Core-Guided MaxSAT with Soft Cardinality Constraints

Antonio Morgado[1], Carmine Dodaro[2], and Joao Marques-Silva[1,3,⋆]

[1] INESC-ID, IST, ULisboa, Portugal
ajrm@sat.inesc-id.pt
[2] Dep. of Mathematics and Computer Science, Unical, Italy
dodaro@mat.unical.it
[3] CASL, University College Dublin, Ireland
jpms@ucd.ie

Abstract. Maximum Satisfiability (MaxSAT) is a well-known optimization variant of propositional Satisfiability (SAT). Motivated by a growing number of practical applications, recent years have seen the development of different MaxSAT algorithms based on iterative SAT solving. Such algorithms perform well on problem instances originating from practical applications. This paper proposes a new core-guided MaxSAT algorithm. This new algorithm builds on the recently proposed unclasp algorithm for ASP optimization problems, but focuses on reusing the encoded cardinality constraints. Moreover, the proposed algorithm also exploits recently proposed weighted optimization techniques. Experimental results obtained on industrial instances from the most recent MaxSAT evaluation, indicate that the proposed algorithm achieves increased robustness and improves overall performance, being capable of solving more instances than state-of-the-art MaxSAT solvers.

1 Introduction

Maximum Satisfiability (MaxSAT) is a well-known optimization version of Propositional Satisfiability (SAT). Recent years have seen a growing number of practical applications of MaxSAT, that include fault localization in C code [12] and design debugging [21], among many others [19]. For practical MaxSAT problem instances, the most effective solutions are based on iterative SAT solving, and a number of alternative approaches exist. One approach iteratively pre-relaxes every clause (by adding to each clause a fresh relaxation variable) and refines bounds on the number of unsatisfied clauses [19]. A recent example of such a MaxSAT solver is QMaxSAT [13]. An alternative approach is based on iterative identification of unsatisfiable cores [10]. Different algorithms based on the identification of unsatisfiable cores have been developed in recent years, e.g. [19]. One additional approach is based on finding minimum hitting sets of a formula representing disallowed sets of clauses [8]. This paper builds on

⋆ This work is partially supported by SFI grant BEACON (09/IN.1/I2618), by FCT grant POLARIS (PTDC/EIA-CCO/123051/2010), by INESC-IDs multiannual PIDDAC funding PEst-OE/EEI/LA0021/2013, and by the European Commission, European Social Fund of Regione Calabria.

B. O'Sullivan (Ed.): CP 2014, LNCS 8656, pp. 564–573, 2014.

recent work on using unsatisfiable cores for solving optimization problems in ASP [1] and shows how the algorithm can be optimized for the case of MaxSAT. Experimental results, obtained on problem instances from the industrial categories of the MaxSAT evaluation, indicate that the new algorithm is more robust in practice than state-of-the-art MaxSAT solvers, being able to solve more problem instances. The paper is organized as follows. Section 1 introduces the paper, followed by the notation and definitions used in the paper in Section 2. The OLL algorithm is presented in Section 3. The experimental results are presented in Section 4 and Section 5 concludes the paper.

2 Preliminaries

This section introduces the notation used throughout the paper. Standard definitions are assumed (e.g. [14,19]). Let $X = \{x_1, x_2 \ldots\}$ be a set of Boolean variables. A *literal* l_i is either a variable x_i or its negation $\neg x_i$. A *clause* c is a disjunction of literals. A *conjunctive normal form* (CNF) formula φ is a conjunction of clauses. An *assignment* $\mathcal{A}$ is a mapping $\mathcal{A} : X \rightarrow \{0, 1\}$, where $\mathcal{A}$ satisfies(falsifies) x_i if $\mathcal{A}(x_i) = 1(\mathcal{A}(x_i) = 0)$. Assignments are extended to literals and clauses in the usual way, that is, $\mathcal{A}(l_i) = \mathcal{A}(x_i)$ if $l_i = x_i$, and $\mathcal{A}(l_i) = 1 - \mathcal{A}(x_i)$ otherwise, while for clauses $\mathcal{A}(c) = \max\{\mathcal{A}(l_i) | l_i \in c\}$. Given a CNF formula φ, a *model* of the formula is an assignment that satisfies all the clauses in φ. The *Propositional Satisfiability* problem (SAT) is the problem of deciding whether there exists a model to a given formula. A subformula of a given unsatisfiable formula which is still unsatisfiable is referred as an *unsatisfiable core* (or simply a core). The calls to the SAT solver are done through function $SATSolver(\varphi)$, that receives a formula and returns a triple $(\text{st}, \varphi_C, \mathcal{A})$, where st is either *true* or *false*.If st is true, then $\mathcal{A}$ is a model, otherwise φ_C is a core. Given a clause c and an integer w greater than 0 referred to as *weight*, the pair (c, w) is a *weighted clause*. Weighted clauses may be classified as *hard* or *soft* clauses. Hard clauses have to be satisfied and are associated with the special weight $\top$. Soft clauses may or may not be satisfied, and their weight represents the cost of falsifying the clause. A *weighted CNF formula* (WCNF) is a set of weighted clauses. A *model* of a WCNF formula is an assignment that satisfies all the hard clauses, and the *cost* associated to the model is the sum of the weights of the falsified soft clauses. The *Weighted Partial MaxSAT* problem is the problem of determining the minimum cost of the models of a given WCNF formula.

In the paper, we refer to *Relaxation Variables*, which are fresh Boolean variables. The process of augmenting a clause with a relaxation variable is referred as *relaxing* the clause. We also refer to a special type of constraints called *cardinality constraints*, which have the form $\sum_i x_i \leq k$. The sum $\sum x_i$ is referred to as the *left hand side* (LHS) of the constraint while k is referred to as the *right hand side* (RHS).

3 The OLL Algorithm

Algorithm OLL has been introduced in the unclasp tool for solving ASP optimization problems [1]. Unclasp is the base of the ASP-based Linux configuration system aspuncud [11], which won four tracks of the 2011 Mancoosi International Solver Competition[1]. Algorithm OLL has been reported [1] to be able to solve a higher number of

[1] `http://www.mancoosi.org/misc/`

MISC optimization instances than clasp (the base solver on which unclasp was built upon).

This section shows how to adapt the OLL algorithm to MaxSAT, but additionally considering the reuse of the cardinality constraints as they are discovered. The idea of the OLL algorithm is to mix the strengths of two MaxSAT algorithms, namely the Fu & Malik algorithm [10] and MSU3 [17,18]. These are core-guided algorithms, which means that the algorithms make use of the unsatisfiable cores in order to relax clauses. Like MSU3 only one relaxation variable is added per clause identified in a core, but similarly to the Fu & Malik algorithm a new cardinality constraint is added for each core as it is found. One of the main difference of the OLL algorithm is that the soft clauses are transformed into hard clauses after relaxing them, while the cardinality constraints are added as soft. Consider the following Example 1.

Example 1. In the example we will abuse the notation and refer to soft constraints as if they were clauses. Consider for example the partial formula $\varphi = \varphi_S \cup \varphi_H$, where φ_S is the set of soft clauses $\varphi_S = \{(x_1, 1), (x_2, 1), (x_3, 1)\}$ and φ_H is the set of hard clauses $\varphi_H = \{(\neg x_1 \vee \neg x_2, \top), (\neg x_1 \vee \neg x_3, \top), (\neg x_2 \vee \neg x_3, \top)\}$. The initial working formula φ_W is $\varphi_S \cup \varphi_H$, which is unsatisfiable. Let the soft clauses in the core returned by the SAT solver be $(x_1, 1)$ and $(x_2, 1)$. The OLL algorithm relaxes both clauses and makes them hard, and adds a cardinality constraint as a soft constraint. As such, the sets of clauses in the working formula are updated as:

$$\varphi_S \leftarrow (\varphi_S \setminus \{(x_1, 1), (x_2, 1)\}) \cup \{(r_1 + r_2 \leq 1, 1)\}$$
$$\varphi_H \leftarrow \varphi_H \cup \{(x_1 \vee r_1, \top), (x_2 \vee r_2, \top)\}$$

where r_1 and r_2 are the new relaxation variables.

The resulting working formula φ_W is again unsatisfiable. Now the unsatisfiable core contains the soft clause $(x_3, 1)$ and the soft constraint $(r_1 + r_2 \leq 1, 1)$. As before the OLL algorithm is going to relax the soft clause and make it hard, that is $(x_3 \vee r_3, \top)$. It will also remove the soft constraint $(r_1 + r_2 \leq 1, 1)$ from the working formula, and add two new constraints $(r_3 + \neg(r_1 + r_2 \leq 1) \leq 1, 1)$ and $(r_1 + r_2 \leq 2, 1)$. The first constraint says that in order to satisfy the constraint, you either have $(r_1 + r_2 \leq 1)$ (the previous constraint) falsified or you are allowed to set r_3 to true.

The second constraint added $(r_1 + r_2 \leq 2, 1)$, if satisfied, allows one more of the previous relaxation variables to be satisfied. The sets in the working formula are then updated as:

$$\varphi_S \leftarrow (\varphi_S \setminus \{(x_3, 1), (r_1 + r_2 \leq 1, 1)\}) \cup \{(r_3 + \neg(r_1 + r_2 \leq 1) \leq 1, 1), (r_1 + r_2 \leq 2, 1)\}$$
$$\varphi_H \leftarrow \varphi_H \cup \{(x_3 \vee r_3, \top)\}$$

Now the resulting working formula is satisfiable and the algorithm returns 2, which is the cost of the satisfying assignment.

As the previous example illustrates, the idea of OLL is to go through unsatisfiable iterations until a satisfiable working formula is obtained. Whenever a new unsatisfiable core is identified, then the working formula is updated such that either all the previous soft constraints in the core are satisfied and allowing a new relaxation variable to be set to true, or one of the soft constraints is allowed to increase its bound by 1.

The example uses soft constraints that correspond to cardinality constraints. On the other hand, SAT solvers only handle clauses. In order to use the OLL algorithm in a MaxSAT solver using a SAT solver, it is necessary to encode the cardinality constraints into CNF each time they are identified. Observe that both cardinality constraints $r_1+r_2 \leq 1$ and $r_1+r_2 \leq 2$, share the same LHS r_1+r_2. In fact, some of the existing encodings of cardinality constraints, encode the sum on the LHS into an array of Boolean variables to represent it as a unary number. In this paper, we propose to use this fact in order to reuse the encodings of the sums of the LHS of the constraints between cardinality constraints that only differ on their RHS. In the previous example, $r_1 + r_2$ would be encoded using an auxiliary function $([s_1, s_2], clauses) \leftarrow createSum(\{r_1, r_2\})$, which receives a set of variables for which we want to encode the sum, and returns a pair containing an array of the Boolean variables that encode the sum in a unary number (the unary number $s_2 s_1$), and a set of clauses that encodes the sum. Whenever a new cardinality constraint is required with the same LHS (sum), then the same variables s_1 and s_2 are set to the appropriate values in order to encode the cardinality constraint.

The pseudo-code of OLL is shown in Algorithm 1. In the following we assume that the input formula is unweighted (weighted case explained later on), partial and the set of hard clauses is satisfiable. Given an input formula φ, OLL maintains three sets of clauses: the current representation of the input formula called the working formula φ_W; the current set of soft clauses φ_S and the set of soft cardinalities φ_{SC} containing (unit) clauses associated to cardinality constraints. Those sets are initialized in line 1. Moreover, function map associates with a literal l (related to one of the cardinality constraints) a pair corresponding to the outputs of the associated sum and a bound (its RHS). OLL starts by calling the SAT solver on the current working formula φ_W. If the formula is satisfiable, then the algorithm terminates and returns the cost of $\mathcal{A}$ (line 5). Otherwise, the working formula is unsatisfiable and an unsatisfiable core is computed. The algorithm proceeds by relaxing all soft clauses of the core that are in φ_S, and making them hard clauses. This is done through function $RelaxAndHarden(\varphi_W, \varphi_C \cap \varphi_S)$, which receives the working formula φ_W, and a set of soft clauses that need to be relaxed. Function $RelaxAndHarden$ returns a pair (L, φ_W), where L is the set of new relaxation variables, and φ_W is updated to the clauses that were in φ_W, but to which the clauses that were in $\varphi_C \cap \varphi_S$, have been relaxed and transformed into hard clauses.

After relaxing soft clauses, the remaining clauses in the core related to cardinality constraints (i.e. clauses in $\varphi_C \cap \varphi_{SC}$) are processed and removed from the working formula φ_W (line 9) and from the set φ_{SC} (line 10). Each of those clauses is a unit clause.The outputs $sumOtps$ of the sum associated with the cardinality constraint corresponding to $\neg s$ are obtained with the function $map(\neg s)$, from which the corresponding bound b (RHS) is also obtained. In fact, the variable s represents the b-th output variable of the sum in $sumOtps$. In line 11, the set L is extended with the variable s. This corresponds to the negation of the previous cardinality constraint with b as the RHS, i.e. if s is true then the sum is greater than b, thus negating the cardinality constraint. The algorithm proceeds by creating a new unit clause $(\neg sumOtps[b+1])$ that encodes the sum to be less or equal to b. The clause is then added to the working formula φ_W (line 14) and to φ_{SC} (line 15). Moreover, in line 16, the pair (sum, $b+1$) is added to the map for the $(b+1)$-th output variable. Note that this is done only if $b+1$

Algorithm 1. OLL algorithm for (non-weighted) (partial) MaxSAT

Input: A formula φ

1 $(\varphi_W, \varphi_S, \varphi_{SC}) \leftarrow (\varphi, \texttt{Soft}(\varphi_W), \emptyset)$;
2 $\texttt{map} \leftarrow \emptyset$; // $\texttt{map}(lit) = (sumOtps, bound)$
3 **while** true **do**
4 $(\mathsf{st}, \varphi_C, \mathcal{A}) \leftarrow \texttt{SATSolver}(\varphi_W)$;
5 **if** $\mathsf{st} = true$ **then return** $\sum_{(c,1)\in\varphi_S}(1 - \mathcal{A}(c))$
6 **else**
7 $(L, \varphi_W) \leftarrow \texttt{RelaxAndHarden}(\varphi_W, \varphi_C \cap \varphi_S)$;
8 **foreach** $(\neg s, 1) \in \varphi_C \cap \varphi_{SC}$ **do**
9 $\varphi_W \leftarrow \varphi_W \setminus \{(\neg s, 1)\}$;
10 $\varphi_{SC} \leftarrow \varphi_{SC} \setminus \{(\neg s, 1)\}$;
11 $L \leftarrow L \cup \{s\}$;
12 $(sumOtps, b) \leftarrow \texttt{map}(\neg s)$;
13 **if** $b + 1 < |sumOtps|$ **then**
14 $\varphi_W \leftarrow \varphi_W \cup \{(\neg sumOtps[b+1], 1)\}$;
15 $\varphi_{SC} \leftarrow \varphi_{SC} \cup \{(\neg sumOtps[b+1], 1)\}$;
16 $\texttt{map}(\neg sumOtps[b+1]) \leftarrow (sumOtps, b+1)$;
17 $(sumOtps_{New}, sumCls_{New}) \leftarrow createSum(L)$;
18 $\varphi_W \leftarrow \varphi_W \cup \{(c, \top) \mid c \in sumCls_{New}\} \cup \{(\neg sumOtps_{New}[1], 1)\}$;
19 $\varphi_{SC} \leftarrow \varphi_{SC} \cup \{(\neg sumOtps_{New}[1], 1)\}$;
20 $\texttt{map}(\neg sumOtps_{New}[1]) \leftarrow (sumOtps_{New}, 1)$;

is less than the size of the sum, i.e. if incrementing the bound by one does not make the sum trivially satisfied.

When all clauses in the core related to soft cardinality constraints have been processed, a new cardinality constraint, with the corresponding new sum is created, containing all variables in L (line 17). The clauses encoding the sum are added to the working formula as hard clauses while a new unit soft clause $(\neg sumOtpts_{New}[1])$ is added to φ_W and to φ_{SC}. This clause encodes that at most one of the variables in L is true. In addition, the pair $(sumOtps_{New}, 1)$ is associated to the literal $\neg sumOtps_{New}[1]$ by adding a new entry to the map.

Proposition 1. *Given a (partial) MaxSAT formula, Algorithm 1 is correct and returns the optimum MaxSAT solution.*

Proof (sketch). The OLL algorithm goes thought unsatisfiable instances until a satisfiable instance is obtained. Initially the algorithm tries to satisfy all the soft clauses (added to a working formula together with the hard clauses). Whenever the working formula is unsatisfiable, then it is updated such that at most one more of the initial soft clauses is allowed to be falsified (than the previous iteration). When the working formula is satisfiable the algorithm stops and the number of initial soft clauses simultaneously falsified corresponds the optimum MaxSAT solution. This process is similar to other MaxSAT algorithms as MSU3.

Nevertheless, in OLL, the restriction on the number of initial soft clauses that are allowed to be falsified is achieved by adding relaxation variables to soft clauses that

belong to a core and have not been relaxed before, and by the addition of soft cardinality constraints (At-Most-K constraints). The soft cardinality constraints (added on line 18, initially with a RHS of 1) allow at most one of the newly relaxed clauses to be falsified or at most one of previous soft cardinality constraints that appeared in the core to be falsified. Falsifying a previous soft cardinality constraint forces the number of associated initial soft clauses that are falsified to increase. The increase is contrained to be at most one by adding a new soft cardinality constraint (added on line 14) equal to the previous soft cardinality but with the RHS increased by 1.

The previous algorithm deals with non-weighted (partial) MaxSAT formulas. In the weighted case the procedure is similar to the MSU1/WPM1 algorithms [15,3], that is, every time a new core is found, the minimum weight of the soft clauses in the core min is computed. Then each clause (c_i, w_i) with a weight greater than the minimum is replaced by two clauses: (c_i, min) and $(c_i, w_i - min)$. Then the algorithm proceeds as in the partial case but as if the core obtained contained only clauses with the same weight min . The result is obtained as in the partial case by considering the cost of the satisfying assignment in the original soft clauses, but considering the original weights.

4 Experimental Results

This section presents the experimental results obtained to validate the performance of the MaxSAT algorithm proposed in Section 3. All experiments were run on an HPC cluster, each node having two processors E5-2620 @2GHz, with each processor having 6 cores, and with a total of 128 GByte of physical memory. Each process was limited to 4GByte of RAM and to a time limit of 1800 seconds. All the industrial instances from the most recent MaxSAT Evaluation[2] 2013 [5] were used, that is the following three categories of benchmarks were considered: (plain) MaxSAT industrial; partial MaxSAT industrial; and weighted partial MaxSAT industrial.

For the experiments, the OLL algorithm proposed in the previous section was implemented in MSUnCore [20][3]. MSUnCore is a state-of-the-art (generic) MaxSAT solver, that won third place in the partial MaxSAT category of the 2013 MaxSAT Evaluation (second place, if portfolio solvers are excluded). The underlying SAT solver in MSUnCore is PicoSAT [7] (version 935). Three different cardinality constraint encodings that are able to encode the sum of all the input variables as a unary number, were considered. Namely Sorting Networks [9], Sequential Counters [22], and Totalizer [6]. In the results the OLL algorithm with the cardinality constraints are referred as *msu-oll-sn*, *msu-oll-sc* and *msu-oll-to* respectively. For weighted instances, we have implemented an OLL solver which includes recent weighted boolean optimizations techniques proposed for MaxSAT solving. When considering the weighted optimizations, and previously to solving, a weighted instance is checked for the BMO condition [16], in which case the instance is solved according to the BMO approach. Otherwise, the stratification technique [4] is considered. The resulting solver is referred in the results as *msu-oll-xx-wo*, where xx corresponds to the cardinality constraint considered. Additionally the experiments include the

[2] `http://www.maxsat.udl.cat`

[3] Logs in `http://sat.inesc-id.pt/~ajrm/oll_statlogs.tgz`

	MSi	PMSi	WPMSi	ALLi
#Instances	55	627	396	1078
msu-oll-sn-wo	25	512	330	867
msu-oll-to-wo	19	517	329	865
msu-oll-to	19	517	315	851
msu-oll-sn	25	512	314	851
msu-oll-sc-wo	18	494	331	843
msu-oll-sc	18	494	289	801
msu-bcd2	22	500	265	787

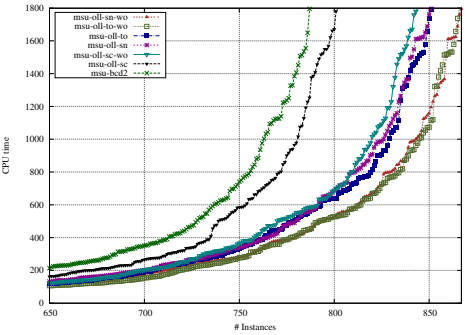

Fig. 1. Cactus plot and statistics for the different configurations of the solvers in MSUnCore

two best solvers for each of the industrial categories from the 2013 MaxSAT Evaluation (among non-portfolio solvers): MiFuMax[4], MSUnCore [20] (BCD2 version), QMaxSAT 0.21 [13], WPM1 [4] (2011 and 2013 versions), and WPM2 [4,2] (2013 version). The solvers are referred in the results as *mifumax*, *msu-bcd2*, *qmaxsat*, *wpm1*, *wpm1-2011*, and *wpm2*.

The table in Figure 1 shows the number of instances solved by each of the algorithms in MSUnCore, that is, the OLL algorithms and msu-bcd2. The first column of the table shows the name of the solver. The second to fourth columns show the number of solved instances by each of the solvers, for the (plain) MaxSAT industrial (MSi), partial MaxSAT industrial (PMSi) and weighted partial MaxSAT industrial instances respectively. The last column shows the total number of solved instances among all of the industrial instances. The first row does not present the number of solved instances, but instead the total number of instances in the category considered in the column. In the table the solvers are ordered according to the number of instances solved in ALLi.

The results in the table show that among the msu-oll-xx solvers, both msu-oll-sn and msu-oll-to have similar performance, being msu-oll-sn slightly better for (plain) MaxSAT instances, while msu-oll-to is slightly better for partial MaxSAT instances. The msu-oll-sc solver performs consistently worse than the other two with a total of 50 less instances solved in ALLi. The table in Figure 1 also allows to conclude that the weighted optimizations included are consistently beneficial for all the msu-oll solvers. This is especially true for msu-oll-sc-wo which solved 32 more instances than msu-oll-sc. Comparing the msu-oll solvers with msu-bcd2 (since they are implemented in the same platform), the results show that for MSi instances, the msu-oll solvers are comparable to msu-bcd2, where msu-oll-sn(-wo) solves 3 more instances than msu-bcd2. In the case of PMSi instances, the difference between msu-bcd2 and the msu-oll solvers is greater, and both the msu-oll-sn and msu-oll-to are able to solve more 12 and 17 instances than msu-bcd2. For weighted instances, all the OLL algorithms outperform msu-bcd2, including the versions that do not make use of weighted optimizations. This can be related to the fact that OLL requires only cardinality constraints to deal with the weights, while msu-bcd2 uses pseudo-Boolean constraints. In fact, the best performing OLL algorithm (msu-oll-sn-wo) is able to solve 80 more instances than msu-bcd2.

[4] `http://sat.inesc-id.pt/~mikolas/sw/mifumax`

	MSi	PMSi	WPMSi	ALLi
#Instances	55	627	396	1078
msu-oll-sn-wo	25	512	330	867
wpm2	12	490	320	822
msu-bcd2	22	500	265	787
wpm1	19	384	342	745
wpm1-2011	37	265	304	606
mifumax	38	273	258	569
qmaxsat	–	540	–	–

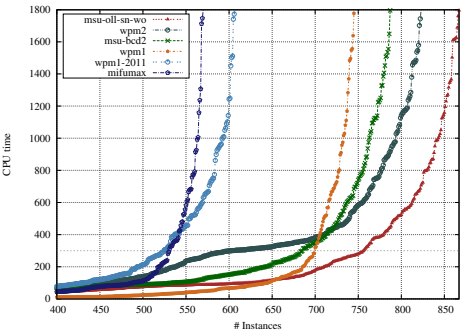

Fig. 2. Cactus plot and statistics for the best OLL algorithm vs non-OLL algorithms

These results are confirmed by the cactus plot in Figure 1, where the msu-bcd2 is the left-most solver, meaning that it solves less instances. On the other end, both msu-oll-sn-wo and msu-oll-to-wo appear close together on the right-most side of the plot.

In order to compare the best performing msu-oll solver (msu-oll-sn-wo) with the remaining solvers, we present in the table of Figure 2 the number of solved instances for the remaining solvers along with msu-oll-sn-wo and msu-bcd2. The table in the Figure 2 has a similar structure to the table in the Figure 1. As before the solvers are ordered according to the number of instances solved in ALLi. The only exception is qmaxsat for which the tested solver only allows to solve partial MaxSAT instances. From the table it is possible to see that for each category, msu-oll-sn-wo is either the third (for MSi) or the second (for PMSi and WPMSi) solver in terms of number of instances solved. Nevertheless, overall msu-oll-sn-wo solves more instances than any of the other solvers (shown in the ALLi column). The closest solver is wpm2 with 822 instances solved, which means a difference of 45 instances to msu-oll-sn-wo. These results are also confirmed by the cactus plot show in Figure 2, where the right-most line corresponds to msu-oll-sn-wo and the gap between the line of msu-oll-sn-wo and next line (wpm2) corresponds to the 45 instances difference. Note that in the figure, qmaxsat is not represented since it only allows solving partial MaxSAT instances.

5 Conclusions

This paper describes how to transform the OLL algorithm, proposed in unclasp for optimization problems in ASP [1], into a core-guided MaxSAT using a modern SAT solver. Additionally, the paper shows how to reuse the encodings of the cardinality constraints as they are added to the working formula. The experimental results indicate that the proposed OLL algorithm represents the currently most robust approach for MaxSAT, being able to solve more instances than state-of-the-art MaxSAT solvers. Despite not being in general the top performer for any specific category of instances, overall the OLL algorithm solves more instances than any of the best performing solvers from the 2013 MaxSAT Evalution, including MiFuMax, MSUnCore (BCD2), WPM1 and WPM2.

Future work will investigate alternative approaches for aggregating soft cardinality constraints, as well as improving the quality of computed unsatisfiable cores.

References

1. Andres, B., Kaufmann, B., Matheis, O., Schaub, T.: Unsatisfiability-based optimization in clasp. In: International Conference on Logic Programming (Technical Communications), pp. 211–221 (2012)
2. Ansótegui, C., Bonet, M.L., Gabàs, J., Levy, J.: Improving WPM2 for (weighted) partial MaxSAT. In: Schulte, C. (ed.) CP 2013. LNCS, vol. 8124, pp. 117–132. Springer, Heidelberg (2013)
3. Ansótegui, C., Bonet, M.L., Levy, J.: Solving (weighted) partial MaxSAT through satisfiability testing. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 427–440. Springer, Heidelberg (2009)
4. Ansótegui, C., Bonet, M.L., Levy, J.: SAT-based MaxSAT algorithms. Artificial Intelligence 196, 77–105 (2013)
5. Argelich, J., Li, C.M., Manyà, F., Planes, J.: The first and second Max-SAT evaluations. Journal on Satisfiability, Boolean Modeling and Computation 4(2-4), 251–278 (2008)
6. Bailleux, O., Boufkhad, Y.: Efficient CNF encoding of boolean cardinality constraints. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, pp. 108–122. Springer, Heidelberg (2003)
7. Biere, A.: Picosat essentials. Journal on Satisfiability, Boolean Modeling and Computation 4(2-4), 75–97 (2008)
8. Davies, J., Bacchus, F.: Solving MAXSAT by solving a sequence of simpler SAT instances. In: Lee, J. (ed.) CP 2011. LNCS, vol. 6876, pp. 225–239. Springer, Heidelberg (2011)
9. Een, N., Sörensson, N.: Translating pseudo-Boolean constraints into SAT. Journal on Satisfiability, Boolean Modeling and Computation 2, 1–26 (2006)
10. Fu, Z., Malik, S.: On solving the partial MAX-SAT problem. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 252–265. Springer, Heidelberg (2006)
11. Gebser, M., Kaminski, R., Schaub, T.: aspcud: A Linux package configuration tool based on answer set programming. In: International Workshop on Logics for Component Configuration (LoCoCo 2011). Electronic Proceedings in Theoretical Computer Science (EPTCS), vol. 65, pp. 12–25 (2011), `http://www.cs.uni-potsdam.de/wv/aspcud/`
12. Jose, M., Majumdar, R.: Cause clue clauses: error localization using maximum satisfiability. In: International Conference on Programming Language Design and Implementation, pp. 437–446 (2011)
13. Koshimura, M., Zhang, T., Fujita, H., Hasegawa, R.: QMaxSAT: A partial Max-SAT solver. Journal on Satisfiability, Boolean Modeling and Computation 8(1-2), 95–100 (2012)
14. Li, C.M., Manyà, F.: MaxSAT, hard and soft constraints. In: Handbook of Satisfiability, pp. 613–632. IOS Press (2009)
15. Manquinho, V., Marques-Silva, J., Planes, J.: Algorithms for weighted boolean optimization. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 495–508. Springer, Heidelberg (2009)
16. Marques-Silva, J., Argelich, J., Graça, A., Lynce, I.: Boolean lexicographic optimization: algorithms & applications. Annals of Mathematics and Artificial Intelligence 62(3-4), 317–343 (2011)
17. Marques-Silva, J., Planes, J.: On using unsatisfiability for solving maximum satisfiability. Computing Research Repository abs/0712.0097 (December 2007)
18. Marques-Silva, J., Planes, J.: Algorithms for maximum satisfiability using unsatisfiable cores. In: Design, Automation and Testing in Europe Conference, pp. 408–413 (March 2008)

19. Morgado, A., Heras, F., Liffiton, M.H., Planes, J., Marques-Silva, J.: Iterative and core-guided maxsat solving: A survey and assessment. Constraints 18(4), 478–534 (2013)
20. Morgado, A., Heras, F., Marques-Silva, J.: Improvements to core-guided binary search for MaxSAT. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 284–297. Springer, Heidelberg (2012)
21. Safarpour, S., Mangassarian, H., Veneris, A., Liffiton, M.H., Sakallah, K.A.: Improved design debugging using maximum satisfiability. In: International Conference on Formal Methods in Computer-Aided Design (2007)
22. Sinz, C.: Towards an optimal CNF encoding of boolean cardinality constraints. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 827–831. Springer, Heidelberg (2005)