SAT solver.

1 Introduction

Weighted Partial MaxSAT (WPMS) [3] is a well-known optimization variant of Boolean Satisfiability (SAT) that finds a wide range of practical applications [3]. WPMS divides the formula in two sets of clauses: The *hard* clauses that must be satisfied and the *soft* clauses that can be unsatisfied with a penalty of their associated *weight*.

Early complete algorithms for MaxSAT solving were based on branch-and-bound search [3]. These algorithms perform very well on *crafted* and *random* instances, but are in general inefficient for industrial instances. An alternative approach is based on iteratively calling a SAT solver. The most widely used approach consists on relaxing the *soft* clauses and then iteratively refining upper bounds on the optimum solution (e.g. [2]). Recent work proposed to guide the search with *unsatisfiable subformulas* [3] (or *cores*) and is most often based on refining lower bounds (e.g. [4, 11, 1]). Other approaches refine both an upper bound and a lower bound [12]. Finally, a more recent approach based on combining *binary search* and *core-guided search* [7] computes the middle value between both bounds. Observe that all the above approaches could benefit from higher quality initial lower bounds and upper bounds to boost the search.

An alternative way to solve MaxSAT is *stochastic local search* (*SLS*). Such methods are incomplete but they can find approximate solutions for problem instances. However, SLS algorithms have a number of drawbacks. First, they are known to provide low quality solutions (ie. upper bounds) for industrial instances. Second, they are unable to take advantage of *partial MaxSAT* instances with hard and soft clauses.

This paper studies existing lower bounds and upper bounds based on calling a SAT solver, presents some improvements and relate them with recent work in the field. The empirical study shows that (i) SLS can improve its performance when initializing its search with an assignment computed by a complete SAT solver, (ii) the new bounds are tighter than the previous ones and finally that (iii) *core-guided MaxSAT algorithms* boost their performance when enhanced with the new bounds in some benchmarks.

^{*} This work was partially supported by SFI PI grant BEACON (09/IN.1/I2618).

2 Computing Lower and Upper Bounds

In this section lower bounds (LB) and upper bounds (UB) for MaxSAT are introduced. In what follows, standard SAT and MaxSAT definitions are introduced (e.g. [3]).

Let $X = \{x_1, x_2, \dots, x_n\}$ be a set of Boolean variables. A *literal* is either a variable x_i or its negation \bar{x}_i . A *clause* C is a disjunction of literals. An *assignment* is a set of literals $\mathcal{A} = \{l_1, l_2, \dots, l_k\}$. If variable x_i is assigned to *true* (*false*), literal x_i (\bar{x}_i) is *satisfied* and literal \bar{x}_i (x_i) is *falsified*. An assignment *satisfies* a literal iff it belongs to the assignment, it satisfies a clause iff it satisfies one or more of its literals and it *falsifies* a clause iff it contains the negation of all its literals. A *model* is a complete assignment that satisfies all the clauses in a CNF formula φ . SAT is the problem of deciding whether there exists a model for a given propositional formula. Given an unsatisfiable SAT formula φ , a subset of clauses φ_C whose conjunction is still unsatisfiable is called an *unsatisfiable core* (or *core*) of the original formula. Modern SAT solvers can be instructed to generate an unsatisfiable core [17].

A *weighted clause* is a pair (C, w) , where C is a clause and the *weight* w is the cost of its falsification. Weighted clauses that must be satisfied are called *mandatory* (or *hard*) and are associated with a special weight \top . Non-mandatory clauses are called *soft* clauses and have a weight $w < \top$. A *weighted formula in conjunctive normal form* (WCNF) φ is a set of weighted clauses. A *model* is a complete assignment \mathcal{A} that satisfies all hard clauses. Given a WCNF formula, the *Weighted Partial MaxSAT* is the problem of finding a model of minimum cost.

The remainder of this section introduces the notation used to describe bound computation algorithms. φ_W is the current working formula. Soft clauses may be extended with additional variables called *relaxation variables*. The bounds may use these functions: $\text{Soft}(\varphi)$ returns the set of all *soft* clauses in φ and $\text{SAT}(\varphi)$ makes a call to the SAT solver which returns whether φ (ignoring weights) is satisfiable (SAT or UNSAT). Without loss of generality, this paper assumes that the input formula has a model.

2.1 Lower Bounds

Consider Algorithm 1. Let λ be the lower bound, initially $\lambda = 0$. A SAT solver is iteratively called while the formula is unsatisfiable. For each core φ_C , the minimum weight $\min(\varphi_C)$ among the soft clauses is computed, the lower bound is updated as $\lambda = \lambda + \min(\varphi_C)$ and the weight of the soft clauses in φ_C is decreased by $\min(\varphi_C)$. Besides, each soft clause that reaches a weight of 0 is removed from the working formula. This lower bound will be referred to as *sat-lb-s*. The lower bound in [7] is similar to the described one but all the soft clauses in φ_C are removed from the formula which provides a weaker lower bound (for weighted MaxSAT but equivalent for unweighted MaxSAT) and will be referred to as *sat-lb*. In [10], cores are detected by unit propagation (UP), whereas the LB in [11, 7] additionally detects cores that cannot be identified by UP. Given that a SAT solver always detects first all the cores by solely applying UP and then the remaining ones, it is straightforward that the LB *sat-lb* is stronger than the one in [10], but *sat-lb* makes calls to a SAT solver which can require exponential time. *sat-lb-s* is an extension of [10] for weighted MaxSAT that provides a stronger LB because it is not restricted to UP.

Algorithm 1: Lower Bound

```
Input:  $\varphi$ 
1  $(\varphi_W, \lambda, \varphi_R) \leftarrow (\varphi, 0, \emptyset)$ 
2 while true do
3    $(st, \varphi_C, \mathcal{A}) \leftarrow \text{SAT}(\varphi_W)$ 
4   if  $st = \text{SAT}$  then return  $(\lambda, \varphi_R)$ 
5    $(\lambda, \varphi_R) \leftarrow (\lambda + \min(\varphi_C), \varphi_R \cup \text{Soft}(\varphi_C))$ 
6   foreach  $(C, w) \in \text{Soft}(\varphi_C)$  do
7      $w \leftarrow w - \min(\varphi_C)$ 
8     if  $w = 0$  then  $\varphi_W \leftarrow \varphi_W \setminus \{(C, w)\}$ 
9   end
10 end
```

Algorithm 2: Upper Bound

```
Input:  $\varphi$ 
1  $(\mu, last\mathcal{A}) \leftarrow (\sum_{i=1}^m w_i + 1, \emptyset)$ 
2  $(R, \varphi_W) \leftarrow \text{Relax}(\emptyset, \varphi, \text{Soft}(\varphi))$ 
3  $(st, \varphi_C, \mathcal{A}) \leftarrow \text{SAT}(\varphi_W)$ 
4 if  $st = \text{true}$  then  $(last\mathcal{A}, \mu) \leftarrow (\mathcal{A}, \sum_{i=1}^m w_i \times (1 - \mathcal{A}\langle C_i \setminus \{r_i\} \rangle))$ 
5 return  $SLS(last\mathcal{A}, \varphi)$ 
```

2.2 Upper Bounds

Consider Algorithm 2. Let μ be an UB. Initially, each soft clause is extended with a relaxation variable in function `Relax`. Then, the SAT solver is called and it returns a satisfying assignment \mathcal{A} . Then, the sum of weights of the soft clauses for which the relaxation variable has been assigned to true provides an UB in \mathcal{A} [7]. Note that, for non-optimal assignments \mathcal{A} , a relaxation variable assigned to true does not mean necessarily that the soft clause associated to such variable must be unsatisfied. As a result, a slight improvement is to sum the weights of unsatisfied soft clauses by \mathcal{A} disregarding the relaxation variables in the soft clauses. Such UB will be referred to as `sat-ub` and is inspired in [5]. Additionally, a stochastic local search (SLS) solver is called providing the previous computed assignment restricted to original variables. Recall that such assignment satisfies all hard clauses. The SLS solver may return an improved solution (or the given one, in the worse case). This UB will be referred to as `sat-ub+s`.

Using *non-random* initial assignments to improve the performance of a local search procedure was first studied in [13] for partial MaxSAT. The work in [9] executes in parallel a SAT solver and an SLS procedure. The variables to be *flipped* by the SLS depend on the current *partial assignment* of the SAT solver. However, such approach is (i) unable to take advantage of hard and soft clauses and (ii) cannot improve the SLS solver in the instances from MaxSAT Evaluations, essentially because the SAT solver proves the unsatisfiability very quickly and cannot guide the SLS procedure. Differently, `sat-ub+s` provides an assignment that satisfies (i) all hard clauses and (ii) its performance only depends on the ability of the SAT solver to find such an assignment. As a result, it can be applied on the benchmarks of MaxSAT Evaluations and still obtain significant improvements as shown in the empirical section.

Benchmark	#Inst.	sls	sat-ub	sat-ub+s	sat-lb	sat-lb-s
circ	9	94892	99	35	4	4
sean	112	69595	265	171	16	16
fir	59	4570	36	27	22	22
simp	138	31	41	28	25	25
msp	148	20787	375	350	227	227
mtg	215	515	18	16	6	6
haplo	6	3690	1151	1068	352	352
frb	25	447	449	446	233	233
mo3sat	80	46	55	37	26	26
mostr	60	39244	246	239	139	139
plan	56	294881	2171	2169	760	1371
spot	21	146940	159734	146739	63408	68743
rnet	78	156099	296800	156230	113019	143922
upgrade	100	-	10849700000	-	251240000	416861000
time	32	19354800	742	704	13	18
pedi	100	216139000	110344	91520	13792	15391
Aborted	-	0	27	27	30	33
AverageTime	-	34.61	3.65	5.73	17.16	21.11

Table 1. Quality of the upper bounds and lower bounds.

3 Experimental Evaluation

Experiments were conducted on a HPC cluster (3GHz) with linux. For each run, the time limit was set to 1200 seconds and a memory limit of 4GB. The bounds were implemented in the MSUNCORE [14] system. All benchmarks from 2009-2011 MaxSAT Evaluations (2067 instances) were considered.

3.1 Analysis of the bounds

Table 1 summarizes the quality of the computed bounds only for some benchmark sets, but similar improvements are observed in the remaining ones. The first column shows the name of the set of benchmarks, the second column shows the number of instances in the set. The three following columns show three different upper bounds. The two final columns show two different lower bounds. All five columns present the average value of the bound for all instances in the benchmark set. Column *sls* refers to an upper bound computed by the SLS procedure ADAPTNOVELTY+ [8] included in the UBCSAT (with default parameters) [16] solver but any other SLS algorithm could be used. *sat-ub+s* uses ADAPTNOVELTY+ as the SLS algorithm. Regarding the upper bounds, the solutions provided by the SLS algorithm are of very low quality. Differently, *sat-ub* provides a solution orders of magnitude better than the previous one. Finally, *sat-ub+s* is more accurate than the previous one. One of the reasons why *sat-ub* and *sat-ub+s* are better than *sls* is because calling a SAT solver with the additional relaxation variables provides a good initial assignment that *satisfies all hard clauses*. Note that the benchmark set *upgrade* contains very large weights and the *sls* algorithm cannot handle such weights. For this reason they are omitted from the average for 2 upper bounds.

Recall that the approach [9] is unable to improve the upper bound provided by a SLS procedure in the MaxSAT Evaluation instances. Regarding the lower bounds, both *sat-lb* and *sat-lb-s* provide the same value for unweighted MaxSAT as expected given that

in such case they are equivalent. Differently, for weighted MaxSAT sat-lb-s provides substantially higher lower bounds.

Note the last two rows in the Table 1 that show summarized results over the 2067 instances. One shows the number of *aborted* instances within the time limit while computing the bounds. The other one shows the average time in seconds to compute the bounds. The upper bounds based on calling a SAT solver can be aborted for some very hard instances, but they usually require much less time than SLS.

3.2 Improving core-guided MaxSAT algorithms with the bounds

In what follows, the performance of several *core-guided* MaxSAT algorithms [7] is studied. Each sub-table in Table 2 shows the results for msu3 [11] (left), msu4 [12] (mid), and core-guided binary search [7] (right), respectively. All three algorithms use exactly *one relaxation variable per soft clause*. Once the LBs are computed, the algorithms will add one relaxation variable to each soft clause returned in φ_R (See Algorithm 1). For each sub-table in Table 2, the first and second columns show the benchmark set and its number of instances, respectively. The remaining three columns show the performance of an algorithm with different bounds in terms of solved instances within the time limit. Note that the necessary time to compute the bounds *is included* in the time limit for each execution. For each algorithm some sets of instances are shown where significant differences in the performance are reported.

msu3 [11] iteratively refines a LB. Table 2 (left) shows the performance of msu3 without LB (3rd column), with sat-lb (4th col.) and with sat-lb-s (5th col.). Clearly, the use of lower bounds improve the performance of msu3. For unweighted problem sets (msp and frb), both lower bounds provide the same improvement as expected. For weighted problem sets (planning, upgrade and pedigree), sat-lb-s is noticeable better than sat-lb.

msu4 [12] refines both a LB and a UB but empirical observation shows that in most of its iterations, msu4 refines an UB. Hence, msu4 may benefit from both bounds but specially from a good initial UB. Table 2 (mid) shows the performance of msu4 where the LB is fixed to sat-lb-s, while the UBs considered are none (3rd col.), sat-ub (4th col.) and sat-ub+s (5th col.). Clearly, the use of UBs improve the performance of msu4, being sat-ub+s the one that provide the best results.

Core-guided binary search [7] refines both a lower bound and upper bound, and at each iteration it asks for the middle value between them. Table 2 (right) shows the performance of core-guided binary search without bounds (3rd col.), with both sat-lb and sat-ub as in [7] (4th col.) and with the two new bounds sat-lb-s and sat-ub+s (5th col.). The additional sixth column shows the results for sat-lb and sat-ub+s. The performance of core-guided binary search is quite good without the bounds and their use improves the performance in 4 of 5 sets. Note that the efficiency for the *upgrade* set of problems is slightly worsened. While the use of bounds can save calls to the SAT solver in binary search, they may move the search to *harder* calls of the SAT solver [15].

Set	#I.	None	sat-lb	sat-lb-s	Set	#I.	None	sat-ub	sat-ub+s	Set	#I.	None	sat-lb sat-ub	sat-lb-s sat-ub+s	sat-lb sat-ub+s
msp	148	89	92	92	sean	112	51	77	78	sean	112	72	77	78	78
frb	25	0	14	14	fir	59	46	53	53	frb	25	0	15	15	15
plan.	56	38	40	44	mostr	60	44	44	59	msp	148	98	107	107	107
upgr.	100	0	0	10	msp	148	75	86	108	upgr.	100	63	59	52	59
pedi.	100	24	40	44	plan.	56	21	35	50	pedi.	100	32	34	34	33
total	370	151	186	204	total	435	237	295	348	total	485	265	292	286	292

Table 2. Bounds on msu3 (left), msu4 (mid) and core-guided binary search (right)

4 Conclusions and Future Work

This paper introduces new LB and UB based on calling a SAT solver and studies their effect on the performance core-guided MaxSAT solvers. The bounds presented in this paper can be integrated in *branch and bound* MaxSAT solvers and MaxSAT solvers based on computing unsatisfiable cores that exploit *disjoint cores* [1, 7], and which add more than one relaxation variable per soft clause [4]. Additionally, the bounds can be extended to other boolean optimization frameworks [6].

References

1. C. Ansótegui, M. L. Bonet, and J. Levy. A new algorithm for weighted partial MaxSAT. In *AAAI*, 2010.
2. D. Le Berre and A. Parrain. The Sat4j library, release 2.2. *JSAT*, 7:59–64, 2010.
3. A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, 2009.
4. Z. Fu and S. Malik. On solving the partial MAX-SAT problem. In *SAT*, pages 252–265, August 2006.
5. E. Giunchiglia and M. Maratea. Solving optimization problems with DLL. In *ECAI*, pages 377–381, August 2006.
6. F. Heras, V. M. Manquinho, and J. Marques-Silva. On applying unit propagation-based lower bounds in pseudo-boolean optimization. In *FLAIRS Conference*, pages 71–76, 2008.
7. F. Heras, A. Morgado, and J. Marques-Silva. Core-guided binary search algorithms for maximum satisfiability. In *AAAI*, 2011.
8. H. H. Hoos. An adaptive noise mechanism for WalkSAT. In *AAAI*, pages 655–660, 2002.
9. L. Kroc, A. Sabharwal, C. P. Gomes, and B. Selman. Integrating systematic and local search paradigms: A new strategy for MaxSAT. In *IJCAI*, pages 544–551, 2009.
10. C. M. Li, F. Manyà, and J. Planes. Exploiting unit propagation to compute lower bounds in branch and bound Max-SAT solvers. In *CP*, pages 403–414, 2005.
11. J. Marques-Silva and J. Planes. On using unsatisfiability for solving maximum satisfiability. *Computing Research Repository*, abs/0712.0097, December 2007.
12. J. Marques-Silva and J. Planes. Algorithms for maximum satisfiability using unsatisfiable cores. In *DATE*, pages 408–413, 2008.
13. M. E. Menai and M. Batouche. An effective heuristic algorithm for the maximum satisfiability problem. *Appl. Intell.*, 24(3):227–239, 2006.
14. A. Morgado, F. Heras, and J. Marques-Silva. The MSUnCore MaxSAT solver. In *POS*, 2011.
15. M. Sellmann and S. Kadioglu. Dichotomic search protocols for constrained optimization. In *CP*, pages 251–265, 2008.
16. D. A. D. Tompkins and H. H. Hoos. UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT & MAX-SAT. In *SAT*, 2004.
17. L. Zhang and S. Malik. Validating sat solvers using an independent resolution-based checker: Practical implementations and other applications. In *DATE*, pages 10880–10885, 2003.