Towards More Effective Unsatisfiability-Based Maximum Satisfiability Algorithms

Joao Marques-Silva and Vasco Manquinho

School of Electronics and Computer Science, University of Southampton, UK
IST/INESC-ID, Technical University of Lisbon, Portugal
jpms@ecs.soton.ac.uk,vmm@sat.inesc-id.pt

Abstract. The MaxSAT problem and some of its well-known variants find an increasing number of practical applications in a wide range of areas. Examples include different optimization problems in system design and verification. However, most MaxSAT problem instances from these practical applications are too hard for existing branch and bound algorithms. One recent alternative to branch and bound MaxSAT algorithms is based on unsatisfiable subformula identification. A number of different unsatisfiability-based MaxSAT algorithms have been developed, which are effective at solving different classes of problem instances. All MaxSAT algorithms based on unsatisfiable subformula identification require using additional Boolean variables, either to allow relaxing some of the clauses or to encode cardinality constraints used by these algorithms. As a result, these algorithms may require using a significant number of additional Boolean variables, that can exceed the original number of variables for some problem instances. This paper proposes techniques for effectively reducing the number of auxiliary variables that must be used in unsatisfiability-based MaxSAT algorithms. Experimental results indicate that the techniques for reducing the number of auxiliary variables are effective, and contribute to more efficient MaxSAT algorithms.

1 Introduction

Maximum Satisfiability (MaxSAT) and variants allow modeling an increasingly large number of optimization problems in an also growing number of practical settings. The recent application of MaxSAT and variants in design debugging and verification of complex designs [11, 4, 5] motivated the development of new MaxSAT algorithms, capable of solving large structured problem instances common to these application domains. Despite the significant improvements made in recent years to standard branch and bound MaxSAT algorithms, in practice existing branch and bound algorithms are unable to solve the vast majority of problem instances from practical applications.

One recent promising line of research is the development of MaxSAT solvers based on the identification of unsatisfiable subformulas (or cores) [4, 8, 9]. These MaxSAT algorithms are built on top of SAT solvers, and so can exploit the most effective SAT techniques [2]. Moreover, these algorithms rely extensively on the ability of modern SAT solvers for producing certificates of unsatisfiability [12]. Even though the organization of existing unsatisfiability-based MaxSAT algorithms is fairly different, these algorithms also share a number of key common characteristics. For example, all unsatisfiability-based MaxSAT algorithms iteratively identify and relax unsatisfiable subformulas. The approach for relaxing unsatisfiable subformulas is well-known (e.g. see [8] for an overview), and consists of adding *relaxing* (or *blocking*) variables to each clause in each identified unsatisfiable subformula. Even though existing experimental results suggest great promise for unsatisfiability-based MaxSAT algorithms, many problem instances are still too complex even for the most effective algorithms. One clear potential drawback of unsatisfiability-based MaxSAT algorithms is the iterated addition of auxiliary variables. For many problem instances, it is possible that the number of additional variables becomes far larger than the original number of variables. As a result, besides the increase of search space, the much larger number of variables can often have a negative effect on SAT solvers. This paper proposes techniques for reducing the number of additional variables used in unsatisfiability-based MaxSAT algorithms. The first technique addresses the encoding of the cardinality constraints relating blocking variables. The second technique addresses the reduction of the actual number of blocking variables. Experimental results, obtained on a wide range of practical problem instances, indicates that the reduction of additional variables can often contribute to significantly reduce run times. The paper is organized as follows. The next two sections introduce MaxSAT and variants, existing branch and bound algorithms, and recent unsatisfiability-based algorithms for MaxSAT. Afterwards, Section 4 proposes the new techniques for reducing the number of variables. The new MaxSAT solvers are evaluated in Section 5 and the paper concludes in Section 6.

2 Maximum Satisfiability

This section provides definitions and background knowledge for the MaxSAT problem; familiarity with SAT and related topics is assumed [2]. The maximum satisfiability (MaxSAT) problem can be stated as follows: given a SAT instance represented in Conjunctive Normal Form (CNF), compute an assignment to the variables that maximizes the number of satisfied clauses. Variants of the MaxSAT problem include the partial MaxSAT, the weighted MaxSAT problem and the partial weighted MaxSAT problem. In the partial MaxSAT problem some clauses (i.e. the *hard* clauses) must be satisfied, whereas others (i.e. the *soft* clauses) may not be satisfied. Weighted variants are addressed elsewhere (e.g. see [5]). MaxSAT algorithms have been subject to significant improvements over the last decade (see for example [6, 5] for a review of past work). Despite the clear relationship with the SAT problem, most modern SAT techniques cannot be applied directly to the MaxSAT problem [6, 5]. As a result, the most successful MaxSAT algorithms implement branch and bound search, and integrate sophisticated lower bounding and inference techniques [5, 6]. Effective lower bounding techniques are based on unit propagation, whereas effective inference techniques can be viewed as based on specific resolution patterns. One alternative approach for solving the MaxSAT problem is to use Pseudo-Boolean Optimization (PBO). An overview is provided in [8].

3 Unsatisfiability-Based MaxSAT Algorithms

As mentioned in the previous section, one of the major drawbacks of the PBO model for MaxSAT is the large number of blocking variables that must be considered. The ability to reduce the number of required blocking variables is expected to improve significantly the ability of SAT/PBO based solvers for tackling instances of MaxSAT. Moreover, any solution to the MaxSAT problem will be unable to satisfy clauses that *must* be part of an unsatisfiable subformula. Consequently, one approach for reducing the number of blocking variables is to associate blocking variables only with clauses that are part of unsatisfiable subformulas. However, it is not simple to identify all clauses that are part of unsatisfiable subformulas. One alternative is the identification and relaxation of unsatisfiable subformulas. A number of unsatisfiability-based MaxSAT algorithms have been proposed in recent years [4, 8, 9]. The first algorithm [4] (referred to as `msu1`) iteratively finds unsatisfiable cores, adds new blocking variables to the non-auxiliary clauses in the unsatisfiable core, and requires that exactly one of the new blocking variables must be assigned value 1. The algorithm terminates whenever the CNF formula is satisfiable, and the number of assigned blocking variables is used for computing the solution to the MaxSAT problem instance. The clauses used for implementing the cardinality constraints are declared auxiliary; all other clauses are non-auxiliary. Observe that each non-auxiliary clause may receive more than one blocking variable, and the total number of blocking variables a clause receives corresponds to the number of times the clause is part of an unsatisfiable core. In the `msu1` algorithm [4] the pairwise encoding is used for encoding AtMost1 constraints. In contrast, `msu1.1` [8][1] proposes different linear encodings. Also, `msu1.1` uses AtMost1 constraints on the blocking variables associated with each clause. Alternative unsatisfiability-based MaxSAT algorithms, `msu3` and `msu4`, were proposed recently [8, 9]. Both `msu3` and `msu4` use a *single* cardinality constraint to constrain the number of blocking variables that can be assigned value 1, and so ensure that at most one blocking variable is required for each clause. `msu4` iterates between lower and upper bounds on the number of blocking variables. In contrast, `msu3` resembles `msu1` and variants, where only a lower bound on the number of blocking variables is updated. Existing experimental results indicate that `msu1.1` is often more efficient than either for `msu3` or `msu4` for most problem instances.

4 Reducing the Number of Additional Variables

This section describes two techniques for reducing the number of variables. The first one addresses the encoding of cardinality constraints, while the second one extends the same ideas to the way blocking variables are used. The original unsatisfiability-based (partial) MaxSAT algorithm [4] used the pairwise encoding for the AtMost 1 cardinality constraints (this algorithm will be referred to as `msu1`). A more effective approach is to use a linear encoding for the AtMost1 cardinality constraint (e.g. [8] compares a number of alternative linear encodings). For problem instances with large unsatisfiable cores, the linear encodings are significantly more effective. The linear encodings use a linear number of additional variables, the auxiliary variables, and a linear number of clauses. The number of additional variables, albeit linear, can be a potential drawback for some problem instances.

One approach to reduce the number of additional variables is to use the recently proposed *bitwise encoding* [10]. Consider an AtMost1 constraint $\sum_{i=0}^{k-1} x_i \leq 1$. Create

[1] For consistency, the algorithm `msu2` in [8] is renamed to `msu1.1`.

r auxiliary variables, where $r = 1$ if $k = 1$ and $r = \lceil \log k \rceil$ if $k > 1$. Let $v_0, \ldots, v_{r-1}$ be the auxiliary variables. Now associate with each x_i the binary representation of $i-1$. Finally, for each x_i create the clauses: $(\neg x_i \vee p_j)$, $j = 0, \ldots, r-1$, where $p_j = v_j$ if the binary representation of $i-1$ has value 1 in position j, and $p_j = \neg v_j$ otherwise. For an AtMost1 constraint with k variables, the bitwise encoding requires $\mathcal{O}(\log k)$ variables and $\mathcal{O}(k \ \log k)$ clauses, i.e. $\mathcal{O}(\log k)$ for each variable in the AtMost1 constraint. Observe that linear encodings (e.g. [8]) require a linear number of auxiliary variables and a linear number of clauses. Hence, the bitwise encoding trades off variables for clauses.

Given that the number of iterations of `msu1` and `msu1.1` is $\mathcal{O}(m)$ [8], where m is the number of clauses in the original formula, the number of additional variables for these algorithms is in $\mathcal{O}(m^2)$. By using the bitwise encoding, this asymptotic complexity remains unchanged, but the actual constant is considerably smaller. Also, observe that the number of blocking variables will be the same; only the number of auxiliary variables used for encoding the AtMost1 constraint is reduced. One should also observe that the created binary clauses only selects which variable of the AtMost1 constraint *can* be assigned value 1; all the other variables are required to be assigned value 0. Hence, the encoding effectively represents an AtMost1 constraint. In order to encode the constraint Exactly1, it would suffice to simply add a clause to capture the AtLeast1 constraint (e.g. [8]). The modified algorithm, using a logarithmic number of auxiliary variables for representing AtMost1 constraints, is referred to as `msu1.2`.

The use of the bitwise encoding [10] above for reducing the number of auxiliary variables, motivates using the same ideas for actually reducing the number of blocking variables. Instead of selecting at most one blocking variable out of set of k blocking variables, the bitwise encoding now operates on the clauses of each identified unsatisfiable core. This essentially allows eliminating the blocking variables by working directly with the auxiliary variables used in the bitwise encoding. For an unsatisfiable core with k clauses, the proposed encoding will require r auxiliary variables, where $r = 1$ if $k = 1$ and $r = \lceil \log k \rceil$ if $k > 1$. Moreover, the encoding will require $\mathcal{O}(\log k)$ variables and $\mathcal{O}(k \ \log k)$ new clauses, i.e. $\mathcal{O}(\log k)$ new clauses for each original clause in the unsatisfiable core. Hence each original clause ω_i in an unsatisfied core *generates* $\mathcal{O}(\log k)$ new clauses. The proposed approach needs to take into consideration when a clause has already been relaxed. Assume clause ω_{ij}, relaxed from an original clause ω_i, is included in an identified unsatisfiable core. Then *all* clauses generated from ω_i need to be re-relaxed. The modified algorithm, using a logarithmic number of blocking variables for each unsatisiifiable core, is referred to as `msu2`.

5 Results

This section summarizes results on MaxSAT and partial MaxSAT instances from practical applications. A number of classes of instances were considered. Class DEBUG [11] represents design debugging MaxSAT instances. FIR [7] represents filter design partial MaxSAT instances. Class SYN [7] represents logic synthesis partial MaxSAT instances. Finally, class MTG [7] represents minimum size test pattern generation partial MaxSAT instances. All partial MaxSAT instances are obtained by translating restricted pseudo-Boolean problem instances into partial MaxSAT (e.g. using a recently proposed

Table 1. Number of aborted instances, with a 1000 seconds timeout

Class	#I	`maxsatz`	`minimaxsat`	`minisat+`	`msu1`	`msu1.1`	`msu1.2`	`msu2`	`msu3`	`msu4`
DEBUG	65	62	65	63	22	14	**8**	11	24	23
FIR	59	–	45	37	15	14	**9**	12	32	44
SYN	74	–	46	44	48	44	**42**	**42**	47	51
MTG	215	–	7	**0**	44	44	52	60	11	16
Total	413	–	163	144	129	116	**111**	125	114	134

translation [5]). A number of MaxSAT solvers was considered, namely: `maxsatz` [6], the best performing solver in the MaxSAT 2007 evaluation [1]; `minimaxsat` [5], a recent competitive MaxSAT solver; and the PBO formulation of the MaxSAT problem solved with `minisat+` [3], one of the best performing PBO solvers [7]. Moreover, the unsatisfiability-based MaxSAT solvers considered were `msu1` [4]; `msu1.1` [8] (renamed from `msu2` for naming consistency); `msu3` [8]; `msu4` [9]; and the new algorithms described in this paper `msu1.2` and `msu2`. All `msu` algorithms are built on top of the same unsatisfiable core extractor, implemented with `minisat` 1.14 [2]. Other alternative MaxSAT algorithms (see [8, 9] for an overview) are known not to be efficient for these instances. Moreover, `minisat+` was run with its best configuration for these classes of instances and, for the partial MaxSAT instances, the original PBO instances were considered. The results for all MaxSAT solvers on all problem instances were obtained on a Linux server running RHE Linux, with a Xeon 5160 3.0 GHz dual-core processor. For the experiments, the available physical memory of the server was 2 GByte. The time limit was set to 1000 seconds per instance. Table 1 summarizes the number of aborted instances for the MaxSAT solvers considered. Overall, CPU times correlate well with the number of aborted instances and are not shown due to lack of space (see [8, 9] for plots for the other algorithms). For `maxsatz`, only MaxSAT results are shown, since `maxsatz` cannot be used with partial MaxSAT instances. The solvers exhibiting the best performance are highlighted in bold. The results indicate that the new algorithms are more efficient than previous MaxSAT algorithms. With the exception of class MTG, MaxSAT algorithms are now vastly superior to minisat+, one of the best performing PBO solvers [3]. For classes DEBUG and FIR, the instances aborted by either `msu1.2` and `msu2` are a fraction of the instances aborted by `minisat+`. Similarly, the two branch and bound algorithms considered, `maxsatz` and `minimaxsat`, perform much worse than *any* of the unsatisfiability-based MaxSAT algorithms. The results indicate that `msu1.2` is the overall best performing algorithm. `msu2` does not perform as well, especially for class MTG. This is in part explained by the growth in the number of clauses that `msu2` often requires. The improvements introduced by `msu1.2` allow significantly reducing the number of aborted instances for classes DEBUG and FIR, and also reducing the number of aborted instances for class SYN. For class MTG the results for `msu1.2` are worse than for `msu1.1`, and also worse than for `msu3` and `msu4`. Moreover, the results also suggest that `msu1.2` is the preferred algorithm for classes DEBUG, FIR and SYN, and that `minisat+` is the preferred algorithm for class MTG. Together, `msu1.2` and `minisat+` abort only 50 instances, thus motivating a portfolio of algorithms for MaxSAT. The experimental results confirm that techniques for reducing the number of additional variables are often effective. The performance of `msu2` is affected by the significant increase in the number of clauses that can often take place. As a result, a mixed approach, involving `msu1.2` and `msu2` should be considered.

6 Conclusions

Despite the significant improvements in MaxSAT algorithms over the last few years [6, 5], current state of the art MaxSAT solvers are ineffective on many problem instances obtained from practical applications [11, 5, 9]. This paper continues recent work on developing MaxSAT algorithms based on identification of unsatisfiable sub-formulas [4, 8, 9], by proposing effective techniques for reducing the number of additional variables that must be used. Two different algorithms are developed `msu1.2` and `msu2`. The experimental results indicate that the proposed techniques are effective, and that `msu1.2` is the most effective algorithm. The results also suggest that a mixed approach between `msu1.2` and `msu2` is expected to provide the most efficient approach.

Acknowledgement. This work is partially supported by EPSRC grant EP/E012973/1, by EU grants IST/033709 and ICT/217069, and by FCT grants POSC/EIA/61852/2004 and PTDC/EIA/76572/2006.

References

1. J. Argelich, C. M. Li, F. Manyà, and J. Planes. MaxSAT evaluation. www.maxsat07.udl.es.
2. N. Een and N. Sörensson. An extensible SAT solver. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 502–518, May 2003.
3. N. Een and N. Sörensson. Translating pseudo-Boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2, March 2006.
4. Z. Fu and S. Malik. On solving the partial MAX-SAT problem. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 252–265, August 2006.
5. F. Heras, J. Larrosa, and A. Oliveras. MiniMaxSat: a new weighted Max-SAT solver. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 41–55, May 2007.
6. C. M. Li, F. Manyà, and J. Planes. New inference rules for Max-SAT. *Journal of Artificial Intelligence Research*, 30:321–359, 2007.
7. V. Manquinho and O. Roussel. Pseudo-Boolean evaluation. www.cril.univ-artois.fr/PB07.
8. J. Marques-Silva and J. Planes. On using unsatisfiability for solving maximum satisfiability. *Computing Research Repository*, abs/0712.0097, December 2007.
9. J. Marques-Silva and J. Planes. Algorithms for maximum satisfiability using unsatisfiable cores. In *Design, Automation and Testing in Europe Conference*, March 2008.
10. S. D. Prestwich. Variable dependency in local search: Prevention is better than cure. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 107–120, May 2007.
11. S. Safarpour, H. Mangassarian, A. Veneris, M. H. Liffiton, and K. A. Sakallah. Improved design debugging using maximum satisfiability. In *Formal Methods in Computer-Aided Design*, pages 13–19, November 2007.
12. L. Zhang and S. Malik. Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications. In *Design, Automation and Testing in Europe Conference*, pages 10880–10885, March 2003.