Universitat de Lleida

SAT-based approaches for constraint optimization

Joel Gabàs Masip

http://hdl.handle.net/10803/396610

SAT-based Approaches for Constraint Optimization

PhD Thesis Dissertation by Joel Gabàs
Supervisor: Dr. Carlos Ansótegui

Doctorat en Enginyeria i
Tecnologies de la Informació

Universitat de Lleida

2016

Resum

L'optimització amb restriccions ha estat utilitzada amb èxit per a resoldre problemes en molts dominis reals (industrials). Aquesta tesi es centra en les aproximacions lògiques, concretament en Màxima Satisfactibilitat (MaxSAT) que és la versió d'optimització del problema de Satisfactibilitat booleana (SAT). A través de MaxSAT, s'han resolt molts problemes de forma eficient. Famílies d'instàncies de la majoria d'aquests problemes han estat sotmeses a la MaxSAT Evaluation (MSE), creant així una col·lecció pública i accessible d'instàncies de referència.

En les edicions recents de la MSE, els algorismes *SAT-based* han estat les aproximacions que han tingut un millor comportament per a les instàncies industrials. Aquests algorismes reformulen el problema d'optimització MaxSAT en una seqüència de problemes de decisió SAT amb un valor donat per al cost que s'ha d'optimitzar. Les instàncies SAT amb un valor per sota de l'òptim són insatisfactibles y les altres satisfactibles. Depenent de quines instàncies SAT es resolen per a guiar la cerca, hi ha dos tipus principals d'algorismes *SAT-based*: els algorismes *core-guided* refinen la cota inferior amb l'ajuda de subproblemes insatisfactibles (*cores*) obtinguts de las instàncies SAT insatisfactibles; i els algorismes *model-guided* refinen la cota superior amb les assignacions (*models*) obtingudes de las instàncies SAT satisfactibles.

Hem centrat la nostra investigació inicial en trobar un balanç eficient entre les aproximacions *core-guided* i *model-guided*. Concretament, després de millorar alguns algorismes *core-guided* inicials, hem incorporat alguna de las fortaleses dels algorismes *model-guided* a l'esquema general d'un algorisme *core-guided*. Fent això, la cota inferior va convergir més ràpidament cap a l'òptim i vam ser capaços d'obtenir assignacions y cotes superiores, permetent així a un algorisme complet treballar de forma natural com una aproximació incompleta donat un temps limitat. La nostra contribució final al desenvolupament de les aproximacions *SAT-based* és el disseny d'un nou algorisme, per al qual hem desenvolupat tècniques addicionals per a explotar l'estructura global dels *cores* insatisfactibles.

Tot aquest treball ha contribuït a tancar varies instàncies obertes i a reduir dramàticament el temps de resolució en moltes altres. A més, hem trobat sorprenentment que reformular y resoldre el problema MaxSAT a través de programació lineal sencera era especialment adequat per algunes famílies. Finalment, hem desenvolupat el primer portfoli altament eficient per a MaxSAT que ha dominat en totes las categories de la MSE des de 2013.

Resumen

La optimización con restricciones ha sido utilizada con éxito para resolver problemas en muchos dominios reales (industriales). Esta tesis se centra en las aproximaciones lógicas, concretamente en Máxima Satisfacibilidad (MaxSAT) que es la versión de optimización del problema de Satisfacibilidad booleana (SAT). A través de MaxSAT, se han resuelto muchos problemas de forma eficiente. Familias de instancias de la mayoría de ellos han sido sometidas a la MaxSAT Evaluation (MSE), creando así una colección pública y accesible de instancias de referencia.

En las ediciones recientes de la MSE, los algoritmos *SAT-based* han sido las aproximaciones que han tenido un mejor comportamiento para las instancias industriales. Estos algoritmos reformulan el problema de optimización MaxSAT en una secuencia de problemas de decisión SAT con un valor dado para el coste que hay que optimizar. Las instancias SAT con un valor por debajo del óptimo son insatisfacibles y las otras satisfacibles. Dependiendo de qué instancias SAT se resuelven para guiar la búsqueda, hay dos tipos principales de algoritmos *SAT-based*: los algoritmos *core-guided* refinan la cota inferior con la ayuda de subproblemas insatisfacibles (*cores*) obtenidos de las instancias SAT insatisfacibles; y los algoritmos *model-guided* refinan la cota superior con las asignaciones (*models*) obtenidas de las instancias SAT satisfacibles.

Hemos centrado nuestra investigación inicial en encontrar un balance eficiente entre las aproximaciones *core-guided* y *model-guided*. Concretamente, tras mejorar algunos algoritmos *core-guided* iniciales, hemos incorporado algunas de las fortalezas de los algoritmos *model-guided* al esquema general de un algoritmo *core-guided*. Haciendo esto, la cota inferior convergía más rápidamente hacia el óptimo y éramos capaces de obtener asignaciones y cotas superiores, permitiendo así a un algoritmo completo trabajar de forma natural como una aproximación incompleta dado un tiempo limitado. Nuestra contribución final al desarrollo de las aproximaciones *SAT-based* es el diseño de un nuevo algoritmo, para el cual hemos desarrollado técnicas adicionales para explotar la estructura global de los *cores* insatisfacibles.

Todo este trabajo ha contribuido a cerrar varias instancias abiertas y a reducir dramáticamente el tiempo de resolución en muchas otras. Además, hemos encontrado sorprendentemente que reformular y resolver el problema MaxSAT a través de programación lineal entera era especialmente adecuado para algunas familias. Finalmente, hemos desarrollado el primer portfolio altamente eficiente para MaxSAT que ha dominado en todas las categorías de la MSE desde 2013.

Abstract

Constraint optimization has been successfully used to solve problems in many real world (industrial) domains. This PhD thesis is focused on logic-based approaches, in particular, on Maximum Satisfiability (MaxSAT) which is the optimization version of Satisfiability (SAT). There have been many problems efficiency solved through MaxSAT. Instance families on the majority of them have been submitted to the international MaxSAT Evaluation (MSE), creating a collection of publicly available benchmark instances.

At recent editions of MSE, SAT-based algorithms were the best performing single algorithm approaches for industrial problems. These algorithms reformulate the MaxSAT optimization problem into a sequence of SAT decision problems with a given value of the cost to be optimized. The SAT instances with a value below the optimum are unsatisfiable and the others satisfiable. Depending on which SAT instances are tested to guide the search, there are two main types of SAT-based algorithms: core-guided algorithms refine the lower bound with the help of unsatisfiable subproblems (cores) obtained from unsatisfiable SAT instances; and model-guided algorithms refine the upper bound with the help of satisfying assignments (models) obtained from satisfiable SAT instances.

We have focused our initial work on finding an efficient balance between core-guided and model-guided approaches. In particular, after improving some initial core-guided algorithms, we have incorporated to the general schema of a core-guided algorithm some of the strengths of model-guided algorithms. The main idea of our approach was optimizing the subproblems related to the unsatisfiable cores. By doing so, the lower bound converged more quickly to the optimum and we were also able to get assignments and upper bounds, allowing in this way a complete algorithm to work naturally as an incomplete approach given limited time. Our final contribution to the development of SAT-based approaches is the design of a new algorithm, for which we have developed additional techniques to exploit the global structure of unsatisfiable cores.

All this work has contributed to close up some open instances and to reduce dramatically the solving time in many others. In addition, we have surprisingly found that reformulating and solving the MaxSAT problem through Integer Linear Programming (ILP) was extremely well suited for some families. Finally, we have developed the first highly efficient MaxSAT portfolio that dominated all categories of MSE since 2013.

Acknowledgements

First of all, I would like to thank my supervisor, Dr. Carlos Ansótegui, who introduced me to the field of constraint optimization and guided my research. He has always been very dedicated, supported me throughout all difficulties and inspired me to do my best. Without him this PhD thesis would not have been possible.

I would also like to thank all the institutions that supported this work. In particular, it has been partially founded by the CICYT research project TASSAT (TIN2010-20967-C04-01/03/04), the CICYT research project ARINF (TIN2009-14704-C03-01), the Ministerio de Economía y Competividad research project TASSAT2 (TIN2013-48031-C4-4-P) and Google Faculty Research Award program.

Finally, I am also in debt with my family for their support in all aspects.

1 Introduction

Constraint optimization is the process of optimizing an objective function of variables whose values are subject to some constraints. This process has been successfully used to solve problems in many real world (industrial) domains. The literature shows us that some problems are more efficiently solved with mixed integer programming and others with logic-based approaches. The work done during this PhD thesis is focused on the second group, in particular, on Maximum Satisfiability (MaxSAT), the optimization version of Satisfiability (SAT). The list of problems efficiency solved through MaxSAT includes: software package upgrade [1], debugging of hardware designs [2, 3], bioinformatics [4, 5], fault localization in C code [6], course timetabling [7], planing [8, 9], scheduling [10], routing [11], electronic markets [12], combinatorial auctions [13] and many others [14, 15]. In order to assess the state-of-the art in the field of MaxSAT solvers, it is interesting to have a collection of publicly available MaxSAT instances including a wide number of families on the aforementioned problems. With this aim, the international MaxSAT Evaluation (MSE) [16, 17] was created in 2006.

MaxSAT expresses the optimization problem as a formula of Boolean variables grouped in clauses. The idea is that sometimes not all these clauses can be satisfied, and we try to satisfy the maximum number of them. The basic MaxSAT problem can be further generalized to the Weighted Partial MaxSAT (WPMS) problem. In this case, we can divide the constraints into two groups: the clauses that must be satisfied (hard), and the ones that may or may not be satisfied (soft). Since not all constraints are equally important, each soft clause may have a different weight indicating the cost of falsifying it. The presence of soft clauses with different weights makes a MaxSAT instance Weighted and the presence of hard clauses makes it Partial. Then, to solve the MaxSAT problem we have to find an assignment to the variables that satisfies all hard clauses and minimizes the objective function corresponding to the aggregated cost of falsified soft clauses.

Solving exactly the MaxSAT problem, i.e. finding and certifying an optimal assignment, can be hard from a computational point of view. There are however many industrial problems slightly beyond the reach of state-of-the-art techniques and many times the goal is not finding an optimal assignment but a good assignment in a reasonable time. For some domains, even a small gain in the quality of the assignment can lead to important practical consequences. Although many incomplete algorithms have been developed in this sense, the experience achieved from the MSE shows us that a reasonable strategy is to improve complete algorithms to solve exactly the problem and modify them so that they return assignments when improved. This way, given a limit of time, they can also work as incomplete algorithms.

There are two main types of MaxSAT complete algorithms: branch and bound [18–22] and SAT-based [14, 15, 23, 24]. At recent editions of MSE, we have seen that the best performing solvers on industrial instances are those implementing SAT-based algorithms. These algorithms proceed by reformulating the MaxSAT optimization problem into a sequence of SAT decision problems.

Intuitively, each SAT instance of the sequence encodes whether it exists an assignment to the MaxSAT instance with a cost less than or equal to a certain k. The SAT instances with a k less than the optimal cost are unsatisfiable, while the others are satisfiable. Therefore, when SAT-based algorithms find the phase transition point, they find the optimum. Depending on which SAT instances of the sequence are tested to guide the search, the two main types of SAT-based algorithms are: core-guided [25–30] and model-guided [31–34]. Core-guided algorithms refine the lower bound and guide the search with unsatisfiable subproblems (cores) obtained from unsatisfiable SAT instances. Model-guided algorithms refine the upper bound and guide the search with satisfying assignments (models) obtained from satisfiable SAT instances. Both have strengths and weaknesses and there have been already some hybrid approaches [35,36]. In particular, the work done during this PhD thesis aims to find an efficient balance between both approaches.

Initially, we focused our research on developing techniques that could be incorporated to the general schema of the core-guided WPM1/WBO algorithm [25,26], which is the weighted version of the Fu and Malik algorithm [23,24] (see Subsection 2.2). We presented this work in [37] where we basically showed how to apply stratification and hardening techniques to a core-guided algorithm. The idea was to improve the quality of the unsatisfiable cores retrieved from the unsatisfiable SAT instances, i.e., to get cores as small as possible with weights as similar as possible. Both techniques are still in use in many state-of-the-art core-guided solvers. In addition, we also developed a specific technique for the WPM1 algorithm to break the symmetries introduced during its execution.

Then, we continued our research incorporating some of the strengths of model-guided algorithms to the core-guide WPM2 [27] algorithm (see Subsection 2.3). In particular, we extended it with the possibility of obtaining assignments (models for the hard clauses) that can be used to refine the upper bound. In this way, we transformed a complete algorithm so that it can also work as an incomplete approach given limited time. We achieved this by performing the optimization of subproblems related to the unsatisfiable cores. In this sense, our approach integrated the best of core-guided algorithms (i.e. focusing on reduced parts of the formula) and model-guided algorithms (i.e. returning upper bounds and assignments for the whole formula). Moreover, by applying the subproblem optimization technique, the lower bound converged more quickly to the optimal cost. Furthermore, we developed an extra technique to exploit the assignments obtained during the optimization of subproblems, using them as a heuristics to guide the search. We presented this work partially in [38] and extended it in [39], where we have also provided a detailed analysis on how all these techniques improved the performance of our WPM2 solver.

Both WPM1 and WPM2, like the majority of SAT-based MaxSAT algorithms, use Pseudo-Boolean (PB) constraints to create the SAT instances of the sequence into which the input MaxSAT instance is reformulated. The PB constraints are used to express the arithmetic and comparison needed to only allow assignments with a cost less than or equal to a certain k. Currently, in most state-of-the-art SAT-based MaxSAT solvers, PB constraints are translated into SAT, since they use internally a SAT solver. However, there is no known SAT encoding which can guarantee the original propagation power of the constraint, i.e, what we call arc-consistency, while keeping a linear complexity in terms of size. The best approach so far, has a quadratic complexity [40]. Instead of translating the PB constraint into SAT, we can explore other options. In particular, to treat PB constraints with specialized inference mechanisms and a moderate cost in size, while preserving the strength of SAT techniques for the rest of the problem, we can use the Satisfiability Modulo Theories (SMT) technology [41]. We used this technology in [38, 39], showing that it is an alternative for implementing efficiently MaxSAT algorithms. However, SAT technology is the default option and SAT encodings for PB constraints are still being improved. In this sense, we are currently developing a a SAT encoding for PB constraints that allows to adjust the balance between size and arc-consistency. We provide in [42] a technical report explaining this encoding that is potentially useful to solve many MaxSAT instances efficiently.

Our final contribution to SAT-based MaxSAT algorithms is the design of the WPM3 algorithm, combining efficiently and effectively some techniques (see Subsection 2.5). In particular, this algorithm is able to optimize subproblems efficiently, while only using a simpler version of PB constraints, called Cardinality constraints. These constraints have all the coefficients of the variables equal to 1 and their best SAT encoding has a quasilinear complexity [43]. We have presented this work in [44], where we have also developed some techniques to exploit the global structure of unsatisfiable cores. We have used this structure both to build Cardinallity constraints that boost the performance of the solvers and to select the subproblems to be optimized. Additionally, we have further developed the technique to exploit the assignments obtained from the subproblems that was presented in [39]. In this sense, we have adapted to MaxSAT a very effective technique used in SAT solvers called phase saving [45]. We have extended [44] in [46], where we explain how our algorithm is related to other well performing core-guided algorithms, Eva [47] and OLL [48, 49], and which advantages has compared with them.

Although all the problem families included in the collection of the MSE are supposed to be more suitable for logic-based approaches, our study would not have been exhaustively conducted without showing the performance of integer programming techniques on them. Actually, we can easily translate a whole MaxSAT instance into an Integer Linear Programming (ILP) instance and apply a Mixed Integer Programming (MIP) solver. In [50] and [39], we confirmed that the ILP approach was overall not competitive on the industrial instances of the MSE. However, we surprisingly found that it was extremely well suited for many non-random instance families, among them some few industrial.

All the aforementioned approaches have different performances depending on the problem, being none of them dominant across all families of industrial instances we have experimented with, i.e. the ones of MSE. Hence, it is reasonable to use meta-algorithmic techniques to devise a portfolio. In this direction, we have adapted and improved some techniques of solver automatic configuration and selection algorithms, to make the first highly efficient MaxSAT portfolio ISAC+ [51] that dominates the MSE (see Subsection 2.6). The basic idea is that we have to predict which one of a set of solvers is the best option for an input instance and we have available a set of instances to make this prediction. Our approach consists basically of a configuration phase and a selection phase. During the configuration phase, we extend the set of solvers of our portfolio with different parametrizations of some solvers. To find this parametrizations, we cluster the set of instances and we apply an automatic configurator for each cluster. Then, we run all the solvers/parametrizations for the whole set of instances. During the selection phase, at runtime, we determine the solver/parametrization to use for the input instance. To determine this, we have the solving times and the features of the set of instances and we apply a selector that does not take into account the clusters of the configuration phase. Our portfolio has not only outperform all single solvers at each category of MSE since 2013, but has also been competitive on each subcategory.

Finally, to evaluate independently our research, we have submitted to the MSE, not only our portfolio, but also all the single solvers developed during this PhD thesis since 2012. With respect to the industrial instances, which are our main goal, we have competed in 18 subcategories, including complete and incomplete track. Our solvers placed 9 times in the first position, 8 times in the second position and 3 times in the third position, being the team with the best results in this period. Our work has contributed to close up some open instances and to reduce dramatically the solving time in many others. Moreover if we take into account that, some of the techniques we developed are incorporated to many other state-of-the-art solvers. In the next section we present the contributions of this PhD thesis with the references to the more relevant papers which are included at the end of this dissertation. Section 3 reports the results achieved at MSE and Section 4 concludes.

2 Research

In this section, we summarize the research done during this PhD thesis. The main goal was to boost the state-of-the-art performance in MaxSAT solving. To this end, we have: (i) identified new solving techniques to incorporate to SAT-based MaxSAT algorithms, (ii) designed and implemented a new complete SAT-based MaxSAT algorithm that can also work as incomplete, (iii) compared MaxSAT solvers with new approaches making use of ILP technology and (iv) designed and implemented a portfolio approach. The methodology was to: (a) study the state-of-the-art, (b) propose new techniques, evaluating theoretically their soundness, (c) implement them and (d) conduct an intensive experimentation. To evaluate independently our work, we have submitted the resulting solvers to the international MaxSAT Evaluation (MSE), as we explain in Section 3. We have also reported the results in form of papers, some of them published in leading conferences and journals (the most relevant ones are A, E, F and G):

A C. Ansótegui, M. L. Bonet, J. Gabàs, J. Levy, **Improving SAT-based Weighted MaxSAT Solvers**, in: Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming (CP'12), 2012, pp. 86 - 101.

B C. Ansótegui, J. Gabàs, **Solving (Weighted) Partial MaxSAT with ILP**, in: Proceedings of the 10th International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) techniques in Constraint Programming (CPAIOR'13), 2013, pp. 403 - 409.

C C. Ansótegui, M. L. Bonet, J. Gabàs, J. Levy, **Improving WPM2 for (Weighted) Partial MaxSAT**, in: Proceedings of the 19th International Conference on Principles and Practice of Constraint Programming (CP'13), 2013, pp. 117 - 132.

D C. Ansótegui, F. Didier, J. Gabàs, **Exploiting the Structure of Unsatisfiable Cores in MaxSAT**, in: Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI'15), 2015, pp. 283 - 289.

E C. Ansótegui, J. Gabàs, J. Levy, **Exploiting subproblem optimization in SAT-based MaxSAT algorithms**, Journal of Heuristics 22(1), 2016, pp. 1 - 53.

F C. Ansótegui, J. Gabàs, Y. Malitsky, M. Sellmann, **MaxSAT by improved instance-specific algorithm configuration**, Artificial Intelligence 235, 2016, pp. 26 - 39.

G C. Ansótegui, J. Gabàs, **WPM3: an (in)complete algorithm for Weighted Partial MaxSAT**, submitted to Artificial Intelligence

H C. Ansótegui, J. Gabàs, **A new SAT encoding for PB constraints allowing to adjust the balance between size and arc-consistency**, Internal Technical Report 05-06-2016.

These papers are also referred in the bibliography of this dissertation as: A [37], B [50], C [38], D [44], E [39], F [51], G [46] and H [42]. We explain in the following subsection the work done during the elaboration of the papers. First of all, Subsection 2.1 gives a concise overview of SAT-based MaxSAT algorithms. In Subsection 2.2, we explain the work done in [37] where we improved the WPM1 algorithm. Subsection 2.3 corresponds to [39], the extended version of [38], where we have introduced the optimization of subproblems to the WPM2 algorithm, improving its performance and allowing it to work as an incomplete approach given limited time. This subsection also mentions the comparison with a full translation into ILP, initially published in [50] and incorporated to [39]. In Subsection 2.4, we explain how to manage PB constraints in SAT-based MaxSAT algorithms and we present a technical report [42] describing a new SAT encoding.

Subsection 2.5 corresponds to [46], the extended version of [44], where we have designed the new (in)complete algorithm WPM3 that incorporates some new techniques to exploit the global structure of the unsatisfiable cores. Finally, in Subsection 2.6 we explain the work done in [51] that corresponds to the portfolio approach.

2.1 SAT-based MaxSAT Algorithms

The latest editions of MSE show us that the best option to solve efficiently industrial MaxSAT instances are SAT-based MaxSAT algorithms. Since we are mainly interested in real world problems, we have focused our research on these algorithms. The main idea is that they reformulate the MaxSAT optimization problem into a sequence of SAT decision problems. Each SAT instance of the sequence answers the question if it exists an assignment for the MaxSAT optimization problem with a cost less than or equal to a certain k. Thus, the SAT instances with a k less than the optimal cost are unsatisfiable, while the others are satisfiable. By finding the phase transition point, SAT-based algorithms find the optimum.

One of the key points of SAT-based algorithms is how to construct the SAT instances in the sequence. To construct one of these SAT instances, we need to detect which soft clauses are falsified under a certain assignment, sum up their cost and compare with k. To detect if a clause is falsified, we can extend it with an auxiliary variable, that becomes true if the clause is falsified. To add the weights of the falsified soft clauses and compare the cost with k, we use PB constraints, that can be managed by their translation into SAT (we discuss other options in Subsection 2.4).

A naive SAT-based algorithm would cover all the soft clauses with a unique PB constraint. Then, it would look for the phase transition point between the unsatisfiable instance with the maximum k (the optimum minus one) and the satisfiable instance with minimum k (the optimum). This could be achieved, for example, by testing different values of k beginning with 0 and increasing the lower bound by one until the first satisfiable instance. The same could be made beginning with the aggregated cost of all soft clauses and decreasing the upper bound by one until the first unsatisfiable instance. In order to skip some of these steps, model-guided algorithms, like minisat+ [31] and SAT4J [32], refined the upper bound with the assignments obtained from satisfiable SAT instances. Since SAT instances accept all assignments with a cost less than or equal to a certain k, the cost of an assignment retrieved from a SAT instance can be much lower than its k. By refining the upper bound with this assignments, the test of many intermediate values of k can be avoided. Also, the assignments and corresponding upper bound allow model-guided algorithms to work naturally as incomplete approaches given limited time.

One inflection point in the development of SAT-based algorithms was the emergence of the Fu and Malik algorithm [23, 24]. By focusing on unsatisfiable cores from unsatisfiable SAT instances while refining the lower bound, it was able to use smaller PB constraints. Many similar algorithms have been proposed since then, being called core-guided. For example, the WPM1/WBO algorithm [25,26] is the weighted version of the Fu and Malik algorithm. The WPM1 algorithm only uses 1-Cardinality constraints (Cardinality constraints that compare with a k equal to 1), but it may require multiple auxiliary variables per soft clause. Our initial work [37] improved WPM1 algorithm both with specific techniques and more general ones that can be used in other algorithms (see Subsection 2.2). More recent core-guided algorithms using general PB constraints only require at most one auxiliary variable per soft clause. The first algorithm to do so appears in [52]. The first algorithm to use a set of PB constraints, each one covering a disjoint unsatisfiable core (cover), was the PM2 algorithm [25]. The WPM2 algorithm [27], the weighted version of PM2, introduced for the first time an optimization module to refine the lower bound taking profit of the at-least constraints obtained from unsatisfiable cores.

Another interesting core-guided alternative is the MaxHS algorithm that appeared in [53]. It also splits the search into a satisfiability and an optimization module like the WPM2 algorithm. In the case of MaxHS, the optimization module returns an interpretation of minimal cost for the at-least constraints obtained from unsatisfiable cores. Then, the satisfiability module tests whether this interpretation satisfies the whole formula. Notice that, several iterations of the optimization and satisfiability modules may be needed to refine a lower bound. Roughly speaking, in [53] part of the work of the optimization module in [27] is transferred to the satisfiability module. The optimization module can use any desired approach to solve the problem. A SAT-based approach was used in [27] and a MIP solver in [53]. In our work [50], we pointed out that translating the MaxSAT problem into ILP and applying a MIP solver, was extremely well suited for some instance families. In [54], MaxHS transferred some hard clauses of the satisfiability module to the optimization module (MIP solver).

All the aforementioned core-guided algorithms refine the lower bound. There have been however several hybrid approaches of core-guided algorithms that also refine the upper bound, like model-guided algorithms. For example, the BINCD algorithm [35,36] that used a disjoint set of PB constraints like WPM2, but also tested satisfiable SAT instances allowing assignments with a cost greater that the optimum. The phase transition point was searched with a binary search scheme. We have also modified the core-guided WPM2 algorithm [38,39], incorporating the hard clauses to the optimization module and using a model-guided SAT-based algorithm to solve these subproblems. By optimizing these subproblems, we obtain assignments that can be used to refine the upper bound and allow the algorithm to work as an incomplete approach (Subsection 2.3).

Core-guided algorithms have dominated on industrial subcategories at the latest editions of MSE. It seems that this good performance is due to the fact that they focus on restricted parts of the problem and they do not need to cover all the soft clauses with PB constraints. Let us see this in Table 1, appeared in [39] extending the work done in [55]. In the table, we show a fair comparison of three solvers implemented using the same technology, $wpm2$ (core-guided), $sat4j$ (model-guided) and $certify\text{-}opt$. This last solver only has to certify the optimum

				*wpm*2			*certify-opt*			*sat*4*j*		
Subcategory	#	$\#H$	$\#S$	$\#*$	$\%b$	$\#AM$	$\#*$	$\%b$	$\#AM$	$\#*$	$\%b$	$\#AM$
MS	55	0.0	$1.8{\cdot}10^6$	20	0.8	41.2	0	100	1	0	100	1
PMS	627	$3.3{\cdot}10^5$	$1.3{\cdot}10^4$	528	63.6	87.8	267	100	1	247	100	1
WPMS	396	$4.9{\cdot}10^5$	$8.6{\cdot}10^3$	333	33.9	426	59	100	1	55	100	1
Total industrial	1078	$3.7{\cdot}10^5$	$9.4{\cdot}10^4$	881	49.6	212	326	100	1	302	100	1

Table 1. *wpm*2, *certify-opt* and *sat*4*j* on the industrial set of MSE 2013.

given by an oracle, i.e. it just tests the two SAT instances that define the phase transition point between satisfiable and unsatisfiable. To create the instances, *sat*4*j* and *certif-opt* use a unique PB constraint covering all soft clauses. The comparison was performed on the industrial set of MSE 2013. Rows correspond to the three subcategories (MS, PMS and WPMS) and the total. With respect to columns, # stands for the total number of instances, $\#H$ and $\#S$ stand for the mean number of hard and soft clauses, respectively. For each solver, $\#*$ stands for the number of solved instances, within a timeout of 7200 seconds, $\%b$ shows the percentage of covered soft clauses and $\#AM$ stands for the number of PB constraints. We can see that the performance of *sat*4*j* and *certify-opt* is very similar. However, *wpm*2 performs much better than *certify-opt* (a difference of 555 solved instances). We can see that, in effect, *wpm*2 only covers a part of the soft clauses and with a disjoint set of PB constraints. In particular, it only covers an average of 49.6% of the soft clauses with an average of 212 PB constraints. This experimtne shows us that it is worth doing all the work to extract the cores, since *wpm*2 outperforms the solvers that use a unique PB constraint for all soft clauses, even if they only have to certify the known optimumm (*certify-opt*).

Finally, some new algorithms have recently appeared that only require at most one auxiliary variable per soft clause while using Cardinality constraints. To our best knowledge, the first one is the OLL algorithm [48,49]. Although with apparently different foundations, the Eva algorithm has been presented in [47]. It works by applying the MaxSAT resolution rule to create the SAT instances in the sequence into which it reformulates the input MaxSAT instance. In [46], we show that the Eva algorithm is also implicitly working with Cardinality constraints and both OLL and Eva algorithms are in fact very similar. Also in [46], we have made a step forward and designed the WPM3 algorithm. It also works with at most one auxiliary variable per soft clause and Cardinality constraints and, in addition, it is able to perform the optimization of subproblems efficiently like WPM2 [38, 39] (see Subsection 2.5).

2.2 Improving Initial Core-guided Algorithms: Stratification

Our first contributions were improvements to initial core-guided algorithms. They were published in [37], that is included as attached paper A at the end of this dissertation. In this work, we studied the the Fu and Malik algorithm [23,24] and its weighted version, the WPM1/WBO algorithm [25,26]. We identified some weaknesses and we tried to correct them incorporating some new techniques to the algorithms. These techniques consisted basically in breaking symmetries of the auxiliary variables and using stratification and hardening techniques to improve the quality of the unsatisfiable cores retrieved from unsatisfiable SAT instances. The symmetry breaking technique is specific for these algorithms, but the stratification and hardening techniques are more general and can be used in other more recent core-guided algorithms. In fact, they are still in use in many state-of-the-art core-guided solvers.

With respect to the symmetry breaking, the symmetries are generated during the execution of the Fu and Malik and WPM1 algorithms due to the fact that both require multiple auxiliary variables per soft clause. Therefore, there can be different assignments on auxiliary variables that falsify the same soft clauses. These symmetries caused extra steps during the process of testing the SAT instances. The technique we applied to mitigate this problem consisted in introducing some new clauses on the auxiliary variables that did not allow some of the assignments falsifying the same soft clauses.

With respect to the stratification, it consists in solving initially a subformula containing the hard clauses and only some soft clauses. Once solved, it is extended with more soft clauses and the process is repeated until all soft clauses are incorporated. The criteria to apply the stratification was the weight of the soft clauses, i.e. we incorporated them following an order of decreasing weight. By doing so, we forced that the unsatisfiable cores, retrieved from unsatisfiable SAT instances, contained soft clauses with similar weights, which prevented a problem that we had detected in the WPM1 algorithm. The strategy that WPM1 used to handle the weighted soft clauses consisted in replacing each one with two copies whose aggregated weight was equal to the weight of the original. In this way, it was able to have groups of soft clauses with the same weight and treat them like the Fu and Malik algorithm. By having cores with similar weights for the soft clauses, we prevented the generation of instances with a huge number of copies of the same soft clauses. Later, the stratification technique has been successfully applied to many other core-guided algorithms. Applying the stratification to all weights is however not good for all instances. On instances with a high diversity of weights, it can result in as many SAT tests as soft clauses, increasing dramatically the solving time in some cases. This is why, we applied a heuristics that incorporated to the subformula several soft clauses with different weights at the same time, if their diversity of weights was high.

With respect to the hardening, it is intimately related to the stratification. The idea is that, we can consider some clauses as hard if we know that their falsification leads to assignments with a cost greater than a known upper bound. Although WPM1 is a core-guided algorithm and refines only the lower bound, we can obtain also upper bounds thanks to the stratification. The optimum obtained for a solved subformula can be added to the cost of all non-incorporated clauses to obtain an upper bound for the whole formula. By considering as hard those soft clauses with a weight greater than the aggregated cost of all non-incorporated clauses, we obtained cores with less soft clauses.

2.3 Exploiting Subproblem Optimization

In our initial work [37] explained in the previous subsection, we solved restricted parts of the problem thanks to the stratification technique. Each subproblem included the hard clauses and some soft clauses, being its optimal cost a lower bound for the cost of its soft clauses in the whole problem. We noticed that it was desirable to focus on solving those parts of the problem with high optimal costs, so our next contribution was to further exploit the optimization of subproblems. In particular, we improved the core-guided algorithm WPM2 [27] by applying the optimization of the subproblems related to unsatisfiable cores. Roughly speaking, we incorporated the hard clauses to the set of at-least constraints that was used in [27] to compute the new lower bound for a set of soft clauses. By doing so, we made the global lower bound converge more quickly to the optimal cost. This work was partially published in [38] and has been extended in [39], both included at the end of this dissertation as attached papers C and E, respectively.

The optimization of subproblems can be performed through any optimization approach. This allows us to combine the strength of exploiting the information extracted from unsatisfiable cores and other approaches. In particular, we experimented with an ILP approach and three SAT-based approaches. With respect to the ILP approach, we applied the one described in [50], where we had used it to solve the whole problem. Although in [50] the full translation into ILP was overall not competitive on the industrial set of the MSE, we surprisingly found that it was extremely well suited for many non-random families, among them some few industrial. With respect to the three SAT-based approaches, we: (i) refined the lower bound with the *subsetsum* function [27, 56], (ii) refined the upper bound with a model-guided algorithm [32–34], and (iii) used a binary search scheme [35, 57] where the lower bound and upper bound are refined as in the previous approaches. The results confirmed that optimizing the subproblems with the ILP approach was not competitive compared with the SAT-based approaches. The SAT-based approach with the best performance in our experimental analysis was (ii), corresponding to a model-guided algorithm. By applying the subproblem optimization technique, the performance of the WPM2 solver experimented an important boost.

The WPM2 algorithm is a complete core-guide algorithm that initially refined only the lower bound. By applying the subproblem optimization technique, we have extended it with the possibility of obtaining assignments for the subproblems that can be used to refine the global upper bound. In this way, it is now able to work as an incomplete approach given limited time, like model-guided algorithms. The incomplete solvers using this technique that we have submitted to the MSE in 2014 and 2015 have dominated the incomplete track for industrial instances. In order to further exploit the optimization of subproblem, we have studied how to use the assignments as a heuristic to guide and boost the search. We have also incorporated to the WPM2 algorithm other improvements like the stratification technique as described in [37] and a hardening technique establishing a new criteria different from those in [36, 37].

From the perspective of coming up with an efficient implementation, SAT-based algorithms need to be implemented on top of an efficient solver based on SAT technology. In this work, we used the Satisfiability Modulo Theories (SMT) technology [41], showing that it is an alternative for implementing efficiently SAT-based algorithms. (see Subsection 2.4). One of the important features that SAT/SMT solvers should have to result into an efficient MaxSAT solver is that they should allow to assert and retract PB constraints and soft clauses incrementally. In this way, we can preserve some learned lemmas from previous iterations that may help to reduce the search space. We have seen that this incrementality is crucial for the efficiency of the improved WPM2 solver, since it allows us to optimize the subproblems with the same SAT/SMT solver that is used to extract the cores.

Finally, our aim is not only to present a method that performed well, but also to understand why this was the case. This understanding will allow us to identify the interaction with other future improvements in the field and whether they are complementary or not to this work. In [39] we have extended the study originally presented in [58] on the structure of the unsatisfiable cores obtained during the search process of core-guided algorithms (see Table 1). Also, we have showed that our improved WPM2 can obtain high quality assignments in a reasonably short time.

2.4 The Management of PB Constraints in SAT-based Algorithms

SAT-based MaxSAT algorithms use Pseudo-Boolean (PB) constraints to create the SAT instances of the sequence into which the input MaxSAT instance is reformulated. These PB constraints are used to express the arithmetic and comparison needed to only allow satisfying assignments with a cost less than or equal to a certain k. The size, the management and the complexity of these PB constraints are crucial for SAT-based algorithms. With respect to the size, the naive approach uses a unique PB constraint that involves all soft clauses. Fortunately, core-guided algorithms use a set of smaller PB constraints that do not necessarily cover all soft clauses. With respect to the management, the default option is to translate PB constraints into SAT and use internally a SAT solver. However, there are other options, like modeling PB constraints with the Linear Integer Arithmetic theory and using internally an SMT solver [39]. With respect to the complexity, in case PB constraints are managed through their translation into SAT, the best encoding that preserve arc-consistency has a quadratic size with respect to the size of the constraint [40]. Depending on the particular SAT-based algorithm, we may only need to use a simpler form of PB constraints with all the coefficients of the variables equal to 1, i.e. Cardinality constraints. For Cardinality constraints, the size of the best SAT encoding is quasilinear with respect to the size of the constraint [43].

There are many algorithms that only work with Cardinality constraints. For weighted instances in particular, the WPM1 algorithm only needs 1-Cardinality constraints but it requires multiple auxiliary variables per soft clause, what introduces additional complexity. There have been recently other approaches that only require at most one auxiliary variable per soft clause while using Cardinality constraints. For example, the OLL [48,49], the Eva [47] and later our WPM3 [46] algorithms (see Subsection 2.5). All this algorithms manage the PB constraints by its translation into SAT and use an underlying SAT solver.

There are however also many algorithms that work with general PB constraints. These algorithms can find a bottleneck in the quadratic size of the SAT encodings with respect to size of the constraint. For them, the SMT technology is a reasonable alternative. This technology allows to treat PB constraints with specialized inference mechanisms and a moderate cost in size, while preserving the strength of SAT techniques for the rest of the problem. We used this technology in [38, 39], showing that it is an alternative for implementing efficiently MaxSAT algorithms. However, SAT technology is the default option and SAT encodings for PB constraints are still being improved. In this sense, we are currently developing a a SAT encoding for PB constraints that allows to adjust the balance between size and arc-consistency. We provide in [42] a technical report (attached paper H) explaining this encoding that is potentially useful to solve many MaxSAT instances efficiently.

2.5 Exploiting the Global Structure of the Unsatisfiable Cores

Our final contribution to the development of SAT-based MaxSAT algorithms is the design of the WPM3 algorithm, that allows to exploit the global structure of the unsatisfiable cores of a MaxSAT instance. This work has been partially published in [44] and extended in [46], both included at the end of this dissertation as attached papers D and G, respectively. In [46], we begin with a study about the complete core-guided algorithms Eva and OLL, that inspired the best performing solvers on industrial instances at MSE 2014. We bring some light about how they proceed, showing that both are working with Cardinality constraints and in the end they are very similar. Then, we present the complete core-guided algorithm WPM3 that also uses Cardinality constraints and, in addition, is able to perform the optimization of subproblems efficiently and work as an incomplete approach given limited time. Our algorithm is designed to be aware of the global structure of the unsatisfiable cores. We have exploited this structure both to build Cardinality constraints that boost the performance of the solvers and to select the subproblems to be optimized.

As we have mentioned in Subsection 2.4, the quadratic size of SAT encodings for general PB constraints can be a bottleneck for SAT-based algorithms. There have been so far several core-guided approaches using only Cardinality constraints. The WPM1 algorithm only needed 1-Cardinality constraints, but clauses required multiple auxiliary variables, introducing additional complexity. To our best knowledge, the OLL algorithm, presented in [48] originally for ASP (Answer Set Programming), was the first one that only required at most one auxiliary variable per soft clause while using Cardinality constraints. Later, this algorithm was applied to MaxSAT and shown to be competitive in [49]. Previous to this work, the Eva algorithm was presented and also shown to be competitive in [47]. It applies the MaxSAT resolution rule to create the SAT instances in the sequence into which it reformulates the input MaxSAT instance. Although both algorithms are core-guided, they had apparently different foundations. In [46], we demonstrate that the transformation made by Eva at each iteration using the MaxSAT resolution rule corresponds to the introduction of a Cardinality constraint. In particular, it introduces the regular encoding for 1-Cardinality constraints described in [59]. In this work, we have also studied the connections between Eva and OLL and found that both are in fact building Cardinality constraints incrementally in a similar way.

The WPM3 algorithm only needs Cardinality constraints like Eva and OLL and has the advantage that it is designed to be aware of the global structure of the unsatisfiable cores of the MaxSAT instance. We use this structure to build efficient Cardinality constraints and to select the subproblems to be optimized. With respect to the Cardinality constraints, we show in [46] that constructing them according to the core structure boost the performance of the solvers, which is mostly due to this structure and not only to constructing them incrementally. With respect to the optimization of subproblems, we show in [46] that being aware of the global structure of the cores allows us to select subproblems that are worth optimizing, since this will lead to a higher lower bound. Finally, we also show how to exploit the assignments obtained from subproblems to extend to MaxSAT a very effective technique used in SAT solvers called phase saving [45].

2.6 A Portfolio Approach

All the work explained in the previous subsections has boosted the state-of-the-art performance on many industrial instances. Nevertheless, none of the aforementioned approaches dominated across all the instance set of the MSE. This is why, we have used meta-algorithmic techniques to combine all of them into a portfolio. This work has been published in [51] that is included at the end of this dissertation as attached paper F. We have basically employed the instance-specific algorithm configurator ISAC, and improved it with the latest in portfolio technology, resulting in ISAC+. We have applied the new methodology to solve the instance set of the MSE that is divided in categories (MS, PMS and WPMS), subdivided in subcategories (random, crafted and industrial) and finally grouped by families depending on the problem domain. Our portfolio consistently outperformed the best existing solvers on the respective categories and was competitive across all subcategories, being also the best performing approach for many families.

In the practice of combinatorial search algorithms, there is oftentimes no single solver that performs best on every single instance family. Rather, different algorithms and even different parametrizations of the same solver excel on different instance families. This is the underlying reason why portfolios have been so successful in SAT [60, 61], CP [62], and QBF [63]. Namely, all these approaches select and schedule solvers instance-specifically. In the literature, we find two meta-algorithmic approaches for making solvers instance-specific using a set of training instances. The first ones are algorithm portfolio builders for given sets of solvers and the second ones instance-specific configurators for parametrized solvers. Our work efficiently combines both approaches.

In our approach, we first compute features for each training instance and normalize them. We cluster the training instances as represented by their normalized features. For each cluster, we use an automatic configurator to find a good parametrization. At runtime, we compute the features of the input instance to be solved and normalize them. Up to this point, the original ISAC and the new ISAC+ work exactly the same. Now, ISAC computes the cluster nearest to the input instance, as measured by the distance between the normalized feature vectors. But, once the parametrizations for each cluster have been computed, there is no reason why we would need to stick to these clusters for selecting the best parametrization for the input instance. In this sense, ISAC+ uses a selector that does not take into account these clusters to determine which solver/parametrization should be used for the input instance.

From the research perspective, MaxSAT is of particular interest as it requires the ability to reason about both optimality and feasibility. Depending on the particular problem being solved, it is more important to emphasize one or the other of these inherent aspects. In [51], we give a detailed insight on the inner workings of our portfolio approach. In particular, we have analyzed ISAC+2014 that was submitted to the MSE 2014, showing which solver/parametrization was selected for how many instances within each industrial family. For this portfolio, we had parametrized the SMT-based solver WPM2-2013 which had solved at MSE 2013 the most industrial instances and had the highest mean family ratio of solved instances on industrial families. Overall, parametrizations of WPM2-2013 were selected for almost one out of four solved instances.

3 Impact: Results at the MaxSAT Evaluation

The work done during this PhD thesis resulted in several solvers that have been regularly submitted to the yearly edition of the international MaxSAT Evaluation (MSE). The MSE is organized as an affiliated event of the International Conference on Theory and Applications of Satisfiability Testing (SAT). The goal of the MSE is assessing the state-of-the-art in the field of MaxSAT solvers, as well as creating a collection of publicly available MaxSAT benchmark instances [64]. The first edition of MSE was in 2006, initially only for complete solvers. Since 2011, the MSE allows also the submission of incomplete solvers in a special track. We have submitted our solvers to the MSE since 2012. That year, a total of 18 solvers were submitted. Since then, the concurrence has increased and 38 solvers were submitted at MSE 2015. These solvers have been implemented by more than ten teams composed by members of international leading universities. The research resulting from the process of making the solvers has been regularly published in international leading conferences and journals.

The instance families of the MSE are grouped by the variant of the Weighted Partial MaxSAT problem (MS, PMS, WMS and WPMS). Categories were subdivided by the nature of the problem (random, crafted and industrial). There

	MS		PMS		WPMS	
2012	1.	wbo1.6-cnf	1.	QMaxSat-g2	1.	pwbo2.1
	2.	**WPM1**	2.	pwbo2.0	2.	**WPM1**
	3.	PM2	3.	QMaxSat	3.	wbo1.6
2013	1.	pmifumax	1.	**ISAC+-pms**	1.	**WPM1-2013**
	2.	**WPM1-2011**	2.	QMaxSAT2-mt	2.	**ISAC+-wpms**
	3.	**ISAC+-ms**	3.	MSUnCore	3.	**WPM2-2013**
2014	1.	Open-WBO-In	1.	**ISAC+2014-pms**	1.	Eva500a
	2.	clasp	2.	Open-WBO-In	2.	**ISAC+2014-wpms**
	3.	Eva500a	3.	Eva500a	3.	MSCG
2015	1.	**ISAC+-2015-ms**	1.	**ISAC+-2015-pms**	1.	LMHS-I
	2.	mscg2015a	2.	**WPM3-2015-co**	2.	MaxHS
	3.	mscg2015b	3.	Open-WBO-R	3.	mscg2015b

Table 2. MSE 2012-2015 three best complete solvers on industrial subcategories.

	MS		PMS		WPMS	
2014	1.	optimax2-r-i	1.	**WPM-2014-in**	1.	**WPM-2014-in**
	2.	**WPM-2014-in**	2.	optimax2-rn-i	2.	optimax2-g-i
	3.	optimax2-rn-i	3.	optimax2-r-i	3.	optimax2w-r-i
2015	1.	optiriss-def-i	1.	**WPM3-2015-in**	1.	**WPM3-2015-in**
	2.	**WPM3-2015-in**	2.	optiriss-def-i	2.	optiriss-def-i
	3.	optiriss-sel-i	3.	optiriss-sel-i	3.	**ILP-2015-in**

Table 3. MSE 2014-2015 three best incomplete solvers on industrial subcategories.

are no WMS industrial instances, resulting only three industrial subcategories. In Tables 2 and 3 we can find the results of our solvers on industrial instances, which are our main goal. At the complete track, we have submitted since 2012 several complete solvers (in bold) that placed in the three first positions 13 times in the 12 subcategories where they competed. In detail, they placed 5 times in the first position, 6 times in the second position and 2 times in the third position. At the incomplete track, we have submitted since 2014 several incomplete solvers (in bold) that placed in the three first positions 7 times in the 6 subcategories where they competed. In detail, they placed 4 times in the first position, 2 times in the second position and 1 time in the third position. Any other team has reached these results in this period.

The work done during this PhD thesis has contributed to boost the state-of-the-art in the field of MaxSAT solvers. Not only our solvers have been among the best performing ones at the latest editions of the MSE, but also some of the techniques we developed are incorporated to many other state-of-the-art solvers. The work done has contributed to close up some open instances and to reduce dramatically the solving time in many others. This has a special impact in the case of industrial families, that come from real world applications. Among the industrial families where the performance has improved these last years, we can find for example the upgradeability problem [1] or the haplotyping-pedigree problem [5].

With respect to the upgradeability problem, it consists in finding a new configuration of packages that minimizes the impact of installing software in a system. The variables of the objective function represent the state (installed/non-installed) of the packages, which are subject to package incompatibilities and dependencies. The goal is to maximize the installation of required packages while minimizing the packages that are uninstalled and the ones that are installed according to dependencies. The instances were submitted to the MSE for the first time in 2010. The best solver at MSE 2010 solved the whole set of 100 instances in a mean time of 158,56 seconds. At MSE 2015 the mean time of the best solver has been 0.57 seconds, which supposes an improvement of two orders of magnitude in five years.

With respect to the haplotyping-pedigree problem, it arises from the fact that current technology is not able to obtain the haplotype of a chromosome, but of the conflated data of the two chromosomes of a pair. The problem consists in estimating the real haplotypes of a pedigree, i.e. a family tree. The solution should first minimize the number of recombination events (very rare in haplotypes) and then minimize the number of distinct haplotypes. The instances were first submitted to MSE in 2011. The best solver at MSE 2011 was only able to solve 81 out of 100 instances within the timeout of 1800 seconds. At MSE 2015 the best solver has solved the 100 instances in a mean time of 8.28 seconds.

4 Conclusions and Future Work

The work done during this PhD thesis has contributed to boost the state-of-the-art in MaxSAT solving. We have developed new techniques and algorithms that have allowed us to close up some open instances and to reduce dramatically the solving time in many others. In addition, we have also developed a new meta-algorithmic approach that is competitive on almost all the problem instance families at the international MaxSAT Evaluation (MSE) and dominates all categories since 2013.

Initially, we focused our research on improving SAT-based MaxSAT algorithms that, according to the latest editions of MSE, are the best option to solve industrial MaxSAT instances from real world problems. In our work [38] (see Subsection 2.2), we have developed techniques that could be incorporated to the general schema of some core-guided algorithms. These techniques consisted basically in improving the quality of the unsatisfiable cores retrieved from the unsatisfiable SAT instances, by using stratification and hardening. By doing so, we reduced the size of the cores and make their weights as similar as possible. Both techniques are still in use in many state-of-the-art core-guided solvers. In addition, we also developed a specific technique for the Fu and Malik algorithm and its weighted version WPM1 to break the symmetries introduced during its execution.

After our work improving some initial core guided-algorithms, we incorporated to the most recent core-guided algorithm WPM2 some of the strengths of model-guided algorithms. In our work [39] (see Subsection 2.3), we chose to improve the WPM2 algorithm because, in contrast to WPM1, it has no symmetries on auxiliary variables since it only needs one per soft clause. We introduced to it some features of model-guided algorithms by optimizing the subproblems related to the unsatisfiable cores. By doing so, not only the lower bound converged more quickly to the optimal cost, but also we got assignments that can be used to refine the upper bound. In this way, we transformed a complete algorithm so that it can also work as an efficient incomplete approach given limited time. This is of special interest in many industrial domains focused on obtaining better upper bounds and assignments in a reasonable time. We have further exploited the subproblem optimization, developing extra techniques to guide the search using the assignments. We also applied the stratification technique developed in our previous work, and established a new criteria, applicable to WPM2 and other similar algorithms, to decide when soft clauses can be hardened. Also in [39], we have shown that the SMT technology is an underlying efficient technology for solving the MaxSAT problem. Moreover, we have compared our solvers with a full translation into ILP and surprisingly found that, although overall not competitive on industrial instances, it was extremely well suited for some families.

Our final contribution to the development of SAT-based algorithms has been the design of a new algorithm. In our third work [46] (see Subsection 2.5), we describe the complete core-guided algorithm WPM3, that can also work as incomplete given limited time. The design of the algorithm has allowed us to combine several techniques to use only Cardinality constraints and perform the optimization of subproblems efficiently. We have shown how these Cardinality constraints can be efficiently constructed by exploiting the global structure of unsatisfiable cores of the MaxSAT instances. We have also used this structure to select the subproblems to be optimized. Also for this algorithm, the optimization of subproblems makes the lower bound converge more quickly to the optimum and provides assignments that can be used to refine the upper bounds. We have further exploited the assignments to extend effectively the notion of phase saving to MaxSAT.

Besides developing new techniques and algorithms that have boosted the state-of-the-art performance on many instances, we have also developed a new meta-algorithmic approach, competitive across almost all the families of the MSE. In our work [51], (see Subsection 2.6) we have introduced an improved instance-specific algorithm configurator ISAC+ by adding a portfolio stage to the existing ISAC approach. Extensive tests show that the new method consistently outperforms the best instance-specific configurators to date. We have applied this method to MaxSAT, a domain where portfolios had never been used in a competitive setting before we conducted this work. Based on all this work, we have submitted our portfolio ISAC+ to the MSE, where it has dominated in all categories since 2013. These results independently confirm that our work mark a significant step forward in solving MaxSAT instances efficiently.

Finally, we have seen that one of the key points to implement SAT-based MaxSAT algorithms is how to manage the PB constraints. Although the SMT technology is a good alternative, it does not preserve arc-consistency. The default option is still to use a SAT solver and translate the PB constraints into SAT. The problem is that the best SAT encoding that preserve arc-consistency has a quadratic complexity in terms of size [40], which can be a bottleneck for some algorithms. One of the options we have studied is the design of SAT-based algorithms that only use Cardinality constraints, a simpler type of PB constraint with a quasilinear complexity in terms of size [43]. We have seen that this kind of approaches are the most suitable for a wide range of problems. However, for some special cases, the number of Cardinality constraints that they introduce might result in a larger encoding than introducing directly a PB constraint. We think that, in order to have more efficient SAT-based MaxSAT solvers, the most promising avenue is to improve SAT encodings for PB constraints, which is at the same time a great challenge. In this sense, we are currently developing a SAT encoding for PB constraints that allows to adjust the balance between size and arc-consistency. We provide in [42] a technical report explaining this encoding that is potentially useful to solve many MaxSAT instances efficiently. As future work, it would be interesting to further improve SAT encodings for PB constraints and study how can they be constructed taking into account the global structure of unsatisfiable cores of the MaxSAT instances.

References

1. J. Argelich, D. L. Berre, I. Lynce, J. P. M. Silva, P. Rapicault, Solving linux upgradeability problems using boolean optimization, in: Proceedings First International Workshop on Logics for Component Configuration (LoCoCo), 2010, pp. 11–22.
2. S. Safarpour, H. Mangassarian, A. G. Veneris, M. H. Liffiton, K. A. Sakallah, Improved design debugging using maximum satisfiability, in: FMCAD, IEEE Computer Society, 2007, pp. 13–19.
3. Y. Chen, S. Safarpour, J. Marques-Silva, A. G. Veneris, Automated design debugging with maximum satisfiability, IEEE Trans. on CAD of Integrated Circuits and Systems 29 (11) (2010) 1804–1817.
4. D. M. Strickland, E. R. Barnes, J. S. Sokol, Optimal protein structure alignment using maximum cliques, Operations Research 53 (3) (2005) 389–402.
5. A. Graça, I. Lynce, J. Marques-Silva, A. L. Oliveira, Efficient and accurate haplotype inference by combining parsimony and pedigree information, in: Algebraic and Numeric Biology - 4th International Conference, ANB 2010, Hagenberg, Austria, July 31- August 2, 2010, Revised Selected Papers, 2010, pp. 38–56.
6. M. Jose, R. Majumdar, Cause clue clauses: error localization using maximum satisfiability, in: Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2011, San Jose, CA, USA, June 4-8, 2011, 2011, pp. 437–446.

7. R. J. A. Achá, R. Nieuwenhuis, Curriculum-based course timetabling with SAT and maxsat, Annals OR 218 (1) (2014) 71–91.
8. M. C. Cooper, S. Cussat-Blanc, M. de Roquemaurel, P. Régnier, Soft arc consistency applied to optimal planning, in: Proc. of CP'06, 2006, pp. 680–684.
9. L. Zhang, F. Bacchus, MAXSAT heuristics for cost optimal planning, in: Proc. of AAAI'12, 2012, pp. 1846–1852.
10. M. Vasquez, J. Hao, A "logic-constrained" knapsack formulation and a tabu algorithm for the daily photograph scheduling of an earth observation satellite, Comp. Opt. and Appl. 20 (2) (2001) 137–157.
11. H. Xu, R. A. Rutenbar, K. A. Sakallah, sub-sat: a formulation for relaxed boolean satisfiability with applications in routing, IEEE Trans. on CAD of Integrated Circuits and Systems 22 (6) (2003) 814–820.
12. T. Sandholm, An algorithm for optimal winner determination in combinatorial auctions, in: Proc. of IJCAI'99, 1999, pp. 542–547.
13. F. Heras, J. Larrosa, S. de Givry, T. Schiex, 2006 and 2007 max-sat evaluations: Contributed instances, JSAT 4 (2-4) (2008) 239–250.
14. C. Ansótegui, M. L. Bonet, J. Levy, Sat-based maxsat algorithms, Artif. Intell. 196 (2013) 77–105.
15. A. Morgado, F. Heras, M. H. Liffiton, J. Planes, J. Marques-Silva, Iterative and core-guided maxsat solving: A survey and assessment, Constraints 18 (4) (2013) 478–534.
16. J. Argelich, C. M. Li, F. Manyà, J. Planes, Maxsat evaluation, http://www.maxsat.udl.cat (2006-2015).
17. Maxsat evaluations (2015).
URL www.maxsat.udl.cat
18. F. Heras, J. Larrosa, A. Oliveras, MiniMaxSat: A new weighted Max-SAT solver, in: Proc. of SAT'07, 2007, pp. 41–55.
19. H. Lin, K. Su, Exploiting inference rules to compute lower bounds for Max-SAT solving, in: Proc. of IJCAI'07, 2007, pp. 2334–2339.
20. H. Lin, K. Su, C. M. Li, Within-problem learning for efficient lower bound computation in Max-SAT solving, in: Proc. of AAAI'08, 2008, pp. 351–356.
21. C. M. Li, F. Manyà, N. O. Mohamedou, J. Planes, Exploiting cycle structures in Max-SAT, in: Proc. of SAT'09, 2009, pp. 467–480.
22. A. Kügel, Improved exact solver for the weighted MAX-SAT problem, in: POS-10. Pragmatics of SAT, Edinburgh, UK, July 10, 2010, 2010, pp. 15–27.
23. Z. Fu, S. Malik, On solving the partial max-sat problem, in: Proc. of SAT'06, 2006, pp. 252–265.
24. Z. Fu, Extending the power of boolean satisfiability: Techniques and applications, Ph.D. thesis, Princeton (2007).
25. C. Ansótegui, M. L. Bonet, J. Levy, Solving (weighted) partial maxsat through satisfiability testing, in: Proc. of SAT'09, 2009, pp. 427–440.
26. V. Manquinho, J. Marques-Silva, J. Planes, Algorithms for weighted boolean optimization, in: Proc. of SAT'09, 2009, pp. 495–508.
27. C. Ansotegui, M. L. Bonet, J. Levy, A new algorithm for weighted partial maxsat, in: Proc. of AAAI'10, 2010, pp. 867–872.
28. R. Martins, V. M. Manquinho, I. Lynce, Exploiting cardinality encodings in parallel maximum satisfiability, in: ICTAI, 2011, pp. 313–320.
29. R. Martins, V. M. Manquinho, I. Lynce, Clause sharing in parallel maxsat, in: LION 12, 2012, pp. 455–460.
30. R. Martins, S. Joshi, V. M. Manquinho, I. Lynce, Incremental cardinality constraints for maxsat, in: Proc. of CP'14, 2014, pp. 531–548.

31. N. Eén, N. Sörensson, Translating pseudo-boolean constraints into SAT, JSAT 2 (1-4) (2006) 1–26.
32. D. L. Berre, Sat4j, a satisfiability library for java, www.sat4j.org (2006).
33. M. Koshimura, T. Zhang, H. Fujita, R. Hasegawa, Qmaxsat: A partial max-sat solver, JSAT 8 (1/2) (2012) 95–100.
34. K. Honjyo, T. Tanjo, Shinmaxsat, a Weighted Partial Max-SAT solver inspired by MiniSat+, Information Science and Technology Center, Kobe University (2012).
35. F. Heras, A. Morgado, J. Marques-Silva, Core-guided binary search algorithms for maximum satisfiability, in: Proc. of AAAI'11, 2011, pp. 36–41.
36. A. Morgado, F. Heras, J. Marques-Silva, Improvements to core-guided binary search for maxsat, in: Proc. of SAT'12, 2012, pp. 284–297.
37. C. Ansótegui, M. L. Bonet, J. Gabàs, J. Levy, Improving sat-based weighted maxsat solvers, in: Proc. of CP'12, 2012, pp. 86–101.
38. C. Ansótegui, M. L. Bonet, J. Gabàs, J. Levy, Improving wpm2 for (weighted) partial maxsat, in: Proc. of CP'13, 2013, pp. 117–132.
39. C. Ansótegui, J. Gabàs, J. Levy, Exploiting subproblem optimization in sat-based maxsat algorithms, J. Heuristics 22 (1) (2016) 1–53.
40. N. Manthey, T. Philipp, P. Steinke, A more compact translation of pseudo-boolean constraints into CNF such that generalized arc consistency is maintained, in: KI 2014: Advances in Artificial Intelligence - 37th Annual German Conference on AI, Stuttgart, Germany, September 22-26, 2014. Proceedings, 2014, pp. 123–134.
41. R. Sebastiani, Lazy Satisfiability Modulo Theories, Journal on Satisfiability, Boolean Modeling and Computation 3 (3-4) (2007) 141–224.
42. C. Ansótegui, J. Gabàs, A new sat encoding for pb constraints allowing to adjust the balance between size and arc-consistency, internal Technical Report 05-06-2016.
43. R. Asín, R. Nieuwenhuis, A. Oliveras, E. Rodríguez-Carbonell, Cardinality networks: a theoretical and empirical study, Constraints 16 (2) (2011) 195–221.
44. C. Ansótegui, F. Didier, J. Gabàs, Exploiting the structure of unsatisfiable cores in maxsat, IJCAI (2015) 283–289.
45. K. Pipatsrisawat, A. Darwiche, A lightweight component caching scheme for satisfiability solvers, in: Proc. of SAT'07, 2007, pp. 294–299.
46. C. Ansótegui, J. Gabàs, Wpm3: an (in)complete algorithm for weighted partial maxsat, submitted to Artif. Intell.
47. N. Narodytska, F. Bacchus, Maximum satisfiability using core-guided maxsat resolution, in: Proc. of AAAI'14, 2014, pp. 2717–2723.
48. B. Andres, B. Kaufmann, O. Matheis, T. Schaub, Unsatisfiability-based optimization in clasp, in: ICLP (Technical Communications), 2012, pp. 211–221.
49. A. Morgado, C. Dodaro, J. Marques-Silva, Core-guided maxsat with soft cardinality constraints, in: Proc. of CP'14, 2014, pp. 564–573.
50. C. Ansótegui, J. Gabàs, Solving (weighted) partial maxsat with ilp, in: CPAIOR 13, 2013, pp. 403–409.
51. C. Ansótegui, J. Gabàs, Y. Malitsky, M. Sellmann, Maxsat by improved instance-specific algorithm configuration, Artif. Intell. 235 (2016) 26–39.
52. J. Marques-Silva, J. Planes, On using unsatisfiability for solving maximum satisfiability, CoRR abs/0712.1097.
53. J. Davies, F. Bacchus, Solving maxsat by solving a sequence of simpler sat instances, in: Proc. of CP'11, 2011, pp. 225–239.
54. J. Davies, F. Bacchus, Exploiting the power of mip solvers in maxsat, in: Proc. of SAT'13, 2013, pp. 166–181.
55. C. Ansotegui, Tutorial: Maxsat latest developments (2013).

56. T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms (3. ed.), MIT Press, 2009.
57. A. Cimatti, A. Franzén, A. Griggio, R. Sebastiani, C. Stenico, Satisfiability modulo the theory of costs: Foundations and applications, in: TACAS, 2010, pp. 99–113.
58. C. Ansotegui, Maxsat latest developments, invited tutorial at CP 2013 (2013).
59. C. Ansótegui, F. Manyà, Mapping problems with finite-domain variables to problems with boolean variables, in: Proc. of SAT'04, 2004, pp. 1–15.
60. L. Xu, F. Hutter, H. Hoos, K. Leyton-Brown, Satzilla: portfolio-based algorithm selection for sat, JAIR 32 (1) (2008) 565–606.
61. S. Kadioglu, Y. Malitsky, A. Sabharwal, H. Samulowitz, M. Sellmann, Algorithm selection and scheduling, CP (2011) 454–469.
62. E. O'Mahony, E. Hebrard, A. Holland, C. Nugent, B. O'Sullivan, Using case-based reasoning in an algorithm portfolio for constraint solving, AICS.
63. L. Pulina, A. Tacchella, A multi-engine solver for quantified boolean formulas, CP (2007) 574–589.
64. J. Argelich, C. M. Li, F. Manyà, J. Planes, Analyzing the instances of the maxsat evaluation, in: SAT, 2011, pp. 360–361.

Attached papers:

A C. Ansótegui, M. L. Bonet, J. Gabàs, J. Levy, **Improving SAT-based Weighted MaxSAT Solvers**, in: Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming (CP'12), 2012, pp. 86 - 101.

B C. Ansótegui, J. Gabàs, **Solving (Weighted) Partial MaxSAT with ILP**, in: Proceedings of the 10th International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) techniques in Constraint Programming (CPAIOR'13), 2013, pp. 403 - 409.

C C. Ansótegui, M. L. Bonet, J. Gabàs, J. Levy, **Improving WPM2 for (Weighted) Partial MaxSAT**, in: Proceedings of the 19th International Conference on Principles and Practice of Constraint Programming (CP'13), 2013, pp. 117 - 132.

D C. Ansótegui, F. Didier, J. Gabàs, **Exploiting the Structure of Unsatisfiable Cores in MaxSAT**, in: Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI'15), 2015, pp. 283 - 289.

E C. Ansótegui, J. Gabàs, J. Levy, **Exploiting subproblem optimization in SAT-based MaxSAT algorithms**, Journal of Heuristics 22(1), 2016, pp. 1 - 53.

F C. Ansótegui, J. Gabàs, Y. Malitsky, M. Sellmann, **MaxSAT by improved instance-specific algorithm configuration**, Artificial Intelligence 235, 2016, pp. 26 - 39.

G C. Ansótegui, J. Gabàs, **WPM3: an (in)complete algorithm for Weighted Partial MaxSAT**, submitted to Artificial Intelligence

H C. Ansótegui, J. Gabàs, **A new SAT encoding for PB constraints allowing to adjust the balance between size and arc-consistency**, Internal Technical Report 05-06-2016.

A

Improving SAT-based Weighted MaxSAT Solvers

C. Ansótegui, M. L. Bonet, J. Gabàs, J. Levy

in: Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming

CP Québec City

2012

Improving SAT-Based Weighted MaxSAT Solvers*

Carlos Ansótegui[1], Maria Luisa Bonet[2], Joel Gabàs[1], and Jordi Levy[3]

[1] DIEI, Univ. de Lleida
{carlos,joel.gabas}@diei.udl.cat
[2] LSI, UPC
bonet@lsi.upc.edu
[3] IIIA-CSIC
levy@iiia.csic.es

Abstract. In the last few years, there has been a significant effort in designing and developing efficient Weighted MaxSAT solvers. We study in detail the WPM1 algorithm identifying some weaknesses and proposing solutions to mitigate them. Basically, WPM1 is based on iteratively calling a SAT solver and adding blocking variables and cardinality constraints to relax the unsatisfiable cores returned by the SAT solver. We firstly identify and study how to break the symmetries introduced by the blocking variables and cardinality constraints. Secondly, we study how to prioritize the discovery of higher quality cores. We present an extensive experimental investigation comparing the new algorithm with state-of-the-art solvers showing that our approach makes WPM1 much more competitive.

1 Introduction

Many combinatorial optimization problems can be modelled as Weighted Partial MaxSAT formulas. Therefore, Weighted Partial MaxSAT solvers can be used in several domains as: combinatorial auctions, scheduling and timetabling problems, FPGA routing, software package installation, etc.

The Maximum Satisfiability (MaxSAT) problem is the optimization version of the satisfiability (SAT) problem. The goal is to maximize the number of satisfied clauses in a SAT formula, in other words, to minimize the number of falsified clauses. The clauses can be divided into hard and soft clauses, depending on whether they must be satisfied (hard) or they may or may not be satisfied (soft). If our formula only contains soft clauses it is a MaxSAT formula, and if it contains both, hard and soft clauses, it is a Partial MaxSAT formula. The Partial MaxSAT problem can be further generalized to the Weighted Partial MaxSAT problem. The idea is that not all soft clauses are equally important.

* This research has been partially founded by the CICYT research projects TASSAT (TIN2010-20967-C04-01/03/04) and ARINF (TIN2009-14704-C03-01).

The addition of weights to soft clauses makes the formula Weighted, and lets us introduce preferences between them. The weights indicate the penalty for falsifying a clause. Given a Weighted Partial MaxSAT problem, our goal is to find an assignment that satisfies all the hard clauses, and the sum of the weights of the falsified clauses is minimal. Such an assignment will be optimal in this context.

SAT technology has evolved to a mature state in the last decade. SAT solvers are really successful at solving industrial decision problems. The next challenge is to use this technology to solve more efficiently industrial optimization problems. Although there has been important work in this direction, we have not reached the success of SAT solvers yet. The present work is one more step in MaxSAT technology to achieve full industrial applicability.

Originally, MaxSAT solvers such as WMaxSatz [12], MiniMaxSat [10], IncWMaxSatz [13] and akmaxsat where depth-first branch and bound based. Recently, there has been a development of SAT based approaches which essentially iteratively call a SAT solver: SAT4J [5], WBO and MSUNCORE [14], WPM1 [1], WPM2 [2], *BINC* and *BINCD* [11] and maxHS [8]. While branch and bound based solvers are competitive for random and crafted instances, SAT based solvers are better for industrial instances.

The WPM1, WBO and MSUNCORE solvers implement weighted versions of the Fu and Malik's algorithm [9]. Essentially, they perform a sequence of calls to a SAT solver, and if the SAT solver returns an unsatisfiable core, they reformulate the problem by introducing new auxiliary variables and cardinality constraints which relax the clauses in the core. Further details are given in section 3 and 5. In this work, we analyze in more detail the WPM1 algorithm to identify and mitigate some weaknesses. The first weakness we have observed is that the addition of the auxiliary variables naturally introduce symmetries which should be broken to achieve better performance. The second weakness has to do with the quality of the cores returned by the SAT solver. Since the SAT solver is used as a black box, we need to come up with new strategies to lead the solver to find better quality cores.

We have conducted an extensive experimental investigation with the best solvers at the last MaxSAT evaluation and other solvers that did not take part in the evaluation, but have been reported to show very good performance. We can see that our current approach can boost radically the performance of the WPM1 becoming the most robust approach.

This paper proceeds as follows: Section 2 introduces some preliminary concepts; Section 3 presents the Fu and Malik's algorithm; Section 4 describes the problem of symmetries and shows how to break them; Section 5 presents the WPM1 algorithm and describes the problem of the quality of the cores; Section 6 introduces an stratified approach to come up with higher quality cores; Section 7 presents some previous concepts needed to describe a general stratified approach discussed in Section 8 and finally Section 9 presents the experimental evaluation.

2 Preliminaries

We consider an infinite countable set of *boolean variables* $\mathcal{X}$. A *literal* l is either a variable $x_i \in \mathcal{X}$ or its negation x_i. A *clause* C is a finite set of literals, denoted as $C = l_1 \vee \cdots \vee l_r$, or as   for the empty clause. A *SAT formula* φ is a finite set of clauses, denoted as $\varphi = C_1 \wedge \cdots \wedge C_m$.

A *weighted clause* is a pair (C, w), where C is a clause and w is a natural number or infinity, indicating the penalty for falsifying C. A clause is called *hard* if the corresponding weight is infinity, otherwise the clause is called *soft*.

A *(Weighted Partial) MaxSAT formula* is a multiset of weighted clauses

$$\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$$

where the first m clauses are soft and the last m' clauses are hard. The set of variables occurring in a formula φ is noted as $\mathrm{var}(\varphi)$.

A *total truth assignment* for a formula φ is a function $I : \mathrm{var}(\varphi) \to \{0, 1\}$, that can be extended to literals, clauses, SAT formulas and MaxSAT formulas, the following way:

$$I(x_i) = 1 - I(x_i)$$
$$I(l_1 \vee \ldots \vee l_r) = \max\{I(l_1), \ldots, I(l_r)\}$$
$$I(\{C_1, \ldots, C_m\}) = \min\{I(C_1), \ldots, I(C_m)\}$$
$$I(\{(C_1, w_1), \ldots, (C_m, w_m)\}) = w_1 \cdot (1 - I(C_1)) + \ldots + w_m \cdot (1 - I(C_m))$$

We define the *optimal cost* of a MaxSAT formula as

$$\mathrm{cost}(\varphi) = \min\{I(\varphi) \mid I : \mathrm{var}(\varphi) \to \{0, 1\}\}$$

and an *optimal assignment* as an assignment I such that $I(\varphi) = \mathrm{cost}(\varphi)$.

We also define *partial truth assignments* for φ as a partial function $I : \mathrm{var}(\varphi) \to \{0, 1\}$ where instantiated falsified literals are removed and the formula is simplified accordingly.

Example 1. Given $\varphi = \{(y, 6), (x \vee y, 2), (x \vee z, 3), (y \vee z, 2)\}$ and $I : \{y, z\} \to \{0, 1\}$ such that $I(y) = 0$ and $I(z) = 0$, we have $I(\varphi) = \{(x, 5), (\ \ , 2)\}$. We also have $\mathrm{cost}(I(\varphi)) = 2$ and $\mathrm{cost}(\varphi) = 0$.

Notice that, for any MaxSAT formula φ and partial truth assignment I, we have $\mathrm{cost}(\varphi) \leq \mathrm{cost}(I(\varphi))$. Notice also that when w is finite, the pair (C, w) is equivalent to having w copies of the clause $(C, 1)$ in our multiset.

We say that a truth assignment I *satisfies* a literal, clause or a SAT formula if it assigns 1 to it, and *falsifies* it if it assigns 0. A SAT formula is *satisfiable* if there exists a truth assignment that satisfies it. Otherwise, it is *unsatisfiable*. Given an unsatisfiable SAT formula φ, an *unsatisfiable core* φ_c is a subset of clauses $\varphi_c \subseteq \varphi$ that is also unsatisfiable. A *minimal unsatisfiable core* is an unsatisfiable core such that any proper subset of it is satisfiable.

The *Weighted Partial MaxSAT problem* for a weighted partial MaxSAT formula φ is the problem of finding an *optimal assignment*. If the optimal cost is infinity, then the subset of hard clauses of the formula is unsatisfiable, and we say that the formula is *unsatisfiable*. The *Weighted MaxSAT problem* is the Weighted Partial MaxSAT problem when there are no hard clauses. The *Partial MaxSAT problem* is the Weighted Partial MaxSAT problem when the weights of soft clauses are all equal. The *MaxSAT problem* is the Partial MaxSAT problem when there are no hard clauses. Notice that the *SAT problem* is equivalent to the Partial MaxSAT problem when there are no soft clauses.

3 The Fu and Malik's Algorithm

The first SAT-based algorithm for Partial MaxSAT algorithm was the Fu and Malik's algorithm described in [9]. It was implemented in the MaxSAT solver msu1.2 [17,18], and its correctness was proved in [1].

The algorithm consists in iteratively calling a SAT solver on a working formula φ. This corresponds to the line $(st, \varphi_c) := SAT(\{C \mid (C_i, w_i) \in \varphi\})$. The SAT solver will say whether the formula is satisfiable or not (variable st), and in case the formula is unsatisfiable, it will give an unsatisfiable core (φ_c). At this point the algorithm will produce new variables, blocking variables (BV in the code), one for each soft clause in the core. The new working formula φ will consist in adding the new variables to the soft clauses of the core, adding a cardinality constraint saying that exactly one of the new variables should be true ($CNF(\sum_{b\in BV} b = 1)$ in the code), and adding one to the counter of falsified clauses. This procedure is applied until the SAT solver returns SAT.

For completeness, we reproduce the code of the Fu and Malik's algorithm in Algorithm 1.

Next we present an example of execution that will be used in the next section.

Example 2. Consider the pigeon-hole formula PHP_1^5 with 5 pigeons and one hole where the clauses saying that no two pigeons can go to the same hole are hard, while the clauses saying that each pigeon goes to a hole are soft:

$$\varphi = \{(x_1, 1), (x_2, 1), (x_3, 1), (x_4, 1), (x_5, 1), (x_1 \vee x_2, \infty), \ldots, (x_4 \vee x_5, \infty)\}$$

In what follows, the new b variables will have a super-index indicating the number of the unsatisfiable core, and a subindex indicating the index of the original soft clause.

Suppose that applying the FuMalik algorithm, the SAT solver computes the (minimal) unsatisfiable core $C_1 = \{1, 2\}$. Here we represent the core by the set of indexes of the soft clauses contained in the core. The new formula will be as shown on the right. At this point, the variable *cost* takes value 1.

$$\begin{aligned} \varphi_1 = \{\ & (x_1 \vee b_1^1\ , 1), \\ & (x_2 \vee b_2^1\ , 1), \\ & (x_3\ , 1), \\ & (x_4\ , 1), \\ & (x_5\ , 1)\ \} \cup \\ & \{(x_i \vee x_j, \infty) \mid i \neq j\} \cup \\ & CNF(b_1^1 + b_2^1 = 1, \infty) \end{aligned}$$

Algorithm 1. The pseudo-code of the FuMalik algorithm (with a minor correction).

```
    Input: $\varphi = \{(C_1, 1), \ldots, (C_m, 1), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$
 1: if $SAT(\{C_i \mid w_i = \infty\}) = (\text{UNSAT}, \ )$ then return $(\infty, \emptyset)$
                                                   ▷Hard clauses are unsatisfiable
 2: $cost := 0$                                                         ▷Optimal
 3: while $true$ do
 4:     $(st, \varphi_c) := SAT(\{C_i \mid (C_i, w_i) \in \varphi\})$  ▷Call to the SAT solver without weights
 5:     if $st = \text{SAT}$ then return $(cost, \varphi)$
 6:     $BV := \emptyset$                                ▷Set of blocking variables
 7:     foreach $C_i \in \varphi_c$ do
 8:         if $w_i \neq \infty$ then                        ▷If the clause is soft
 9:             $b :=$ new variable( )
10:             $\varphi := \varphi \setminus \{(C_i, 1)\} \cup \{(C_i \vee b, 1)\}$    ▷Add blocking variable
11:             $BV := BV \cup \{b\}$
12:     $\varphi := \varphi \cup \{(C, \infty) \mid C \in CNF(\sum_{b \in BV} b = 1)\}$
                                      ▷Add cardinality constraint as hard clauses
13:     $cost := cost + 1$
```

If the next unsatisfiable cores found by the SAT solver are $C_2 = \{3, 4\}$ and $C_3 = \{1, 2, 3, 4\}$, then the new formula will be:

$$\begin{aligned}\varphi_2 = \{\ &(x_1 \vee b_1^1, 1),\\ &(x_2 \vee b_2^1, 1),\\ &(x_3 \vee b_3^2, 1),\\ &(x_4 \vee b_4^2, 1),\\ &(x_5, 1)\ \} \cup\\ &\{(x_i \vee x_j, \infty) \mid i \neq j\} \cup\\ &CNF(b_1^1 + b_2^1 = 1, \infty) \cup\\ &CNF(b_3^2 + b_4^2 = 1, \infty)\end{aligned}$$

$$\begin{aligned}\varphi_3 = \{\ &(x_1 \vee b_1^1 \vee b_1^3, 1),\\ &(x_2 \vee b_2^1 \vee b_2^3, 1),\\ &(x_3 \vee b_3^2 \vee b_3^3, 1),\\ &(x_4 \vee b_4^2 \vee b_4^3, 1),\\ &(x_5, 1)\ \} \cup\\ &\{(x_i \vee x_j, \infty) \mid i \neq j\} \cup\\ &CNF(b_1^1 + b_2^1 = 1, \infty) \cup\\ &CNF(b_3^2 + b_4^2 = 1, \infty) \cup\\ &CNF(b_1^3 + b_2^3 + b_3^3 + b_4^3 = 1, \infty)\end{aligned}$$

After the third iteration, the variable *cost* has value 3. Finally, after finding the core $C_4 = \{1, 2, 3, 4, 5\}$ we get the following satisfiable MaxSAT formula:

$$\begin{aligned}\varphi_4 = \{\ &(x_1 \vee b_1^1 \vee b_1^3 \vee b_1^4, 1),\\ &(x_2 \vee b_2^1 \vee b_2^3 \vee b_2^4, 1),\\ &(x_3 \vee b_3^2 \vee b_3^3 \vee b_3^4, 1),\\ &(x_4 \vee b_4^2 \vee b_4^3 \vee b_4^4, 1),\\ &(x_5 \vee b_5^4, 1)\ \} \cup\\ &\{(x_i \vee x_j, \infty) \mid i \neq j\} \cup\\ &CNF(b_1^1 + b_2^1 = 1, \infty) \cup\\ &CNF(b_3^2 + b_4^2 = 1, \infty) \cup\\ &CNF(b_1^3 + b_2^3 + b_3^3 + b_4^3 = 1, \infty) \cup\\ &CNF(b_1^4 + b_2^4 + b_3^4 + b_4^4 + b_5^4 = 1, \infty)\end{aligned}$$

At this point *cost* is 4. The algorithm will now call the SAT solver on φ_4, and the solver will return the answer "satisfiable". The algorithm returns $cost = 4$.

4 Breaking Symmetries

It is well known that formulas that contain a great deal of symmetries cause SAT solvers to explore many redundant truth assignments. Adding symmetry breaking clauses to a formula has the effect of removing the symmetries, while keeping satisfiability the same. Therefore it is a way to speed up solvers by pruning the search space.

In the case of executions of the FuMalik algorithm, symmetries can appear in two ways. On one hand, there are formulas that naturally contain many symmetries. For instance, in the case of the pigeon-hole principle we can permute the pigeons or the holes, leaving the formula intact. On the other hand, in each iteration of the FuMalik algorithm, we modify the formula adding new variables and hard constraints. In this process we can also introduce symmetries. In the present paper, we are no concerned with eliminating natural symmetries of a MaxSAT formula as in [16], since that might be costly, and it is not the aim of the present work. Instead we will eliminate the symmetries that appear in the process of performing the algorithm. In this case, it is very efficient to extract the symmetries given our implementation of the algorithm.

Before we formally describe the process of eliminating the symmetries, we will see an example.

Example 3. Consider again the pigeon-hole formula PHP_1^5 of Example 2. The working formula φ_3 from the previous section is still unsatisfiable, this is the reason to find a fourth core C_4. However, if we do not consider the clause x_5 the formula is satisfiable, and has 8 distinct models (two for each variable among $\{x_1, \ldots, x_4\}$ set to true). Here, we show 2 of the models, marking the literals set to true (we do not include the clauses $x_i \vee x_j$, for $i \neq j$ and put the true literals in boxes):

$$\begin{array}{lll}
x_1 \vee \quad b_1^1 \vee \qquad\quad b_1^3 & \qquad\qquad & x_1 \vee \quad b_1^1 \vee \qquad\qquad b_1^3 \\
x_2 \vee \quad b_2^1 \vee \qquad\quad b_2^3 & & x_2 \vee \quad b_2^1 \vee \qquad\quad b_2^3 \\
x_3 \vee \qquad\qquad b_3^2 \vee b_3^3 & & x_3 \vee \qquad\qquad b_3^2 \vee b_3^3 \\
x_4 \vee \qquad\qquad b_4^2 \vee \ b_4^3 & & x_4 \vee \qquad\qquad b_4^2 \vee \ b_4^3 \\
b_1^1 + b_2^1 = 1 & & b_1^1 + b_2^1 = 1 \\
b_3^2 + b_4^2 = 1 & & b_3^2 + b_4^2 = 1 \\
b_1^3 + b_2^3 + b_3^3 + b_4^3 = 1 & & b_1^3 + b_2^3 + b_3^3 + b_4^3 = 1
\end{array}$$

The previous two models are related by the permutation $b_1^1 \leftrightarrow b_2^1, b_1^3 \leftrightarrow b_2^3$. The two ways of assigning values to the b variables are equivalent. The existence of so many *partial* models makes the task of showing unsatisfiability of the formula (including x_5) much harder.

The mechanism to eliminate the symmetries caused by the extra variables is as follows: suppose we are in the s iteration of the FuMalik algorithm, and we have obtained the set of cores $\{\varphi_1, \dots, \varphi_s\}$. We assume that the clauses in the cores follow a total order. For clarity we will name the new variables of core φ_l for l such that $1 \leq l \leq s$ as b_i^l, where i is an index in φ_l. Now, we add the clauses:

$$b_i^s \to b_j^l \qquad \text{for } l = 1, \dots, s-1 \text{ and } i, j \in \varphi_l \cap \varphi_s \text{ and } j > i$$

This clauses implies that in Example 3 we choose the model on the left rather than the one on the right.

Example 4. For the Example 3, after finding the third unsatisfiable core C_3, we would add the following clauses to break symmetries (written in form of implications):

$$\begin{array}{c} b_1^3 \to b_2^1 \\ b_3^3 \to b_4^2 \end{array}$$

Adding these clauses, instead of the 8 partial models, we only have 4, one for each possible assignment of x_i to true.

After finding the fourth core C_4, we also add (written in compact form):

$$\begin{array}{l} b_1^4 \to (b_2^1 \wedge b_2^3 \wedge b_3^3 \wedge b_4^3) \\ b_2^4 \to (b_3^3 \wedge b_4^3) \\ b_3^4 \to (b_4^2 \wedge b_4^3) \end{array}$$

5 The WPM1 Algorithm

Algorithm 2 is the weighted version of the FuMalik algorithm described in section 3 [1,14] In this algorithm, we iteratively call a SAT solver with a weighted working formula, but excluding the weights. When the SAT solver returns an unsatisfiable core, we calculate the minimum weight of the clauses of the core (w_{min} in the algorithm.). Then, we transform the working formula in the following way: we duplicate the core having on one of the copies, the clauses with weight the original minus the minimum weight, and on the other copy we put the blocking variables and we give it the minimum weight. Finally we add the cardinality constraint on the blocking variables, and we add w_{min} to the *cost*.

The process of doubling the clauses might imply to end up converting clauses with weight say w into w copies of the clause of weight 1. When this happens, the process becomes very inefficient. In the following we show a (tiny) example that reflects this situation.

Example 5. Consider the formula $\varphi = \{(x_1, 1), (x_2, m), (x_2, \infty)\}$.

Assume that the SAT solver always includes the first soft clause in the returned unsatisfiable core, even if this makes the core not minimal. After one iteration, the new formula would be:

$$\varphi_1 = \{(x_1 \vee b_1^1, 1), (x_2 \vee b_2^1, 1), (x_2, m-1), (x_2, \infty), (b_1^1 + b_2^1 = 1, \infty)\}$$

Algorithm 2. The pseudo-code of the WPM1 algorithm.

```
    Input: $\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$
1:  if $SAT(\{C_i \mid w_i = \infty\}) = (\text{UNSAT}, \ )$ then return $(\infty, \emptyset)$    ▷Hard clauses are unsatisfiable
2:  $cost := 0$    ▷Optimal
3:  while $true$ do
4:      $(st, \varphi_c) := SAT(\{C_i \mid (C_i, w_i) \in \varphi\})$    ▷Call to the SAT solver without weights
5:      if $st =$ SAT then return $(cost, \varphi)$
6:      $BV := \emptyset$    ▷Blocking variables of the core
7:      $w_{min} := \min\{w_i \mid C_i \in \varphi_c \wedge w_i \neq \infty\}$    ▷Minimum weight
8:      foreach $C_i \in \varphi_c$ do
9:          if $w_i \neq \infty$ then
10:             $b :=$ new variable()
11:             $\varphi := \varphi \setminus \{(C_i, w_i)\} \cup \{(C_i, w_i - w_{min}),\ (C_i \vee b, w_{min})\}$
                    ▷Duplicate soft clauses of the core
12:             $BV := BV \cup \{b\}$
13:     $\varphi := \varphi \cup \{(C, \infty) \mid C \in CNF(\sum_{b \in BV} b = 1)\}$
                    ▷Add cardinality constraint as hard clauses
14:     $cost := cost + w_{min}$
```

If from now on, at each iteration i, the SAT solver includes the first clause along with $\{(x_2, m - i + 1), (x_2, \infty)\}$ in the unsatisfiable core, then at iteration i, the formula would be:

$$\begin{aligned}\varphi_i = \{\, & (x_1 \vee b_1^1 \vee \cdots \vee b_1^i, 1), (x_2 \vee b_2^1, 1), \ldots, (x_2 \vee b_2^i, 1), (x_2, m - i), (x_2, \infty), \\ & (b_1^1 + b_2^1 = 1, \infty), \ldots, (b_1^i + b_2^i = 1, \infty)\}\end{aligned}$$

The WPM1 algorithm would need m iterations to solve the problem.

Obviously, a reasonable good SAT solver would return a better quality core than in previous example. However, unless it can guarantee that it is minimal, a similar example (but more complicated) could be constructed.

6 A Stratified Approach for WPM1

In Algorithm 3 we present a modification of the WPM1 algorithm that tries to prevent the situation described in Example 5 by carrying out a stratified approach. The main idea is to restrict the set of clauses sent to the SAT solver to force it to concentrate on those with higher weights. As a result, the SAT solver returns unsatisfiable cores with clauses with higher weights. These are better quality cores and contribute to increase the cost faster. When the SAT solver returns SAT, then we allow it to use clauses with lower weights.

In Algorithm 3 we use a variable w_{max}, and we only send to the SAT solver the clauses with weight greater or equal than it. As in Algorithm 2, we start by checking that hard clauses are satisfiable. Then, we initialize w_{max} to the

Algorithm 3. The pseudo-code of the stratified approach for WPM1 algorithm.

```
    Input: $\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$
1:  if $SAT(\{C_i \mid w_i = \infty\}) = (\text{UNSAT}, \ )$ then return $(\infty, \emptyset)$
2:  $cost := 0$                                                      ▷Optimal
3:  $w_{max} := \max\{w_i \mid (C_i, w_i) \in \varphi \wedge w_i < w_{max}\}$
4:  while true do
5:      $(st, \varphi_c) := SAT(\{C_i \mid (C_i, w_i) \in \varphi \wedge w_i \geq w_{max}\})$     ▷Call without weights
6:      if $st = \text{SAT}$ and $w_{max} = 0$ then return $(cost, \varphi)$
7:      else
8:          if $st = \text{SAT}$ then $w_{max} := \max\{w_i \mid (C_i, w_i) \in \varphi \wedge w_i < w_{max}\}$
9:          else
10:             $BV := \emptyset$                                 ▷Blocking variables of the core
11:             $w_{min} := \min\{w_i \mid C_i \in \varphi_c \wedge w_i \neq \infty\}$        ▷Minimum weight
12:             foreach $C_i \in \varphi_c$ do
13:                 if $w_i \neq \infty$ then
14:                     $b :=$ new variable()
15:                     $\varphi := \varphi \setminus \{(C_i, w_i)\} \cup \{(C_i, w_i - w_{min}), \ (C_i \vee b, w_{min})\}$
                        ▷Duplicate soft clauses of the core
16:                     $BV := BV \cup \{b\}$
17:             $\varphi := \varphi \cup \{(C, \infty) \mid C \in CNF(\sum_{b \in BV} b = 1)\}$
                                               ▷Add cardinality constraint as hard clauses
18:             $cost := cost + w_{min}$
```

highest weight smaller than infinite. If the SAT solver returns SAT, there are two possibilities. Either w_{max} is zero (it means that we have already sent all clauses to the SAT solver) and we finish; or it is not yet zero, and we decrease w_{max} to the highest weight smaller than w_{max}, allowing the SAT solver to use clauses with smaller weights. If the SAT solver returns UNSAT, we proceed like in Algorithm 2. This algorithm was submitted to the MaxSAT evaluation 2011 as WPM1 (version 2011). It was the best performing solver for the weighted partial industrial category. The description of the solver was never published in a paper before.

We can use better strategies to decrease the value of w_{max}. Notice that, in the worst case, we could need more executions of the SAT solver than Algorithm 2, because the calls that return SAT but $w_{max} > 0$ do not contribute to increase the computed cost. Therefore, we need to find a balance between the number of those unproductive SAT calls, and the minimum weight of the cores. For example, one of the possible strategies is to decrease w_{max} until the following condition is satisfied

$$\frac{|C_i \mid (C_i, w_i) \in \varphi \wedge w_i < w_{max}\}|}{|\{w_i \mid (C_i, w_i) \in \varphi \wedge w_i < w_{max}\}|} > \alpha$$

or $w_{max} = 0$. This strategy tends to send more new clauses to the SAT solver when they have bigger diversity of weights. In our implementation of WPM1 submitted to the MaxSAT evaluation 2012, we use this strategy, called *diversity heuristic*, with $\alpha = 1.25$.

The proof of the correctness of this algorithm is like the proof for WPM1. The only additional point is that the new algorithm is forcing the SAT solver to find some cores before others. In the proof of correctness of WPM1 there is no assumption on what cores the SAT solver finds first.

7 MaxSAT Reducibility

Our algorithms solve a MaxSAT formula by successively transforming it until we get a satisfiable formula. To prove the soundness of the algorithms it suffices to prove that these transformations preserve the cost of the formula. However, apart from this notion of *cost-preserving transformation*, we can define other (stronger) notions of formula transformation, like *MaxSAT equivalence* and *MaxSAT reducibility.*

Definition 1.
We say that φ_1 *and* φ_2 *are* **cost-equivalent** *if* $\mathrm{cost}(\varphi_1) = \mathrm{cost}(\varphi_2)$*.*
We say that φ_1 *and* φ_2 *are* **MaxSAT equivalent** *if, for any assignment* $I : var(\varphi_1) \cup var(\varphi_2) \to \{0,1\}$*, we have* $\mathrm{cost}(I(\varphi_1)) = \mathrm{cost}(I(\varphi_2))$*.*
We say that φ_1 *is* **MaxSAT reducible** *to* φ_2 *if, for any assignment* $I : var(\varphi_1) \to \{0,1\}$*, we have* $\mathrm{cost}(I(\varphi_1)) = \mathrm{cost}(I(\varphi_2))$*.*

Notice that the distinction between MaxSAT equivalence and MaxSAT reduction is the domain on the partial assignment. In one case it is $\mathrm{var}(\varphi_1) \cup \mathrm{var}(\varphi_2)$, and in the other $\mathrm{var}(\varphi_1)$.

The notion of cost-preserving transformation is the weakest of all three notions, and suffices to prove the soundness of the algorithms. However, it does not allow us to replace *sub*-formulas by cost-equivalent *sub*-formulas, in other words $\mathrm{cost}(\varphi_1) = \mathrm{cost}(\varphi_2)$ does not imply $\mathrm{cost}(\varphi_1 \cup \varphi_3) = \mathrm{cost}(\varphi_2 \cup \varphi_3)$. On the other hand, the notion of MaxSAT equivalence is the strongest of all three notions, but too strong for our purposes, because the formula transformations we use does not satisfy this notion. When φ_2 has variables not occurring in φ_1, it is convenient to use the notion of MaxSAT reducibility, that, in these cases, is weaker than the notion of MaxSAT equivalence.

In the following we show some examples of the notions of Definition 1.

Example 6. The following example shows a formula transformation that preserves the cost, but not MaxSAT reducibility. Consider $\varphi_1 = \{(x,2),(\overline{x},1)\}$ and $\varphi_2 = \{(\ ,1)\}$. We have $\mathrm{cost}(\varphi_1) = \mathrm{cost}(\varphi_2) = 1$, hence the transformation of φ_1 into φ_2 is cost-preserving. However, φ_1 is not MaxSAT reducible to φ_2, because the assignment $I : \{x\} \to \{0,1\}$ with $I(x) = 0$, makes $\mathrm{cost}(I(\varphi_1)) = 2 \neq 1 = \mathrm{cost}(I(\varphi_2))$.

On the contrary, φ_2 is MaxSAT reducible to φ_1, because there is a unique assignment $I : \emptyset \to \{0,1\}$, and it satisfies $\text{cost}(I(\varphi_1)) = \text{cost}(I(\varphi_2))$. Hence, MaxSAT reducibility is not a symmetric relation.

The following example shows that MaxSAT reducibility does not imply MaxSAT equivalence. Consider $\varphi_1 = \{(x, 2), (x, 1)\}$ and $\varphi_3 = \{(\quad, 1), (x, 1), (x \vee y, 1), (x \vee z, 1), (y \vee z, \infty)\}$. We have that φ_1 is MaxSAT reducible to φ_3. To prove this, we must consider two interpretations I_1 and I_2, defined by $I_1(x) = 0$ and $I_2(x) = 1$. In the first case, we obtain $I_1(\varphi_1) = \{(\quad, 2)\}$ and $I_1(\varphi_3) = \{(\quad, 2), (y, 1), (y \vee z, \infty)\}$ that have the same cost 2. In the second case, we obtain $I_2(\varphi_1) = \{(\quad, 1)\}$ and $I_2(\varphi_3) = \{(\quad, 1), (z, 1), (y \vee z, \infty)\}$ that have also the same cost 1. However, φ_1 and φ_3 are not MaxSAT equivalent because for $I : \{x, y, z\} \to \{0, 1\}$ defined by $I(x) = I(y) = I(z) = 1$ we have $\text{cost}(I(\varphi_1)) = 1 \neq \infty = \text{cost}(I(\varphi_3))$.

Finally, φ_1 is MaxSAT equivalent to $\varphi_4 = \{(\quad, 1), (x, 1)\}$.

The notion of MaxSAT equivalence was implicitly defined in [7]. In this paper a MaxSAT resolution rule that preserves MaxSAT equivalence is defined, and proved complete for MaxSAT.

For lack of space we state without proof:

Lemma 1. (1) *If φ_1 is MaxSAT-reducible to φ_2 and var$(\varphi_2) \cap$ var$(\varphi_3) \subseteq$ var(φ_1), then $\varphi_1 \cup \varphi_3$ is MaxSAT-reducible to $\varphi_2 \cup \varphi_3$.*
(2) *MaxSAT-reducibility is transitive: if φ_1 is MaxSAT-reducible to φ_2, φ_2 is MaxSAT-reducible to φ_3, and var$(\varphi_1) \cap$ var$(\varphi_3) \subseteq$ var(φ_2), then φ_1 is MaxSAT-reducible to φ_3.*

Example 7. Notice that the side condition of Lemma 1 (1) is necessary. For instance, if we take $\varphi_1 = \{(\quad, 1)\}$, $\varphi_2 = \{(x, 1), (x, \infty)\}$ and $\varphi_3 = \{(x, 1)\}$, where the side condition $\text{var}(\varphi_2) \cap \text{var}(\varphi_3) = \{x\} \not\subseteq \emptyset = \text{var}(\varphi_1)$ is violated, we have that φ_1 is MaxSAT reducible to φ_2, but $\varphi_1 \cup \varphi_3$ is not MaxSAT reducible to $\varphi_2 \cup \varphi_3$.

Similarly, the side condition in Lemma 1 (2) is also necessary. For instance, if we take $\varphi_1 = \{(x, 1), (x, 1)\}$, $\varphi_2 = \{(\quad, 1)\}$ and $\varphi_3 = \{(x, 1), (x, \infty)\}$, where the side condition $\text{var}(\varphi_1)) \cap \text{var}(\varphi_3) = \{x\} \not\subseteq \emptyset = \text{var}(\varphi_2)$ is also violated, we have that φ_1 is MaxSAT reducible to φ_2 and this to φ_3. However, φ_1 is not MaxSAT reducible to φ_3.

There are two side conditions in Lemma 1 (1) and (2) (see Example 7) that restrict the set of variables that can occur in the MaxSAT problems. However, if we ensure that problem transformations only *introduce fresh* variables, i.e. when φ_1 is MaxSAT reduced to φ_2, all new variables introduced in φ_2 do not occur elsewhere, then these conditions are trivially satisfied. In our algorithms, all formula transformations satisfy this restriction.

8 Generic Stratified Approach

In Algorithm 4 we show how the stratified approach can be applied to any *generic* weighted MaxSAT solver WPM. In the rest of the section we will describe what

Algorithm 4. The pseudo-code of a generic MaxSAT algorithm that follows a stratified approach heuristics.

```
    Input: $\varphi = \{(C_1, w_1), \ldots, (C_m, w_m)\}$
 1: $cost := 0$
 2: $w_{max} = \infty$
 3: while $true$ do
 4:     $\varphi_{w_{max}} := \{(C_i, w_i) \in \varphi \mid w_i \geq w_{max}\}$
 5:     $(cost', \varphi_{sat}, \varphi_{res}) = \mathrm{WPM}(\varphi_{w_{max}})$
 6:     $cost = cost + cost'$
 7:     if $cost = \infty$ or $w_{max} = 0$ then return $(cost, \varphi_{sat})$
 8:     $W = \sum\{w_i \mid (C_i, w_i) \in \varphi \setminus \varphi_{w_{max}} \cup \varphi_{res}\}$
 9:     $\varphi_{sat} = \{(C_i, \mathrm{harden}(w_i, W)) \mid (C_i, w_i) \in \varphi_{sat}\}$
10:     $\varphi = (\varphi \setminus \varphi_{w_{max}}) \cup \varphi_{sat} \cup \varphi_{res}$
11:     $w_{max} = \mathrm{decrease}(w_{max})$
12: return $(cost, \varphi)$

13: function harden(w,W)
14: begin
15:     if $w > W$ then return $\infty$
16:     else return $w$
```

properties the generic algorithm WPM has to satisfy in order to ensure the correctness of this approach.

We assume that, given a weighted MaxSAT formula φ, $WPM(\varphi)$ returns a triplet $(cost, \varphi_{sat}, \varphi_{res})$ such that φ is MaxSAT reducible to $\{(\ \ , cost)\} \cup \varphi_{sat} \cup \varphi_{res}$, φ_{sat} is satisfiable (has cost zero), and clauses of φ_{res} have cost strictly smaller than w_{max}. Given φ, WPM1 return a pair $(cost, \varphi')$ where φ is MaxSAT reducible to $\{(\ \ , cost)\} \cup \varphi'$ and φ is satisfiable, hence satisfies the requirements taking $\varphi_{res} = \emptyset$. Moreover, we can also think of WPM as an algorithm that *partially* solves the formula, and returns a lower bound cost, a satisfiable part of the formula φ_{sat}, and an unsolved residual φ_{res}.

The algorithm uses a variable w_{max} to restrict the clauses sent to the MaxSAT solver. The first time $w_{max} = \infty$, and we run WPM only on the hard clauses. Then, in each iteration we send clauses with weight w_{max} or bigger to WPM. We add the return cost to the current cost, and decrease w_{max}, until w_{max} is zero.

Algorithm 3 is an instance of this generic schema where WPM is a partial execution of WPM1 where clauses generated during duplication with weight smaller than w_{max} are put apart in φ_{res}.

Lines 8 and 9 are optional and can be removed from the algorithm without affecting to its correctness. They are inspired in [15]. The idea is to harden all soft clauses with weight bigger than the sum of the weights of the clauses not sent to the WPM plus the clauses returned in φ_{res}. The proof of the correctness of these lines is based in the following lemma (not proved for lack of space).

Lemma 2. *Let* $\varphi_1 = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$ *be a satisfiable MaxSAT formula,* $\varphi_2 = \{(C'_1, w'_1), \ldots, (C'_r, w'_r)\}$ *be a MaxSAT formula without hard clauses and* $W = \sum_{j=1}^{r} w'_j$*. Let*

$$harden(w) = \begin{cases} w & \text{if } w \leq W \\ \infty & \text{if } w > W \end{cases}$$

and $\varphi'_1 = \{(C_i, harden(w_i)) \mid (C_i, w_i) \in \varphi_1\}$*. Then* $\mathrm{cost}(\varphi_1 \cup \varphi_2) = \mathrm{cost}(\varphi'_1 \cup \varphi_2)$*.*

Notice that we check the applicability of this lemma dynamically, recomputing the value W in every iteration in line 8 of Algorithm 4.

Theorem 1. *Assuming that WPM, given a formula* φ*, returns a triplet* $(cost, \varphi_{sat}, \varphi_{res})$ *such that* φ *is MaxSAT reducible to* $\{(\ , cost)\} \cup \varphi_{sat} \cup \varphi_{res}$*,* φ_{sat} *is satisfiable, and* φ_{res} *only contain clauses with weight strictly smaller than* w_{max}*, Algorithm 4 is a correct algorithm for Weighted Partial MaxSAT. Moreover, when for a formula* φ*, the algorithm returns* (c, φ')*, then* $c = \mathrm{cost}(\varphi)$ *and any assignment satisfying* φ' *is an optimal assignment of* φ*.*

9 Experimental Results

We conducted our experimentation on the same environment as the MaxSAT evaluation [4] (processor 2 GHz). We increased the timeout from half hour to two hours, and the memory limit from 0.5G to 1G. The solvers that implement our Weighted Partial MaxSAT algorithms are built on top of the SAT solver picosat (v.924) [6]. The solver *wpm1* implements the original WPM1 algorithm [1]. The cardinality constraints introduced by WPM1 are translated into SAT through the regular encoding [3]. This encoding assures a linear complexity on the size of the cardinality constraint. This is particularly important for the last queries where the size of the cores can be potentially close to the number of soft clauses. We use the subscript b to indicate that we break symmetries as described in section 4, s to indicate we apply the stratified approach and d to indicate that we apply the diversity heuristic to compute the next w_{max}, both described in section 6. *wpm1* was the solver submitted to the 2009 and 2010 MaxSAT evaluations, and *wmp1*$_s$ the one submitted to the 2011 evaluation. The hardening soft clauses (lines 8 and 9 in Algorithm 4) had not impact in our implementations' performance.

In the following we present results for the benchmarks of the Weighted Partial MaxSAT categories of the MaxSAT 2011 evaluation. We compare our solvers with the best three solvers of the evaluation, and other solvers which did not compete but have been reported to exhibit good performance, such as, *binc* and *bincd* [11], maxhs [8] and the Weighted CSP solver toulbar2 [19].

We present the experimental results following the same classification criteria as in the MaxSAT evaluation. For each solver and set of instances, we present the number of solved instances in parenthesis and the mean time required to solve them. Solvers are ordered from left to right according to the total number

Table 1. Experimental results

set	#	wpm1$_{bsd}$	wpm1$_{bs}$	wpm1$_s$	wpm1$_b$	wbo1.6	wpm1	wpm2	maxhs	bincd	maxhs	sat4j	binc	toulbar2
haplotyping	100	**423(95)**	425(95)	378(88)	376(79)	93.4(72)	390(65)	350(42)	1043(34)	196(21)	1247(35)	108(20)	408(27)	383(5)
timetabling	26	685.60(9)	**683(9)**	585(8)	992(9)	776(4)	1002(7)	707(9)	1238(4)	766(6)	2388(5)	0(0)	1350(4)	0(0)
upgradeability	100	35.8(100)	37.2(100)	36.5(100)	36.9(100)	63.3(100)	114(100)	311(100)	**29(100)**	637(78)	28.4(50)	844(30)	0(0)	0(0)
Total	226	**204**	204	196	188	176	172	151	138	105	90	50	31	5

(a) Weighted Partial - Industrial

set	#	wpm1$_{bsd}$	incw maxsatz	ak maxsat	toulbar2	wpm1$_{bs}$	wmax satz09z	maxhs	sat4j	wpm1$_s$	bincd	binc	wpm1	wbo1.6	wpm1$_b$
auc-paths	86	274(53)	7.59(86)	**4.6(86)**	28.8(86)	332(53)	570(80)	72.2(86)	994(44)	22(33)	1828(2)	0(0)	0(0)	0(0)	0(0)
auc-sched	84	**11.9(84)**	220(84)	123(84)	133(84)	12.2(84)	92(84)	1125(69)	716(80)	7.6(80)	130(50)	103(45)	0(0)	0(0)	0(0)
planning	56	27.9(52)	92.2(38)	354(40)	149(41)	**26.3(56)**	220(50)	306(29)	3.27(55)	12.5(54)	59.5(47)	51.1(46)	1.46(28)	1.33(30)	3.63(29)
warehouses	18	44(14)	1184(18)	37(2)	0.03(1)	571(3)	0.32(1)	0.37(1)	1.34(1)	1644(3)	7.03(1)	8.67(1)	**4.23(18)**	0.51(4)	0.88(12)
miplib	12	1187(4)	**1419(5)**	0.47(2)	63.2(3)	1165(4)	266(3)	0.07(1)	693(4)	34(3)	618(3)	699(3)	0.21(1)	0(0)	1507(1)
random-net	74	**241(39)**	1177(1)	1570(2)	0(0)	0(0)	0(0)	2790(6)	0(0)	0(0)	0(0)	0(0)	615(27)	63(37)	439(12)
spot5dir	21	257(10)	1127(5)	1106(5)	217(5)	383(10)	11.5(2)	199(6)	1.95(2)	1.41(5)	**66.7(11)**	51.7(6)	1.03(4)	2.60(5)	12.8(6)
spot5log	21	**532(14)**	0.63(4)	200(5)	170(5)	574(14)	15.6(2)	710(6)	6.04(3)	44.4(6)	124(11)	79.3(7)	21.5(6)	25.5(6)	131(7)
Total	372	**270**	241	226	225	224	222	204	189	184	125	108	84	82	67

(b) Weighted Partial - Crafted

Instance set	#	maxhs	wbo1.6	wpm1$_b$	wpm1$_{bsd}$	wpm1	wpm1$_{bs}$	wpm1$_s$	bincd	sat4j	binc	toulbar2
Table4 [8]	13	**4.41(13)**	5.03(13)	5.54(13)	8.84(13)	18.92(13)	534.13(13)	559.23(13)	56.69(11)	3485.52(6)	19.50(1)	0.00(0)
Total	13	**13**	13	13	13	13	13	13	11	6	1	0

(c) Table 4 from [8]. Linux upgradibility family forcing diversity of weights of instances they solved. We present in bold the results for the best performing solver in each set. '# ' stands for number of instances of the given set.

Table 1(a) presents the results for the industrial instances of the Weighted Partial MaxSAT category. As we can see, our original solver *wpm1* would have ranked as the second best solver after *wbo1.6*. By breaking symmetries ($wpm1_b$) we solve 12 more instances than *wbo1.6*, and 20 more if we apply the stratified approach. Combining both, we solve 28 more instances. The addition of the diversity heuristic to the stratified approach has no impact for the instances of this category. We do not present any result on branch and bound based solvers since they typically do not perform well on industrial instances.

Table 1(b) presents the results for the crafted instances of the Weighted Partial MaxSAT category. The best ranked solvers in this category for the MaxSAT 2011 evaluation were: *incwmaxsatz*, *akmaxsat* and *wmaxsatz09*, in this order. All are branch and bound based solvers, which typically dominate the crafted and random categories. We can see that our solver *wpm1* shows a poor performance in this category. However, by applying the stratified approach ($wpm1_s$) we jump from 84 solved instances to 184. If we also break symmetries ($wpm1_{bs}$) we solve 224 instances, ranking as the third best solver respect to the participants of the MaxSAT 2011 evaluation, very close to *akmaxsat*. If we compare carefully the results of *wpm1* and $wpm1_{bs}$, we notice that there are two sets where *wpm1* behaves much better (warehouses and random-net). This suggests that we must make our stratified approach more flexible, for example, by incorporating the diversity heuristic ($wpm1_{bsd}$). Using $wpm1_{bsd}$ we solve up to 270 instances, outperforming all the branch and bound solvers.

In [8] it is pointed out that instances with a great diversity of weights can be a bottleneck for some Weighted MaxSAT solvers. To test this hypothesis they generate 13 instances from the Linux upgradibility set in the Weighted Partial MaxSAT industrial category preserving the underlying CNF but modifying the weights to force a greater diversity. We have reproduced that experiment in Table 1(c). As we can see, *wpm1* compares well to the best performing solvers, and by breaking symmetries ($wpm1_b$) we reach the performance of *maxhs* and *wbo1.6*. On the other hand, the stratified approach impacts negatively ($wpm1_s$ or $wpm1_{bs}$), but the diversity heuristic fixes this problem.

Taking into consideration the experimental results obtained in the different categories, we can see that our approach $wpm1_{bsd}$ is the most robust solver for Weighted Partial MaxSAT instances. We also checked the effectiveness of breaking symmetries for Unweighted Partial MaxSAT instances. For industrial instances we improve from 181 to 262 solved instances, and for crafted from 55 to 115.

References

1. 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)
2. Ansótegui, C., Bonet, M.L., Levy, J.: A new algorithm for weighted partial MaxSAT. In: AAAI 2010 (2010)

3. Ansótegui, C., Manyà, F.: Mapping Problems with Finite-Domain Variables to Problems with Boolean Variables. In: Hoos, H.H., Mitchell, D.G. (eds.) SAT 2004. LNCS, vol. 3542, pp. 1–15. Springer, Heidelberg (2005)
4. Argelich, J., Li, C.M., Manyà, F., Planes, J.: MaxSAT evaluations (2006, 2007, 2008, 2009, 2010, 2011), `http://www.maxsat.udl.cat`
5. Berre, D.L.: SAT4J, a satisfiability library for java (2006), `http://www.sat4j.org`
6. Biere, A.: PicoSAT essentials. Journal on Satisfiability 4, 75–97 (2008)
7. Bonet, M.L., Levy, J., Manyà, F.: Resolution for Max-SAT. Artif. Intell. 171(8-9), 606–618 (2007)
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. 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)
10. Heras, F., Larrosa, J., Oliveras, A.: MiniMaxSat: A New Weighted Max-SAT Solver. In: Marques-Silva, J., Sakallah, K.A. (eds.) SAT 2007. LNCS, vol. 4501, pp. 41–55. Springer, Heidelberg (2007)
11. Heras, F., Morgado, A., Marques-Silva, J.: Core-guided binary search algorithms for maximum satisfiability. In: AAAI 2011 (2011)
12. Li, C.M., Manyà, F., Mohamedou, N., Planes, J.: Exploiting Cycle Structures in Max-SAT. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 467–480. Springer, Heidelberg (2009)
13. Lin, H., Su, K., Li, C.M.: Within-problem learning for efficient lower bound computation in Max-SAT solving. In: AAAI 2008, pp. 351–356 (2008)
14. 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)
15. Marques-Silva, J., Argelich, J., Graça, A., Lynce, I.: Boolean lexicographic optimization: algorithms & applications. Ann. Math. Artif. Intell. 62(3-4), 317–343 (2011)
16. Marques-Silva, J., Lynce, I., Manquinho, V.: Symmetry Breaking for Maximum Satisfiability. In: Cervesato, I., Veith, H., Voronkov, A. (eds.) LPAR 2008. LNCS (LNAI), vol. 5330, pp. 1–15. Springer, Heidelberg (2008)
17. Marques-Silva, J., Manquinho, V.: Towards More Effective Unsatisfiability-Based Maximum Satisfiability Algorithms. In: Kleine Büning, H., Zhao, X. (eds.) SAT 2008. LNCS, vol. 4996, pp. 225–230. Springer, Heidelberg (2008)
18. Marques-Silva, J., Planes, J.: On using unsatisfiability for solving maximum satisfiability. CoRR abs/0712.1097 (2007)
19. Sanchez, M., Bouveret, S., Givry, S.D., Heras, F., Jégou, P., Larrosa, J., Ndiaye, S., Rollon, E., Schiex, T., Terrioux, C., Verfaillie, G., Zytnicki, M.: Max-CSP competition 2008: toulbar2 solver description (2008)

B

Solving (Weighted) Partial MaxSAT with ILP

C. Ansótegui, J. Gabàs

in: Proceedings of the 10th International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) techniques in Constraint Programming

CPAIOR Yorktown Heights

2013

Solving (Weighted) Partial MaxSAT with ILP [⋆]

Carlos Ansótegui[1] and Joel Gabàs[1]

DIEI, Univ. de Lleida
carlos@diei.udl.cat
joel.gabas@diei.udl.cat

Abstract. Several combinatorial optimization problems can be translated into the Weighted Partial Maximum Satisfiability (WPMS) problem. This is an optimization variant of the Satisfiability (SAT) problem. There are two main families of WPMS solvers based on SAT technology: *branch and bound* and *SAT-based*. From the MaxSAT evaluations, we have learned that SAT-based solvers dominate on industrial instances while branch and bound dominate on random. For crafted instances it depends on the category.
In this work, we study the performance of an Integer Linear Programming approach. In particular, we translate the WPMS problem into ILP and apply the Mixed Integer Programming (MIP) solver, IBM-CPLEX. We present an extensive experimental evaluation showing that this approach clearly dominates on crafted instances.

1 Introduction

The Maximum Satisfiability (MaxSAT) problem is the optimization version of SAT and has several application domains [2, 4, 20–24]. The idea behind this formalism is that sometimes not all constraints of a problem can be satisfied, and we try to satisfy the maximum number of them. The MaxSAT problem can be further generalized to the Weighted Partial MaxSAT (WPMS) problem. In this case, we can divide the constraints in two groups: the constraints that must be satisfied (hard), and the ones that may or may not be satisfied (soft). In the last group, we may put different weights to the constraints, where the weight is the penalty to falsify the constraint.

There are two main classes of WPMS algorithms: branch and bound [7, 11, 13–15] and SAT-based [1, 5, 8–10, 17–19]. SAT-based MaxSAT algorithms consist in the reformulation of the problem as a sequence of SAT instances.

From the last international MaxSAT evaluation 2012 [3], we can conclude that SAT-based solvers dominate on industrial instances while branch and bound dominate on random instances. For crafted instances, it is not so clear. Branch and bound solvers have dominated since 2006, but at the last 2012 evaluation, two SAT-based solvers were reported to be the best for crafted instances at the Weighted Partial MaxSAT category.

[⋆] This research has been partially founded by the CICYT research projects TASSAT (TIN2010-20967-C04-01/03/04) and ARINF (TIN2009-14704-C03-01).

In this paper, we focus our attention on Mixed Integer Programming (MIP) techniques from Operation Research (OR). They have been very successful on solving optimization problems and it makes sense to study the performance of these techniques on WPMS instances. In particular, we explore the approach which consists in translating the WPMS instances into ILP and then apply a MIP solver. It is easy to see that the *soft* clauses of a WPMS instance represent the objective function of the corresponding ILP instance, while the *hard* clauses represent the region of feasible solutions. Section 3 provides a detailed description.

Our experimental evaluation shows that this approach clearly dominates on (Weighted) Partial crafted instances. This is a surprising result not reported before and worth to be known in the community. On the other hand, we also show that this approach is not competitive on the industrial instances. Therefore, further work is required to identify the strength of ILP for crafted instances and how to take advantage of it.

To our best knowledge, the first work using MIP technology for MaxSAT solving can be found in [6]. However, the experimental results did not show such a good performance on crafted instances. A more recent work, using MIP technology can be found in [5]. The authors present a hybrid approach where a SAT solver and a MIP solver interact. This approach also solves a sequence of SAT instances as SAT-based solvers. These instances are simpler than the ones generated by SAT-based solvers since all the arithmetic constraints are extracted and managed by the MIP solver. This is an interesting approach, but from the experimental evaluation we can see it is not yet competitive enough.

This paper proceeds as follows. Section 2 presents some preliminary concepts. Section 3 presents the translation from WPMS into ILP. Section 4 presents the experimental evaluation. Finally, Section 5 shows the conclusions and the future work.

2 Preliminaries

A *literal* is either a Boolean variable x or its negation $\overline{x}$. A *clause* C is a disjunction of literals. A *weighted clause* is a pair (C, w), where C is a clause and w is a natural number or infinity, indicating the penalty for falsifying the clause C. A *Weighted Partial MaxSAT formula* is a multiset of weighted clauses

$$\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$$

where the first m clauses are soft and the last m' clauses are hard. The set of variables occurring in a formula φ is noted as $\mathrm{var}(\varphi)$.

A *(total) truth assignment* for a formula φ is a function $I : \mathrm{var}(\varphi) \rightarrow \{0, 1\}$, that can be extended to literals, clauses, SAT formulas. For MaxSAT formulas is defined as $I(\{(C_1, w_1), \ldots, (C_m, w_m)\}) = \sum_{i=1}^{m} w_i\,(1 - I(C_i))$. The *optimal cost* of a formula is $\mathrm{cost}(\varphi) = \min\{I(\varphi) \mid I : \mathrm{var}(\varphi) \rightarrow \{0, 1\}\}$ and an *optimal assignment* is an assignment I such that $I(\varphi) = \mathrm{cost}(\varphi)$.

The *Weighted Partial MaxSAT problem* for a Weighted Partial MaxSAT formula φ is the problem of finding an *optimal assignment.*

3 Translation of Weighted Partial MaxSAT into ILP

Encodings translating WPMS into ILP can be found in the literature [12, 16]. Here, we describe the precise encoding we used in our evaluation. Given a WPMS formula, $\{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$, we can translate it into a ILP instance, as follows:

Let $s = (\cup_{i=1}^{m} CNF(\overline{b}_i \leftrightarrow C_i))$ and $h = (\cup_{j=m+1}^{m+m'} C_j)$, where $CNF(\varphi)$ transforms φ into Conjuntive Normal Form and the b_i's are new fresh Boolean variables. The ith element of s ensures that b_i is true iff the soft clause C_i is falsified and h is the set of hard clauses of the WPMS problem. The soft clauses of a WPMS instance represent the *objective function* of the *equivalent* ILP instance which can be described as follows:

$$\begin{aligned}
&\text{Minimize: } \textstyle\sum_1^m w_i \cdot b_i \\
&\text{Subject to:} \\
&ILP(s \cup h) \\
&0 \leq x_i \leq 1, x_i \in var(s \cup h)
\end{aligned}$$

where function $ILP(\varphi)$ maps every clause $C_i \in \varphi$ into a linear inequality with operator $>$. The left-hand side of the linear inequality corresponds to the sum of the literals in C_i once mapped into integer terms, such that, literal $x(\overline{x})$ is mapped to integer term $x(1-x)$. The right-hand side corresponds to constant 0. After moving the constants to the rigth, the right-hand side corresponds to constant $-k$, where k is the number of negative literals in clause C_i. Finally, we add the bounding box constraints that ensure that every integer variable in the ILP instance has domain $\{0, 1\}$. It can be easily seen that the implication $C_i \rightarrow \overline{b}_i$ from $\overline{b}_i \leftrightarrow C_i$ is unnecessary (as we are optimizing).

Example 1. Given the WPMS formula, $\{(x_1 \vee x_2, 2), \ldots, (x_1 \vee \overline{x}_2, 3), (\overline{x}_1 \vee x_2, \infty), (\overline{x}_1 \vee \overline{x}_2, \infty)\}$, the corresponding ILP formulation is [1]:

Minimize: $2 \cdot b_1 + 3 \cdot b_2$

Subject to:

$$\begin{array}{ll}
x_1 + x_2 + b_1 > 0; & \backslash\ \overline{b}_1 \rightarrow (x_1 \vee x_2) \\
-x_1 - b_1 > -2; & \\
-x_2 - b_1 > -2; & \backslash\ (x_1 \vee x_2) \rightarrow \overline{b}_1 \\
x_1 - x_2 + b_2 > -1; & \backslash\ \overline{b}_2 \rightarrow (x_1 \vee \overline{x}_2) \\
-x_1 - b_2 > -2; & \\
x_2 - b_2 > -1; & \backslash\ (x_1 \vee \overline{x}_2) \rightarrow \overline{b}_2 \\
-x_1 + x_2 > -1; & \backslash\ \overline{x}_1 \vee x_2 \\
-x_1 - x_2 > -2; & \backslash\ \overline{x}_1 \vee \overline{x}_2
\end{array}$$

Bounds: $0 \leq x_1 \leq 1;\ 0 \leq x_2 \leq 1;\ 0 \leq b_1 \leq 1;\ 0 \leq b_2 \leq 1;$

[1] For example, for the first soft clause we get: $ILP(\{CNF(\overline{b}_1 \leftrightarrow (x_1 \vee x_2)\}) = ILP(\{CNF(\overline{b}_1 \rightarrow (x_1 \vee x_2)), CNF((x_1 \vee x_2) \rightarrow \overline{b}_1)\ \}) = ILP(\{(x_1 \vee x_2 \vee b_1), (\overline{x}_1 \vee \overline{b}_1), (\overline{x}_2 \vee \overline{b}_1)\ \}) = \{(x_1 + x_2 + b_1 > 0), ((1-x_1) + (1-b_1) > 0), ((1-x_2) + (1-b_2) > 0)\ \} = \{(x_1 + x_2 + b_1 > 0), (-x_1 - b_1 > -2), (-x_2 - b_1 > -2)\ \}$.

4 Experimental Results

In this section we present our intensive experimental investigation on the PMS and WPMS crafted and industrial instances from the 2012 MaxSAT Evaluation. Results on random instances are not included since the translation of WPMS into ILP did not win on any family. We provide results for the ILP translation, the best two solvers for each category of the 2012 MaxSAT Evaluation, and two solvers which did not participate but have been reported to exhibit good performance. We run our experiments on a cluster featured with 2.27 GHz processors, memory limit of 3.9 GB and a timeout of 7200 seconds per instance [2].

The results are presented in Table 1 following the same criteria as in the 2012 MaxSAT Evaluation. For each solver and family of instances, we present the number of solved instances in parenthesis and the mean solving time. Solvers are ordered from left to right according to the total number of solved instances. The results for the best performing solver in each family are presented in bold. The number of instances of each family is specified in the column under the sign '#'. Since different families may have different number of instances, we also include for each solver the mean ratio of solved instances.

Table 1 shows the results of our experimentation where we compare the following solvers. The solver *ilp* which corresponds to the translation of MaxSAT into ILP (see Section 3) [3], and the application of the MIP solver IBM-CPLEX studio124 (through C++ API and default parameters). The best two solvers for each category of the 2012 MaxSAT Evaluation: WPMS crafted (*wpm*1 [1], *shinms* [9]), WPMS industrial (*pwbo*2.1 [17, 18], *wpm*1), PMS crafted (*qms*0.21 [10], *akms_ls* [11] and PMS industrial (*qms*0.21*g*2, *pwbo*2.1). Two other solvers that have been reported to exhibit good performance: *bincd*2 , which is the new version of the BINCD algorithm [8, 19], with the best configuration reported by authors, and *maxhs* from [5], which is a hybrid SAT-MIP approach.

Table 1(a) presents the results for the PMS crafted instances. The *ilp* approach solves 332 of 372 instances, 35 more than *akms_ls*. PMS solver *qms*0.21 is the third in solved instances but the first in mean ratio with 81.1%.

Table 1(b) presents the results for the WPMS crafted instances. Again, the *ilp* approach is the best one, solving 332 of 372 instances, 14 more than the second one, *wpm*1. It has a clear impact on the *auc-paths*, *auc-scheduling* and *mini-encoding-warehouses* families, solving 100% of the instances in half a second.

Table 1(c) presents the results for the PMS industrial instances. We can see that, although the *ilp* approach is in general not competitive for industrial instances, it wins for *aes*, *bcp-fir* and *bcp-syn* families. In *bcp-syn* it performs much better than the rest of the solvers.

Table 1(d) presents the results for the WPMS industrial instances. Although we can confirm that the *ilp* approach is not competitive for industrial instances, it performs quite well in *upgradeability-problem* family.

[2] We thank the Artificial Intelligence Research Institute (IIIA) of the Spanish Research Council (CSIC) for the access to their high-performance computing clusters.

[3] Not considering implication $C_i \rightarrow \overline{b}_i$ gave similar results.

Family	#	*ilp*	*akms_ls*	*qms*0.21	*shinms*	*bincd*2	*pwbo*2.1	*wpm*1	*maxhs*
frb	25	1153(13)	160(5)	**347(25)**	44(23)	0.0(0)	151(15)	0.0(0)	2099(5)
job-shop	3	0.0(0)	0.0(0)	42(3)	**36(3)**	100(3)	93(1)	83(2)	0.0(0)
mxc_ran	96	45(96)	**1.1(96)**	270(83)	339(76)	85(71)	80(64)	0.0(0)	2164(12)
mxc_str	62	326.5(38)	**282(41)**	800(30)	402(23)	109(21)	37(19)	92(9)	1039(13)
mxo_3st	80	13.1(80)	**0.5(80)**	198(80)	695(78)	8.3(80)	37(63)	208(78)	234(46)
mxo_str	60	337.6(59)	482(38)	**6.4(60)**	3.5(59)	57(60)	7.7(60)	618(41)	0.0(0)
min-enc_kt	42	**163(42)**	3199(34)	248(6)	514(5)	275(6)	307(2)	689(3)	0.0(0)
ps_ml	4	34(4)	259(3)	**1.8(4)**	3.4(4)	49(4)	93(4)	5.3(3)	92(4)
Total	372	**332**	297	291	271	245	228	136	80
Ratio		76.5%	63.2%	**81.1%**	77.0%	65.3%	59.3%	41.1%	26.4%

(a) Partial Crafted

Family	#	*ilp*	*wpm*1	*shinms*	*akms_ls*	*pwbo*2.1	*maxhs*	*bincd*2
auc-pat	86	**0.5(86)**	484(59)	318(84)	2.6(86)	111(19)	35(86)	1415(12)
auc-sch	84	**0.4(84)**	5.7(84)	5.8(84)	68(84)	7.7(81)	965(78)	142(81)
min-enc-p	56	297(56)	36(53)	8.2(52)	141(40)	**0.5(56)**	459(31)	33(54)
min-enc-w	18	**0.5(18)**	20(14)	0.4(1)	20(2)	3.8(14)	0.2(1)	2.1(1)
ps-ml	12	82.82(3)	845.0(5)	**128(5)**	0.3(2)	4.0(3)	0.1(1)	1073(4)
ran-net	74	**533(59)**	160(41)	0.0(0)	4061(8)	42(35)	2771(10)	0.0(0)
wcsp-s5-d	21	43(18)	549(14)	**744(21)**	1556(6)	62(8)	101(6)	128(12)
wcsp-s5-l	21	323(8)	277(15)	**201(17)**	109(5)	1.7(6)	357(6)	299(13)
Total	372	**332**	285	264	233	222	219	177
Ratio		**78.6%**	72.0%	64.8%	45.3%	54.4%	41.6%	45.6%

(b) Weighted Partial Crafted

Family	#	*bincd*2	*qms*0.21*g*2	*pwbo*2.1	*shinms*	*wpm*1	*ilp*	*maxhs*
aes	7	453(1)	3155(1)	0.0(0)	0.0(0)	0.0(0)	**1311(3)**	453(2)
bcp-fir	59	44(58)	108(56)	68(56)	14(22)	9.6(55)	**63(59)**	481(25)
bcp-h-y_si	17	171(16)	**358(17)**	175(15)	41(16)	137(16)	667(6)	277(11)
bcp-h-y_su	38	245(32)	**106(35)**	98(25)	282(34)	310(24)	0.0(0)	307(21)
bcp-msp	64	**214(38)**	452(30)	96(26)	281(22)	320(9)	856(37)	120(1)
bcp-mtg	40	1.2(40)	**0.2(40)**	0.6(40)	0.6(40)	13(40)	769(29)	115(6)
bcp-syn	74	29(43)	284(35)	22(39)	87(33)	32(41)	**19(71)**	86(61)
cir-tra-com	4	109(4)	**45(4)**	200(2)	52(4)	544(4)	6922(1)	0.0(0)
hap-ass	6	728(5)	153(5)	**9.1(5)**	0.0(0)	289(4)	2125(5)	14(5)
pbo-mqc_ne	84	279(84)	**59(84)**	222(68)	146(84)	486(35)	1101(6)	400(34)
pbo-mqc_nl	84	79(84)	**24(84)**	72(82)	180(79)	423(49)	508(6)	364(37)
pbo-rou	15	**1.1(15)**	3.6(15)	28(15)	4.8(15)	1.2(15)	20(15)	28(14)
pro-ins	12	314(3)	**129(12)**	0.1(1)	207(4)	0.3(1)	2.7(1)	7.6(1)
Total	504	**423**	418	374	353	293	239	218
Ratio		78.2%	**83.0%**	66.3%	63.6%	61.2%	48.9%	43.0%

(c) Partial Industrial

Family	#	*wpm*1	*pwbo*2.1	*bincd*2	*maxhs*	*ilp*	*shinms*
hap-ped	100	**203(95)**	123(87)	545(73)	1089(39)	1892(18)	1204(47)
tim	26	**1168(11)**	672(7)	169(8)	1250(6)	0.0(0)	2261(5)
upg-pro	100	16(100)	33(100)	76(100)	**13(100)**	19(100)	0.0(0)
Total	226	**206**	194	181	145	118	52
Ratio		**79.1%**	71.3%	67.9%	54.0%	39.3%	22.1%

(d) Weighted Partial Industrial

Table 1. Experimental results of ILP translation compared with other solvers.

5 Conclusions and Future Work

From the experimentation, we conclude that the translation of WPMS into ILP has the best performance on crafted instances. This is a quite remarkable result since branch and bound solvers, like *akms_ls*, have always dominated this category since 2006. The ILP translation is however not competitive on industrial and random instances. This is something to be studied in depth, and may constitute the seed for new hybrid approaches of ILP and other WPMS algorithms.

References

1. C. Ansótegui, M. L. Bonet, J. Gabàs, and J. Levy. Improving sat-based weighted maxsat solvers. In *Proc. of the 18th Int. Conf. on Principles and Practice of Constraint Programming (CP'12)*, pages 86–101, 2012.
2. J. Argelich, D. L. Berre, I. Lynce, J. P. M. Silva, and P. Rapicault. Solving linux upgradeability problems using boolean optimization. In *Proceedings First International Workshop on Logics for Component Configuration (LoCoCo)*, pages 11–22, 2010.
3. J. Argelich, C. M. Li, F. Manyà, and J. Planes. Maxsat evaluations, 2011, 2012. http://www.maxsat.udl.cat.
4. R. Asín and R. Nieuwenhuis. http://www.lsi.upc.edu/∼rasin/timetabling.html, 2010.
5. J. Davies and F. Bacchus. Solving maxsat by solving a sequence of simpler sat instances. In *Proc. of the 17th Int. Conf. on Principles and Practice of Constraint Programming (CP'11)*, pages 225–239, 2011.
6. Z. Fu and S. Malik. On solving the partial max-sat problem. In *Proc. of the 9th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'06)*, pages 252–265, 2006.
7. F. Heras, J. Larrosa, and A. Oliveras. MiniMaxSat: A new weighted Max-SAT solver. In *Proc. of the 10th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'07)*, pages 41–55, 2007.
8. F. Heras, A. Morgado, and J. Marques-Silva. Core-guided binary search algorithms for maximum satisfiability. In *Proc. the 25th National Conference on Artificial Intelligence (AAAI'11)*, 2011.
9. K. Honjyo and T. Tanjo. Shinmaxsat. A Weighted Partial Max-SAT solver inspired by MiniSat+, Information Science and Technology Center, Kobe University.
10. M. Koshimura, T. Zhang, H. Fujita, and R. Hasegawa. Qmaxsat: A partial max-sat solver. *JSAT*, 8(1/2):95–100, 2012.
11. A. Kügel. Improved exact solver for the weighted max-sat problem. To appear.
12. C. M. Li and F. Manyà. Maxsat, hard and soft constraints. In *Handbook of Satisfiability*, pages 613–631. 2009.
13. C. M. Li, F. Manyà, N. O. Mohamedou, and J. Planes. Exploiting cycle structures in Max-SAT. In *Proc. of the 12th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'09)*, 2009.
14. H. Lin and K. Su. Exploiting inference rules to compute lower bounds for Max-SAT solving. In *IJCAI'07*, pages 2334–2339, 2007.
15. H. Lin, K. Su, and C. M. Li. Within-problem learning for efficient lower bound computation in Max-SAT solving. In *Proc. the 23th National Conference on Artificial Intelligence (AAAI'08)*, pages 351–356, 2008.
16. V. Manquinho, J. Marques-Silva, and J. Planes. Algorithms for weighted boolean optimization. In *Proc. of the 12th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'09)*, pages 495–508, 2009.
17. R. Martins, V. M. Manquinho, and I. Lynce. Exploiting cardinality encodings in parallel maximum satisfiability. In *ICTAI*, pages 313–320, 2011.
18. R. Martins, V. M. Manquinho, and I. Lynce. Clause sharing in parallel maxsat. In *LION*, pages 455–460, 2012.
19. A. Morgado, F. Heras, and J. Marques-Silva. Improvements to core-guided binary search for maxsat. In *Proc. of the 15th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'12)*, pages 284–297, 2012.

20. J. D. Park. Using weighted max-sat engines to solve mpe. In *Proc. the 24th National Conference on Artificial Intelligence (AAAI'10)*, pages 682–687, 2002.
21. S. Safarpour, H. Mangassarian, A. G. Veneris, M. H. Liffiton, and K. A. Sakallah. Improved design debugging using maximum satisfiability. In *FMCAD*, pages 13–19. IEEE Computer Society, 2007.
22. D. M. Strickland, E. Barnes, and J. S. Sokol. Optimal protein structure alignment using maximum cliques. *Oper. Res.*, 53(3):389–402, May 2005.
23. M. Vasquez and J. kao Hao. A "logic-constrained" knapsack formulation and a tabu algorithm for the daily photograph scheduling of an earth observation satellite. *Computational Optimization and Applications*, 20:137–157, 2001.
24. H. Xu, R. A. Rutenbar, and K. A. Sakallah. sub-sat: a formulation for relaxed boolean satisfiability with applications in routing. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 22(6):814–820, 2003.

C

Improving WPM2 for (Weighted) Partial MaxSAT

C. Ansótegui, M. L. Bonet, J. Gabàs, J. Levy

in: Proceedings of the 19th International Conference on Principles and Practice of Constraint Programming

CP Uppsala

2013

Improving WPM2 for (Weighted) Partial MaxSAT*

Carlos Ansótegui[1], Maria Luisa Bonet[2], Joel Gabàs[1], and Jordi Levy[3]

[1] DIEI, Univ. de Lleida
{carlos,joel.gabas}@diei.udl.cat
[2] LSI, UPC
bonet@lsi.upc.edu
[3] IIIA-CSIC
levy@iiia.csic.es

Abstract. Weighted Partial MaxSAT (WPMS) is an optimization variant of the Satisfiability (SAT) problem. Several combinatorial optimization problems can be translated into WPMS. In this paper we extend the state-of-the-art WPM2 algorithm by adding several improvements, and implement it on top of an SMT solver. In particular, we show that by focusing search on solving to optimality subformulas of the original WPMS instance we increase the efficiency of WPM2. From the experimental evaluation we conducted on the PMS and WPMS instances at the 2012 MaxSAT Evaluation, we can conclude that the new approach is both the best performing for industrial instances, and for the union of industrial and crafted instances.

1 Introduction

In the last decade Satisfiability (SAT) solvers have progressed dramatically in performance due to new algorithms, such as, conflict directed clause learning [36], and better implementation techniques. Thanks to these advances, nowadays the best SAT solvers can tackle hard decision problems. Our aim is to push this technology forward to deal with optimization problems.

The Maximum Satisfiability (MaxSAT) problem is the optimization version of SAT. The idea behind this formalism is that sometimes not all constraints of a problem can be satisfied, and we try to satisfy the maximum number of them. The MaxSAT problem can be further generalized to the Weighted Partial MaxSAT (WPMS) problem.

In the MaxSAT community, we find two main classes of algorithms: branch and bound [17, 22, 24, 26, 27] and SAT-based [2, 14, 19–21, 31–33]. The latter clearly dominate on industrial and some crafted instances, as we can see in the results of the last 2012 MaxSAT Evaluation. SAT-based MaxSAT algorithms basically reformulate a MaxSAT instance into a sequence of SAT instances. By solving these SAT instances the MaxSAT problem can be solved [6].

* This research has been partially founded by the CICYT research projects TASSAT (TIN2010-20967-C04-01/03/04) and ARINF (TIN2009-14704-C03-01).

In this paper we revisit the SAT-based MaxSAT algorithm WPM2 [5] which belongs to a family of algorithms that exploit the information from the unsatisfiable cores the underlying SAT solver provides. This algorithm is the natural extension to the weighted case of the Partial MaxSAT algorithm PM2 [3, 4]. In our experimental investigation the original WPM2 algorithm solves 796 out of 1474 from the whole benchmark of PMS and WPMS industrial and crafted instances at the 2012 MaxSAT Evaluation. We have extended WPM2 with several complementary improvements. First of all, we apply the stratification approach described in [2], what results in solving 74 additional instances. Secondly, we introduce a new criteria to decide when soft clauses can be hardened, that provides 66 additional solved instances. The hardening of soft clauses in MaxSAT SAT-based solvers has been previously studied in [2, 33]. Finally, our most effective contribution is to introduce a new strategy that focuses search on solving to optimality subformulas of the original MaxSAT instance. Actually, the new WPM2 algorithm is parametric on the approach we use to optimize these subformulas. This allows to combine the strength of exploiting the information extracted from unsatisfiable cores and other optimization approaches. By solving these smaller optimization problems we get the most significant boost in our new WPM2 algorithm. In particular, we experiment with three approaches: (i) refine the lower bound on these subformulas with the subsetsum function [5, 13], (ii) refine the upper bound with the strategy applied in minisat+ [15], SAT4J [10], qmaxsat [21] or ShinMaxSat [20], and (iii) a binary search scheme where the lower bound and upper bound are refined as in the previous approaches. The best performing approach in our experimental analysis is the second one and it allows to solve up to 238 additional instances. As a summary, the overall speed-up we achieved on the original WPM2 solver is about 378 additional solved instances, a 47% more.

As we mentioned, SAT-based MaxSAT algorithms reformulate a MaxSAT instances into a sequence of SAT instances. Obviously, it is important to use an efficient SAT solver. Also, most SAT-based MaxSAT algorithms require the addition of Pseudo-Boolean (PB) linear constraints as a result of the reformulation process. These PB constraints are used to bound the cost of the optimal assignment. Currently, in most state-of-the-art SAT-based MaxSAT solvers, PB constraints are translated into SAT. However, there is no known SAT encoding which can guarantee the original propagation power of the constraint, i.e, what we call arc-consistency, while keeping the translation low in size. The best approach so far, has a cubic complexity [8]. This can be a bottleneck for WPM2 [5] and also for other algorithms such as, BINCD [19] or SAT4J [10].

In order to treat PB constraints with specialized inference mechanisms and a moderate cost in size, while preserving the strength of SAT techniques for the rest of the formula, we use the Satisfiability Modulo Theories (SMT) technology [35]. Related work in this sense can be found in [34]. Also, in [1] a Weighted Constraint Satisfaction Problems (WCSP) solver implementing the original WPM1 [4] algorithm is presented.

An SMT instance is a generalization of a Boolean formula in which some propositional variables have been replaced by predicates with predefined interpretations from background theories such as, e.g., linear integer arithmetic. Most modern SMT solvers integrate a SAT solver with decision procedures (theory solvers) for sets of literals belonging to each theory. This way, we can hopefully get the best of both worlds: in particular, the efficiency of the SAT solver for the Boolean reasoning and the efficiency of special-purpose algorithms for the theory reasoning.

Another reasonable choice would be to use a PB solver, which can be seen as a particular case of an SMT solver specialized on the theory of PB constraints [28, 29]. However, if we also want to solve problems modeled with richer formalisms like WCSP, the SMT approach seems a better choice since we can take advantage of a wide range of theories [1].

In this work, we implemented both the last version of the WPM1 algorithm [2] and the revisited version of the WPM2 algorithm on top the of the SMT solver Yices. Then, we performed an extensive experimental evaluation comparing them with the best two solvers for PMS and WPMS categories at the 2012 MaxSAT Evaluation and with three additional solvers that did not take part but have been reported to exhibit good performance: *bincd*2, which is the new version of the BINCD algorithm [19] described in [33], with the best configuration reported by authors, *maxhs* from [14], which consists in an hybrid SAT and Integer Linear Programming (ILP) approach, and *ilp* which performs a translation of WPMS into ILP solved with IBM-CPLEX studio124 [7].

We observe that the implementation on SMT of our new WPM2 algorithm with the second approach for optimizing the subformulas is the best performing solver for both PMS and WPMS industrial instances. We also observe that it is the best performing for the union of PMS and WPMS industrial and crafted instances, what shows this is a robust approach. These results make us conjecture that by improving the interaction of our new WPM2 algorithm with diverse optimization techniques applied on the subformulas we can get additional speed-ups.

This paper proceeds as follows. Section 2 presents some preliminary concepts. Section 3 describes WPM2 [5] and the new improvements. Section 4 describes the SMT problem and discuss some implementation details of the SMT-based MaxSAT algorithms. Section 5 presents the experimental evaluation. Finally, Section 6 shows the conclusions and the future work.

2 Preliminaries

A *literal* is either a Boolean variable x or its negation $\overline{x}$. A *clause* C is a disjunction of literals. A *weighted clause* is a pair (C, w), where C is a clause and w is a natural number or infinity, indicating the penalty for falsifying the clause C. A *Weighted Partial MaxSAT formula* is a multiset of weighted clauses

$$\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$$

where the first m clauses are soft and the last m' clauses are hard. The set of variables occurring in a formula φ is noted as $\text{var}(\varphi)$.

A *(total) truth assignment* for a formula φ is a function $I : \text{var}(\varphi) \to \{0, 1\}$, that can be extended to literals, clauses and SAT formulas. For MaxSAT formulas is defined as $I(\{(C_1, w_1), \ldots, (C_m, w_m)\}) = \sum_{i=1}^{m} w_i \, (1 - I(C_i))$. The *optimal cost* of a formula is $\text{cost}(\varphi) = \min\{I(\varphi) \mid I : \text{var}(\varphi) \to \{0, 1\}\}$ and an *optimal assignment* is an assignment I such that $I(\varphi) = \text{cost}(\varphi)$.

The *Weighted Partial MaxSAT problem* for a Weighted Partial MaxSAT formula φ is the problem of finding an *optimal assignment.*

3 WPM2 Algorithm

The WPM2 algorithm [5] is described in Algorithm 1. The fragments in gray (lines 4, 10, 11, 13- 18 and 20) correspond to the new improvements we have incorporated.

In the WPM2 algorithm, we extend soft clauses C_i with a unique fresh auxiliary blocking variable b_i obtaining $\varphi_w = \{C_i \vee b_i\}_{i=1 \ldots m} \cup \{C_{m+i}\}_{i=1 \ldots m'}$. Notice that b_i will be set to true by a SAT solver on φ_w if C_i is false. We also work with a set AL of at-least PB constraints of the form $\sum_{i \in A} w_i \, b_i \geq k$ on the variables b_i, and a similar set AM of at-most constraints of the form $\sum_{i \in A} w_i \, b_i \leq k$, that are modified at every iteration of the algorithm.

Intuitively, the WPM2 algorithm refines at every iteration the lower bound on φ till it reaches the optimum $\mathit{cost}(\varphi)$. The AM constraints are used to bound the cost of the falsified clauses. The AL constraints are used to impose that subsets of soft clauses have a minimum cost and to compute the AM constraints, as we will see later. The algorithm ends when $\varphi_w \cup CNF(AL \cup AM)$ becomes satisfiable[1], where CNF is the translation to SAT of the PB constraints.

Technically speaking, the AL constraints give lower bounds on $\mathit{cost}(\varphi)$. The AM constraints enforce that all solutions of the set of constraints $AL \cup AM$ are the solutions of AL of minimal cost. This ensures that any solution of the formula sent to the solver, $\varphi_w \cup \mathit{CNF}(AL \cup AM)$, if there is any, is an optimal assignment of φ. Therefore, given a set of at-least constraints AL we compute a *corresponding* set of at-most constraints AM as follows. First, we need to introduce the notion of *core* and *cover.* A core is a set of indexes A such that $\sum_{i \in A} w_i \, b_i \geq k \in AL$. Function $\mathit{core}(\sum_{i \in A} w_i \, b_i \geq k)$ returns the core A and function $\mathit{cores}(AL)$ returns $\{\mathit{core}(al) \mid al \in AL\}$. Covers are defined from cores as follows.

Definition 1. *Given a set of cores L, we say that the set of indexes A is a* cover *of L, if it is a minimal non-empty set such that, for every $A' \in L$, if $A' \cap A \neq \emptyset$, then $A' \subseteq A$. Given a set of cores L, we denote the set of covers of L as* $\text{SC}(L)$.

[1] The AL constraints are redundant, i.e., not required to be sent to the SAT solver for the soundness of the algorithm but help to speed up the search.

Algorithm 1. Revisited WPM2 algorithm.

```
    Input: $\varphi = \{(C_1, w_1), \dots, (C_m, w_m), (C_{m+1}, \infty), \dots, (C_{m+m'}, \infty)\}$
1:  if $sat(\{C_i \in \varphi \mid w_i = \infty\}) = (UNSAT, \_, \_)$ then return $(\infty, \emptyset)$
2:  $\varphi_w := \{C_1 \vee b_1, \dots, C_m \vee b_m, C_{m+1}, \dots, C_{m+m'}\}$          ▷Extend all soft clauses
3:  $AL := \{w_1\, b_1 \geq 0, \dots, w_m\, b_m \geq 0\}$          ▷Set of at-least constraints
4:  $w_{max} := \infty$
5:  while $true$ do
6:      $AM := \emptyset$          ▷Set of at-most constraints
7:      foreach $(\sum_{i \in A} w_i\, b_i \geq k) \in AL$ do
8:          if $A \in SC(cores(AL))$ then
9:              $AM := AM \cup \{\sum_{i \in A} w_i\, b_i \leq k\}$
10:     $(st, \varphi_c, \mathcal{I}) := sat(\varphi_w \setminus \{C_i \vee b_i \mid (C_i, w_i) \in \varphi \wedge w_i < w_{max}\} \cup CNF(AL \cup AM))$
11:     if $st =$ SAT and $w_{max} = 0$ then return $(\mathcal{I}(\varphi), \mathcal{I})$
12:     else
13:         if $st =$ SAT then
14:             $W := \sum\{w_i \mid (C_i, w_i) \in \varphi \wedge w_i < w_{max}\}$
15:             $\varphi_h := harden(\varphi, AM, W)$
16:             $w_{max} := decrease(w_{max}, \varphi)$
17:         else
18:             $A := \{i \mid (C_i \vee b_i) \in (\varphi_c \setminus \varphi_h)\}$          ▷New core
19:             $A := \bigcup_{\substack{A' \in cores(AL) \\ A' \cap A \neq \emptyset}} A'$          ▷New cover
20:             $k := newbound(AL \cup \varphi_w, A)$
21:             $AL := \{al \in AL \mid core(al) \neq A\} \cup \{\sum_{i \in A} w_i\, b_i \geq k\}$
```

Given a set AL, the set AM is the set of at-most constrains $\sum_{i \in A} w_i\, b_i \leq k$ such that $A \in SC(cores(AL))$ and k is the solution of minimizing $\sum_{i \in A} w_i\, b_i$ subject to AL and $b_i \in \{0, 1\}$.

The algorithm starts with $AL = \{w_1\, b_1 \geq 0, \dots, w_m\, b_m \geq 0\}$ and the corresponding $AM := \{w_1\, b_1 \leq 0, \dots, w_m\, b_m \leq 0\}$ that ensures that the unique solution of $AL \cup AM$ is $b_1 = \cdots = b_m = 0$ with cost 0[2]. At every iteration, the algorithm calls a SAT solver with $\varphi_w \cup CNF(AL \cup AM)$. If it returns SAT, then the interpretation $\mathcal{I}$ is a MaxSAT solution of φ and we return the optimal cost $\mathcal{I}(\varphi)$. If it returns UNSAT, then we use the information of the unsatisfiable core φ_c obtained by the SAT solver to enlarge the set AL, excluding more interpretations on the b_i's that are not partial solutions of φ_w. Before calling again the SAT solver, we update AM conveniently, to ensure that solutions to the new constraints $AL \cup AM$ are still minimal solutions of the new AL constraint set. Notice that in every iteration the set of solutions of $\{b_1, \dots, b_m\}$ defined by AL is decreased, whereas the set of solutions of AM is increased.

[2] In the implementation, we do not add a blocking variable to a soft clause till it appears into a core.

One key point in WPM2 is to compute the $newbound(AL, A)$ (line 20) which corresponds to the following optimization problem:

$$\text{minimize} \sum_{i \in A} w_i \cdot b_i \quad \text{subject to } \{\sum_{i \in A} w_i \cdot b_i \geq k\} \cup AL \tag{1}$$

where $k = 1 + \sum\{k' \mid \sum_{i \in A'} w_i\, b_i \leq k' \in AM \wedge A' \subseteq A\}$.

Notice that by removing the AL constraints in (1), we get the subsetsum problem [13]. In the original WPM2 algorithm [5], the subsetsum problem is progressively solved until we get a solution that also satisfies the AL constraints. This satisfiability check in the original WPM2 is performed with a SAT solver.

In what follows, we present how we have modified the original WPM2 algorithm (fragments in gray in Algorithm 1) by incorporating several improvements: the application of a stratified approach, the hardening of soft clauses and the optimization of the subformulas defined by the covers.

3.1 Stratified Approach

As in [4] for WPM1, we apply a stratified approach. The stratified approach (lines 4, 10, 11 and 16) consists in sending to the SAT solver only those soft clauses with weight $w_i \geq w_{max}$. Then, when the SAT solver returns SAT, if there are still unsent clauses, we decrease w_{max} to include additional clauses to the formula. From [4], we also apply the diversity heuristic (line 16) which supplies us with an efficient method to calculate how we have to reduce the value of w_{max} in the stratified approach, so that, when there is a big variety of distinct weights, w_{max} decreases faster, and, when there is a low diversity, w_{max} is decreased to the following value of w_i. Similar approach with an alternative heuristic for grouping clauses can be found in [32].

3.2 Clause Hardening

The hardening of soft clauses in MaxSAT SAT-based solvers has been previously studied in [2, 11, 18, 23, 25, 30, 33]. Inspired by these works we study a hardening scheme for WPM2. While clause hardening was reported to have no positive effect in WPM1 [2], we will see that it boosts efficiency in WPM2.

The clause hardening (lines 14, 15 and 18) consists in considering hard those soft clauses whose satisfiability we know does not need to be reconsidered. We need some lemma ensuring that falsifying those soft clauses would lead us to suboptimal solutions. In the case of WPM1, all soft clauses satisfying $w_i > W$, where $W = \sum\{w_i \mid (C_i, w_i) \in \varphi \wedge w_i < w_{max}\}$ is the sum of weights of clauses not sent to the SAT solver, can be hardened. The correctness of this transformation is ensured by the following lemma:

Lemma 1 (Lemma 24 in [6])
Let $\varphi_1 = \{(C_1, w_1), \dots, (C_m, w_m), (C_{m+1}, \infty), \dots, (C_{m+m'}, \infty)\}$ be a MaxSAT formula with cost zero, let $\varphi_2 = \{(C'_1, w'_1), \dots, (C'_r, w'_r)\}$ be a MaxSAT formula without hard clauses and $W = \sum_{j=1}^{r} w'_j$. Let

$$\mathit{harden}(w) = \begin{cases} w & \mathit{if}\ w \leq W \\ \infty & \mathit{if}\ w > W \end{cases}$$

and $\varphi'_1 = \{(C_i, \mathit{harden}(w_i)) \mid (C_i, w_i) \in \varphi_1\}$*. Then,* $\mathrm{cost}(\varphi_1 \cup \varphi_2) = \mathrm{cost}(\varphi'_1 \cup \varphi_2)$*, and any optimal assignment for* $\varphi'_1 \cup \varphi_2$ *is an optimal assignment of* $\varphi_1 \cup \varphi_2$*.*

However, this lemma is not useful in the case of WPM2 because we do not proceed by transforming the formula, like in WPM1. Therefore, we generalize this lemma. For this, we need to introduce the notion of *optimal* of a formula.

Definition 2. *Given a MaxSAT formula* $\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$*, we say that* k *is a (possible) optimal of* φ *if there exists a subset* $A \subseteq \{1, \ldots, m\}$ *such that* $\sum_{i \in A} w_i = k$*.*

Notice that, for any interpretation I of the variables of φ, we have that $I(\varphi)$ is an optimal of φ. However, if k is an optimal, there does not exist necessarily an interpretation I satisfying $I(\varphi) = k$. Notice also that, given φ and k, finding the *next optimal*, i.e. finding the smallest $k' > k$ such that k' is an optimal of φ is equivalent to the subset sum problem.

Lemma 2. *Let* $\varphi_1 \cup \varphi_2$ *be a MaxSAT formula and* k_1 *and* k_2 *values such that:* $\mathrm{cost}(\varphi_1 \cup \varphi_2) = k_1 + k_2$ *and any assignment* I *satisfies* $I(\varphi_1) \geq k_1$ *and* $I(\varphi_2) \geq k_2$*. Let* k' *be the smallest possible optimal of* φ_2 *such that* $k' > k_2$*. Let* φ_3 *be a set of soft clauses with* $W = \sum\{w_i \mid (C_i, w_i) \in \varphi_3\}$*.*

Then, if $W < k' - k_2$*, then any optimal assignment* I' *of* $\varphi_1 \cup \varphi_2 \cup \varphi_3$ *assigns* $I'(\varphi_2) = k_2$

Proof. Let I' be any optimal assignment of $\varphi_1 \cup \varphi_2 \cup \varphi_3$. On the one hand, as for any other assignment, we have $I'(\varphi_2) \geq k_2$.

On the other hand, any of the optimal assignments I of $\varphi_1 \cup \varphi_2$ can be extended (does not matter how) to the variables of $\mathrm{var}(\varphi_3) \setminus \mathrm{var}(\varphi_1 \cup \varphi_2)$, such that

$$I(\varphi_1 \cup \varphi_2 \cup \varphi_3) = I(\varphi_1) + I(\varphi_2) + I(\varphi_3) \leq k_1 + k_2 + W < k_1 + k' \tag{2}$$

Now, assume that $I'(\varphi_2) \neq k_2$, then $I'(\varphi_2) \geq k'$. As any other assignment, $I'(\varphi_1) \geq k_1$. Hence, $I'(\varphi_1 \cup \varphi_2 \cup \varphi_3) \geq k_1 + k' > I(\varphi_1 \cup \varphi_2 \cup \varphi_3)$, but this contradicts the optimality of I'. Therefore, $I'(\varphi_2) = k_2$.

□

In order to apply this lemma we have to consider partitions of the formula $\varphi_1 \cup \varphi_2$ ensuring $\mathrm{cost}(\varphi_1 \cup \varphi_2) = k_1 + k_2$ and $I(\varphi_1) \geq k_1$ and $I(\varphi_2) \geq k_2$, for any assignment I. This can be easily ensured, in the case of WPM2, if both φ_1 and φ_2 are unions of covers. Then, we only have to check if the next possible optimal k' of φ_2 exceeds the previous one k_2 more than the sum W of the weights of the clauses not sent to the SAT solver. In such a case, we can consider all soft clauses of φ_2 and their corresponding AM constraint with k_2 as hard clauses. In other words, we do not need to recompute the partial optimal k_2 of φ_2.

Finally, in line 15 of Algorithm 1, function $harden(\varphi, AM, W)$ returns the set of soft clauses φ_h that needs to be considered hard based on the previous analysis according to: the current set of covers AM, the next optimals of these covers and the sum of the weights W of soft clauses beyond the current w_{max}, i.e., not yet sent to the SAT solver.

3.3 Cover Optimization

As we have mentioned earlier, one key point in WPM2 is how to compute the $newbound(AL, A)$ (line 20). Actually, we can solve to optimality the subformulas defined by the union of the soft clauses related to the cover A and the hard clauses.

Definition 3. *Given a MaxSAT formula* $\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$ *and a set of indexes A, we define the subformula, $\varphi[A]$, as follows:* $\varphi[A] = \{(C_i, w_i) \in \varphi \mid i \in A \vee w_i = \infty)\}$

Solving to optimality $\varphi[A]$ give us the optimal value $k = cost(\varphi[A])$ for the AM constraint related to cover A. In order to do this, while taking advantage of the AL constraints generated so far, we only have to extend the minimization problem corresponding to the *newbound* (1) function, by adding φ_w to the constraints, i.e, $newbound(AL \cup \varphi_w, A)$[3]. Notice that $newbound(AL \cup \varphi_w, A) \geq newbound(AL, A)$.

In order to optimize $\varphi[A]$, we can use any exact approach related to MaxSAT, such as, MaxSAT branch and bound algorithms, MaxSAT SAT-based algorithms, saturation under the MaxSAT resolution rule, or we can use other solving techniques such as PB solvers or ILP techniques, etc. Our new WPM2 algorithm is parametric on any suitable optimization solving approach. In this work, we present three approaches.

The first and natural approach consists in iteratively refining (increasing) the lower bound on the optimal k for $\varphi[A]$ by applying the subsetsum function as in the original WPM2. The procedure stops when we satisfy the constraints $AL \cup \varphi_w$. Notice that since we have included φ_w into the set of constraints, the solution we will eventually get has to be optimal for $\varphi[A]$.

The second approach consists in iteratively refining (decreasing) the upper bound following the strategy applied in minisat+ [15], SAT4J [10], qmaxsat [21] or ShinMaxSat [20]. The upper bound ub is initially set to the sum of the weights w_i of the soft clauses in $\varphi[A]$. Then, we iteratively test whether $k = ub - 1$ is feasible or not. Whenever we get a satisfying assignment, we update ub to the sum of the weights w_i of those soft clauses where b_i evaluates to true under the satisfying assignment. If we get an unsatisfiable answer, the previous ub is the optimal value for $\varphi[A]$.

The third approach applies a binary search scheme [12, 16, 19]. We additionally refine the lower bound as in our first approach and the upper bound as in the second approach.

[3] We can actually exclude from φ_w all the soft clauses not in $\varphi[A]$.

The worst case complexity, in terms of the number of calls to the SAT solver, of the new WPM2 algorithm is the number of times that the newbound function is called (bounded by the number of clauses) multiplied by the number of SAT calls needed in each call to the newbound function. This latter number is logarithmic on the sum of the weights of the clauses of the core if we use a binary search, hence essentially the number of clauses. Therefore, the worst case complexity, when using a binary search to solve to optimality the subformulas, is quadratic on the number of soft clauses.

In order to see that the number of calls to the newbound function is bounded by the number of clauses we just need to recall that WPM2 merges the covers. Consider a binary tree where the soft clauses are the leaves, and the internal nodes represent the merges (calls to the newbound function). A binary tree of n leaves has n-1 internal nodes.

Solving to optimality all the covers can be very costly since these are NP-hard problems. Depending on the unsatisfiable cores we get in the general loop of the WPM2 algorithm some covers have to be merged. Therefore, we may argue that part of the work we did in order to optimize these covers can be useless[4]. For example, a reasonable strategy is to optimize the current cover only if it was not the result of merging other covers, i.e., when the last unsatisfiable core is contained into a cover. In the experimental evaluation, we will see that although the number of solved instances does not vary too much, the mean time for solving some families can be decreased.

4 Engineering Efficient SMT-Based MaxSAT Solvers

We have implemented both the last version of the WPM1 algorithm [2] and the revisited version of the WPM2 algorithm on top the of the SMT solver Yices.

As we have said, an SMT instance is a generalization of SAT where some propositional variables are replaced by predicates with predefined interpretations from background theories. Among the theories considered in the SMT library [9] we are interested in QF_LIA (Quantifier-Free *Linear Integer Arithmetic*). With the QF_LIA theory we can model the PB constraints that SAT-based MaxSAT algorithms generate during their execution. Therefore, for the SMT-based MaxSAT algorithm, we just need to replace the conversion to CNF (line 10 in Algorithm 1) by the proper linear integer arithmetic predicates.

As suggested in [16, 31], we can preserve some learned lemmas from previous iterations that may help to reduce the search space. In order to do that, we execute the SMT solver in incremental mode. Within this mode, we can call the solve routine and add new clauses (assertions) on demand, while preserving learned lemmas. However, notice that our algorithms delete parts of the formula between iterations. For example, in lines 7 to 9 of Algorithm 1 we recompute the set AM, possibly erasing some of the at-most constraints. Therefore, we have to take care also of any learned lemma depending on them.

[4] The related *AL* constraints can still be kept.

The SMT solver Yices gives the option of marking assertions as *retractable*. If the SMT solver does not support the deletion of assertions but supports the usage of assumptions, we can replace every retractable assertion C, with $a \rightarrow C$, where a is an assumption. Before each call, we activate the assumptions of assertions that have not been retracted by the algorithm. Notice that assertions that do have been retracted will have a pure literal ($\overline{a}$) such that a has not been activated. Therefore, the solver can safely set to false a deactivating the clause. Moreover, any learned lemma on those assertions will also include $\overline{a}$. For example, Z3 and Mathsat SMT solvers do not allow to delete clauses, but they allow the use of assumptions.

5 Experimental Results

In this section we present an intensive experimental investigation on the PMS and WPMS industrial and crafted instances from the 2012 MaxSAT Evaluation. We provide results for our new WPM2 SMT-based MaxSAT solver, for a WPM1 [2] SMT-based MaxSAT solver, the best two solvers for each category of the 2012 MaxSAT Evaluation, and three solvers which did not participate but the authors have reported to exhibit good performance. We run our experiments on a cluster featured with 2.27 GHz processors, memory limit of 3.9 GB and a timeout of 7200 seconds per instance.

The experimental results are presented in Tables 1 and 2 following the same classification criteria as in the 2012 MaxSAT Evaluation. For each solver and family of instances, we present the number of solved instances in parenthesis and the mean solving time. Solvers are ordered from left to right according to the total number of solved instances. The results for the best performing solver in each family are presented in bold. The number of instances of every family is specified in the column under the sign '#'. Since different families may have different number of instances, we also include for each solver the mean ratio of solved instances.

Our new WPM2 algorithm is implemented on top of the Yices SMT solver (version 1.0.29). The different versions of WPM2 and corresponding implementations are named $wpm2$ where subindexes can be $_s$ that stands for stratified approach with diversity heuristic and $_h$ for hardening. Regarding to how we perform the cover optimization, $_l$ stands for lower bound refinement based on subsetsum, $_u$ for upper bound refinement based on satisfying truth assignment, and $_b$ for binary search. Finally, $_a$ stands for optimizing all the covers and $_c$ for optimizing only covers that contain the last unsatisfiable core.

Table 1 shows our first experiment, where we evaluate the impact of each variation on the original $wpm2$. By using a stratified approach with the diversity heuristic ($wpm2_s$) we solve some additional instances in all categories having the best improvement in WPMS crafted. Overall, we solve 74 more instances. By adding hardening ($wpm2_{sh}$) we solve 66 more instances, mainly in WPMS industrial family *haplotyping-pedigrees*.

Table 1. Experimental results of different versions of *wpm*2

Instance set	#	$wpm2$	$wpm2_{s}$	$wpm2_{sh}$	$wpm2_{shlc}$	$wpm2_{shla}$	$wpm2_{shbc}$	$wpm2_{shba}$	$wpm2_{shuc}$	$wpm2_{shua}$
PMS-Industrial										
aes	7	0.00(0)	0.00(0)	0.00(0)	0.00(0)	0.00(0)	0.00(0)	**10.34(1)**	1836.54(1)	514.14(1)
bcp-fir	59	83.29(57)	23.44(57)	23.44(57)	40.87(57)	104.75(57)	113.13(57)	146.03(57)	60.91(58)	**57.36(58)**
bcp-hipp-yRa1_simp	17	504.81(13)	155.95(12)	155.95(12)	95.16(12)	239.17(13)	151.30(13)	813.36(16)	**107.24(16)**	160.55(16)
bcp-hipp-yRa1_su	38	380.47(19)	164.70(16)	164.70(16)	181.49(18)	667.59(19)	226.66(25)	585.82(28)	555.92(33)	**315.87(34)**
bcp-msp	64	821.88(25)	756.59(26)	756.59(26)	283.50(28)	606.35(28)	283.78(31)	464.47(30)	**711.08(36)**	912.70(34)
bcp-mtg	40	1363.97(18)	852.58(23)	852.58(23)	1320.80(28)	786.09(34)	1296.44(32)	940.62(37)	997.38(35)	**578.75(39)**
bcp-syn	74	60.29(41)	302.54(41)	302.54(41)	296.12(41)	83.15(39)	242.78(42)	69.23(42)	**103.21(43)**	78.22(42)
circuit-trace-compaction	4	285.74(3)	835.72(4)	835.72(4)	230.85(3)	134.79(4)	145.72(4)	151.41(4)	**118.24(4)**	129.10(4)
haplotype-assembly	6	**2.87(5)**	7.17(5)	7.17(5)	9.80(5)	42.86(5)	15.53(5)	51.44(5)	18.28(4)	65.94(5)
pbo-mqc_nencdr	84	866.38(84)	821.16(84)	821.16(84)	142.43(84)	**107.70(84)**	130.90(84)	127.99(84)	245.77(84)	257.06(84)
pbo-mqc_nlogencdr	84	362.20(84)	353.14(84)	353.14(84)	24.48(84)	**17.19(84)**	58.42(84)	65.79(84)	124.43(84)	140.44(84)
pbo-routing	15	**0.46(15)**	2.14(15)	2.14(15)	3.89(15)	6.59(15)	4.88(15)	6.72(15)	5.42(15)	6.73(15)
protein-ins	12	2626.34(10)	2162.13(9)	2162.13(9)	476.03(12)	552.46(12)	360.76(12)	284.63(12)	**234.31(12)**	333.85(12)
Total	504	374	376	376	387	394	404	415	425	**428**
		69.6%	70.9%	70.9%	72.5%	76.0%	77.5%	81.4%	81.7%	**83.6%**
WPMS-Industrial										
haplotyping-pedigrees	100	16.01(22)	292.29(25)	242.08(90)	154.12(92)	142.59(92)	199.61(96)	86.67(95)	202.70(98)	**176.02(98)**
timetabling	26	1017.27(8)	720.89(8)	651.84(8)	1008.59(9)	**430.67(9)**	1544.43(9)	931.17(8)	932.77(8)	1438.59(9)
upgradeability-problem	100	19.19(100)	19.67(100)	**15.27(100)**	96.88(100)	375.70(100)	99.51(100)	365.70(100)	97.83(100)	371.96(100)
Total	226	130	133	198	201	201	205	203	206	**207**
		50.9%	51.9%	73.6%	75.5%	75.5%	76.9%	75.3%	76.3%	**77.5%**
Total Industrial	730	504	509	574	588	595	609	618	631	**635**
		66.1%	67.3%	71.4%	73.0%	75.9%	77.4%	80.2%	80.7%	**82.5%**
PMS-Crafted										
frb	25	0.00(0)	0.00(0)	0.00(0)	0.00(0)	0.00(0)	0.00(0)	0.00(0)	0.00(0)	0.00(0)
job-shop	3	68.31(3)	63.37(3)	63.37(3)	50.84(3)	53.11(3)	51.29(3)	**49.87(3)**	88.41(3)	58.75(3)
maxclique_random	96	478.65(76)	471.97(76)	471.97(76)	435.07(77)	533.97(78)	423.89(83)	392.04(82)	565.44(87)	**512.17(89)**
maxclique_structured	62	779.71(21)	592.07(21)	592.07(21)	511.52(22)	783.37(24)	650.87(25)	838.38(26)	802.09(26)	**876.90(27)**
maxone_3sat	80	16.42(80)	18.11(80)	18.11(80)	10.90(80)	9.39(80)	5.34(80)	**5.29(80)**	6.65(80)	6.47(80)
maxone_structured	60	156.20(58)	89.15(59)	89.15(59)	14.42(60)	32.46(60)	13.11(60)	46.68(60)	**9.24(60)**	50.43(60)
min-enc_kbtree	42	1607.47(4)	707.46(4)	707.46(4)	2057.16(5)	2086.38(5)	2254.64(6)	1777.41(5)	**921.09(6)**	1906.23(5)
pseudo_miplib	4	122.23(4)	93.95(4)	93.95(4)	109.21(4)	64.35(4)	**27.35(4)**	37.81(4)	54.30(4)	34.23(4)
Total	372	246	247	247	251	254	261	260	266	**268**
		64.9%	65.1%	65.1%	65.9%	66.5%	67.6%	67.4%	68.4%	**68.5%**
WPMS-Crafted										
auc-paths	86	0.00(0)	376.67(1)	370.25(1)	706.38(33)	546.57(33)	1019.93(74)	948.64(75)	435.71(82)	**271.16(82)**
auc-scheduling	84	0.00(0)	133.39(51)	136.89(51)	71.37(84)	69.59(84)	2.73(84)	1.66(84)	1.25(84)	**1.07(84)**
min-enc-planning	56	491.39(25)	141.80(38)	138.29(38)	3.24(56)	2.47(56)	0.75(56)	0.77(56)	**0.74(56)**	0.80(56)
min-enc-warehouses	18	11.36(1)	3.36(1)	3.43(1)	0.07(1)	0.08(1)	0.04(1)	0.05(1)	**1227.86(2)**	0.05(1)
pseudo-miplib	12	2936.02(3)	1533.63(3)	1487.40(3)	13.72(3)	11.81(3)	471.93(4)	990.65(5)	299.53(4)	**695.31(5)**
random-net	74	0.00(0)	0.00(0)	0.00(0)	1041.56(15)	1943.41(17)	1633.69(33)	**1529.94(33)**	3369.16(17)	3151.03(15)
wcsp-spot5-dir	21	187.40(9)	310.45(10)	248.53(12)	420.10(14)	428.10(14)	495.17(14)	470.92(14)	**8.27(14)**	113.90(14)
wcsp-spot5-log	21	1067.69(8)	469.38(10)	120.60(9)	100.07(13)	59.16(13)	133.75(14)	48.91(13)	23.84(14)	**16.50(14)**
Total	372	46	114	115	219	221	280	**281**	273	271
		19.5%	31.9%	32.5%	52.2%	52.6%	62.9%	**63.4%**	62.0%	62.0%
Total Crafted	744	292	361	362	470	475	541	**541**	539	539
		42.2%	48.5%	48.8%	59.1%	59.5%	65.2%	**65.4%**	65.2%	65.3%
Total (W)PMS	1474	796	870	936	1058	1070	1150	1159	1170	**1174**
		54.1%	57.9%	60.1%	66.1%	67.7%	71.3%	72.8%	72.9%	**73.9%**

Table 2. Experimental results of best *wpm*2 version compared with other solvers

(a) Partial Industrial

Instance set	#	$wpm2_{shua}$	*bincd*2	*qms*0.21*g*2	*pwbo*2.1	*shinms*	*wpm*1	*ilp*
aes	7	514.14(1)	453.22(1)	3154.99(1)	0.00(0)	0.00(0)	3073.19(1)	**1310.95(3)**
bcp-fir	59	57.36(58)	44.09(58)	108.17(56)	68.10(56)	13.54(22)	10.87(57)	**62.86(59)**
bcp-hipp-yRa1_simp	17	160.55(16)	170.49(16)	**358.14(17)**	174.98(15)	40.66(16)	70.27(16)	666.94(6)
bcp-hipp-yRa1_su	38	315.87(34)	244.97(32)	**105.60(35)**	97.91(25)	282.26(34)	244.23(28)	0.00(0)
bcp-msp	64	912.79(34)	**213.47(38)**	451.50(30)	96.14(26)	281.37(22)	1053.47(7)	855.96(37)
bcp-mtg	40	578.75(39)	1.15(40)	**0.15(40)**	0.57(40)	0.60(40)	8.54(40)	769.27(29)
bcp-syn	74	78.22(42)	28.56(43)	283.64(35)	21.82(39)	86.98(33)	59.30(45)	**18.95(71)**
circuit-trace-compaction	4	129.10(4)	109.31(4)	**45.01(4)**	200.11(2)	52.22(4)	118.43(4)	6921.80(1)
haplotype-assembly	6	65.94(5)	728.30(5)	153.22(5)	9.09(5)	0.00(0)	**2.63(5)**	2124.72(5)
pbo-mqc_nencdr	84	257.06(84)	278.45(84)	**58.78(84)**	222.19(68)	145.71(84)	804.25(54)	1109.89(6)
pbo-mqc_nlogencdr	84	140.44(84)	78.58(84)	**23.69(84)**	71.86(82)	180.37(79)	403.38(55)	508.21(6)
pbo-routing	15	6.73(15)	**1.14(15)**	3.61(15)	27.67(15)	4.80(15)	1.75(15)	19.68(15)
protein-ins	12	333.85(12)	314.09(3)	**128.58(12)**	0.11(1)	206.51(4)	1812.03(3)	2.72(1)
Total	504	**428**	423	418	374	353	330	239
Mean ratio		**83.6%**	78.2%	83.0%	66.3%	63.6%	68.3%	48.9%

(b) Weighted Partial Industrial

Instance set	#	$wpm2_{shua}$	*wpm*1	*pwbo*2.1	*bincd*2	*maxhs*	*ilp*	*shinms*
haplotyping-pedigrees	100	**176.02(98)**	212.76(93)	123.00(87)	544.80(73)	1089.24(39)	1892.16(18)	1203.99(47)
timetabling	26	1438.59(9)	**1347.39(11)**	671.45(7)	168.55(8)	1249.85(6)	0.00(0)	2261.00(5)
upgradeability-problem	100	371.96((100)	**4.57(100)**	32.67(100)	76.40(100)	13.41(100)	19.26(100)	0.00(0)
Total	226	**207**	204	194	181	145	118	52
Mean ratio		77.5%	**78.4%**	71.3%	67.9%	54.0%	39.3%	22.1%

(c) Partial Crafted

Instance set	#	*ilp*	*akms_ls*	*qms*0.21	*shinms*	$wpm2_{shua}$	*bincd*2	*pwbo*2.1	*wpm*1
frb	25	1152.97(13)	159.47(5)	**346.73(25)**	43.52(23)	0.00(0)	0.00(0)	151.16(15)	0.00(0)
job-shop	3	0.00(0)	0.00(0)	**41.51(3)**	36.44(3)	58.75(3)	100.43(3)	93.17(1)	835.23(3)
maxclique_random	96	45.13(96)	**1.09(96)**	269.93(83)	339.30(76)	512.17(89)	85.29(71)	79.45(64)	652.31(59)
maxclique_structured	62	326.51(38)	**281.51(41)**	800.18(30)	401.45(23)	876.90(27)	109.21(21)	37.07(19)	153.39(13)
maxone_3sat	80	13.12(80)	**0.47(80)**	198.39(80)	694.52(78)	6.47(80)	8.28(80)	36.97(63)	97.24(80)
maxone_structured	60	337.60(59)	482.29(38)	**6.35(60)**	3.53(59)	50.43(60)	57.42(60)	7.71(60)	718.41(42)
min-enc_kbtree	42	**162.89(42)**	3199.18(34)	248.02(6)	513.97(5)	1906.23(5)	274.79(6)	306.58(2)	2469.35(6)
pseudo_miplib	4	34.12(4)	258.91(3)	**1.84(4)**	3.44(4)	34.23(4)	48.74(4)	93.19(4)	163.61(4)
Total	372	**332**	297	291	271	268	245	228	207
Mean ratio		76.5%	63.2%	**81.1%**	77.0%	68.5%	65.3%	59.3%	58.3%

(d) Weighted Partial Crafted

Instance set	#	*ilp*	*wpm*1	$wpm2_{shua}$	*shinms*	*akms_ls*	*pwbo*2.1	*maxhs*	*bincd*2
auc-paths	86	**0.49(86)**	108.14(63)	271.16(82)	317.62(84)	2.57(86)	110.67(19)	35.41(86)	1414.73(12)
auc-scheduling	84	**0.38(84)**	1.56(84)	1.07(84)	5.81(84)	68.27(84)	7.65(81)	965.10(78)	141.77(81)
min-enc-planning	56	296.52(56)	2.61(53)	0.80(56)	8.15(52)	141.21(40)	**0.46(56)**	459.18(31)	32.74(54)
min-enc-warehouses	18	**0.49(18)**	78.89(17)	0.05(1)	0.43(1)	20.11(2)	3.78(14)	0.18(1)	2.10(1)
pseudo-miplib	12	82.82(3)	157.33(4)	695.31(5)	**127.99(5)**	0.26(2)	3.97(3)	0.03(1)	1072.69(4)
random-net	74	532.74(59)	**4.03(70)**	3151.03(15)	0.00(0)	4060.60(8)	42.15(35)	2770.76(10)	0.00(0)
wcsp-spot5-dir	21	42.88(18)	37.70(13)	113.90(14)	**743.63(21)**	1555.46(6)	61.89(8)	101.11(6)	127.73(12)
wcsp-spot5-log	21	322.93(8)	399.59(14)	16.50(14)	**200.73(17)**	108.70(5)	1.71(6)	357.32(6)	299.32(13)
Total	372	**332**	318	271	264	233	222	219	177
Mean ratio		**78.6%**	77.4%	62.0%	64.8%	45.3%	54.4%	41.6%	45.6%

Table 3. Summary of solved instances and mean ratio % for best solvers

solvers	pms	wpms	Ind.	pms	wpms	Cra.	Total
	428	**207**	**635**	268	271	539	**1174**
*wpm*2*shua*	**83.6 %**	77.5 %	**82.5 %**	68.5 %	62.0 %	65.3 %	**73.9 %**
	330	204	534	207	318	525	1059
*wpm*1	68.3 %	**78.4 %**	70.2 %	58.3 %	77.4 %	67.9 %	69.0 %
	423	181	604	245	177	422	1026
*bincd*2	78.2 %	67.9 %	76.3 %	65.3 %	45.6 %	55.5 %	65.9 %
	239	118	357	**332**	**332**	**664**	1021
ilp	48.9 %	39.3 %	47.1 %	76.5 %	**78.6 %**	**77.6 %**	62.3 %
	374	194	568	228	222	450	1018
*pwbo*2.1	66.3 %	71.3 %	67.2 %	59.3 %	54.5 %	56.9 %	62.1 %
	353	52	405	271	264	535	940
shinms	63.6 %	22.1 %	55.8 %	77.0 %	64.8 %	70.9 %	63.4 %
	418			291			
qms	83.0 %			**81.1 %**			

Regarding our three approaches for optimizing the covers, we can see that by optimizing with subsetsum ($wpm2_{shla}$) we solve some additional instances in all categories having the best improvement in WPMS industrial with 18 more and in WPMS crafted with 106 more. It is important to highlight that optimizing covers with subsetsum, instead of applying the subsetsum as in the original WPM2 algorithm, leads to a total improvement of 134 additional solved instances, with respect to $wpm2_{sh}$.

Optimizing all covers by refining the upper bound ($wpm2_{shua}$), we get an additional boost with respect to $wpm2_{shla}$. We can see that we solve some additional instances in all categories. We get the best improvement for PMS industrial, solving 34 additional instances, and for WPMS crafted, 50 more. Notice that the overall increase with respect to $wpm2_{sh}$ is of 238 additional solved instances.

Binary search ($wpm2_{shba}$) improves 10 instances in WPMS crafted with respect to $wpm2_{shua}$. But the global performance with respect to $wpm2_{sh}$, 223, is not as good as only refining the upper bound ($wpm2_{shua}$).

Optimizing only covers that contain the last unsatisfiable core solves almost the same instances as optimizing all covers but improves the average running time in the WPMS industrial family *upgradeability-problem* by a factor of 4.

Table 2 shows the results of our second experiment where we compare the best variation and implementation of our new WPM2 algorithm ($wpm2_{shua}$) with several solvers. In particular, we compare with the best two solvers for the PMS and WPMS industrial and crafted instances of the 2012 MaxSAT Evaluation: PMS industrial (*qms*0.21*g*2, *pwbo*2.1), WPMS industrial (*pwbo*2.1 [31, 32], *wpm*1 [2][5]), PMS crafted (*qms*0.21 [21], *akms_ls* [22] and WPMS crafted (*wpm*1, *shinms* [20]). We also compare with three additional MaxSAT solvers: *bincd*2, which is the new version of the BINCD algorithm [19] described in [33], with the best configuration reported by authors, *maxhs* from [14], which consists in an hybrid SAT-ILP approach, and *ilp*, which translates WPMS into ILP and applies the MIP solver IBM-CPLEX studio124 [7].

[5] We present in this paper a version implemented on top of the Yices SMT solver.

Table 2(a) presents the results for the PMS industrial instances. Our $wpm2_{shua}$ is the first one in solved instances with 428 and mean ratio with 93.6%, closely followed by *bincd*2 and *qms*0.21*g*2.

Table 2(b) presents the results for the WPMS industrial instances. As we can see, our $wpm2_{shua}$ and *wpm*1 dominate this category with 207 and 204 solved instances and 77.5% and 78.4% mean ratio, resp.

As a summary of industrial instances, we can conclude that our $wpm2_{shua}$ is the best performing solver with a total of 635 solved instances, followed by *bincd*2 with a total of 604. We do not have results for any version of *qms* since it only works for PMS instances. The closest solver to the search scheme of *qms* would be *shinms* but it does not perform well for WPMS industrial.

Table 2(c) presents the results for the PMS crafted instances. The *ilp* approach solves 332 of 372 instances, 35 more than *akms_ls*. This is remarkable since branch and bound solvers, like *akms_ls*, have always dominated this category since 2006. PMS solver *qms*0.21 is the third in solved instances but the first in mean ratio with 81.1%. Our $wpm2_{shua}$ is the fifth in solved instances with 268 and the fourth in mean ratio with 68.5%.

Table 2(d) presents the results for the WPMS crafted instances. Again, the *ilp* approach is the best one, solving 332 of 372 instances, 14 more than the second one, *wpm*1. Our $wpm2_{shua}$ is the third in solved instances with 271 and the fourth in mean ratio with 62.0%.

As a summary of crafted instances, we can conclude that *ilp* is the best performing approach, and our $wpm2_{shua}$ is the second in total solved instances.

In Table 3 we can see a summary of the solved instances and mean ratio per category for best solvers. We recall that all solvers accept weights except *qms* that is only for PMS. Our $wpm2_{shua}$ is the first in solved instances for both PMS industrial and WPMS industrial. In crafted categories it is the second in total solved instances. However, for both PMS crafted and WPMS crafted categories *ilp* is the first in solved instances. We can conclude that our $wpm2_{shau}$ is the most robust solver across all four PMS and WPMS industrial and crafted categories, followed by *wpm*1 and *bincd*2.

6 Conclusions and Future Work

From the experimental evaluation, we conclude that the new WPM2 solver is the best performing solver for PMS and WPMS industrial instances and the best on the union of PMS and WPMS industrial and crafted instances. In particular, we have shown that solving to optimality the subformulas defined by covers really works in practice. As future work, we will study how to improve the interaction with the optimization of the subformulas. A portfolio that selects the most suitable optimization approach depending on the structure of the subformula seems another way of achieving additional speed-ups. Finally, we have also shown that SMT technology is an underlying efficient technology for solving the MaxSAT problem.

References

1. Ansótegui, C., Bofill, M., Palahí, M., Suy, J., Villaret, M.: A Proposal for Solving Weighted CSPs with SMT. In: Proceedings of the 10th International Workshop on Constraint Modelling and Reformulation (ModRef 2011), pp. 5–19 (2011)
2. Ansótegui, C., Bonet, M.L., Gabàs, J., Levy, J.: Improving sat-based weighted maxsat solvers. In: Milano, M. (ed.) CP 2012. LNCS, vol. 7514, pp. 86–101. Springer, Heidelberg (2012)
3. Ansótegui, C., Bonet, M.L., Levy, J.: On solving MaxSAT through SAT. In: Proc. of the 12th Int. Conf. of the Catalan Association for Artificial Intelligence (CCIA 2009), pp. 284–292 (2009)
4. 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)
5. Ansotegui, C., Bonet, M.L., Levy, J.: A new algorithm for weighted partial maxsat. In: Proc. the 24th National Conference on Artificial Intelligence (AAAI 2010) (2010)
6. Ansótegui, C., Bonet, M.L., Levy, J.: Sat-based maxsat algorithms. Artif. Intell. 196, 77–105 (2013)
7. Ansotegui, C., Gabas, J.: Solving maxsat with mip. In: CPAIOR (2013)
8. Bailleux, O., Boufkhad, Y., Roussel, O.: New encodings of pseudo-boolean constraints into CNF. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 181–194. Springer, Heidelberg (2009)
9. Barrett, C., Stump, A., Tinelli, C.: The Satisfiability Modulo Theories Library (SMT-LIB) (2010), `http://www.SMT-LIB.org`
10. Berre, D.L.: Sat4j, a satisfiability library for java (2006), `http://www.sat4j.org`
11. Borchers, B., Furman, J.: A two-phase exact algorithm for max-sat and weighted max-sat problems. J. Comb. Optim. 2(4), 299–306 (1998)
12. Cimatti, A., Franzén, A., Griggio, A., Sebastiani, R., Stenico, C.: Satisfiability modulo the theory of costs: Foundations and applications. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 99–113. Springer, Heidelberg (2010)
13. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. MIT Press (2009)
14. 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)
15. Eén, N., Sörensson, N.: Translating pseudo-boolean constraints into SAT. JSAT 2(1-4), 1–26 (2006)
16. 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)
17. Heras, F., Larrosa, J., Oliveras, A.: MiniMaxSat: A new weighted Max-SAT solver. In: Marques-Silva, J., Sakallah, K.A. (eds.) SAT 2007. LNCS, vol. 4501, pp. 41–55. Springer, Heidelberg (2007)
18. Heras, F., Larrosa, J., Oliveras, A.: Minimaxsat: An efficient weighted max-sat solver. J. Artif. Intell. Res (JAIR) 31, 1–32 (2008)
19. Heras, F., Morgado, A., Marques-Silva, J.: Core-guided binary search algorithms for maximum satisfiability. In: Proc. the 25th National Conference on Artificial Intelligence (AAAI 2011) (2011)

20. Honjyo, K., Tanjo, T.: Shinmaxsat, a Weighted Partial Max-SAT solver inspired by MiniSat+, Information Science and Technology Center, Kobe University
21. Koshimura, M., Zhang, T., Fujita, H., Hasegawa, R.: Qmaxsat: A partial max-sat solver. JSAT 8(1/2), 95–100 (2012)
22. Kügel, A.: Improved exact solver for the weighted max-sat problem (to appear)
23. Larrosa, J., Heras, F., de Givry, S.: A logical approach to efficient max-sat solving. Artif. Intell. 172(2-3), 204–233 (2008)
24. Li, C.M., Manyà, F., Mohamedou, N., Planes, J.: Exploiting cycle structures in Max-SAT. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 467–480. Springer, Heidelberg (2009)
25. Li, C.M., Manyà, F., Planes, J.: New inference rules for Max-SAT. J. Artif. Intell. Res (JAIR) 30, 321–359 (2007)
26. Lin, H., Su, K.: Exploiting inference rules to compute lower bounds for Max-SAT solving. In: IJCAI 2007, pp. 2334–2339 (2007)
27. Lin, H., Su, K., Li, C.M.: Within-problem learning for efficient lower bound computation in Max-SAT solving. In: Proc. the 23rd National Conference on Artificial Intelligence (AAAI 2008), pp. 351–356 (2008)
28. 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)
29. Manquinho, V.M., Martins, R., Lynce, I.: Improving unsatisfiability-based algorithms for boolean optimization. In: Strichman, O., Szeider, S. (eds.) SAT 2010. LNCS, vol. 6175, pp. 181–193. Springer, Heidelberg (2010)
30. Marques-Silva, J., Argelich, J., Graça, A., Lynce, I.: Boolean lexicographic optimization: algorithms & applications. Ann. Math. Artif. Intell. 62(3-4), 317–343 (2011)
31. Martins, R., Manquinho, V.M., Lynce, I.: Exploiting cardinality encodings in parallel maximum satisfiability. In: ICTAI, pp. 313–320 (2011)
32. Martins, R., Manquinho, V., Lynce, I.: Clause sharing in parallel MaxSAT. In: Hamadi, Y., Schoenauer, M. (eds.) LION 2012. LNCS, vol. 7219, pp. 455–460. Springer, Heidelberg (2012)
33. 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)
34. Nieuwenhuis, R., Oliveras, A.: On SAT modulo theories and optimization problems. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 156–169. Springer, Heidelberg (2006)
35. Sebastiani, R.: Lazy Satisfiability Modulo Theories. Journal on Satisfiability, Boolean Modeling and Computation 3(3-4), 141–224 (2007)
36. Silva, J.P.M., Sakallah, K.A.: Grasp: A search algorithm for propositional satisfiability. IEEE Trans. Computers 48(5), 506–521 (1999)

D

Exploiting the Structure of Unsatisfiable Cores in MaxSAT

C. Ansótegui, F. Didier, J. Gabàs

in: Proceedings of the 28th International Joint Conference on Artificial Intelligence

IJCAI Buenos Aires

2015

Exploiting the Structure of Unsatisfiable Cores in MaxSAT*

Carlos Ansótegui
DIEI, Univ. of Lleida, Spain
carlos@diei.udl.es

Frederic Didier
Google Paris, France
fdid@google.com

Joel Gabàs
DIEI, Univ. of Lleida, Spain
jgabas@diei.udl.es

Abstract

We propose a new approach that exploits the good properties of core-guided and model-guided MaxSAT solvers. In particular, we show how to effectively exploit the structure of unsatisfiable cores in MaxSAT instances. Experimental results on industrial instances show that the proposed approach outperforms both complete and incomplete state-of-the-art MaxSAT solvers at the last international MaxSAT Evaluation in terms of robustness and total number of solved instances.

1 Introduction

Maximum Satisfiability (MaxSAT) is an optimization version of the well-known Satisfiability (SAT) problem. Recently, we have seen the successful application of MaxSAT techniques to solve several industrial or real combinatorial optimization problems: software package upgrade, debugging of hardware designs, fault localization in C code, bioinformatics, course timetabling, planing, scheduling, routing, electronic markets, combinatorial auctions, etc. See [Ansótegui *et al.*, 2013b; Morgado *et al.*, 2013] for citations.

Solving exactly combinatorial optimization problems, i.e., finding and certifying the best possible assignment, can be NP-hard from a computational point of view. However, many industrial problems are slightly beyond the reach of state-of-the-art techniques, and for many real domains we are often interested on improving in a reasonable time the best current assignment. Notice that even a small gain in the quality of the assignment can lead to important practical consequences for real domains.

This seems to suggest we should focus on developing incomplete algorithms for industrial problems. The experience achieved from the international SAT and MaxSAT competitions, shows us that the right research avenue is to focus first on improving complete or exact algorithms. Then, with the proper modifications, we can get an incomplete algorithm. Besides, we can always incorporate our complete algorithm into incomplete approaches such as Large Neighborhood Search [Shaw, 1998]. In LNS, we heuristically dive into promising regions of the search space (neighborhoods) that are explored with complete solvers.

The aim of this paper is to improve and combine effectively and efficiently techniques from complete state-of-the-art solvers MaxSAT solvers. In particular, we pay special attention to how to exploit the structure of the unsatisfiable cores in a MaxSAT instance. Finally, develop a complete algorithm that can be used in an incomplete approach.

In the MaxSAT community, we find two main classes of complete algorithms: branch and bound [Heras *et al.*, 2007; Li *et al.*, 2009; Kügel, 2010] and SAT-based [Ansótegui *et al.*, 2013b; Morgado *et al.*, 2013]. SAT-based approaches clearly dominate on industrial instances. SAT-based MaxSAT algorithms reformulate a MaxSAT instance into a sequence of SAT instances. By solving these SAT instances the MaxSAT problem can be solved. Intuitively, the SAT instances encode whether it is possible to find an assignment to the MaxSAT instances with a cost less than or equal to a certain k. The sequence is built in increasing order of k and it can be split into two parts. The instances in the first part are all unsatisfiable while the instances in the second one are all satisfiable. The value of k where the first satisfiable instance is located gives us the optimum of the MaxSAT instance. This transition from unsatisfiable to satisfiable SAT instances is usually associated with an easy-hard-easy pattern in terms of the hardness of the SAT instances [Shen and Zhang, 2003].

By solving the unsatisfiable SAT instances MaxSAT solvers refine the lower bound, while by solving satisfiable instances they refine the upper bound. Within SAT-based MaxSAT algorithms we find two main classes: (i) those that refine the lower bound, and guide the search with the unsatisfiable cores obtained from unsatisfiable SAT instances (core-guided algorithms) and, (ii) those that refine the upper bound, and guide the search with the satisfiable assignments obtained from satisfiable SAT instances (model-guided algorithms). Both approaches have strengths and weaknesses and there have been already some hybrid approaches [Morgado *et al.*, 2012; Ansótegui and Gabàs, 2013].

SAT-based MaxSAT solvers use Pseudo-Boolean (PB) constraints to create the SAT instances in the sequence. It is known that efficiency of SAT-based MaxSAT solvers heavily depends on how complex are these PB constraints, and how efficiently we manage them. Respect complexity, our current approach, as in [Andres *et al.*, 2012; Mor-

*Research partially supported by the Ministerio de Economía y Competividad research project TASSAT2: TIN2013-48031-C4-4-P and Google Faculty Research Award program.

gado *et al.*, 2014], only adds cardinality (Card) constraints (PB constraints with coefficients equal to 1) even if the MaxSAT instance is weighted. Respect management, unlike [Morgado *et al.*, 2014; Narodytska and Bacchus, 2014; Martins *et al.*, 2014], where Card constraints are incrementally constructed, our best approach does not use yet these incremental strategies for Card constraints. It exploits the structure of the unsatisfiable cores to produce a more efficient encoding for the Card constraints.

As we have mentioned, the algorithm we present is able to produce upper bounds during the search process. We also use these upper bounds to extend a very effective technique used in SAT solvers called *phase saving* [Pipatsrisawat and Darwiche, 2007] to MaxSAT.

Finally, we show an extensive experimental investigation on industrial instances. From the results, we can conclude that our approach outperforms clearly both in terms of robustness and number of solved instances the winners of the international MaxSAT Evaluation 2014 (MSE14) at the complete and incomplete tracks.

This paper proceeds as follows. Section 2 introduces some preliminary concepts. Section 3 presents the complete MaxSAT algorithm. Section 4 discusses how to encode efficiently in SAT the Card constraints generated by the MaxSAT algorithm. Section 5 explains how to exploit the upper bounds generated by the MaxSAT algorithm. Section 6 shows the experimental evaluation. Finally, Section 7 concludes.

2 Preliminaries

Definition 1 *A* literal *l is either a Boolean variable x or its negation $\overline{x}$. A* clause *c is a disjunction of literals. A* SAT formula *is a set of clauses that represents a Boolean formula in Conjunctive Normal Form (CNF), i.e. a conjunction of clauses.*

Definition 2 *A* weighted clause *is an ordered pair $\langle c, w\rangle$, where c is a clause and w is a natural number or infinity (indicating the cost of falsifying c, see Definitions 4 and 5). If w is infinite the clause is* hard*, otherwise it is* soft.

Definition 3 *A* Weighted Partial MaxSAT (WPMS) formula *is an ordered multiset of weighted clauses:*

$$\varphi = \langle \langle c_1, w_1\rangle, \ldots, \langle c_s, w_s\rangle, \langle c_{s+1}, \infty\rangle, \ldots, \langle c_{s+h}, \infty\rangle\rangle$$

where the first s clauses are soft and the last h clauses are hard. The presence of soft clauses with different weights makes the formula Weighted and the presence of hard clauses makes it Partial. The ordered multiset of weights of the soft clauses in the formula is noted as $w(\varphi)$. The top weight of the formula is noted as $W(\varphi)$, and defined as $W(\varphi) = \sum w(\varphi) + 1$. By φ_S we refer the set of soft clauses and by φ_H to the set of hard clauses. Finally, the set of variables occurring in the formula is noted as $var(\varphi)$.

Example 1 *Given the following WPMS formula: $\varphi = \langle\langle x_1, 5\rangle, \langle x_2, 3\rangle, \langle x_3, 3\rangle, \langle \overline{x}_1 \vee \overline{x}_2, \infty\rangle, \langle \overline{x}_1 \vee \overline{x}_3, \infty\rangle, \langle \overline{x}_2 \vee \overline{x}_3, \infty\rangle\rangle$, we have that $w(\varphi) = \langle 5, 3, 3\rangle$, $W(\varphi) = 12$, $\varphi_S = \langle\langle x_1, 5\rangle, \langle x_2, 3\rangle, \langle x_3, 3\rangle\rangle$, $\varphi_H = \langle\langle \overline{x}_1 \vee \overline{x}_2, \infty\rangle, \langle \overline{x}_1 \vee \overline{x}_3, \infty\rangle, \langle \overline{x}_2 \vee \overline{x}_3, \infty\rangle\rangle$ and $var(\varphi) = \{x_1, x_2, x_3\}$.*

Definition 4 *An* assignment *for a set of Boolean variables X is a function $\mathcal{I} : X \rightarrow \{0, 1\}$, that can be extended to literals, (weighted) clauses, SAT formulas and WPMS formulas as follows:*

$$\mathcal{I}(\overline{x}) = 1 - \mathcal{I}(x)$$
$$\mathcal{I}(l_1 \vee \ldots \vee l_m) = \max\{\mathcal{I}(l_1), \ldots, \mathcal{I}(l_m)\}$$
$$\mathcal{I}(\{c_1, \ldots, c_n\}) = \min\{\mathcal{I}(c_1), \ldots, \mathcal{I}(c_n)\}$$
$$\mathcal{I}(\langle c, w\rangle) = w\,(1 - \mathcal{I}(c))$$
$$\mathcal{I}(\langle\langle c_1, w_1\rangle, \ldots, \langle c_{s+h}, w_{s+h}\rangle\rangle) = \sum_{i=1}^{s+h} \mathcal{I}(\langle c_i, w_i\rangle)$$

We will refer to the value returned by an assignment $\mathcal{I}$ on a weighted clause or a WPMS formula as the cost of $\mathcal{I}$.

Definition 5 *We say that an assignment $\mathcal{I}$ satisfies a clause or a SAT formula if the value returned by $\mathcal{I}$ is equal to 1. In the case of SAT formulas, we will refer also to this assignment as a satisfying assignment or solution. Otherwise, if the value returned by $\mathcal{I}$ is equal to 0, we say that $\mathcal{I}$ falsifies the clause or the SAT formula.*

Definition 6 *The* SAT problem *for a SAT formula φ is the problem of finding a solution for φ. If a solution exists the formula is satisfiable, otherwise it is unsatisfiable.*

Definition 7 *Given an unsatisfiable SAT formula φ, an* unsatisfiable core *φ_C is a subset of clauses $\varphi_C \subseteq \varphi$ that is also unsatisfiable.*

Definition 8 *A SAT algorithm for the SAT problem, takes as input a SAT formula φ and returns an assignment $\mathcal{I}$ such that $\mathcal{I}(\varphi) = 1$ if the formula is satisfiable. Otherwise, it returns an unsatisfiable core φ_C.*

Given unlimited resources of time and memory, we say that a SAT algorithm is complete *if it terminates for any SAT formula. Otherwise, it is* incomplete.

Definition 9 *The* optimal cost (or optimum) *of a WPMS formula φ is* $\text{cost}(\varphi) = \min\{\mathcal{I}(\varphi) \mid \mathcal{I} : var(\varphi) \rightarrow \{0, 1\}\}$ *and an* optimal assignment *is an assignment $\mathcal{I}$ such that $\mathcal{I}(\varphi) = \text{cost}(\varphi)$. We will refer to this assignment as a solution for φ if $\mathcal{I}(\varphi) \neq \infty$. Any cost above (below) $cost(\varphi)$ is called an upper (lower) bound for φ.*

Example 2 *Given the WPMS formula φ of Example 1, we have that $\text{cost}(\varphi) = \min\{6, 8, 11, \infty\} = 6$ and the optimal assignment $\mathcal{I}$ maps $\langle x_1, x_2, x_3\rangle$ to $\langle 1, 0, 0\rangle$.*

Definition 10 *The* Weighted Partial MaxSAT problem *for a WPMS formula φ is the problem of finding a solution for φ. If a solution does not exist the formula is unsatisfiable.*

Definition 11 *A WPMS algorithm for the WPMS problem, takes as input a WPMS formula φ and returns an assignment $\mathcal{I}$, such that, $\mathcal{I}(\varphi) \geq cost(\varphi)$.*

Given unlimited resources of time and memory, we say that a WPMS algorithm is complete or exact *if for any input WPMS formula φ and returned $\mathcal{I}$, $\mathcal{I}(\varphi) = cost(\varphi)$. Otherwise, we say it is* incomplete.

Definition 12 *An* integer linear Pseudo-Boolean (PB) constraint *is an inequality of the form $w_1 x_1 + \cdots + w_n x_n$ op k, where $op \in \{\leq, \geq, =, >, <\}$, k and w_i are integer coefficients, and x_i are Boolean variables. A PB* at-most constraint *is a PB constraint where op is $\leq$. A* cardinality (Card) constraint *is a PB constraint where the coefficients w_i are equal to* 1.

3 WPM3 MaxSAT Algorithm

Algorithm 1: WPM3

```
Input: $\varphi = \langle\langle c_1, w_1\rangle, ..., \langle c_s, w_s\rangle, \langle c_{s+1}, \infty\rangle, ..., \langle c_{s+h}, \infty\rangle\rangle$
1  $\langle AM, w_{strat}\rangle := \langle\langle\langle\langle 1\rangle, 0, w_1\rangle, \ldots, \langle\langle s\rangle, 0, w_s\rangle\rangle, \infty\rangle$
2  while $true$ do
3      $\langle st, C, \mathcal{I}\rangle := sat(\varphi, AM, w_{strat})$
4      if $st =$ SAT then
5          if $w_{strat} = \min_{w_j \in w(AM), w_j \neq 0}(\{w_j\})$ then return $\langle \mathcal{I}, \mathcal{I}(\varphi)\rangle$
6          $w_{strat} = decrease(AM, w_{strat})$
7      else
8          if $w_{strat} = \infty$ then return $\langle \emptyset, \infty\rangle$
9          $w_{min} := min(w(AM_C))$
10         $AM := AM[\langle A_j, k_j, w_j\rangle_{j\in C} / \langle A_j, k_j, w_j - w_{min}\rangle_{j \in C}]$
11         $\langle A, k_A\rangle := optimize(\varphi, AM_C)$
12         $AM := AM + \langle\langle A, k_A, w_{min}\rangle\rangle$
```

Function sat(φ, AM, w_{strat})

```
1  $\varphi^k := \varphi_H \cup \{c_i \vee b_i\}_{\langle c_i, w_i\rangle \in \varphi_S} \cup \{\ CNF(A_j, k_j) \vee a_j, \overline{a}_j\ \}_{\langle A_j, k_j, w_j\rangle \in AM, w_j \geq w_{strat}}$
2  $\langle st, \varphi^k_C, \mathcal{I}\rangle := satsolver(\varphi^k)$
3  $C := \{j \mid \{\overline{a}_j\} \cap \varphi^k_C \neq \emptyset\}$
4  return $\langle st, C, \mathcal{I}\rangle$
```

Function optimize(φ, AM)

```
1  $A := +_{\langle A_j, K_j, w_j\rangle \in AM} A_j$
2  $ub := \sum_{\langle A_j, k_j, w_j\rangle \in AM} \mid A_j \mid$
3  while $true$ do
4      $k := ub - 1$
5      $\langle st, \_, \mathcal{I}\rangle := sat(\varphi, \langle\langle A, k, \_\rangle\rangle, \_)$
6      if $st =$ SAT then $ub := \mathcal{I}(\langle\langle c_i, 1\rangle \mid \langle c_i, w_i\rangle \in \varphi\rangle_{i \in A})$
7      else return $\langle A, ub\rangle$
```

In this section, we present the WPM3 complete algorithm for the MaxSAT problem. Given an input WPMS formula φ, WPM3 solves the formula by testing the satisfiability of a sequence of SAT instances φ^k where $0 \leq k \leq W(\varphi)$. Each SAT instance φ^k encodes whether there is an assignment to φ with a cost $\leq k$. Notice that SAT instances with $k < cost(\varphi)$ are unsatisfiable, otherwise they are satisfiable. The optimum corresponds to the k of the first satisfiable SAT instance.

Roughly speaking, from every unsatisfiable SAT instance the algorithm finds and keeps an unsatisfiable core. WPM3 is designed to be aware of the global structure of theses cores. This is used both for producing more efficient cardinality (Card) constraint encodings (see Section 4) and focus the search on subproblems of the input MaxSAT instance.

The algorithm maintains a set of soft at-most Card constraints AM. We note these constraints as $\langle A, k, w\rangle$, where A is an ordered multiset of indexes of the original soft clauses, k indicates at most how many clauses from A can be falsified, and w is the cost for falsifying this soft constraint. The at-most constraints are used to do not accept those solutions whose cost exceeds the current k, where $k = \sum_{\langle A_j, k_j, w_j\rangle \in AM} k_j \cdot w_j$. Moreover, the algorithm guarantees that $k \leq cost(\varphi)$. The idea of maintaining multiple at-most Card constraints instead of a single one was originally introduced in [Ansótegui *et al.*, 2009a] for PM2 algorithm. Notice that from the AM set the global core structure can be obtained.

We start (line 1) by adding to AM a soft at-most constraint for each original soft clause. Then, the algorithm will iterate (line 2) till it is able to determine $cost(\varphi)$. This will happen if it detects that the set of hard clauses is unsatisfiable (line 8, $cost(\varphi) = \infty$) or when it is able to generate the first satisfiable instance (line 5, $cost(\varphi) = k = \mathcal{I}(\varphi)$). We obviate for the moment the role of w_{stat} and we consider it is ∞ for the first iteration and $\min_{w_j \in w(AM), w_j \neq 0}(\{w_j\})$ for the rest.

Function *sat* (line 3) builds the SAT instance φ^k at the current iteration and sends it to the SAT solver. φ^k is constructed through the union of the following sets expressed as SAT clauses: (i) the set of hard clauses, (ii) the reification to variables b_i of soft clauses, (iii) the reification to variables a_j of the translation into CNF of the at-most constraints in AM, and finally (iv) the unit clauses $\overline{a}_j$. The new b_i variables are true if the respective original soft clause becomes false. They are used to encode the at-most constraints which restrict the number of falsified soft clauses. The new a_j variables are true if the respective at-most constraint becomes false. The unit clauses $\overline{a}_j$ encode that we would like to satisfy all the at-most constraints. If this is not possible, some of them will appear into the unsatisfiable core φ^k_C.

Example 3 *Given* $\varphi = \langle\langle x_1, 1\rangle, \langle x_2, 1\rangle, \langle x_3, 1\rangle, \langle \overline{x}_1 \vee \overline{x}_2, \infty\rangle, \langle \overline{x}_1 \vee \overline{x}_3, \infty\rangle, \langle \overline{x}_2 \vee \overline{x}_3, \infty\rangle\rangle$*. At an intermediate step we have the set* $AM = \langle\langle\langle 1, 2\rangle, 1, 1\rangle, \langle\langle 3\rangle, 0, 1\rangle\rangle$*. Then,*

$$\varphi^k = \{\overline{x}_1 \vee \overline{x}_2,\ \overline{x}_1 \vee \overline{x}_3,\ \overline{x}_2 \vee \overline{x}_3\} \cup \{x_1 \vee b_1,\ x_2 \vee b_2,\ x_3 \vee b_3\}$$
$$\cup\ \{CNF(b_1 + b_2 \leq 1) \vee a_1, \overline{a}_1, CNF(b_3 \leq 0) \vee a_2, \overline{a}_2\}$$

To our best knowledge, the idea of introducing Card constraints as soft constraints in a core-guided algorithm was initially proposed in [Andres *et al.*, 2012] for Answer Set Programming (ASP) optimization problems, and recently adapted to MaxSAT problems in [Morgado *et al.*, 2014]. The approach in [Narodytska and Bacchus, 2014] (best solver for industrial instances at MSE14) also uses soft Card constraints. These approaches work locally and are not aware of the global core structure and therefore they can not exploit it.

Going back to WPM3 algorithm, if function *sat* returns satisfiable ($st = SAT$) (line 4), then we return the optimal assignment $\mathcal{I}$ and its cost (line 5). Otherwise (line 7), C is the set of indexes of at-most constraints involved into the last unsatisfiable core. If the core only involves original hard clauses, we can return unsatisfiable (line 8). If there are at-most constraints involved in the core, then, we need to relax some of them since they only allow assignments or solutions with a cost strictly less than $cost(\varphi)$.

At this stage (lines 9-12), we need to relax the set of AM constraints, but guaranteeing we do not exceed $cost(\varphi)$. Basically, we need to replace the set of at-most constraints AM_C involved in the last core with a new set of at-most constraints which allows assignments with a higher cost.

Since we will only use Card constraints we first apply the idea described in the WPM1/WBO [Ansótegui *et al.*, 2009b; Manquinho *et al.*, 2009] MaxSAT algorithms to deal with Weighted instances (lines 9-10). It basically prevents the algorithm to introduce PB constraints instead of Card constraints when the at-most constraints involved in the core have different weights. In this case, we compute the minimum weight w_{min} of the constraints in AM_C (line 9), and replace every soft constraint $\langle A_j, k_j, w_j\rangle$ by two copies with weights $w_j - w_{min}$ and w_{min}. The first set of copies will remain in AM (line 10) while the second will be replaced by the new at-most constraint $\langle A, k_A, w_{min}\rangle$ (line 12). Notice we guarantee that $\sum_{\langle A_j,k_j,w_j\rangle\in AM} \mid A_j \mid \cdot w_j = \sum w(\varphi)$.

Function *optimize* (line 11) allows to determine which is the new φ_k we will test. This function, basically, solves exactly the subproblem that involves the new at-most constraint we will generate on the original soft clauses, and the hard clauses. The result is the set of indexes of original soft clauses A of the new at-most constraint (notice that and index can be repeated) and the number of clauses k_A that we will at most allow to be falsified. To our best knowledge, the idea of solving a subproblem of the original optimization instance φ was originally applied for MaxSAT in WPM2 algorithm [Ansotegui *et al.*, 2010]. In [Davies and Bacchus, 2011] a similar approach is applied calling a MIP solver to solve the subproblem. Recently, in [Ansótegui *et al.*, 2013a], this process is extended and named as *cover optimization*. The best strategy reported consists on refining iteratively the upper bound on the subproblem using the model-guided MaxSAT algorithm in [Berre, 2006]. We apply it within function *optimize*, although depending on the particular family of instances other strategies or algorithms can have better performance. At this point we have increased k by $(k_A - \sum_{\langle A_j,k_j,w_j\rangle\in AM_C} k_j)\cdot w_{min}$ towards $cost(\varphi)$. Otherwise, without the *optimize* step, we can only increase k by w_{min}.

Treating explicitly the new at-most constraint (line 12) and its relation to the constraints it substitutes is fundamental, not only for function *optimize*, but also to encode more efficient Card constraints (see Section 4). In contrast, this information, the global core structure, is not explicitly present in recent approaches as [Morgado *et al.*, 2014; Narodytska and Bacchus, 2014] and therefore harder to be exploited efficiently.

During this description, we have obviated the role of w_{strat} (lines 5 and 6). It corresponds to the application of the stratified approach introduced in [Ansotegui *et al.*, 2009c]. The stratified approach consists in sending to the SAT solver only a subset of the soft clauses, i.e., those with a weight $\geq w_{strat}$. Function *decrease* updates conveniently w_{strat}. This can help to reduce the size of the unsatisfiable cores, produce simpler at-most constraints and progress faster to the optimum. We also apply *hardening* techniques like the ones described in [Ansótegui *et al.*, 2012; Morgado *et al.*, 2012; Ansótegui *et al.*, 2013a].

4 Efficient Card Constraints for MaxSAT

In the last decade, we have seen many contributions on how to encode efficiently PB and Card constraints into SAT [Bailleux and Boufkhad, 2003; Sinz, 2005; Eén and Sörensson, 2006; Bailleux *et al.*, 2009; Asín *et al.*, 2011]. The goal is to achieve an arc-consistent encoding (i.e., with good propagation properties) as small as possible.

Since WPM3 only uses Card constraints, let us consider the Card constraint: $b_1 + \cdots + b_n \leq k$. From the sake of clarity, the encoder is split into two black boxes: the *sum* and the operator *op* (in our case representing $\leq$). The *sum* takes as input a list of n variables $[b_1,\ldots,b_n]$ and returns a set of SAT clauses S and a list of m variables $[o_1,\ldots,o_m]$. The operator takes as input the o variables and integer k and returns a set of SAT clauses OP. The encoding of the Card constraint into SAT corresponds to the union $S \cup OP$.

$$\langle S, [o_1,\ldots,o_m]\rangle := sum([b_1,\ldots,b_n])$$
$$OP := op(k, [o_1,\ldots,o_m])$$

In our case, we use the Cardinality Networks encoding in [Asín *et al.*, 2011]. There, $m = k+1$ and the sum builds a SAT formula such that if i of the input b variables are set to true then the first i of the output o variables are set to true and the rest to false. Therefore, *op* returns the unit clause $\overline{o}_{k+1}$.

Our first observation is that it is crucial for the efficiency of the MaxSAT solver in which order the b variables are fed into the sum. In previous MaxSAT solvers, the order of the b variables was not taken into account. They were just added in the same order of their respective soft clauses.

However, the b variables should be added taking into account the structure of the unsatisfiable cores. In particular, variables belonging to the same core should be as close as possible. In our algorithm the set A in an at-most constraint $\langle A, k, w\rangle$ is ordered. In line 1 of function *optimize* (see Section 3) when we generate the set A of the new at most constraint, we concatenate the sets of b variables of the at-most constraints that it replaces. Respect to latest advances in MaxSAT [Morgado *et al.*, 2014; Narodytska and Bacchus, 2014] a deeper explanation of their efficiency could be that these algorithms implicitly preserve the order.

Our second observation has to do again with the structure of the unsatisfiable cores we have detected so far. As we have just commented, the new at-most constraint $\langle A, k_A, w_{min}\rangle$ (line 12 of WPM3) replaces/merges other at-most constraints. In the end, we can consider that there is a tree structure that represents how we have merged the unsatisfiable cores and where the root node is the new at-most constraint. Instead of creating a Cardinality Network for the new constraint we can reproduce this tree structure. We basically reproduce the totalizer encoding proposed originally in [Bailleux and Boufkhad, 2003]. The leaf nodes join the at-most constraints related with a single soft clause of the input formula (i.e., with $k = 0$) that appeared in the same core. The leaf nodes are encoded with Cardinality Networks.

Example 4 *Imagine at-most constraint $\langle A_3, 5, 1\rangle$ (root node) replaces at-most constraints $\langle A_1, 3, 1\rangle$ and $\langle A_2, 1, 1\rangle$ (child 1 and child 2). Let us see how we start constructing the tree. Children are processed recursively in the same way.*

$$\langle S', \overset{\substack{|A_3|=10\\ sum\leq 5}}{[o^3_1, \dots, o^3_6]}\rangle := totalizer(\overset{\substack{|A_1|=6\\ sum\leq 5}}{[o^1_1, ..., o^1_6]}, \overset{\substack{|A_2|=4\\ sum\leq 4}}{[o^2_1, ..., o^2_5]})$$

$$S^3 := S' \cup S^1 \cup S^2 \quad OP^3 := \{\overline{o}^3_6\} \cup OP^1 \cup OP^2$$

The advantage of preserving this structure, in contrast to having a single Card constraint, is that we can again exploit it to derive smaller encodings. In particular, we can restrict the sums of the nodes using the lower bounds of their siblings. The upper bound for a non root node is set to the difference between the upper bound of its parent and the sum of the lower bounds of its siblings. We apply this in a top-down update process.

Example 5 *At example 4, the upper bound for the root node is* 5*. Child 1 (child 2) already contributes with a lower bound of* 3 *(*1*), therefore the new upper bound for child 1 (child 2) is* $5 - 1 = 4$ *(*$5 - 3 = 2$*).*

$$\langle S', \overset{\substack{|A_3|=10\\ sum\leq 5}}{[o^3_1, \dots, o^3_6]}\rangle := totalizer(\underset{lb=3}{\overset{\substack{|A_1|=6\\ sum\leq 4}}{[o^1_1, ..., o^1_5]}}, \underset{lb=1}{\overset{\substack{|A_2|=4\\ sum\leq 2}}{[o^2_1, ..., o^2_3]}})$$

$$S^3 := S' \cup S^1 \cup S^2 \quad OP^3 := \{\overline{o}^3_6\} \cup OP^1 \cup OP^2$$

Function *optimize* can provide assignments I_{ub} which are upper bounds (see Section 5) for the whole formula φ. Using this upper bound, we can set the upper bound for the root node which corresponds to the cost of the assignment I_{ub} on the set of soft clauses related to it.

Finally, notice that for a given at-most constraint we build from scratch its SAT encoding when we feed it into the SAT solver. Recently, in [Martins *et al.*, 2014] it has been proposed to build Card constraints incrementally. We will explore extending our approach in this sense. As we will see in Section 6, even without the incremental extension, our approach outperforms the rest of the solvers.

5 Upper Bounds and Phase Saving

In Section 3, we have described a complete core-guided algorithm that keeps increasing a lower bound towards the optimum. However, its design and, in particular, function *optimize*, which implements a model-guided MaxSAT algorithm, allows to produce upper bounds for φ during its execution. Whenever function *optimize* finds a new solution to the subproblem, we can just extend this solution to the rest of the formula and check its cost. We do this by considering that, those clauses in a undefined state under the solution to the subproblem, are falsified [1]. Function *optimize* keeps track of the assignment to the subproblem whose extension to the rest of the formula had the lowest cost. The cost of this assignment is an upper bound for the whole problem. We will refer to it as the best global assignment found by function *optimize*.

We can exploit further the best global assignment we get from function *optimize*. State-of-the-art SAT solvers apply a technique called *phase saving* introduced in [Pipatsrisawat and Darwiche, 2007]. In SAT solvers, when non-chronological backtracking is applied due to a conflict, lots of variable assignments get lost and have to be revealed again during search. The idea is to avoid redoing this work by storing the phase of the variables when we find a conflict. Then, we assign to the next decision variables the stored polarity till we find a new conflict and update the polarities again. This technique has been shown to be quite effective. We can extend this idea to MaxSAT in the following way. Within function *optimize* we let the SAT solver apply phase saving in the regular way (line 5). Then, if the best global assignment found so far has the same cost for the subproblem as the solution found by function *optimize*, we store the polarity of the variables in the best global assignment. We use these polarities to guide the search of the SAT solver in line 3 of WPM3, disabling the regular phase saving that would be applied. In particular, we only store the polarities of the original variables in φ.

6 Experimental Results

We have evaluated our approach on the industrial instances from the MaxSAT Evaluation 2014 (MSE14) [Argelich *et al.*, 2006 2014]. We run our experiments on a cluster featured with 2.6GHz processors and a memory limit of 3.5 GB. The instance set of MSE14 is divided into three categories depending on the variant of the MaxSAT problem: MaxSAT (MS), Partial MaxSAT (PMS) or Weighted Partial MaxSAT (WPMS). In each category, instances are grouped by families: 2 for MS, 22 for PMS and 8 for WPMS. Since families have different number of instances, we considered more fair to present the solvers ranked by mean family ratio of solved instances.

We provide results for the new $wpm3$ MaxSAT solver and the best solvers of the MSE14. We have excluded the MaxSAT solver $ISAC+$ [Ansótegui *et al.*, 2014], since it is a portfolio and our intention here is to compare ground solvers. The ground solvers with the best overall performance at MSE14 for industrial instances were: $eva500a$ [Narodytska and Bacchus, 2014], $mscg$ [Morgado *et al.*, 2014] and $open\text{-}wbo$ [Martins *et al.*, 2014]. We also present results for an initial version we implemented within the $or\text{-}tools$ package [Google, 2009] which is only core-guided.

Table 1 shows our first experiment, where we evaluate the impact of each improvement on WPM3 (with a timeout of 1800 seconds). All the variations on WPM3 are implemented on top of the glucose SAT solver (version 3.0) [Audemard and Simon, 2009] . The different variations and corresponding implementations are named $wpm3$ with different subindexes. Subindex $_o$ stands for *cover optimization* (see Section 3). Regarding how Card constraints are encoded, $_t$ stands for core based tree, $_{tk}$ stands for core based tree with refinement of the

[1]This can be improved looking at the structure of the undefined clauses.

	MS Ind. 100% 55	PMS Ind. 100% 568	WPMS Ind. 100% 410	Total Ind. 100% 1033
$wpm3_{tkop}$	**88,5% 43**	**84,0% 499**	**74,0% 367**	**81,7% 909**
$wpm3_{tko}$	87,5% 42	83,4% 494	73,9% 366	81,3% 902
$wpm3_{tk}$	87,5% 42	82,5% 483	73,0% 359	80,4% 884
$wpm3_{to}$	87,5% 42	83,4% 494	72,9% 358	81,0% 894
$wpm3_{t}$	87,5% 42	81,9% 480	72,9% 358	80,0% 880
$wpm3_{o}$	87,5% 42	82,9% 489	71,3% 349	80,3% 880
$wpm3$	87,5% 42	80,7% 470	70,4% 346	78,6% 858
$pm2_{o}$	84,0% 39	78,9% 458	-	-
$pm2$	84,0% 39	77,5% 445	-	-
$wpm1/wbo$	80,0% 36	61,8% 349	67,4% 348	64,4% 733

Table 1: WPM3 and improvements.

k upper bound in sub-sums (see Section 4). Finally, $_p$ stands for phase saving extended to MaxSAT (see Section 5).

At the table, we present for each category and solver the mean ratio of solved instances per family and the total number of instances. We introduce results for $wpm1/wbo$ [Ansótegui *et al.*, 2009b; Manquinho *et al.*, 2009] and $pm2$ algorithms [2]. $wpm3$ can be considered as a hybridization of these two algorithms (see Section 3). As we can see, $wpm3$ clearly outperforms $pm2$. This is because, as described in Section 4, we build Card constraints taking into account the order imposed by the unsatisfiable cores. If we add the *cover optimization* technique, see $wpm3_o$, then we get a version that would have ranked the first at MSE14 for industrial instances in terms of the total mean family ratio. The next two versions, $wpm3_t$ and $wpm3_{tk}$, that further exploit the structure of the cores and improve the construction of the Card constraint, provide a total of 13 additional solved instances for PMS and 13 for WPMS. As we can see, the *cover optimization* technique always improves, in particular for Weighted instances at version $wpm3_{tko}$. Finally, the extension of phase saving for MaxSAT improves the average running time, and it helps to solve 7 additional instances within the timeout.

	MS Ind. 100% 55	PMS Ind. 100% 568	WPMS Ind. 100% 410	Total Ind. 100% 1033
$wpm3_{tkop}$	**88.5% 43**	**84.0% 499**	**74.0%** 367	**81.7% 909**
$or\text{-}tools$	81.7% 36	81.9% 482	73.9% **369**	79.8% 887
$eva500a$	86.5% 41	80.0% 476	72.8% 368	78.7% 885
$mscg$	86.5% 41	80.2% 468	68.5% 361	77.7% 870
$open\text{-}wbo$	87.5% 42	81.0% 472	64.9% 335	77.3% 849

Table 2: WPM3 and $or\text{-}tools$ compared to best MSE14 solvers

Table 2 compares our best version $wpm3_{tkop}$ with the best performing complete solvers at MSE14 for industrial instances. Clearly, $wpm3_{tkop}$ dominates both on mean family ratio and total number of solved instances. A deeper analysis reveals that $wpm3_{tkop}$ has best ratio on 30 out of the 32 families that compose the categories, while $eva500a$ (the second one) has best ratio on 20.

Our last experiment is presented in Table 3. Since $wpm3_{tkop}$ is able to produce upper bounds we also compared

[2]$pm2$ algorithm is only designed for Partial MaxSAT instances

	$wpm3_{tkop}$	$open\text{-}wbo$	qms	$wpm2014$	$optimax$
$wpm3_{tkop}$		**50** **556 394**	**50** 502 **379**	**49** **547 386**	**53** **524 390**
$open\text{-}wbo$	45 466 344		**45** 458 **361**	44 **477** 341	**45** **466 378**
qms	15 **511** 281	19 **522** 286		17 **534** 279	18 **530 302**
$wpm2014$	48 427 369	**48** **477 387**	**48** 402 **379**		**51** 420 **367**
$optimax$	39 368 259	43 387 270	**47** 337 284	39 **423** 260	

Table 3: WPM3 compared to MSE14 best incomplete solvers.

it with the best performing solvers $wpm2014$ and $optimax$ [3] for industrial instances at the incomplete track of the MSE14. We also compared with other MaxSAT solvers that did not take part in the incomplete track but they are able to produce upper bounds: *open-wbo* [Martins *et al.*, 2014] and qms [Koshimura *et al.*, 2012]. We do not include $eva500a$, $mscg$ or $or\text{-}tools$ since they can not produce upper bounds.

The timeout for the incomplete track at MSE is set to 300 seconds. For a given instance, the winners are the solvers that produce the best upper bound. The best solver is the one that won on more instances. Since these results give us a partial order, it can be misleading to report an overall winner. In table 3, we present the dominance relation between pairs of solvers on the three categories. For example, $wpm3_{tkop}$ (*open-wbo*) is able to obtain a better or equal upper bound than *open-wbo* ($wpm3_{tkop}$) on 50 (45) ms instances, 556 (466) PMS instances and 394 (344) WPMS instances. As we can see, $wpm3_{tkop}$ practically dominates the rest of the solvers. The exception is qms on PMS where qms outperforms by 9 instances $wpm3_{tkop}$. For this case, we extended the timeout to 1800 seconds. We found that $wpm3_{tkop}$ outperformed qms by 5 instances. This is somehow expected since $wpm3_{tkop}$, within this timeout, solves to optimality 40 instances more than qms.

7 Conclusions

We have proposed a complete algorithm for MaxSAT that can be also used in an incomplete approach. We show how to combine several techniques. We have shown how that the design of the algorithm allows to exploit the structure of unsatisfiable cores in MaxSAT instances to build more efficient Card constraints. This opens a window to understand better the performance of the latest solvers and probably further improve them. Finally, we have shown that an extended notion of phase saving for MaxSAT is effective. Our resulting MaxSAT solver outperforms the winners for the complete and incomplete track at the last MaxSAT Evaluation.

References

[Andres *et al.*, 2012] Benjamin Andres, Benjamin Kaufmann, Oliver Matheis, and Torsten Schaub. Unsatisfiability-based optimization in clasp. In *ICLP (Technical Communications)*, pages 211–221, 2012.

[3]$optimax$ builds on top on the bincd2 algorithm [Morgado *et al.*, 2012]

[Ansótegui and Gabàs, 2013] Carlos Ansótegui and Joel Gabàs. Solving (weighted) partial maxsat with ilp. In *CPAIOR 13*, pages 403–409, 2013.

[Ansótegui *et al.*, 2009a] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. On solving MaxSAT through SAT. In *Proc. of CCIA'09*, pages 284–292, 2009.

[Ansótegui *et al.*, 2009b] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. Solving (weighted) partial maxsat through satisfiability testing. In *Proc. of SAT'09*, pages 427–440, 2009.

[Ansotegui *et al.*, 2009c] Carlos Ansotegui, Maria Luisa Bonet, and Jordi Levy. Solving (weighted) partial maxsat through satisfiability testing. In *Proc. of SAT'09*, pages 427–440, 2009.

[Ansotegui *et al.*, 2010] Carlos Ansotegui, Maria Luisa Bonet, and Jordi Levy. A new algorithm for weighted partial maxsat. In *Proc. of AAAI'10*, pages 867–872, 2010.

[Ansótegui *et al.*, 2012] Carlos Ansótegui, Maria Luisa Bonet, Joel Gabàs, and Jordi Levy. Improving sat-based weighted maxsat solvers. In *Proc. of CP'12*, pages 86–101, 2012.

[Ansótegui *et al.*, 2013a] Carlos Ansótegui, Maria Luisa Bonet, Joel Gabàs, and Jordi Levy. Improving wpm2 for (weighted) partial maxsat. In *Proc. of CP'13*, pages 117–132, 2013.

[Ansótegui *et al.*, 2013b] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. Sat-based maxsat algorithms. *Artif. Intell.*, 196:77–105, 2013.

[Ansótegui *et al.*, 2014] Carlos Ansótegui, Yuri Malitsky, and Meinolf Sellmann. Maxsat by improved instance-specific algorithm configuration. In *Proc. of AAAI'14*, pages 2594–2600, 2014.

[Argelich *et al.*, 2006 2014] Josep Argelich, Chu Min Li, Felip Manyà, and Jordi Planes. Maxsat evaluation, 2006-2014. http://www.maxsat.udl.cat.

[Asín *et al.*, 2011] Roberto Asín, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. Cardinality networks: a theoretical and empirical study. *Constraints*, 16(2):195–221, 2011.

[Audemard and Simon, 2009] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In *Proc. of IJCAI'09*, pages 399–404, 2009.

[Bailleux and Boufkhad, 2003] Olivier Bailleux and Yacine Boufkhad. Efficient CNF encoding of boolean cardinality constraints. In *Proc. of CP'03*, pages 108–122, 2003.

[Bailleux *et al.*, 2009] Olivier Bailleux, Yacine Boufkhad, and Olivier Roussel. New encodings of pseudo-boolean constraints into cnf. In *Proc. of SAT'09*, pages 181–194, 2009.

[Berre, 2006] Daniel Le Berre. Sat4j, a satisfiability library for java, 2006. www.sat4j.org.

[Davies and Bacchus, 2011] Jessica Davies and Fahiem Bacchus. Solving maxsat by solving a sequence of simpler sat instances. In *Proc. of CP'11*, pages 225–239, 2011.

[Eén and Sörensson, 2006] Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into SAT. *JSAT*, 2(1-4):1–26, 2006.

[Google, 2009] Google. Or-tools: The google operations research suite, 2009. https://developers.google.com/optimization/.

[Heras *et al.*, 2007] Federico Heras, Javier Larrosa, and Albert Oliveras. MiniMaxSat: A new weighted Max-SAT solver. In *Proc. of SAT'07*, pages 41–55, 2007.

[Koshimura *et al.*, 2012] Miyuki Koshimura, Tong Zhang, Hiroshi Fujita, and Ryuzo Hasegawa. Qmaxsat: A partial max-sat solver. *JSAT*, 8(1/2):95–100, 2012.

[Kügel, 2010] Adrian Kügel. Improved exact solver for the weighted MAX-SAT problem. In *POS-10. Pragmatics of SAT, Edinburgh, UK, July 10, 2010*, pages 15–27, 2010.

[Li *et al.*, 2009] Chu Min Li, Felip Manyà, Nouredine Ould Mohamedou, and Jordi Planes. Exploiting cycle structures in Max-SAT. In *Proc. of SAT'09*, pages 467–480, 2009.

[Manquinho *et al.*, 2009] Vasco Manquinho, João Marques-Silva, and Jordi Planes. Algorithms for weighted boolean optimization. In *Proc. of SAT'09*, pages 495–508, 2009.

[Martins *et al.*, 2014] Ruben Martins, Saurabh Joshi, Vasco M. Manquinho, and Inês Lynce. Incremental cardinality constraints for maxsat. In *Proc. of CP'14*, pages 531–548, 2014.

[Morgado *et al.*, 2012] António Morgado, Federico Heras, and João Marques-Silva. Improvements to core-guided binary search for maxsat. In *Proc. of SAT'12*, pages 284–297, 2012.

[Morgado *et al.*, 2013] António Morgado, Federico Heras, Mark H. Liffiton, Jordi Planes, and Joao Marques-Silva. Iterative and core-guided maxsat solving: A survey and assessment. *Constraints*, 18(4):478–534, 2013.

[Morgado *et al.*, 2014] António Morgado, Carmine Dodaro, and Joao Marques-Silva. Core-guided maxsat with soft cardinality constraints. In *Proc. of CP'14*, pages 564–573, 2014.

[Narodytska and Bacchus, 2014] Nina Narodytska and Fahiem Bacchus. Maximum satisfiability using core-guided maxsat resolution. In *Proc. of AAAI'14*, pages 2717–2723, 2014.

[Pipatsrisawat and Darwiche, 2007] Knot Pipatsrisawat and Adnan Darwiche. A lightweight component caching scheme for satisfiability solvers. In *Proc. of SAT'07*, pages 294–299, 2007.

[Shaw, 1998] Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Proc. of CP'98*, pages 417–431, 1998.

[Shen and Zhang, 2003] Haiou Shen and Hantao Zhang. An empirical study of MAX-2-SAT phase transitions. *Electronic Notes in Discrete Mathematics*, 16:80–92, 2003.

[Sinz, 2005] Carsten Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In *Proc. of CP'05*, pages 827–831, 2005.

E

Exploiting subproblem optimization in SAT-based MaxSAT algorithms

C. Ansótegui, J. Gabàs, J. Levy

Journal of Heuristics 22(1)

2016

J Heuristics (2016) 22:1–53
DOI 10.1007/s10732-015-9300-7

Exploiting subproblem optimization in SAT-based MaxSAT algorithms

Carlos Ansótegui[1] · **Joel Gabàs**[1] · **Jordi Levy**[2]

Received: 20 October 2014 / Revised: 23 July 2015 / Accepted: 16 August 2015 /
Published online: 15 September 2015

Abstract The Maximum Satisfiability (MaxSAT) problem is an optimization variant of the Satisfiability (SAT) problem. Several combinatorial optimization problems can be translated into a MaxSAT formula. Among exact MaxSAT algorithms, SAT-based MaxSAT algorithms are the best performing approaches for real-world problems. We have extended the WPM2 algorithm by adding several improvements. In particular, we show that by solving some subproblems of the original MaxSAT instance we can dramatically increase the efficiency of WPM2. This led WPM2 to achieve the best overall results at the international MaxSAT Evaluation 2013 (MSE13) on industrial instances. Then, we present additional techniques and heuristics to further exploit the information retrieved from the resolution of the subproblems. We exhaustively analyze the impact of each improvement what contributes to our understanding of why they work. This architecture allows to convert exact algorithms into efficient incomplete algorithms. The resulting solver had the best results on industrial instances at the incomplete track of the latest international MSE.

Keywords Constraint optimization · Satisfiability · Maximum Satisfiability

✉ Carlos Ansótegui
carlos@diei.udl.es

Joel Gabàs
joel.gabas@udl.cat

Jordi Levy
levy@iiia.csic.es

1 DIEI, Universitat de Lleida, Lleida, Spain

2 IIIA-CSIC, Barcelona, Spain

1 Introduction

Combinatorial optimization problems arise in many domains: scheduling and planning, software and hardware verification, knowledge compilation, probabilistic modeling, bioinformatics, energy systems, smart cities, social networks, computational sustainability, etc. From a computational point of view, many optimization problems are NP-hard meaning that is unlikely that they admit a polynomial-time algorithm. The good news is that some real problems are already efficiently solved by state-of-the-art Constraint Programming techniques (Rossi et al. 2006) and many others are only slightly beyond the reach of these techniques.

In recent decades, Satisfiability (SAT) solvers (Biere et al. 2009) have progressed spectacularly in performance thanks to better implementation techniques and conceptual enhancements, such as Conflict Driven Clause Learning-based algorithms, which are able to reduce the size of the search space significantly in many instances of real NP-Complete problems. Every year, the community celebrates competitions where the performance of all these solvers is compared, and every year we contemplate how the number of solvable problems increases, and how instances that were considered very difficult, become easy. Thanks to these advances, nowadays the best SAT solvers can tackle industrial problems with hundreds of thousands of variables and millions of clauses. We use the term industrial in the sense of practical or real-world applications. Some extensions of SAT that have also attracted the interest of the scientific community in recent years include: Pseudo-Boolean (PB) satisfiability, Satisfiability Modulo Theories (SMT), satisfiability of Quantified Boolean Formulas, Maximum Satisfiability (MaxSAT), Model Counting (#SAT), etc. There exist also solvers for all these extensions of SAT, and competitions where they are tested.

The MaxSAT problem is the optimization version of SAT. The idea behind this formalism is that sometimes not all the constraints of a problem can be satisfied, and we try to satisfy the maximum number of them. The MaxSAT problem can be further generalized to the Weighted Partial MaxSAT (WPMS) problem. In this case, we can divide the constraints into two groups: the clauses (constraints) that must be satisfied (hard), and the ones that may or may not be satisfied (soft). In the last group, we may associate different weights with the clauses, where the weight is the cost of falsifying the clause. The idea is that not all constraints are equally important. The addition of weights to clauses makes the instance Weighted, and the separation into hard and soft clauses makes the instance Partial. The WPMS problem is a natural combinatorial optimization problem, and it has been already applied in many domains (Ansótegui et al. 2013b; Morgado et al. 2013a).

In the MaxSAT community, we find two main classes of complete algorithms: branch and bound (Heras et al. 2007; Kügel 2010; Li et al. 2009; Lin and Su 2007; Lin et al. 2008) and SAT-based (Ansótegui et al. 2012; Davies and Bacchus 2011; Heras et al. 2011; Honjyo and Tanjo 2012; Koshimura et al. 2012; Martins et al. 2011, 2012; Morgado et al. 2012).

SAT-based approaches clearly dominate on industrial and some crafted instances, as we can see in the results of the international MaxSAT Evaluation (Argelich et al. 2006-2004). SAT-based MaxSAT algorithms basically reformulate a MaxSAT instance into a sequence of SAT instances. By solving these SAT instances the MaxSAT problem can be solved (see (Ansótegui et al. 2013b; Morgado et al. 2013a) for further information). Intuitively, this sequence is built such that it can be split into two parts where the instances in the first part are all unsatisfiable while the instances in the second one are all satisfiable. By locating the phase transition point, i.e., the last unsatisfiable instance and the first satisfiable instance, we can locate the optimum of the optimization problem. As we will see, once we solve an unsatisfiable SAT instance we can refine the lower bound, while when we solve a satisfiable SAT instance we can refine the upper bound. Among SAT-based MaxSAT algorithms we also find two main classes: (i) those that focus the search on refining the lower bound, and exploit the information of unsatisfiable cores and, (ii) those that focus the search on refining the upper bound, and exploit the information of the satisfiable assignments. Both approaches have strengths and weaknesses. Our current work aims to find an efficient balance between both approaches.

In this paper, we present the improved version of the SAT-based MaxSAT algorithm WPM2 (Ansotegui et al. 2010) (see Sect. 4) presented originally in Ansótegui et al. (2013a), a more detailed experimental analysis of the new algorithm, further improvements and an incomplete version. The aim of this paper is not only to present a method that performs well, but also to understand why this is the case. This way we will be able to identify the interaction with other future improvements in the field and whether they are complementary or not to this work.

In our experimental investigation, our reference point is the original WPM2 algorithm which solves 959 out of 2078 instances from the whole benchmark of industrial and crafted instances of the MSE13.

With respect to the improvements we have incorporated, first of all, we extend the original WPM2 algorithm by applying the stratification approach described in Ansótegui et al. (2012), what results in solving 100 additional instances. Second, we introduce a new criteria to decide when soft clauses can be hardened (Ansótegui et al. 2012; Morgado et al. 2012), that provides 68 additional solved instances. Finally, our most effective contribution is to introduce a new strategy that focuses search on solving subproblems of the original MaxSAT instance. In order to define these subproblems we use the information provided by the unsatisfiable cores we obtain during the solving process. The improved WPM2 algorithm is parametric on the approach we use to solve these subproblems. This allows to combine the strength of exploiting the information extracted from unsatisfiable cores and other optimization approaches. By solving these smaller optimization problems we get the most significant boost in our improved WPM2 algorithm. In particular, we experiment with an Integer Linear Programming (ILP) approach, corresponding to the strategy shown in Sect. 3, and three MaxSAT approaches: (i) refine the lower bound for these subproblems with the *subsetsum* function (Cormen et al. 2009; Ansotegui et al. 2010), (ii) refine the upper bound with the strategy applied in minisat+ (Eén and Sörensson 2006), SAT4J (Berre 2006), qmaxsat (Koshimura et al. 2012) or ShinMaxSat (Honjyo and Tanjo 2012), and (iii) a binary search scheme (Heras et al. 2011; Cimatti et al. 2010; Fu and Malik 2006) where the lower bound and upper bound are refined as in the previous approaches. The best performing approach in our experimental analysis is the second one and it allows to solve up to 296 additional instances. As a summary, the overall increase in performance we achieved so far compared to the original WPM2 is about 464 additional solved instances, or 48 % more. These improvements led WPM2 to be the overall best performing approach on the industrial instances of the MSE13.

To further explain the good results of our approach we include and extend the study originally presented in Ansotegui (2013a) on the structure of the unsatisfiable cores obtained during the search process of SAT-based algorithms (see Sect. 6). We explain how the improved WPM2 algorithm takes advantage of this structure.

We have also explored how we can exploit information retrieved from the subproblems that are solved. When the strategy used to optimize the subproblems is able to produce satisfying assignments, i.e., it refines the upper bound (see approaches (ii) and (iii) above), we can use these assignments as a heuristic to guide and boost the search. This improvement allows to solve 50 additional instances. The overall increase in performance compared to the original WPM2 is 514 additional solved instances. Actually, if we take into account the timeout of 7200 s (2 h) used in our experiments, we obtain an overall speed-up of about three orders of magnitude (see Sect. 6). Moreover, we show that high quality satisfying assignments can be obtained in a reasonably short time, giving us naturally an incomplete approach. Our experimental results confirm that the incomplete version of our exact improved WPM2 algorithm would have dominated the track for incomplete solvers at the MSE13. Furthermore, at MSE14, even though it was not the best complete approach, it dominates the others as incomplete.

From the perspective of coming up with an efficient implementation of our approach, it is obvious that SAT-based MaxSAT algorithms have to be implemented on top of an efficient solver based on SAT technology. This solver has to be capable of returning an unsatisfiable core when the input instance is unsatisfiable and a satisfying assignment when the instance is satisfiable. Moreover, SAT-based MaxSAT algorithms require the addition of linear PB constraints as a result of the reformulation process of the original problem into a sequence of SAT instances. These PB constraints are used to bound the cost of the optimal assignment. Currently, in most state-of-the-art SAT-based MaxSAT solvers, PB constraints are translated into SAT. However, there is no known SAT encoding which can guarantee the original propagation power of the constraint, i.e, what we call arc-consistency, while keeping the translation low in size. The best approach so far, has a cubic complexity (Bailleux et al. 2009). This can be a bottleneck for WPM2 (Ansotegui et al. 2010) and also for other algorithms such as, BINCD (Heras et al. 2011) or SAT4J (Berre 2006).

In order to treat PB constraints with specialized inference mechanisms and a moderate cost in size, while preserving the strength of SAT techniques for the rest of the problem, we use the SMT technology (Sebastiani 2007) (see Sect. 5). Related work in this sense can be found in Nieuwenhuis and Oliveras (2006). Also, in Ansótegui et al. (2011) a Weighted Constraint Satisfaction Problems (WCSP) solver implementing the original WPM1 (Ansotegui et al. 2009) algorithm is presented.

Finally, we have also seen the development of successful methods for solving combinatorial problems by applying techniques from Operations Research. Although the literature shows us that some NP-hard problems are more suitable for logic-based approaches while others are more efficiently solved with integer programming techniques, our current study would not be exhaustively conducted without showing the performance of integer programming techniques on MaxSAT problems. Actually, we can easily reformulate the WPMS problem into an ILP problem (see Sect. 3) and apply an ILP approach. We show that the ILP approach is not competitive on the industrial instances where a logic-based approach like the one proposed here is more efficient.

This paper proceeds as follows. Section 2 formally defines the main concepts that are used in the paper. Section 3 presents the translation of WPMS into ILP. Section 4 describes the WPM2 algorithm and the new improvements. Section 5 describes the SMT problem and discusses some implementation details of the SMT-based MaxSAT algorithms. Section 6 presents the experimental evaluation. Finally, Sect. 7 shows the conclusions and the future work.

2 Preliminaries

Definition 1 A *literal* l is either a Boolean variable x or its negation $\overline{x}$. A *clause* c is a disjunction of literals. A *SAT formula* is a set of clauses that represents a Boolean formula in Conjunctive Normal Form (CNF), i.e., a conjunction of clauses.

Definition 2 A *weighted clause* is an ordered pair (c, w), where c is a clause and w is a natural number or infinity (indicating the cost of falsifying c, see Definitions 5 and 6). If w is infinite the clause is *hard*, otherwise it is *soft*.

Definition 3 A *Weighted Partial MaxSAT (WPMS) formula* is an ordered multiset of weighted clauses:

$$\varphi = \langle (c_1, w_1), \dots, (c_s, w_s), (c_{s+1}, \infty), \dots, (c_{s+h}, \infty) \rangle$$

where the first s clauses are soft and the last h clauses are hard. The presence of soft clauses with different weights makes the formula Weighted and the presence of hard clauses makes it Partial. The ordered multiset of weights of the soft clauses in the formula is noted as $w(\varphi)$. The top weight of the formula is noted as $W(\varphi)$, and defined as $W(\varphi) = \sum w(\varphi) + 1$. The set of indexes of soft clauses is noted as $S(\varphi)$ and the set of indexes of hard clauses is noted as $H(\varphi)$. When it is clear to which formula φ these soft (hard) clauses belong, we also refer to these sets of indexes as S (H). Finally, the set of variables occurring in the formula is noted as $\mathrm{var}(\varphi)$.

Example 1 Given the WPMS formula $\varphi = \langle (x_1, 5), (x_2, 3), (x_3, 3), (\overline{x}_1 \vee \overline{x}_2, \infty), (\overline{x}_1 \vee \overline{x}_3, \infty), (\overline{x}_2 \vee \overline{x}_3, \infty) \rangle$, we have that $w(\varphi) = \langle 5, 3, 3 \rangle$, $W(\varphi) = 12$, $S(\varphi) = \{1, 2, 3\}$, $H(\varphi) = \{4, 5, 6\}$ and $\mathrm{var}(\varphi) = \{x_1, x_2, x_3\}$.

Definition 4 Given a WPMS formula φ and a set of indexes A, φ_A is the WPMS formula that contains the clauses (c_i, w_i) such that $i \in A$. By φ_S we refer the set of soft clauses and by φ_H to the set of hard clauses.

Example 2 Given the WPMS formula φ of Example 1, we have that $\varphi_{\{1,3.5\}} = \langle (x_1, 5), (x_3, 3), (\overline{x}_1 \vee \overline{x}_3, \infty) \rangle$, $\varphi_S = \langle (x_1, 5), (x_2, 3), (x_3, 3) \rangle$ and $\varphi_H = \langle (\overline{x}_1 \vee \overline{x}_2, \infty), (\overline{x}_1 \vee \overline{x}_3, \infty), (\overline{x}_2 \vee \overline{x}_3, \infty) \rangle$.

Definition 5 An *assignment* for a set of Boolean variables X is a function $\mathcal{I} : X \rightarrow \{0, 1\}$, that can be extended to literals, (weighted) clauses, SAT formulas and WPMS formulas as follows:

$$\begin{aligned}
&\mathcal{I}(\overline{x}) = 1 - \mathcal{I}(x)\\
&\mathcal{I}(l_1 \vee \ldots \vee l_m) = \max\{\mathcal{I}(l_1), \ldots, \mathcal{I}(l_m)\}\\
&\mathcal{I}(\{c_1, \ldots, c_n\}) = \min\{\mathcal{I}(c_1), \ldots, \mathcal{I}(c_n)\}\\
&\mathcal{I}((c, w)) = w\,(1 - \mathcal{I}(c))\\
&\mathcal{I}(\langle (c_1, w_1), \ldots, (c_{s+h}, w_{s+h})\rangle) = \textstyle\sum_{i=1}^{s+h} \mathcal{I}((c_i, w_i))
\end{aligned}$$

We will refer to the value returned by an assignment $\mathcal{I}$ on a weighted clause or a WPMS formula as the cost of $\mathcal{I}$.

Given a WPMS formula φ and a set of indexes A, we will refer to $\mathcal{I}_A$ as an assignment for φ_A.

Definition 6 We say that an assignment $\mathcal{I}$ satisfies a clause or a SAT formula if the value returned by $\mathcal{I}$ is equal to 1. In the case of SAT formulas, we will refer also to this assignment as a satisfying assignment or solution. Otherwise, if the value returned by $\mathcal{I}$ is equal to 0, we say that $\mathcal{I}$ falsifies the clause or the SAT formula.

Definition 7 The *SAT problem* for a SAT formula φ is the problem of finding a solution for φ. If a solution exists the formula is satisfiable, otherwise it is unsatisfiable.

Definition 8 Given an unsatisfiable SAT formula φ, an *unsatisfiable core* φ_C is a subset of clauses $\varphi_C \subseteq \varphi$ that is also unsatisfiable. A *minimal unsatisfiable core* is an unsatisfiable core such that any proper subset of it is satisfiable.

Example 3 Given the SAT formula: $\varphi = \{(x_1), (x_2), (x_3), (\overline{x}_1 \vee \overline{x}_2), (\overline{x}_1 \vee \overline{x}_3), (\overline{x}_2 \vee \overline{x}_3)\}$ we have that $\{(x_1), (x_2), (x_3), (\overline{x}_1 \vee \overline{x}_2)\} \subseteq \varphi$ is an unsatisfiable core and $\{(x_1), (x_2), (\overline{x}_1 \vee \overline{x}_2)\} \subseteq \varphi$ is a minimal unsatisfiable core.

Definition 9 A SAT algorithm for the SAT problem, takes as input a SAT formula φ and returns an assignment $\mathcal{I}$ such that $\mathcal{I}(\varphi) = 1$ if the formula is satisfiable. Otherwise, it returns an unsatisfiable core φ_C.

Given unlimited resources of time and memory, we say that a SAT algorithm is *complete* if it terminates for any SAT formula. Otherwise, we say that it is *incomplete*.

Definition 10 The *optimal cost (or optimum)* of a WPMS formula φ is $\text{cost}(\varphi) = \min\{\mathcal{I}(\varphi) \mid \mathcal{I} : \text{var}(\varphi) \rightarrow \{0, 1\}\}$ and an *optimal assignment* is an assignment $\mathcal{I}$ such that $\mathcal{I}(\varphi) = \text{cost}(\varphi)$. We will refer to this assignment as a solution for φ if $\mathcal{I}(\varphi) \neq \infty$. Any cost above (below) *cost* (φ) is called an upper (lower) bound for φ.

Example 4 Given the WPMS formula φ of Example 1, we have that $\text{cost}(\varphi) = \min\{6, 8, 11, \infty\} = 6$ and the optimal assignment $\mathcal{I}$ maps $\langle x_1, x_2, x_3\rangle$ to $\langle 1, 0, 0\rangle$.

Definition 11 The *Weighted Partial MaxSAT problem* for a WPMS formula φ is the problem of finding a solution for φ. If a solution does not exist the formula is unsatisfiable.

Definition 12 A WPMS algorithm for the WPMS problem, takes as input a WPMS formula φ and returns an assignment $\mathcal{I}$, such that, $\mathcal{I}(\varphi) \geq cost(\varphi)$.

Given unlimited resources of time and memory, we say that a WPMS algorithm is *complete or exact* if for any input WPMS formula φ and returned $\mathcal{I}$, $\mathcal{I}(\varphi) = cost(\varphi)$. Otherwise, we say it is *incomplete*.

Definition 13 An *integer linear Pseudo-Boolean (PB) constraint* is an inequality of the form $w_1 x_1 + \cdots + w_n x_n \; op \; k$, where $op \in \{\leq, \geq, =, >, <\}$, k and w_i are integer coefficients, and x_i are Boolean variables. A *cardinality constraint* is a PB constraint where the coefficients w_i are equal to 1. A PB *At-Most (At-Least) constraint* is a PB constraint where op is $\leq$ ($\geq$).

Example 5 $5x_1 + 3x_2 + 3x_3 \leq 6$ is a PB At-Most constraint and $5x_1 + 3x_2 + 3x_3 \geq 6$ is PB At-Least constraint.

3 Translation of weighted partial MaxSAT into ILP

In order to solve a WPMS problem, a first reasonable approach consists in reformulating the WPMS problem as an ILP problem and applying an ILP solver. Several encodings can be found in the literature (Li and Manyà 2009; Manquinho et al. 2009; Ansótegui and Gabàs 2013; Davies and Bacchus 2013).

Here, we describe the precise encoding we used in Ansótegui and Gabàs (2013). Given a WPMS formula, $\langle (c_1, w_1), \ldots, (c_s, w_s), (c_{s+1}, \infty), \ldots, (c_{s+h}, \infty) \rangle$, we can translate it into an ILP instance, as follows:

$$
\begin{aligned}
&\text{Minimize:} \sum\nolimits_1^s w_i \cdot b_i \\
&\text{Subject to:} \\
&ILP(\varphi'_H \cup \varphi'_S) \\
&0 \leq v_i \leq 1, \; v_i \in var(\varphi'_H \cup \varphi'_S)
\end{aligned}
$$

The *Minimize* section of the ILP formulation defines the objective function of the problem. This corresponds to the aggregated cost of the falsified soft clauses in the WPMS formula we want to minimize. In order to identify which soft clauses are falsified by a given assignment to the original x_i variables of the problem, we introduce an indicator variable b_i for each soft clause (c_i, w_i). These b_i are also known as reification, relaxation or blocking variables.

The *Subject to* section includes the constraints that have to be satisfied under any assignment. This corresponds to the original set of hard clauses of the WPMS formula represented by $\varphi'_H = \cup_{j=s+1}^{s+h} c_j$, and the set of clauses connecting the soft clauses of the WPMS formula with their respective indicator variable represented by $\varphi'_S = \cup_{i=1}^{s} CNF(\overline{c}_i \leftrightarrow b_i)$. Notice that by enforcing consistency, we ensure that b_i is true iff the soft clause c_i is falsified. Function $CNF(\varphi)$ transforms φ into Conjunctive Normal Form while function $ILP(\varphi)$ maps every clause $c_i \in \varphi$ into a linear inequality with operator $>$. The left-hand side of that linear inequality corresponds to the sum of the literals in c_i once mapped into integer terms, such that, literal $x(\overline{x})$ is mapped to integer term $x(1-x)$. The right-hand side corresponds to constant 0. After moving the constants to the right, the right-hand side corresponds to constant $-k$, where k is the number of negative literals in clause c_i. Finally, we add the bounding constraints that ensure that every integer variable in the ILP formulation has domain $\{0, 1\}$. It can be easily seen that, since we are minimizing, the implication $b_i \rightarrow \overline{c}_i$ from $\overline{c}_i \leftrightarrow b_i$ is unnecessary. Notice that, by the same nature of the minimization problem, variables b_i will be 0 (false) whenever it is possible and we do not need the part of the double implication that ensures that $c_i \rightarrow \overline{b}_i$, i.e., $b_i \rightarrow \overline{c}_i$.

Example 6 Given the WPMS formula $\varphi = \langle (x_1 \vee x_2, 2), (x_1 \vee \overline{x}_2, 3), (\overline{x}_1 \vee x_2, \infty), (\overline{x}_1 \vee \overline{x}_2, \infty) \rangle$, the corresponding ILP formulation is:

Minimize: $2 \cdot b_1 + 3 \cdot b_2$

Subject to:

$$\begin{array}{lll}
x_1 + x_2 + b_1 > 0; & \# x_1 \vee x_2 \vee b_1 & \equiv \overline{(x_1 \vee x_2)} \rightarrow b_1 \\
\quad -x_1 - b_1 > -2; & \quad \# \overline{x}_1 \vee \overline{b}_1 & \\
\quad -x_2 - b_1 > -2; & \quad \# \overline{x}_2 \vee \overline{b}_1 & \equiv b_1 \rightarrow \overline{(x_1 \vee x_2)} \\
x_1 - x_2 + b_2 > -1; & \# x_1 \vee \overline{x}_2 \vee b_2 & \equiv \overline{(x_1 \vee \overline{x}_2)} \rightarrow b_2 \\
\quad -x_1 - b_2 > -2; & \quad \# \overline{x}_1 \vee \overline{b}_2 & \\
\quad x_2 - b_2 > -1; & \quad \# x_2 \vee \overline{b}_2 & \equiv b_2 \rightarrow \overline{(x_1 \vee \overline{x}_2)} \\
-x_1 + x_2 > -1; & \# \overline{x}_1 \vee x_2 & \\
-x_1 - x_2 > -2; & \# \overline{x}_1 \vee \overline{x}_2 &
\end{array}$$

$0 \leq x_1 \leq 1;\ 0 \leq x_2 \leq 1;\ 0 \leq b_1 \leq 1;\ 0 \leq b_2 \leq 1;$

4 Original WPM2 algorithm and improvements

In this section, we present the complete SAT-based MaxSAT algorithm WPM2 (Ansotegui et al. 2010) for the WPMS problem and how it has been improved.

At a high description level, given an input WPMS formula φ, the original WPM2 algorithm (Ansotegui et al. 2010), described in Algorithm 1, iteratively calls a SAT solver querying whether there is an assignment to φ with a cost less than or equal to a certain k. The initial value of k is 0 and the last value is exactly the optimal cost of φ, i.e., $cost(\varphi)$. Notice that all SAT queries with a $k < cost(\varphi)$ must have a negative answer while all the queries with $k \geq cost(\varphi)$ must have a positive answer. Therefore, our optimization problem can be reformulated as identifying where the phase transition from negative answers to positive answers occurs.

More formally, the query sent to the SAT solver is a SAT formula φ^k that is satisfiable iff there is an assignment, say $\mathcal{I}$, such that $\mathcal{I}(\varphi) \leq k$. In order to construct such a formula, we need to detect which soft clauses are falsified under a certain assignment $\mathcal{I}$, sum up their cost and compare with k.

To detect if a clause is falsified, we extend every soft clause (c_i, w_i) with a unique auxiliary Boolean indicator variable b_i obtaining $(c_i \vee b_i, w_i)$. Notice that, if c_i is false, then in order to maintain consistency b_i must be true. Therefore, these b_i variables work as indicator variables that become true if a clause c_i is falsified.

To add the weights of the falsified soft clauses and compare the cost with k we use PB constraints that are translated into a SAT formula.

Example 7 Before we go into detail on algorithm WPM2, let us describe how a naive SAT-based MaxSAT algorithm would work. Let us consider a simple WPMS formula representing the Weak Pigeon-Hole Problem (Razborov 2001; Raz 2002) of 5 pigeons and 1 hole. Variable x_i is true if the ith pigeon is in the hole.

$$\varphi = \langle (x_1, 5), (x_2, 5), (x_3, 3), (x_4, 3), (x_5, 1) \rangle \cup \langle CNF(\sum x_i \leq 1, \infty) \rangle$$

The weight of the soft clauses indicates the cost of not having the ith pigeon in the hole and the hard clauses indicate that at most one pigeon can be in the hole. A naive SAT-based MaxSAT solver would search for the optimal cost, solving a sequence of φ^k SAT instances which are satisfiable iff there is an assignment to φ, i.e., to the x_i variables, with a cost less than or equal to k. In order to build such φ^k SAT instances, all the soft clauses are extended and the PB At-Most constraint, $5 \cdot b_1 + 5 \cdot b_2 + 3 \cdot b_3 + 3 \cdot b_4 + 1 \cdot b_5 \leq k$, is used to bound the aggregated cost of the falsified clauses.

$$\begin{aligned}\varphi^k = \{ & (x_1 \vee b_1), (x_2 \vee b_2), (x_3 \vee b_3), (x_4 \vee b_4), (x_5 \vee b_5) \} \cup CNF(\sum x_i \leq 1) \cup \\ & CNF(5 \cdot b_1 + 5 \cdot b_2 + 3 \cdot b_3 + 3 \cdot b_4 + 1 \cdot b_5 \leq k)\end{aligned}$$

Since $cost(\varphi) = 12$, we know that the SAT instances between φ^0 to φ^{11} are unsatisfiable, while the SAT instances between φ^{12} to φ^{18} (the top weight $W(\varphi) = 18$) are satisfiable. A naive binary search between 0 and 18 would check the satisfiability of this sequence: $\varphi^9, \varphi^{13}, \varphi^{11}$ and φ^{12}. Since φ^{11} and φ^{12} are unsatisfiable and satisfiable, respectively, we have found the phase transition point and we can conclude that the optimal cost is 12.

The first characteristic of the original WPM2 algorithm is that it works with two sets of PB constraints: AM and AL. The set AM of PB (At-Most) constraints are used to bound the aggregated cost of the falsified soft clauses. In particular, AM is a set of PB constraints that bound the cost of non-overlapping subsets of soft clauses. More precisely, an $am \in AM$ is a PB constraint of the form $\sum_{i \in A} w_i\, b_i \leq k$ where A is a subset of the indexes of the soft clauses, b_i and w_i are the corresponding indicator variables and weights of the ith soft clause and k the concrete bound for that subset of soft clauses. When describing algorithms, we will use the object oriented programming notation $am.A$ and $am.k$ to refer to the set A and integer k related to an At-Most constraint $am \in AM$. The idea of having multiple and smaller PB At-Most constraints instead of a single one was introduced in Ansótegui et al. (2009).

The set AL of PB (At-Least) constraints are used to impose that the aggregated cost of the falsified clauses in a given subset of soft clauses must be at least some natural number. These are redundant constraints and are not necessary for the soundness of the algorithm but help to improve the performance of the SAT solver. More precisely, an $al \in AL$ is a PB constraint of the form $\sum_{i \in A} w_i\, b_i \geq k$.

For the sake of space and readability, we will use the notation $\langle A, w(\varphi_A), \leq, k\rangle$ and At-Most constraint instead of $\sum_{i \in A} w_i\, b_i \leq k$ and PB At-Most constraint in the algorithms. Mutatis mutandis for the PB At-Least constraints.

In the following, we go into detail on the original WPM2 algorithm (Algorithm 1). First of all, we check whether the set of hard clauses (φ_H) is satisfiable (Algorithm 1 line 1). If it is unsatisfiable, we can already stop since there is no solution to φ. Otherwise, the main loop of the algorithm starts (Algorithm 1 line 3). Notice that we exit this loop iff the *sat* function returns satisfiable. In that case, we have found a solution.

Algorithm 1: Original WPM2.

```
Input: $\varphi = \langle (c_1, w_1), \ldots, (c_s, w_s), (c_{s+1}, \infty), \ldots, (c_{s+h}, \infty) \rangle$
1:  if $\langle \text{UNSAT}, \_, \_ \rangle = sat(\varphi_H, \_, \_)$ then return $\langle \infty, \emptyset \rangle$
2:  $\langle AL, AM \rangle := \langle \emptyset, \emptyset \rangle$
3:  while $true$ do
4:      $\langle st, C, \mathcal{I} \rangle := sat(\varphi, AL, AM)$
5:      if $st = \text{SAT}$ then
6:          return $\langle \mathcal{I}(\varphi), \mathcal{I} \rangle$
7:      else
8:          $\langle A, k \rangle := optimize(\varphi_S, AL, AM, C)$
9:          $AL := \{\langle A, w(\varphi_A), \geq, k \rangle\} \cup AL$
10:         $AM := \{\langle A, w(\varphi_A), \leq, k \rangle\} \cup AM \setminus \{\underset{am.A \subseteq A}{am \in AM}\}$
```

Function sat(φ, AL, AM)

```
1:  $\langle A, k \rangle := \langle \bigcup_{am \in AM} am.A, \sum am.k \rangle$
2:  $\varphi^k := \{\varphi.c_i\}_{i \notin A} \cup \{\varphi.c_i \vee b_i\}_{i \in A} \cup CNF(AL \cup AM)$
3:  $\langle st, \varphi^k_C, \mathcal{I} \rangle := satsolver(\varphi^k)$
4:  $C := \{i \in S(\varphi) \mid (\varphi.c_i \in \varphi^k_C) \vee (\varphi.c_i \vee b_i \in \varphi^k_C)\}$
5:  return $\langle st, C, \mathcal{I} \rangle$
```

Function optimize(φ, AL, AM, C)

```
1:  $A := \bigcup_{am \in AM,\; am.A \cap C \neq \emptyset} am.A \;\cup C$
2:  $k := lb := subsetsum(w(\varphi_A), \sum_{am \in AM,\; am.A \subseteq A} am.k + 1)$
3:  while $true$ do
4:      $\langle st, \_, \_ \rangle := sat(\_, AL, \{\langle A, w(\varphi_A), \leq, k \rangle\})$
5:      if $st = \text{SAT}$ then return $\langle A, lb \rangle$
6:      else $k := lb := subsetsum(w(\varphi_A), k + 1)$
```

The *sat* function (Algorithm 1 line 4) builds the SAT formula at the current iteration and sends it to the SAT solver. As we can see, a SAT formula φ^k is built by extending soft clauses with indicator variables and aggregating to φ^k the conversion to *CNF* of the PB constraints into the sets AL and AM (*sat* line 2). Actually, only a subset of the soft clauses is relaxed, i.e., extended with indicator variables b_i. In particular, those that have appeared previously in some unsatisfiable core. This is done because of efficiency issues.

The SAT solver outputs a triplet $\langle st, \varphi_C^k, \mathcal{I}\rangle$ (*sat* line 3). If the SAT solver returns *satisfiable* ($st = \text{SAT}$), φ_C^k is empty and $\mathcal{I}$ is the satisfying assignment (solution) found by the SAT solver. If the SAT solver returns *unsatisfiable* ($st = \text{UNSAT}$), φ_C^k is the unsatisfiable core found by the solver and $\mathcal{I}$ is empty. Finally, the *sat* function returns the status, st, the indexes of the soft clauses in the unsatisfiable core C (*sat* line 4) if $st = \text{UNSAT}$, and the satisfying assignment $\mathcal{I}$ if $st = \text{SAT}$.

When the *sat* function returns $st = \text{SAT}$ (Algorithm 1 line 5) we return the optimal cost and the optimal assignment (Algorithm 1 line 6) and the algorithm ends. When the *sat* function returns $st = \text{UNSAT}$ (Algorithm 1 line 7) we analyze the unsatisfiable core returned by the solver and we compute, within the *optimize* function (Algorithm 1 line 8), which is the set of indexes of soft clauses and bound $\langle A, k\rangle$ to construct the new At-Most constraint $\langle A, w(\varphi_A), \leq, k\rangle$ and update the AL and AM sets (Algorithm 1 lines 9 and 10). Technically speaking, the AL constraints give lower bounds on $cost(\varphi)$, while the AM constraints enforce that all solutions of the set of constraints $AL \cup AM$ are the solutions of AL of minimal cost. This ensures that any solution returned by $sat(\varphi, AL, AM)$, if there is any, has to be an optimal assignment of φ.

Within the *optimize* function, basically, we check which non-relaxed soft clauses and At-Most constraints on relaxed clauses are involved in the unsatisfiable core. The union of the indexes of all these clauses, set A (*optimize* line 1), gives us the indexes of the new At-Most constraint. Notice that the non-relaxed soft clauses involved in A will be relaxed in the next step within the *sat* function (*sat* line 2). In order to finish the building process of the new At-Most constraint, we have to compute its independent term k (*optimize* lines 2–6). Intuitively, the next k has to be the next lower bound candidate for the subproblem defined by the soft clauses related to A, i.e., φ_A . Notice that the previous candidate was the sum of the k's ($\sum_{am.A \subseteq A}^{am \in AM} am.k$) of the At-Most constraints involved into the unsatisfiable core, which were proven by the SAT solver to be too restrictive. In order to obtain the next candidate, we need to find the minimum integer k that satisfies the next conditions: (i) k is a linear combination of the weights involved in φ_A, (ii) k is greater than or equal to ($\sum_{am.A \subseteq A}^{am \in AM} am.k) + 1$ and (iii) the new At-Most constraint $\langle A, w(\varphi_A), \leq, k\rangle$ is consistent with the set of constraints in AL.

As we can see in the *optimize* function, (i) and (ii) are enforced by the *subsetsum* function (*optimize* lines 2 and 6) while (iii) is checked by the *sat* function (*optimize* line 4). The main idea is that the subset sum problem (Cormen et al. 2009) is progressively solved until we get a solution that also satisfies the AL constraints.

Actually, the *optimize* function represents the following optimization problem:

$$\begin{array}{ll}
\text{Minimize } \sum_{i \in A} w_i \cdot b_i & \text{(i)} \\
\text{Subject to:} & \\
\sum_{i \in A} w_i \cdot b_i \geq k' & \text{(ii)} \\
AL & \text{(iii)} \\
0 \leq b_i \leq 1, i \in A &
\end{array}$$

where $k' = (\sum_{am.A \subseteq A}^{am \in AM} am.k) + 1$. Notice that, by removing the AL constraints, we get the subset sum problem.

Finally, once we obtain $\langle A, k\rangle$ (Algorithm 1 line 8) to construct the new At-Most constraint $\langle A, w(\varphi_A), \leq, k\rangle$, we update the AL and AM sets. Basically, we replace in the AM set the At-Most constraints whose respective soft clauses are already considered in the new At-Most constraint (Algorithm 1 line 10). Notice that the new At-Most constraint enforces that the cost of any assignment to the subproblem φ_A has at most cost k. Optionally, we extend the AL set with an additional redundant constraint stating that the cost of any assignment to the subproblem φ_A has to be at least k (Algorithm 1 line 9).

For further information and detailed proofs on the soundness and completeness of the original WPM2 algorithm see Ansotegui et al. (2010).

Example 8 The original WPM2 algorithm performs the following iterations on the pigeon-hole formula presented in Example 7.

$$\varphi = \langle (x_1, 5), (x_2, 5), (x_3, 3), (x_4, 3), (x_5, 1)\rangle \cup \langle CNF(\sum x_i \leq 1, \infty)\rangle$$

In the first iteration, the SAT formula φ^0 is sent to the SAT solver within the *sat* function. The SAT solver certifies that it is unsatisfiable, and returns an unsatisfiable core (noted with dots •) that involves the soft clauses 1 and 3, and the set of hard clauses. Semantically speaking, this core tells us that pigeons 1 and 3 can not be at the same time in the hole. The *optimize* function computes the new At-Most constraint ($5b_1 + 3b_3 \leq 3$) that corresponds to the subproblem $\varphi_{\{1,3\}}$. Notice that the corresponding At-Most constraint for a subproblem restricts to k the aggregated cost of its falsified soft clauses. Obviously, the first optimal candidate k for $\varphi_{\{1,3\}}$ is 3 since we will try first to leave out the pigeon with the minimum weight. Then, the soft clauses are relaxed with an additional variable and the corresponding constraints are added to AL and AM. These changes (noted with triangles ◀) transform φ^0 into φ^3. The second iteration is similar to the first one, but with the soft clauses 2 and 4. The new At-Most constraint computed by the *optimize* function is ($5b_2 + 3b_4 \leq 3$).

Iteration 1

$$\begin{array}{ll} \varphi^0 = \{ & (x_1), \bullet \\ & (x_2), \\ & (x_3), \bullet \\ & (x_4), \\ & (x_5)\} \cup \\ & CNF(\sum x_i \leq 1) \bullet \end{array} \qquad \begin{array}{ll} \varphi^3 = \{ & (x_1 \vee b_1), \blacktriangleleft \\ & (x_2), \\ & (x_3 \vee b_3), \blacktriangleleft \\ & (x_4), \\ & (x_5)\} \cup \\ & CNF(\sum x_i \leq 1) \cup \\ & CNF(5b_1 + 3b_3 \geq 3) \blacktriangleleft \cup \\ & CNF(5b_1 + 3b_3 \leq 3) \blacktriangleleft \end{array}$$

$sat(\varphi, AL, AM) = \langle \text{UNSAT}, \{1,3\}, \emptyset\rangle;\ \mathcal{I}(\varphi) > 0;$ $\quad optimize(\varphi_S, AL, AM, \{1,3\}) = \langle \{1,3\}, 3\rangle;$

Iteration 2

$$\begin{array}{ll} \varphi^3 = \{ & (x_1 \vee b_1), \\ & (x_2), \bullet \\ & (x_3 \vee b_3), \\ & (x_4), \bullet \\ & (x_5)\} \cup \\ & CNF(\sum x_i \leq 1) \bullet \\ & CNF(5b_1 + 3b_3 \geq 3) \cup \\ & CNF(5b_1 + 3b_3 \leq 3) \end{array} \qquad \begin{array}{ll} \varphi^6 = \{ & (x_1 \vee b_1), \\ & (x_2 \vee b_2), \blacktriangleleft \\ & (x_3 \vee b_3), \\ & (x_4 \vee b_4), \blacktriangleleft \\ & (x_5)\} \cup \\ & CNF(\sum x_i \leq 1) \cup \\ & CNF(5b_1 + 3b_3 \geq 3) \cup \\ & CNF(5b_2 + 3b_4 \geq 3) \blacktriangleleft \cup \\ & CNF(5b_1 + 3b_3 \leq 3) \cup \\ & CNF(5b_2 + 3b_4 \leq 3) \blacktriangleleft \end{array}$$

$sat(\varphi, AL, AM) = \langle \text{UNSAT}, \{2,4\}, \emptyset\rangle;\ \mathcal{I}(\varphi) > 3;$ $\quad optimize(\varphi_S, AL, AM, \{2,4\}) = \langle \{2,4\}, 3\rangle;$

In the third iteration, we get an unsatisfiable core for φ^6 with the soft clauses 1 and 2, the two At-Most constraints from previous subproblems and the set of hard clauses. The *optimize* function returns the subset of indexes that defines the new subproblem $\varphi_{\{1,2,3,4\}}$, with the soft clauses from the core and from the previous subproblems that intersect with it. It also returns the next optimal candidate for $\varphi_{\{1,2,3,4\}}$, $k = 8$, which satisfies the three conditions: (i) it is a linear combination of the weights of $\varphi_{\{1,2,3,4\}}$, (ii) it is greater than or equal to the addition of the subsumed previous subproblem optimal candidates plus one $(3 + 3 + 1)$ and (iii) its corresponding At-Most constraint $(5b_1 + 5b_2 + 3b_3 + 3b_4 \leq 8)$ is consistent with the set of constraints in AL. We have to replace the two At-Most constraints involved in the core with the new one, that is less restrictive. Notice that the previous constraints only allowed the two pigeons with weight 3 to be out of the hole. On the other hand, the current constraint allows both one pigeon with weight 3 and one with weight 5 to be out of the hole. However, this is not enough to solve the subproblem since at-least three pigeons should be out of the hole.

Iteration 3

$$
\begin{array}{ll}
\varphi^6 = \{ & (x_1 \vee b_1), \bullet \\
 & (x_2 \vee b_2), \bullet \\
 & (x_3 \vee b_3), \\
 & (x_4 \vee b_4), \\
 & (x_5) \} \cup \\
 & CNF(\sum x_i \leq 1) \bullet \cup \\
 & CNF(5b_1 + 3b_3 \geq 3) \cup \\
 & CNF(5b_2 + 3b_4 \geq 3) \cup \\
 & CNF(5b_1 + 3b_3 \leq 3) \bullet \\
 & CNF(5b_2 + 3b_4 \leq 3) \bullet
\end{array}
$$

$$
\begin{array}{ll}
\varphi^8 = \{ & (x_1 \vee b_1), \\
 & (x_2 \vee b_2), \\
 & (x_3 \vee b_3), \\
 & (x_4 \vee b_4), \\
 & (x_5) \} \cup \\
 & CNF(\sum x_i \leq 1) \cup \\
 & CNF(5b_1 + 3b_3 \geq 3) \cup \\
 & CNF(5b_2 + 3b_4 \geq 3) \cup \\
 & CNF(5b_1 + 5b_2 + 3b_3 + 3b_4 \geq 8) \blacktriangleleft \cup \\
 & CNF(5b_1 + 5b_2 + 3b_3 + 3b_4 \leq 8) \blacktriangleleft
\end{array}
$$

$sat(\varphi, AL, AM) =$
$\langle \text{UNSAT}, \{1, 2\}, \emptyset \rangle;$
$\mathcal{I}(\varphi) > 6;$

$optimize(\varphi_S, AL, AM, \{1, 2\}) =$
$\langle \{1, 2, 3, 4\}, 8 \rangle;$

In the fourth and the fifth iteration the *sat* function provides a core involving the soft clauses in $\varphi_{\{1,2,3,4\}}$. The *optimize* function provides new bounds 10 and 11, respectively. This last bound does allow three pigeons to be out of the hole. However, this is not yet enough to solve the whole problem.

Iteration 4

$sat(\varphi, AL, AM) =$
$\langle \text{UNSAT}, \{1, 2, 3, 4\}, \emptyset \rangle;$
$\mathcal{I}(\varphi) > 8;$

$optimize(\varphi_S, AL, AM, \{1, 2, 3, 4\}) =$
$\langle \{1, 2, 3, 4\}, 10 \rangle;$

Iteration 5

$sat(\varphi, AL, AM) =$
$\langle \text{UNSAT}, \{1, 2, 3, 4\}, \emptyset \rangle;$
$\mathcal{I}(\varphi) > 10;$

$optimize(\varphi_S, AL, AM, \{1, 2, 3, 4\}) =$
$\langle \{1, 2, 3, 4\}, 11 \rangle;$

In the sixth iteration, we get a core involving all the soft clauses, the At-Most constraint and the set of hard clauses. The At-Most constraint corresponding to subproblem $\varphi_{\{1,2,3,4\}}$, allows three pigeons out of the hole, but one is still in the hole. Since pigeon 5 is also in the hole, there is a conflict. We get the new constraint involving all the soft clauses and the new bound 12. In the seventh iteration, we get that φ^{12} is satisfiable and the original WPM2 algorithm ends. Notice that indeed 12 is the minimum cost that allows four pigeons out of the hole.

Iteration 6

$$\begin{aligned}\varphi^{11} = \{\ & (x_1 \vee\ b_1\ ),\ \bullet\\ & (x_2 \vee\ b_2\ ),\ \bullet\\ & (x_3 \vee\ b_3\ ),\ \bullet\\ & (x_4 \vee\ b_4\ ),\ \bullet\\ & (x_5 \qquad)\ \bullet\,\} \cup\\ & CNF(\textstyle\sum x_i \leq 1)\ \bullet \cup\\ & CNF(5b_1 + 3b_3 \geq 3) \cup\\ & CNF(5b_2 + 3b_4 \geq 3) \cup\\ & CNF(5b_1 + 5b_2 + 3b_3 + 3b_4 \geq 8) \cup\\ & CNF(5b_1 + 5b_2 + 3b_3 + 3b_4 \geq 10) \cup\\ & CNF(5b_1 + 5b_2 + 3b_3 + 3b_4 \geq 11) \cup\\ & CNF(5b_1 + 5b_2 + 3b_3 + 3b_4 \leq 11)\ \bullet\end{aligned}$$

$sat(\varphi, AL, AM) =$

$\langle \text{UNSAT}, \{1, 2, 3, 4, 5\}, \emptyset\rangle;$

$\mathcal{I}(\varphi) > 11;$

$$\begin{aligned}\varphi^{12} = \{\ & (x_1 \vee\ b_1\ ),\\ & (x_2 \vee\ b_2\ ),\\ & (x_3 \vee\ b_3\ ),\\ & (x_4 \vee\ b_4\ ),\\ & (x_5 \vee\ b_5\ )\ \blacktriangleleft\,\} \cup\\ & CNF(\textstyle\sum x_i \leq 1) \cup\\ & CNF(5b_1 + 3b_3 \geq 3) \cup\\ & CNF(5b_2 + 3b_4 \geq 3) \cup\\ & CNF(5b_1 + 5b_2 + 3b_3 + 3b_4 \geq 8) \cup\\ & CNF(5b_1 + 5b_2 + 3b_3 + 3b_4 \geq 10) \cup\\ & CNF(5b_1 + 5b_2 + 3b_3 + 3b_4 \geq 11) \cup\\ & CNF(5b_1 + 5b_2 + 3b_3 + 3b_4 + 1b_5 \geq 12)\ \blacktriangleleft \cup\\ & CNF(5b_1 + 5b_2 + 3b_3 + 3b_4 + 1b_5 \leq 12)\ \blacktriangleleft\end{aligned}$$

$optimize(\varphi_S, AL, AM, \{1, 2, 3, 4, 5\}) =$

$\langle\{1, 2, 3, 4, 5\}, 12\rangle;$

Iteration 7

$sat(\varphi, AL, AM) = \langle \text{SAT}, \emptyset, \mathcal{I}\rangle;\ cost(\varphi) = 12;$

In what follows, we present how the original WPM2 algorithm can be improved by the application of the stratified approach, the hardening of soft clauses, the optimization of subproblems, and the exploitation of the assignments whose costs are upper bounds on the subproblems. The first three improvements were already applied in Ansótegui et al. (2013a).

Finally, we show that a collateral result of optimizing these subproblems by refining the upper bound is to turn the original WPM2 complete algorithm into an incomplete algorithm given a fixed amount of time or memory resources. We present the improvements in Algorithm 2 by extending Algorithm 1. The improvements are underlined.

4.1 Stratified approach

The first improvement corresponds to the stratified approach introduced in Ansótegui et al. (2012) for algorithm WPM1. The stratified approach (Algorithm 2 lines 2, 4, 7 and 9) consists in sending to the SAT solver only a subset of the soft clauses, i.e., φ_M (Algorithm 2 line 4). For the sake of clarity, we will refer to this subset as a module. More precisely, a module M is the set of indexes of the clauses to be sent to the SAT solver. Therefore, when the *sat* function returns $st = \text{SAT}$, it means that we have solved the subproblem φ_M. If φ_M is equal to φ, then we have solved the whole problem and we can finish (Algorithm 2 line 7). A crucial point is how we extend our current module. This action is performed by the *newmodule* function (Algorithm 2 line 9).

In our current approach, we follow the stratified strategy applied in Ansótegui et al. (2012). There, a module M is formed by the indexes of the clauses whose weight is greater than or equal to a certain weight w_{max}. Initially, w_{max} is ∞. In order to extend the current module, we apply the diversity heuristic (Algorithm 2 line 9) which supplies us with an efficient method to calculate how we have to reduce the value of w_{max}. In particular, when there is a low diversity of weights in $w(\varphi \setminus \varphi_M)$, w_{max} is decreased to $max\ w(\varphi \setminus \varphi_M)$, while when there is a high diversity of weights, w_{max} decreases faster to keep the diversity of $w(\varphi \setminus \varphi_M)$ low. A similar approach with an alternative heuristic for grouping clauses can be found in Martins et al. (2012). In Morgado et al. (2013b), it is proposed to only consider in the working formula or the current module, those clauses that have been falsified by a satisfying assignment corresponding to an upper bound.

4.2 Clause hardening

The hardening of soft clauses in MaxSAT SAT-based solvers has been previously studied in Borchers and Furman (1998), Li et al. (2007), Larrosa et al. (2008), Heras et al. (2008), Marques-Silva et al. (2011), Ansótegui et al. (2012), and Morgado et al. (2012). Inspired by these works we study a hardening scheme for WPM2. While clause hardening was reported to have no positive effect in WPM1 (Ansótegui et al. 2012), we will see that it boosts efficiency in WPM2.

The clause hardening (Algorithm 2 lines 2, 8 and 11) consists in considering hard those soft clauses whose satisfiability we know does not need to be reconsidered. We need some lemma ensuring that falsifying those soft clauses would lead us to non-optimal assignments. In the case of WPM1, all soft clauses satisfying $w_i > W$, where $W = \sum\{w_i \mid (c_i, w_i) \in \varphi \wedge w_i < w_{max}\}$ is the sum of weights of clauses not sent to the SAT solver, can be hardened.

The correctness of this transformation is ensured by the following lemma:

Algorithm 2: Improved WPM2.

```
    Input: $\varphi = \langle (c_1, w_1), \ldots, (c_s, w_s), (c_{s+1}, \infty), \ldots, (c_{s+h}, \infty) \rangle$
1:  if $\langle \text{UNSAT}, \_, \_, \_ \rangle = sat(\varphi_H, \_, \_, \_)$ then return $\langle \infty, \emptyset \rangle$
2:  $\langle AL, AM\underline{, \beta, \mathcal{I}_S, H', M} \rangle := \langle \emptyset, \emptyset\underline{, \emptyset, \emptyset, \emptyset, newmodule(\varphi, H(\varphi))} \rangle$
3:  while $true$ do
4:      $\langle st, C, \mathcal{I}_{\underline{M}}\underline{, \beta} \rangle := sat(\varphi_{\underline{M}}, AL, AM\underline{, \beta})$
5:      if $st =$ SAT then
6:          $\underline{\mathcal{I}_S := \operatorname{arg\,min}^{\mathcal{I} \in \{\mathcal{I}_M, \mathcal{I}_S\}} \mathcal{I}(\varphi)}$
7:          $\underline{\textbf{if } \varphi_M = \varphi \textbf{ then}}$ return $\langle \mathcal{I}_{\underline{M}}(\varphi), \mathcal{I}_{\underline{M}} \rangle$
8:          $\underline{H' := harden(\varphi, AM, M)}$
9:          $\underline{M := newmodule(\varphi, M)}$
10:     else
11:         $\langle A, k\underline{, \mathcal{I}_A, \mathcal{I}_S} \rangle := optimize(\varphi_{S\underline{\cup H}}, AL, AM, C\underline{, H', \mathcal{I}_S})$
12:         $\underline{\beta := \{b_i \mid \mathcal{I}_A(\varphi.c_i) = 0\}^{i \in A} \cup \{\overline{b_i} \mid \mathcal{I}_A(\varphi.c_i) = 1\}^{i \in A} \cup \beta_{S \setminus A}}$
13:         $AL := \{\langle A, w(\varphi_A), \geq, k \rangle\} \cup AL$
14:         $AM := \{\langle A, w(\varphi_A), \leq, k \rangle\} \cup AM \setminus \{am \in AM\}_{am.A \subseteq A}$
```

Lemma 1 (Lemma 24 in Ansótegui et al. (2013b))

Let $\varphi_1 = \{(c_1, w_1), \ldots, (c_s, w_s), (c_{s+1}, \infty), \ldots, (c_{s+h}, \infty)\}$ *be a MaxSAT formula with cost zero, let* $\varphi_2 = \{(c'_1, w'_1), \ldots, (c'_r, w'_r)\}$ *be a MaxSAT formula without hard clauses and* $W = \sum_{j=1}^{r} w'_j$. *Let*

$$harden(w) = \begin{cases} w & if\, w \leq W \\ \infty & if\, w > W \end{cases}$$

and $\varphi_1' = \{(c_i, harden(w_i)) \mid (c_i, w_i) \in \varphi_1\}$. *Then,* $cost(\varphi_1 \cup \varphi_2) = cost(\varphi_1' \cup \varphi_2)$, *and any optimal assignment for* $\varphi_1' \cup \varphi_2$ *is an optimal assignment of* $\varphi_1 \cup \varphi_2$.

However, this lemma is not useful in the case of WPM2 because we do not proceed by transforming the formula, like in WPM1. Therefore, we generalize this lemma. For this, we need to introduce the notion of *optimal candidate* of a formula.

Function sat($\varphi, AL, AM\underline{, \beta}$)

```
1: $\langle A, k\rangle := \langle \bigcup_{am \in AM} am.A, \sum am.k\rangle$
2: $\varphi^k := \{\varphi.c_i\}_{i \notin A} \cup \{\varphi.c_i \vee b_i\}_{i \in A} \cup CNF(AL \cup AM)\underline{\cup\beta}$
3: $\underline{\textbf{repeat}}$
4:     $\langle st, \varphi_C^k, \mathcal{I}\rangle := satsolver(\varphi^k)$
5:     $\underline{\langle \beta, \varphi^k\rangle := \langle \beta \setminus \varphi_C^k, \varphi^k \setminus \beta\rangle}$
6: $\underline{\textbf{until } (\beta \cap \varphi_C^k = \emptyset)}$
7: $C := \{i \in S(\varphi) \mid (\varphi.c_i \in \varphi_C^k) \vee (\varphi.c_i \vee b_i \in \varphi_C^k)\}$
8: return $\langle st, C, \mathcal{I}\underline{, \beta}\rangle$
```

Function optimize($\varphi, AL, AM, C\underline{, H', \mathcal{I}_S}$)

```
1: $A := \underline{(}\bigcup_{am \in AM,\, am.A \cap C \neq \emptyset} am.A \cup C\underline{) \setminus H'}$
2: $k := lb := subsetsum(w(\varphi_A), \sum_{am \in AM,\, am.A \subseteq A} am.k + 1)$
3: $\underline{ub := W(\varphi_A)}$
4: while $true$ do
5:     $\underline{k := strategy(\overset{\substack{refine\\lower\ bound}}{lb}, \overset{binary\ search}{\frac{lb+ub}{2}}, \overset{\substack{refine\\upper\ bound}}{ub-1})}$
6:     $\langle st, \_, \mathcal{I}_A, \_\rangle := sat(\underline{\varphi_{A \cup H \cup H'}}, AL, \{\langle A, w(\varphi_A), \leq, k\rangle\}\underline{\cup \overset{am.A \subseteq H'}{\{am \in AM\}}}, \_)$
7:     if $st =$ SAT then
8:         $\underline{\langle \mathcal{I}_S, ub\rangle := \langle \arg\min_{\mathcal{I} \in \{\mathcal{I}_A, \mathcal{I}_S\}} \mathcal{I}(\varphi), \mathcal{I}_A(\varphi_A)\rangle}$
9:         $\underline{\textbf{if } lb = ub \textbf{ then}}$ return $\langle A, lb, \underline{\mathcal{I}_A, \mathcal{I}_S}\rangle$
10:    else
11:        $k := lb := subsetsum(w(\varphi_A), k + 1)$
12:        $\underline{\textbf{if } lb = ub \textbf{ then return } \langle A, lb, \mathcal{I}_A, \mathcal{I}_S\rangle}$
```

Definition 14 Given a WPMS formula $\varphi = \langle (c_1, w_1), \ldots, (c_s, w_s), (c_{s+1}, \infty), \ldots, (c_{s+h}, \infty) \rangle$, we say that k is an optimal candidate of φ if there exists a subset $A \subseteq \{1, \ldots, s\}$ such that $\sum_{i \in A} w_i = k$.

Notice that, for any assignment $\mathcal{I}$ of the variables of φ, we have that $\mathcal{I}(\varphi)$ is an optimal candidate of φ. However, if k is an optimal candidate, there does not exist necessarily an assignment $\mathcal{I}$ satisfying $\mathcal{I}(\varphi) = k$. Notice also that, given φ and k, finding the *next optimal candidate*, i.e., finding the smallest $k' > k$ such that k' is an optimal candidate of φ is equivalent to the subset sum problem.

Lemma 2 *Let $\varphi_1 \cup \varphi_2$ be a WPMS formula and k_1 and k_2 values such that: cost$(\varphi_1 \cup \varphi_2) = k_1 + k_2$ and any assignment $\mathcal{I}$ satisfies $\mathcal{I}(\varphi_1) \geq k_1$ and $\mathcal{I}(\varphi_2) \geq k_2$. Let k' be the smallest possible optimal candidate of φ_2 such that $k' > k_2$. Let φ_3 be a set of soft clauses with $W = \sum \{w_i \mid (c_i, w_i) \in \varphi_3\}$.*

Then, if $W < k' - k_2$, then any optimal assignment $\mathcal{I}'$ of $\varphi_1 \cup \varphi_2 \cup \varphi_3$ assigns $\mathcal{I}'(\varphi_2) = k_2$

Proof Let $\mathcal{I}'$ be any optimal assignment of $\varphi_1 \cup \varphi_2 \cup \varphi_3$. On the one hand, as for any other assignment, we have $\mathcal{I}'(\varphi_2) \geq k_2$.

On the other hand, any of the optimal assignments $\mathcal{I}$ of $\varphi_1 \cup \varphi_2$ can be extended (does not matter how) to the variables of var$(\varphi_3) \setminus$ var$(\varphi_1 \cup \varphi_2)$, such that

$$\mathcal{I}(\varphi_1 \cup \varphi_2 \cup \varphi_3) = \mathcal{I}(\varphi_1) + \mathcal{I}(\varphi_2) + \mathcal{I}(\varphi_3) \leq k_1 + k_2 + W < k_1 + k' \qquad (1)$$

Now, assume that $\mathcal{I}'(\varphi_2) \neq k_2$, then $\mathcal{I}'(\varphi_2) \geq k'$. As any other assignment, $\mathcal{I}'(\varphi_1) \geq k_1$. Hence, $\mathcal{I}'(\varphi_1 \cup \varphi_2 \cup \varphi_3) \geq k_1 + k' > \mathcal{I}(\varphi_1 \cup \varphi_2 \cup \varphi_3)$, but this contradicts the optimality of $\mathcal{I}'$. Therefore, $\mathcal{I}'(\varphi_2) = k_2$. □

Example 9 It may seem that the condition of Lemma 2 is hard to satisfy unless φ_1 and φ_2 are over disjoint sets of variables. This is not the case, and here we present a simple example where φ_1 and φ_2 share variables and Lemma 2 holds:

$$\varphi_H = CNF(((x_1 + x_2 \leq 1), \infty), ((x_3 + x_4 \leq 1), \infty), ((x_1 + x_2 + x_3 + x_4 \leq 2), \infty))$$

$$\varphi_1 = \langle (x_1, 1), (x_2, 1) \rangle \cup \varphi_H$$

$$\varphi_2 = \langle (x_3, 1), (x_4, 1) \rangle \cup \varphi_H$$

$$\varphi_1 \cup \varphi_2 = \langle (x_1, 1), (x_2, 1), (x_3, 1), (x_4, 1) \rangle \cup \varphi_H$$

Notice that the condition of Lemma 2 is satisfied for φ_1 and φ_2, since $k_1 = cost(\varphi_1) = 1$, $k_2 = cost(\varphi_2) = 1$ and $k_1 + k_2 = cost(\varphi_1 \cup \varphi_2) = 2$.

In order to apply this lemma we have to consider formulas $\varphi_1 \cup \varphi_2$ ensuring cost$(\varphi_1 \cup \varphi_2) = k_1 + k_2$ and $\mathcal{I}(\varphi_1) \geq k_1$ and $\mathcal{I}(\varphi_2) \geq k_2$, for any assignment $\mathcal{I}$. This can be easily ensured, in the case of WPM2, if both φ_1 and φ_2 are subproblems. Then, we only have to check if the next optimal candidate k' of φ_2 exceeds the previous one k_2 more than the sum W of the weights of the clauses not sent to the SAT solver. In such a case, we can consider all soft clauses of φ_2 and their corresponding AM constraint with k_2 as hard clauses. In other words, we do not need to recompute the optimal k_2 of φ_2.

The $harden(\varphi, AM, M)$ function (Algorithm 2 line 8) returns the set of indexes of soft clauses H' that needs to be considered hard based on the previous analysis according to: the current set of At-Most constraints AM, the next optimal candidates of these constraints and the sum of the weights W of soft clauses beyond the current w_{max}, i.e., not yet sent to the SAT solver.

Function harden(φ, AM, M)

```
1: $H' = \{i \mid \langle A, w(\varphi_A), \leq, k\rangle \in AM_{\{i\}} \wedge \sum w(\varphi \setminus \varphi_M) < subsetsum(w(\varphi_A), k+1) - k\}$
2: return $H'$
```

Finally, in the *optimize* function (Algorithm 2 line 11) we introduce H' since as we will see in the next subsection, we need to know which are all the hard clauses to this point of the execution.

4.3 Subproblem optimization

As we have mentioned earlier, one key point in WPM2 is how to compute $\langle A, k\rangle$ within the *optimize* function (Algorithm 2 line 11) to construct the new At-Most constraint $(\langle A, w(\varphi), \geq, k\rangle)$. In the original WPM2 algorithm, the idea was to compute the next lower bound candidate, k, for the subproblem φ_A. We can go further and set k to the optimal cost of the subproblem $\varphi_{A\cup H}$, i.e., $k = cost(\varphi_{A\cup H})$.[1]

In order to do this, while taking advantage of the AL constraints generated so far, we only have to extend the definition of the minimization problem corresponding to the original *optimize* function, by adding $\varphi_{A\cup H}$ to the *Subject to* section.

To solve $\varphi_{A\cup H}$, we can use any complete approach related to MaxSAT, such as, MaxSAT branch and bound algorithms, MaxSAT SAT-based algorithms, saturation under the MaxSAT resolution rule (Larrosa and Heras 2005; Bonet et al. 2006), or we can use other solving techniques such as PB solvers or ILP techniques, etc. Therefore the improved WPM2 algorithm is parametric on any suitable optimization solving approach. In this work, we experimented with an ILP approach, corresponding to the strategy shown in Sect. 3, and three MaxSAT approaches (new *optimize* line 5) that we describe in the next lines.

The first and natural approach consists in iteratively refining (increasing) the lower bound ($k = lb$) on $cost(\varphi_{A\cup H})$ by applying the *subsetsum* function as in the original WPM2 (new *optimize* lines 5 and 11). The procedure stops when lb satisfies the constraints $AL \cup \varphi_{A\cup H}$ (new *optimize* line 9). Notice that, since we have included $\varphi_{A\cup H}$ into the set of constraints, we will get an optimal assignment or solution for $\varphi_{A\cup H}$.

The second approach consists in iteratively refining (decreasing) the upper bound following the strategy applied in minisat+ (Eén and Sörensson 2006), SAT4J (Berre

[1] For the sake of clarity, we will obviate in the following mentioning the hardened soft clauses (H') due to the clause hardening technique (see Sect. 4.2).

2006), qmaxsat (Koshimura et al. 2012) or ShinMaxSat (Honjyo and Tanjo 2012). The upper bound ub is initially set to the top weight of φ_A. Then, we iteratively check whether there exists an assignment for $\varphi_{A\cup H}$ with cost $k = ub - 1$. Whenever we get a satisfying assignment ($\mathcal{I}_A$) we update ub to $\mathcal{I}_A(\varphi_A)$, i.e., the sum of the weights w_i of those soft clauses falsified under the satisfying assignment (new *optimize* line 5). Notice that since $\mathcal{I}_A$ is a satisfying assignment, it follows that $\mathcal{I}_A(\varphi_A) = \mathcal{I}_A(\varphi_{A\cup H})$. If we get an unsatisfiable answer, the previous upper bound ($\mathcal{I}_A(\varphi_A)$) is the optimal cost for $\varphi_{A\cup H}$ (new *optimize* lines 11 and 12).

The third approach applies a binary search scheme (Heras et al. 2011; Cimatti et al. 2010; Fu and Malik 2006) on k (new *optimize* line 5). We additionally refine the lower bound (lb) as in our first approach and the upper bound ($\mathcal{I}_A(\varphi_A)$) as in the second approach. The search ends when lb and $\mathcal{I}_A(\varphi_A)$ are equal (new *optimize* lines 9 and 11).

A final remark is that, if we combine this technique with the previous hardening technique, then we simply have to take into account the set of indexes of soft clauses H' that became hard. As we can see in the new *optimize* function (Algorithm 2 line 11), we first compute the set A as in the original function, but excluding the soft clauses that became hard H' (new *optimize* line 1). Then, we call the *sat* function but adding $\varphi_{A\cup H\cup H'}$ and the set of At-Most constraints involved in H', i.e., $\{am \in AM\}_{am.A\subseteq H'}$ (new *optimize* line 6).

The worst case complexity, in terms of the number of calls to the SAT solver, of the improved WPM2 algorithm is the number of times that the *optimize* function is called (bounded by the number of soft clauses) multiplied by the number of SAT calls needed in each call to the *optimize* function. This latter number is logarithmic on the sum of the weights of the clauses of the core if we use a binary search, hence essentially the number of clauses. Therefore, the worst case complexity, when using a binary search to solve the subproblems, is quadratic on the number of soft clauses.

In order to see that the number of calls to the *optimize* function is bounded by the number of clauses we just need to recall that WPM2 merges the At-Most constraints. Consider a binary tree where the soft clauses are the leaves, and the internal nodes represent the merges (calls to the *optimize* function). A binary tree of n leaves has n−1 internal nodes.

Solving all the subproblems exactly can be very costly since these are NP-hard problems. Notice that some of these subproblems can be integrated soon into another subproblem which we will also solve. A reasonable strategy would be to solve a subproblem when it appears for the second time, meaning that the associated k is not the optimal cost. However, in practice, we found that the more efficient strategy was to solve a subproblem only if it incorporates a previous subproblem.

Example 10 The improved WPM2 algorithm performs different iterations from the ones of the original WPM2 in Example 8 on the pigeon-hole formula presented in Example 7.

$$\varphi = \langle (x_1, 5), (x_2, 5), (x_3, 3), (x_4, 3), (x_5, 1)\rangle \ \cup \ \langle CNF(\sum x_i \leq 1, \infty)\rangle$$

In contrast to the original, the improved WPM2 algorithm performs fewer calls to the *sat* function with an unsatisfiable response. Therefore, it performs fewer calls to the *optimize* function and fewer updates to AL and AM. Besides, the *sat* function deals with formulas φ_M with fewer soft clauses and sets AL and AM with shorter constraints. We find the first difference in the first iteration where, applying the stratified approach, we consider only a subproblem φ_M with those clauses whose weight $w_i \geq 5$. The first unsatisfiable core (noted with dots $\bullet$), involves the soft clauses 1 and 2, and the set of hard clauses. The *optimize* function computes the new At-Most constraint that corresponds to the subproblem $\varphi_{\{1,2\} \cup H}$. The optimal cost k for this subproblem is 5, since this is the weight of both pigeons. Notice that, applying the stratified approach, we get a better first lower bound for the formula. The soft clauses are relaxed and the corresponding constraints are added to AL and AM (noted with triangles $\blacktriangleleft$). In the second iteration, we get that φ_M^5 is satisfiable. Since φ_M is not equal to φ, we compute a new module M with those clauses whose weight $w_i \geq 3$.

Iteration 1

$$\begin{array}{ll}
\varphi_M^0 = \{\ (x_1),\ \bullet & \varphi_M^5 = \{\ (x_1 \vee\ b_1\ ),\ \blacktriangleleft \\
\qquad (x_2),\ \bullet\ \} \cup & \qquad (x_2 \vee\ b_2\ ),\ \blacktriangleleft\ \} \cup \\
\qquad CNF(\sum x_i \leq 1)\ \bullet & \qquad CNF(\sum x_i \leq 1) \cup \\
 & \qquad CNF(5b_1 + 5b_2 \geq 5)\ \blacktriangleleft \cup \\
 & \qquad CNF(5b_1 + 5b_2 \leq 5)\ \blacktriangleleft \\
 & \\
sat(\varphi_M, AL, AM, \_) = & optimize(\varphi, AL, AM, \{1,2\}, \{\}, \_) = \\
\langle \text{UNSAT}, \{1,2\}, \emptyset, \_\rangle; & \langle \{1,2\}, 5, \_, \_\rangle; \\
\mathcal{I}(\varphi) > 0; &
\end{array}$$

Iteration 2

$$sat(\varphi_M, AL, AM, \_) = \langle \text{SAT}, \emptyset, \mathcal{I}, \_\rangle;\ \varphi_M \neq \varphi \rightarrow \mathcal{I}(\varphi) \geq 5;\ M = \{1,2,3,4\};$$

The third iteration is similar to the first one, but with the soft clauses 3 and 4.

Iteration 3

$$\begin{array}{ll}
\varphi_M^5 = \{\ (x_1 \vee\ b_1\ ), & \varphi_M^8 = \{\ (x_1 \vee\ b_1\ ), \\
\qquad (x_2 \vee\ b_2\ ), & \qquad (x_2 \vee\ b_2\ ), \\
\qquad (x_3\qquad),\ \bullet & \qquad (x_3 \vee\ b_3\ ),\ \blacktriangleleft \\
\qquad (x_4\qquad),\ \bullet\ \} \cup & \qquad (x_4 \vee\ b_4\ ),\ \blacktriangleleft\ \} \cup \\
\qquad CNF(\sum x_i \leq 1)\ \bullet \cup & \qquad CNF(\sum x_i \leq 1) \cup \\
\qquad CNF(5b_1 + 5b_2 \geq 5) \cup & \qquad CNF(5b_1 + 5b_2 \geq 5) \cup \\
\qquad CNF(5b_1 + 5b_2 \leq 5) & \qquad CNF(3b_3 + 3b_4 \geq 3)\ \blacktriangleleft \cup \\
 & \qquad CNF(5b_1 + 5b_2 \leq 5) \cup \\
 & \qquad CNF(3b_3 + 3b_4 \leq 3)\ \blacktriangleleft \\
 & \\
sat(\varphi_M, AL, AM, \_) = & optimize(\varphi, AL, AM, \{3,4\}, \{\}, \_) = \\
\langle \text{UNSAT}, \{3,4\}, \emptyset, \_\rangle; & \langle \{3,4\}, 3, \_, \_\rangle; \\
\mathcal{I}(\varphi) > 5; &
\end{array}$$

It is in the fourth iteration where we can appreciate the impact of subproblem optimization. In Example 8, we needed three iterations to get the bound 11 for the At-Most constraint corresponding to the subproblem $\varphi_{\{1,2,3,4\}}$. Incorporating the hard clauses to the *optimize* function and solving the subproblem $\varphi_{\{1,2,3,4\} \cup H}$, we get directly the optimal cost 11 in just one iteration. In the fifth iteration, we get that φ_M^5 is satisfiable. As φ_M is not equal to φ we have to compute a new module again. Before doing that, we apply the hardening technique. We get that the soft clauses 1, 2, 3 and 4, and their corresponding At-Most constraint can be considered as hard (H'). This is because the current k of this At-Most constraint is 11 ($5+3+3$) and its next k' candidate is 13 ($5+5+3$), while the addition of weights of those soft clauses not yet in M, i.e., {5}, is only 1. So, reconsidering this At-Most constraint would lead to an assignment with a higher cost than falsifying all the soft clauses not yet in M. Semantically speaking, the increase in cost of any non-allowed distribution of pigeons 1, 2, 3 and 4, is always higher than the cost of allowing the pigeon 5 to be out of the hole. After applying the hardening technique, we compute the new module M (clauses whose weight $w_i \geq 1$) that corresponds to the whole problem (i.e., in the next iteration with a satisfiable response from the *sat* function, we will have found the solution).

Iteration 4

$$\begin{array}{ll}
\varphi_M^8 = \{\ (x_1 \vee b_1\ ),\ \bullet & \varphi_M^{11} = \{\ (x_1 \vee b_1\ ), \\
\quad (x_2 \vee b_2\ ),\ \bullet & \quad (x_2 \vee b_2\ ), \\
\quad (x_3 \vee b_3\ ),\ \bullet & \quad (x_3 \vee b_3\ ), \\
\quad (x_4 \vee b_4\ ),\ \bullet\ \} \cup & \quad (x_4 \vee b_4\ ), \cup \\
\quad CNF(\sum x_i \leq 1)\ \bullet \cup & \quad CNF(\sum x_i \leq 1) \cup \\
\quad CNF(5b_1 + 5b_2 \geq 5) \cup & \quad CNF(5b_1 + 5b_2 \geq 5) \cup \\
\quad CNF(3b_3 + 3b_4 \geq 3) \cup & \quad CNF(3b_2 + 3b_4 \geq 3) \cup \\
\quad CNF(5b_1 + 5b_2 \leq 5)\ \bullet \cup & \quad CNF(5b_1 + 5b_2 + 3b_3 + 3b_4 \geq 11)\ \blacktriangleleft \cup \\
\quad CNF(3b_3 + 3b_4 \leq 3)\ \bullet & \quad CNF(5b_1 + 5b_2 + 3b_3 + 3b_4 \leq 11)\ \blacktriangleleft \\
 & \\
sat(\varphi_M, AL, AM, \_) = & optimize(\varphi, AL, AM, \{1,2,3,4\}, \{\}, \_) = \\
\langle \text{UNSAT}, \{1,2,3,4\}, \emptyset, \_\rangle; & \langle \{1,2,3,4\}, 11, \_, \_\rangle; \\
\mathcal{I}(\varphi) > 8; &
\end{array}$$

Iteration 5

$$sat(\varphi_M, AL, AM, \_) = \langle \text{SAT}, \emptyset, \mathcal{I}, \_\rangle;\ \varphi_M \neq \varphi \rightarrow \mathcal{I}(\varphi) \geq 11;\ H' = \{1,2,3,4\};\ M = \{1,2,3,4,5\};$$

In the sixth iteration, the *sat* function returns a core with all the soft clauses, the At-Most constraint and the set of hard clauses. Applying the hardening technique, the soft clauses with indexes in H' and the corresponding At-Most constraint are considered as hard. Therefore, the new At-Most constraint computed by the *optimize* function involves only the soft clause 5 and has the new bound $k = 1$. In the seventh iteration, we get that φ_M^{12} is satisfiable. Since φ_M is equal to φ, 12 is the solution to the problem.

Iteration 6

$$\begin{array}{ll}
\varphi_M^{11} = \{ (x_1 \vee b_1),\ \bullet & \varphi_M^{12} = \{\ (x_1 \vee b_1\ ), \\
\quad (x_2 \vee b_2),\ \bullet & \quad (x_2 \vee b_2\ ), \\
\quad (x_3 \vee b_3),\ \bullet & \quad (x_3 \vee b_3\ ), \\
\quad (x_4 \vee b_4),\ \bullet & \quad (x_4 \vee b_4\ ), \\
\quad (x_5 \qquad)\ \bullet\ \} \cup & \quad (x_5 \vee b_5\ )\ \blacktriangleleft \} \cup \\
\quad CNF(\sum x_i \leq 1)\ \bullet \cup & \quad CNF(\sum x_i \leq 1) \cup \\
\quad CNF(5b_1 + 5b_2 \geq 5) \cup & \quad CNF(5b_1 + 5b_2 \geq 5) \cup \\
\quad CNF(3b_2 + 3b_4 \geq 3) \cup & \quad CNF(3b_2 + 3b_4 \geq 3) \cup \\
\quad CNF(5b_1 + 5b_2 + 3b_3 + 3b_4 \geq 11) \cup & \quad CNF(5b_1 + 5b_2 + 3b_3 + 3b_4 \geq 11) \cup \\
\quad CNF(5b_1 + 5b_2 + 3b_3 + 3b_4 \leq 11)\ \bullet & \quad CNF(5b_1 + 5b_2 + 3b_3 + 3b_4 \leq 11) \cup \\
 & \quad CNF(1b_5 \geq 1)\ \blacktriangleleft \cup \\
 & \quad CNF(1b_5 \leq 1)\ \blacktriangleleft \\
 & \\
sat(\varphi_M, AL, AM, \_) = & optimize(\varphi, AL, AM, \{1,2,3,4,5\}, \{1,2,3,4\}, \_) = \\
\langle \text{UNSAT}, \{1,2,3,4,5\}, \emptyset, \_\rangle & \langle \{5\}, 1\rangle; \\
\mathcal{I}(\varphi) > 11; &
\end{array}$$

Iteration 7

$$sat(\varphi_M, AL, AM, \_) = \langle \text{SAT}, \emptyset, \mathcal{I}, \_\rangle;\ \varphi_M = \varphi \rightarrow cost(\varphi) = 12;$$

4.4 Exploiting satisfying assignments from subproblems

Whenever we obtain an upper bound or solution for a subproblem, we can obtain an upper bound for the whole problem. Consider $\varphi_{A\cup H}$, where A is a subset of the indexes of the soft clauses in a WPMS formula φ and H the set of indexes of the hard clauses, as the subproblem we want to solve. According to the search strategy we use in the *optimize* function (see Sect. 4.3) we may obtain assignments $\mathcal{I}_A$ (Algorithm 2 line 6) that are upper bounds or directly a solution for $\varphi_{A\cup H}$. If we extend this assignment by assigning a random value in {0, 1} to every variable in $var(\varphi) \setminus \text{var}(\varphi_{A\cup H})$, then it is not difficult to see that $\mathcal{I}_A(\varphi) \geq cost(\varphi)$, i.e., $\mathcal{I}_A(\varphi)$ is an upper bound for φ. Therefore, by comparing $\mathcal{I}_A(\varphi)$ with the cost of the best assignment found so far $\mathcal{I}_S(\varphi)$ (Algorithm 2 line 8), the improved WPM2 algorithm becomes naturally an incomplete approach. It is incomplete in the sense that it reports the assignment with the best cost found within restricted time and memory resources.

Also, due to the stratification approach, once we obtain a satisfying assignment $\mathcal{I}_M$ for a module M (Algorithm 2 line 4), we may have obtained a better upper bound for φ. Therefore, following the same idea, we update conveniently $\mathcal{I}_S$ (Algorithm 2 line 6) as in the *optimize* function.

Obviously, this incomplete approach makes sense if we are able to obtain quickly good quality upper bounds. We will show that this is the case in the experimental section (see Sect. 6). However, let us visualize the behavior of the improved WPM2 algorithm on a particular instance. In Fig. 1, we show the upper bounds $\mathcal{I}_A(\varphi)$ and lower bounds obtained during the execution of the improved WPM2 on an industrial Partial MaxSAT formula φ. The upper bound refinement strategy was used for the subproblem optimization phase.

For the sake of comparison, we also show the lower bounds obtained with the original WPM2 algorithm. The objective is to show that the improved WPM2 algorithm not only is able to provide good upper bounds, it also converges faster to the optimal cost.

In the x-axis, we show the elapsed seconds of the search in a logarithmic scale. The y-axis correspond to the range of the objective function (from 0 to top weight). In the upper half (from optimum to top weight) we show the value of the obtained upper bounds and in the lower half (from optimum to 0) we show the value of the lower bounds.

As we can see, the original WPM2 does not provide any upper bound until the optimum is found. In contrast, the improved WPM2 does provide several upper bounds coming from the subproblem optimization phase.

Both algorithms provide lower bounds. Every lower bound update corresponds to a new k for a subproblem obtained after a call to the *optimize* function and is followed by a call to the *sat* function to check the satisfiability of the whole problem. Notice that the subproblem optimization occurs always between lower bound updates. In the

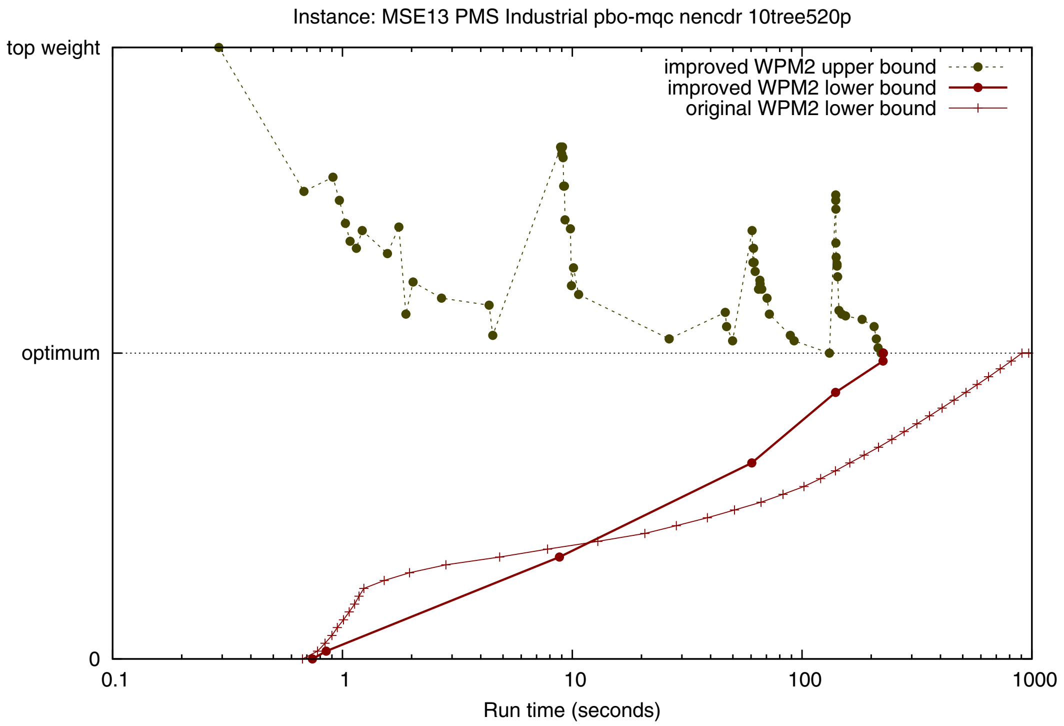


Fig. 1 Upper and lower bounds obtained with the original and the improved WPM2

improved WPM2 algorithm, during the subproblem optimization phase, the upper bound for the particular subproblem is always improved, i.e., it decreases. However, if we extend this assignment to the whole problem this monotonic behavior is not guaranteed. This is why, in Fig. 1, during the subproblem optimization phase, the upper bounds for the whole problem tend to decrease but can also increase.

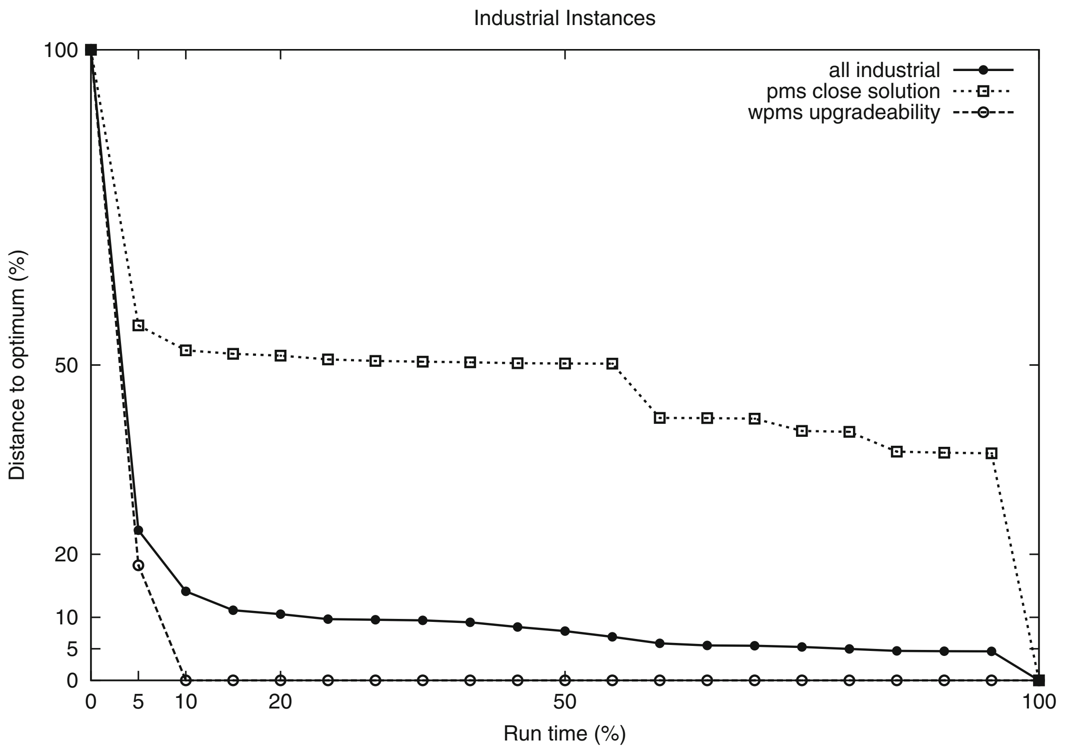
With respect to the quality of these upper bounds, notice that, in less than 5 s, an upper bound very close to the optimum (less than 6 % error) is obtained. Also, an upper bound $\mathcal{I}_A(\varphi)$ equal to the optimal cost $cost(\varphi)$ (0 % error) is obtained in 132 s, earlier than the solution itself (226 s). This is interesting because it means that we can obtain very high quality assignments or even a solution before solving exactly the problem, i.e., certifying that there is no any other assignment with a lower cost.

Finally, we can see that, thanks to subproblem optimization, the improved WPM2 only needs 6 calls (lower bounds updates) on the whole problem, while the original WPM2 needs more than 40.

For more detailed information, we analyze the quality of upper bounds for all the instances of our experimentation in Sect. 6 (Figs. 2 and 3).

We can further exploit the satisfying assignments obtained during the search. In modern SAT solvers, the polarity of the decision variables is chosen according to the most recent polarity they were assigned in a previous partial assignment. This technique is called phase saving (Pipatsrisawat and Darwiche 2007) and its main goal is to avoid redoing work. Notice that, during backtracking, many variable assignments are undone and the suitable polarity needs to be revealed again during search.

In our SAT-based MaxSAT algorithms, we perform independent queries to a SAT solver. Therefore, some suitable information can be lost. For example, in Sect. 5 we discuss how to preserve learned clauses by using the SAT solvers in incremental



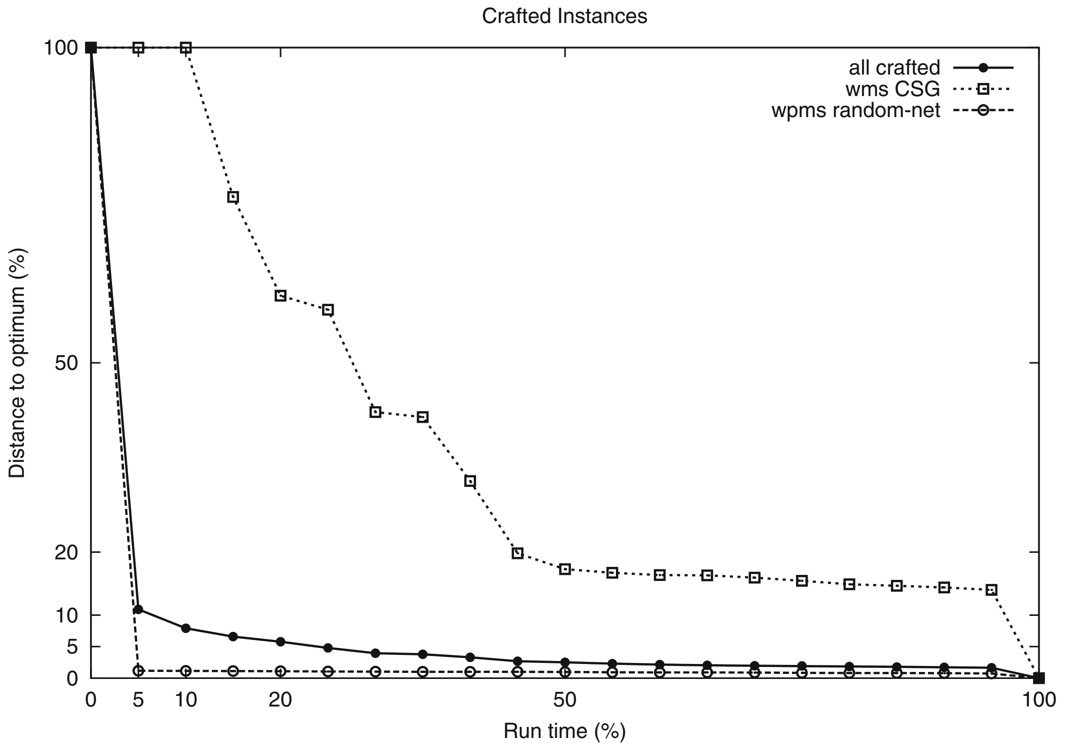
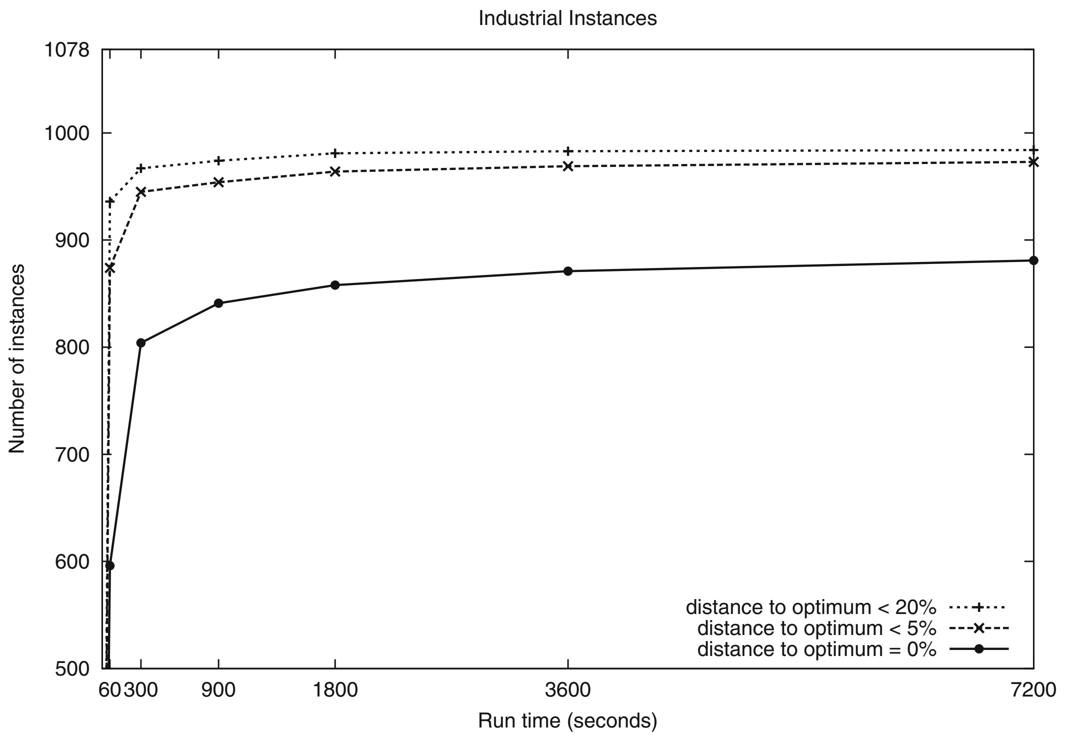


Fig. 2 Upper bound quality relative to the resolution time (at the top industrial instances, at the bottom crafted instances)



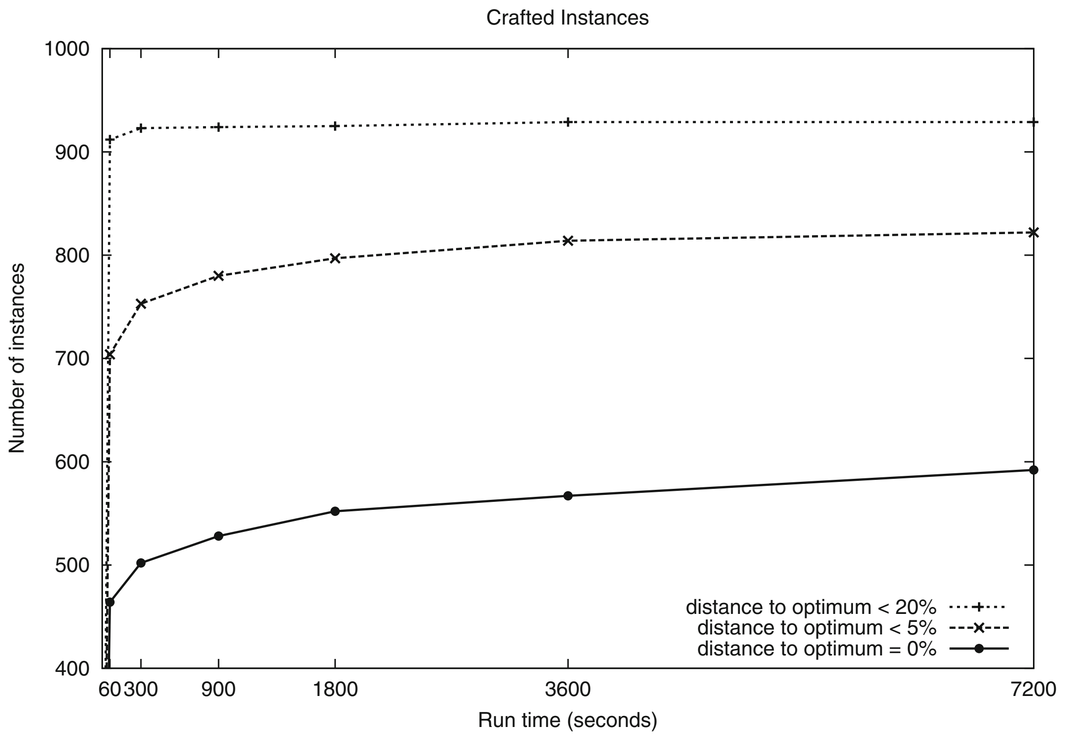


Fig. 3 Number of instances on which a certain upper bound quality is reached (at the top industrial instances, at the bottom crafted instances)

mode. Here, the idea is to use the optimal assignment $\mathcal{I}_A$ for a subproblem $\varphi_{A \cup H}$ (Algorithm 2 line 11) to guide the search in the next call to the SAT solver. Basically, the set β (Algorithm 2 line 12) is updated to contain the unit clauses that represent whether the ith soft clause was satisfied ($\overline{b}_i$) or falsified (b_i) by the $\mathcal{I}_A$ of the most recent subproblem it took part in. We expect assignments $\mathcal{I}_A$ to be more informed as search proceeds and therefore be able to guess the satisfaction status of soft clauses in optimal assignments for φ. In the *sat* function the set β is appended to the set of clauses sent to the SAT solver (new *sat* line 2). Although this gives us extra propagation, it may be the case that our guess is wrong, therefore we iteratively call the SAT solver until no unit clause in β is involved in the unsatisfiable core (new *s*at lines 3, 5 and 6). Notice that the unit clauses that do not appear in any core, do remain in the set β providing us extra propagation power (Algorithm 2 line 4). Therefore, we use the optimal assignments from subproblems to guess the phase of the variables in an optimal assignment to the whole problem φ. In addition, we can also use the satisfying assignments $\mathcal{I}_A$, obtained within the subproblem optimization, to guide the search during this phase.

5 Engineering efficient SMT-based MaxSAT solvers

We have implemented both the last version of the WPM1 algorithm (Ansótegui et al. 2012) and the improved WPM2 algorithm on top of the Yices SMT solver (Dutertre and de Moura 2014).

An SMT formula is a generalization of a Boolean formula in which some propositional variables have been replaced by predicates with predefined assignments from background theories such as, e.g., linear integer arithmetic. For example, an SMT formula can contain clauses like $x_1 \vee x_2 \vee (b_1 + 2 \leq b_1)$ and $(b_1 \geq 2 \cdot b_2 + 3 \cdot b_3)$, where x_1 and x_2 are Boolean variables and b_1, b_2 and b_3 are integer variables. Predicates over non-Boolean variables, such as linear integer inequalities, are evaluated according to the rules of a background theory. Leveraging the advances made in SAT solvers in the last decade, SMT solvers have proved to be competitive with classical decision methods in many areas. Most modern SMT solvers integrate a SAT solver with decision procedures (theory solvers) for sets of literals belonging to each theory. This way, we can hopefully get the best of both worlds: in particular, the efficiency of the SAT solver for the Boolean reasoning and the efficiency of special-purpose algorithms for the theory reasoning.

Another reasonable choice would be to use a PB solver, which can be seen as a particular case of an SMT solver specialized on the theory of PB constraints (Manquinho et al. 2009, 2010). However, if we also want to solve problems modeled with richer formalisms like WCSP, the SMT approach seems a better choice since we can take advantage of a wide range of theories (Ansótegui et al. 2011).

Among the theories considered in the SMT-LIB (Barrett et al. 2010) (SMT Library) we are interested in QF_LIA (Quantifier-Free *Linear Integer Arithmetic*). With the QF_LIA theory we can model the PB constraints that SAT-based MaxSAT algorithms generate during their execution. To this end, the PB variables can be declared as integer variables whose domain is {0, 1}. Therefore, for the SMT-based MaxSAT algorithm, we just need to replace the conversions to *CNF* by the proper linear integer arithmetic predicates. As we can see in Example 11, the SMT-LIB language v2.0 is a standard language with a prefix notation where the operator is placed at the beginning of the predicates.

Example 11 Given the SAT instance of Example 8 Iteration 3, $\varphi^6 = \{(x_1 \vee b_1), (x_2 \vee b_2), (x_3 \vee b_3), (x_4 \vee b_4), (x_5)\ \} \ \cup CNF(\sum x_i \ \leq 1) \cup CNF(5 \cdot b_1 + 3 \cdot b_3 \geq 3) \cup CNF(5 \cdot b_2 + 3 \cdot b_4 \geq 3) \cup CNF(5 \cdot b_1 + 3 \cdot b_3 \leq 3) \cup CNF(5 \cdot b_2 + 3 \cdot b_4 \leq 3)$ The SMT instance φ^6 in the SMT-LIB language v2.0 under QF_LIA (Barrett et al. 2010) would be as follows:

```
; Set QF_LIA theory

(set-logic QF_LIA)

; Declaration of variables

(declare-fun x1 () Bool)
(declare-fun x2 () Bool)
(declare-fun x3 () Bool)
(declare-fun x4 () Bool)
(declare-fun x5 () Bool)
(declare-fun b1 () Int)
(declare-fun b2 () Int)
(declare-fun b3 () Int)
(declare-fun b4 () Int)
(declare-fun b5 () Int)

; Bounds for PB variables

(assert (>= b1 0))
(assert (<= b1 1))
(assert (>= b2 0))
(assert (<= b2 1))
(assert (>= b3 0))
(assert (<= b3 1))
(assert (>= b4 0))
(assert (<= b4 1))
(assert (>= b5 0))
(assert (<= b5 1))

; Soft clauses

(assert (or x1 (= b1 1)))
(assert (or x2 (= b2 1)))
(assert (or x3 (= b3 1)))
(assert (or x4 (= b4 1)))
(assert x5)

; Hard clauses

(assert (or (not x1) (not x2)))
(assert (or (not x1) (not x3)))
(assert (or (not x1) (not x4)))
(assert (or (not x1) (not x5)))
(assert (or (not x2) (not x3)))
(assert (or (not x2) (not x4)))
(assert (or (not x2) (not x5)))
(assert (or (not x3) (not x4)))
(assert (or (not x3) (not x5)))
(assert (or (not x4) (not x5)))

; PB constraints

(assert (>= (+ (* b1 5) (* b3 3)) 3))
(assert (>= (+ (* b2 5) (* b4 3)) 3))
(assert (<= (+ (* b1 5) (* b3 3)) 3))
(assert (<= (+ (* b2 5) (* b4 3)) 3))

; Check satisfiability

(check-sat)
```

As suggested in Fu and Malik (2006) and Martins et al. (2011), we can preserve some learned lemmas from previous iterations that may help to reduce the search space. In order to do that, we execute the SMT solver in incremental mode. Within this mode, we can call the solve routine and add new clauses (assertions) on demand, while preserving learned lemmas. However, notice that our algorithms delete parts of the formula between iterations. For example, when we have to update the AM set in the WPM2 algorithm (see Sect. 4) by deleting some At-Most constraints. Therefore, we also have to take care of any learned lemma depending on them.

The Yices SMT solver gives the option of marking assertions as *retractable*. If the SMT solver does not support the deletion of assertions but supports the usage of assumptions, we can replace every retractable assertion c, with $a \rightarrow c$, where a is an assumption. Before each call, we activate the assumptions of assertions that have not been retracted by the algorithm. Notice that assertions that do have been retracted will have a pure literal ($\overline{a}$) such that a has not been activated. Therefore, the solver can safely set to false a, deactivating the clause. Moreover, any learned lemma on those assertions will also include $\overline{a}$. For example, Z3 and Mathsat SMT solvers do not allow to delete clauses, but they allow the use of assumptions.

From the point of view of incrementality it is also quite recommendable to reuse as much as possible the PB constraints we modify during the search, since we will also be able to reuse learned clauses depending on them. Pioneering works in this sense can be found in Bofill et al. (2013) and Andres et al. (2012). Our current implementation does not incorporate these complementary improvements, but it would certainly benefit from them.

6 Experimental results

In this section we present an intensive experimental investigation on the industrial and crafted instances of the MaxSAT Evaluation 2013 (MSE13) (Argelich et al. 2006-2004). We provide results for the improved WPM2 SMT-based MaxSAT solver and the best solvers of the MSE13. We run our experiments on a cluster with Intel Xeon CPU E7-8837 @ 2.67GHz processors and a memory limit of 3.5 GB. These are exactly the same specs as in the MSE13.

The instance set of the MSE13 is divided in four categories depending on the variant of the MaxSAT problem: MaxSAT (MS), Partial MaxSAT (PMS), Weighted MaxSAT (WMS) or WPMS. In addition, instances are further divided according to their nature into three subcategories: random, crafted or industrial (we are actually only interested in industrial and crafted). In each subcategory, instances are grouped by families.

We use the same set of instances for experiments on complete and incomplete solvers. The experimental results for complete and incomplete solvers are presented in Sects. 6.1 and 6.2, respectively.

6.1 Complete solvers

In this subsection, we analyze the performance of the improved WPM2 complete solver. First of all, we present the summarized results of the complete solvers at MSE13 in Tables 1 and 2. Second, we analyze the impact of each improvement on the original WPM2 algorithm in Table 3 (full detailed information in Tables 11 and 12). Then, we compare the results of the improved WPM2 solver with the best solvers of the MSE13 in Table 4 (full detailed information in Tables 13 and 14). Finally, in Tables 5 and 6, we further analyze the results of the improved WPM2 solver, discussing how it is able to exploit more efficiently the structure of the instances.

Tables 1 and 2 show the results of the MSE13 (with a timeout of 1800 s). Since families have different numbers of instances, we considered it was more fair to present the solvers ordered by mean family ratio of solved instances. In Table 1 we see the four best performing solvers on each industrial and crafted subcategory. In Table 2 we see the best performing solvers on the whole set of industrial and crafted instances. We have excluded $ISAC+$, since it is a portfolio based solver and our intention here is to compare ground solvers. Notice that $ISAC+$ already includes some of the ground solvers. The ground solvers with the best overall performance were: $wpm2$-13$*$ which actually corresponds to a WPM2 variation ($wpm2_{shua}*$ in Table 3), $maxhs13$ which consists in an hybrid SAT-ILP approach described in Davies and Bacchus (2011), $qms2$ which is based on an upper bound refinement described in Koshimura et al. (2012), $optim\text{-}ni$ and $msunc$ which implement the core-guided binary search algorithm described in Heras et al. (2011) and Morgado et al. (2012), $pwbo2.3$ which takes advantage of parallel processing as described in Martins et al. (2011), Martins et al. (2012), $wpm1$-2013 which is the SMT-based version of the improved WPM1 algorithm described in Ansótegui et al. (2012), ilp which translates WPMS into ILP and applies the MIP solver IBM-CPLEX studio124 as described in Sect. 3, and $wmsz09$ which implements a branch and bound algorithm described in Li et al. (2006), Li et al. (2007), Li et al. (2009). Further information about solvers and authors can be found in Argelich et al. (2006-2004).

We have completed the evaluation of the MSE13 by providing results of some solvers on those categories where originally they did not take part. For example, results

Table 1 MSE13 best solvers ordered by mean ratio of solved instances

		MS		PMS		WMS		WPMS
Random	1.	*msz13f*	1.	*ISAC+*	1.	*ckm-s*	1.	*ISAC+*
	2.	*ISAC+*	2.	*wmsz09*	2.	*ISAC+*	2.	*wmsz09*
	3.	*ckm-s*	3.	*wmsz+*	3.	*msz13f*	3.	*wmsz+*
	4.	*wmsz+*	4.	*ckm-s*	4.	*wmsz+*	4.	*ckm-s*
Crafted	1.	*ahms*	1.	*ISAC+*	1.	*ISAC+*	1.	*maxhs13*
	2.	*ISAC+*	2.	*qms2-m*	2.	*wmsz+*	2.	*ISAC+*
	3.	*msz13f*	3.	*qms2-mt*	3.	*wmsz09*	3.	*ilp13*
	4.	*ckm-s*	4.	*antom_s1*	4.	*msz13f*	4.	*wpm1-13*
Industrial	1.	*pmifu*	1.	*ISAC+*	1.	–	1.	*ISAC+*
	2.	*wpm1-11*	2.	*qms2-mt*	2.	–	2.	*wpm1-13*
	3.	*ISAC+*	3.	*wpm2-13∗*	3.	–	3.	*wpm2-13∗*
	4.	*optim-ni*	4.	*optim-ni*	4.	–	4.	*msunc*

Table 2 MSE13 best solvers

		Ind. (%)	1078	Cra. (%)	1000	Total (%)	2078
1.	*wpm2-13∗*	**75.0**	**820**	46.3	521	**61.5**	1341
2.	*maxhs13*	59.7	719	59.9	670	59.8	**1389**
3.	*qms2*	68.7	640	47.3	481	58.6	1121
4.	*optim-ni*	70.1	671	39.7	435	55.8	1106
5.	*msunc*	71.4	784	37.7	429	55.5	1215
6.	*pwbo2.3*	63.0	686	45.5	521	54.7	1207
7.	*wpm1-13*	65.0	743	40.1	452	53.3	1195
8.	*ilp13*	46.3	575	**61.1**	723	53.3	1298
9.	*wmsz09*	19.5	238	59.2	**745**	38.1	983

Mean ratio and number of solved instances; Best results are in bold

Table 3 Impact of WPM2 improvements, compared on the industrial and crafted instances of MSE13 (number and mean ratio of solved instances)

Solvers	MS	PMS	WMS	WPMS	Ind.	MS	PMS	WMS	WPMS	Cra.	Total
$wpm2$	20	429	–	202	651	12	247	10	39	308	959
	50.6 %	67.9 %	–	49.5 %	61.8 %	14.0 %	59.9 %	6.6 %	19.2 %	31.5 %	47.6 %
$wpm2_s$	**21**	467	–	203	691	12	248	10	98	368	1059
	51.6 %	74.2 %	–	52.5 %	66.9 %	14.0 %	58.1 %	6.6 %	29.9 %	34.0 %	51.4 %
$wpm2_{sh}$	**21**	464	–	269	754	12	248	15	98	373	1127
	51.6 %	73.5 %	–	62.4 %	69.0 %	14.0 %	58.1 %	10.3 %	29.9 %	34.6 %	52.8 %
$wpm2_{shia}$	18	239	–	261	518	**18**	248	**23**	253	542	1060
	48.7 %	37.8 %	–	55.4 %	43.2 %	**16.5 %**	46.8 %	**21.0 %**	55.5 %	40.0 %	41.7 %
$wpm2_{shic}$	19	259	–	267	545	17	257	22	258	554	1099
	49.7 %	40.5 %	–	57.2 %	45.5 %	16.0 %	48.7 %	20.6 %	56.5 %	40.8 %	43.5 %
$wpm2_{shla}$	18	480	–	319	817	12	256	16	199	483	1300
	48.7 %	76.3 %	–	74.0 %	73.7 %	14.0 %	61.6 %	12.0 %	50.1 %	42.0 %	58.8 %
$wpm2_{shlc}$	**21**	486	–	326	833	12	255	16	198	481	1314
	51.6 %	75.6 %	–	75.5 %	73.8 %	14.0 %	61.4 %	12.0 %	49.9 %	41.9 %	58.8 %
$wpm2_{shba}$	17	503	–	326	846	15	261	19	264	559	1405
	47.8 %	80.6 %	–	74.6 %	76.6 %	15.2 %	62.1 %	20.3 %	68.4 %	49.2 %	63.7 %
$wpm2_{shbc}$	20	497	–	**339**	856	14	265	19	263	561	1417
	50.6 %	78.7 %	–	**77.4 %**	76.3 %	14.8 %	62.8 %	20.3 %	67.3 %	49.0 %	63.4 %
$wpm2_{shua}*$	16	502	–	326	844	13	270	18	255	556	1400
	46.8 %	80.7 %	–	75.5 %	76.8 %	14.4 %	63.2 %	18.6 %	67.1 %	48.8 %	63.6 %
$wpm2_{shuc}$	18	513	–	334	865	14	271	18	255	558	1423
	48.7 %	81.9 %	–	76.7 %	78.1 %	14.8 %	63.5 %	18.6 %	66.1 %	48.7 %	64.3 %
$wpm2_{shucg}$	20	524	–	336	880	14	267	18	272	571	1451
	50.6 %	82.5 %	–	77.0 %	78.6 %	14.8 %	63.0 %	18.6 %	68.8 %	49.2 %	64.8 %
$wpm2_{shucgo}$	20	**528**	–	333	**881**	14	**272**	18	**288**	**592**	**1473**
	50.6 %	**82.9 %**	–	76.5 %	**78.9 %**	14.8 %	**63.7 %**	18.6 %	**73.7 %**	**51.0 %**	**65.7 %**

Best results are in bold

of solvers $wpm1$-13 and $wpm2$-13$*$ have been added in the MS category. Results of solver $qms2$ have been also added to MS and WMS categories. We had to change the format of these instances so that $qms2$ could read them. These additional results do not change the overall performance, but they give the full picture.

From Table 2, we emphasize that the solver implementing the variation of the WPM2 algorithm ($wpm2$-13$*$) was already the best in family ratio of solved instances on the whole set of industrial and crafted instances at MSE13. Although it solved fewer instances than $maxhs13$ on the whole set, $wpm2$-13$*$ dominated both in family ratio and number of solved instances on the set of industrial instances.

Table 3 shows our first experiment, where we evaluate the impact of each improvement on the original WPM2 algorithm (with a timeout of 7200 s). All the variations on the WPM2 algorithm are implemented on top of the Yices SMT solver (version

Table 4 $wpm2_{shucgo}$ compared to MSE13 best complete solvers on the industrial and crafted instances of MSE13 (number and mean ratio of solved instances)

Solvers	MS	PMS	WMS	WPMS	Ind.	MS	PMS	WMS	WPMS	Cra.	Total
$wpm2_{shucgo}$	20	528	–	333	**881**	14	272	18	288	592	**1473**
	50.6 %	82.9 %	–	76.5 %	**78.9 %**	14.8 %	63.7 %	18.6 %	73.7 %	51.0 %	**65.7 %**
$wpm2$-13∗	16	502	–	326	844	13	270	18	255	556	1400
	46.8 %	80.7 %	–	75.5 %	76.8 %	14.4 %	63.2 %	18.6 %	67.1 %	48.8 %	63.6 %
$maxhs13$	7	486	–	259	752	7	304	43	**330**	684	1436
	22.4 %	70.6 %	–	53.8 %	62.7 %	12.0 %	71.1 %	45.3 %	**89.1 %**	62.2 %	62.5 %
$qms2$-$g2$	19	**543**	–	119	681	10	284	30	207	531	1212
	34.0 %	**85.0 %**	–	46.5 %	71.2 %	13.2 %	**73.2 %**	22.2 %	58.2 %	50.3 %	61.4 %
$optim$-ni	38	503	–	165	706	7	261	32	165	465	1171
	83.7 %	80.8 %	–	50.7 %	73.2 %	7.4 %	64.6 %	25.0 %	44.8 %	42.7 %	58.8 %
$ilp13$	7	354	–	259	620	46	**333**	76	311	766	1386
	22.4 %	54.1 %	–	55.3 %	52.0 %	37.7 %	68.4 %	64.7 %	78.0 %	**65.5 %**	58.4 %
$wpm1$-13	21	422	–	**351**	794	12	207	11	301	531	1325
	51.6 %	68.4 %	–	**79.6 %**	70.1 %	14.0 %	48.3 %	7.4 %	74.9 %	43.5 %	57.6 %
$pwbo2.33$	7	445	–	269	721	8	254	12	204	478	1199
	22.4 %	71.9 %	–	58.5 %	64.8 %	12.4 %	65.2 %	14.6 %	66.7 %	48.4 %	57.1 %
$msunc$	26	512	–	270	808	8	249	17	164	438	1246
	56.4 %	77.9 %	–	67.0 %	73.5 %	7.8 %	58.5 %	13.6 %	43.6 %	38.2 %	56.9 %
$wmsz09$	0	200	–	85	285	**156**	318	82	234	**790**	1075
	0.0 %	29.0 %	–	18.9 %	24.2 %	86.4 %	60.4 %	61.2 %	56.1 %	63.6 %	42.7 %
$msz13f$	0	152	–	132	284	**156**	317	**83**	232	788	1072
	0.0 %	22.8 %	–	26.0 %	22.0 %	86.4 %	60.2 %	**66.2 %**	52.7 %	63.4 %	41.5 %
$w/pmifu$	**42**	287	–	261	590	5	61	37	125	228	818
	87.5 %	46.2 %	–	50.9 %	50.5 %	6.6 %	35.1 %	27.1 %	30.9 %	27.8 %	39.8 %
ckm-s	0	119	–	12	131	**156**	293	79	23	551	682
	0.0 %	15.2 %	–	7.2 %	12.0 %	**91.0 %**	55.0 %	46.2 %	7.4 %	45.6 %	27.8 %
$ahms$	0	34	–	10	44	146	224	73	179	622	666
	0.0 %	6.0 %	–	5.5 %	5.4 %	86.5 %	40.3 %	42.5 %	44.7 %	49.6 %	26.2 %

Best results are in bold

1.0.29). The different variations (see Sect. 4) and corresponding implementations are named $wpm2$ with different subindexes. Subindex $_s$ stands for stratified approach and $_h$ for clause hardening. Regarding to how we perform the subproblem optimization, $_i$ stands for ILP translation, $_l$ stands for lower bound refinement based on subset sum, $_u$ for upper bound refinement based on satisfying truth assignment, and $_b$ for binary search. Subindex $_a$ stands for optimizing all the subproblems and $_c$ for optimizing only subproblems that contain already extended clauses. Finally, $_g$ stands for guiding the search with the optimal assignments from subproblems and $_o$ for guiding the search also with the satisfying assignments within the subproblem optimization.

The original $wpm2$ has a performance of 47.6 % (959) family ratio (number) of solved instances. By using the stratified approach explained in Sect. 4.1 ($wpm2_s$)

we solve some additional instances in all categories having the highest increase on WPMS crafted subcategory. Overall, we solve 100 more instances. By applying also the clause hardening explained in Sect. 4.2 ($wpm2_{sh}$) we solve 68 more instances, mainly on WPMS industrial subcategory. This last one, with 52.8 % (1127) family ratio (number) of solved instances, increases in performance by 5.2 % (168) compared to $wpm2$.

Regarding the different variations for optimizing the subproblems (see Sect. 4.3), we can see that optimizing the subproblems through ILP ($wpm2_{shia}$) is not competitive on industrial instances. It solves 236 fewer industrial instances than $wpm2_{sh}$. This is expected since, as we will see in Table 4, ilp-13, the approach based on a full translation to ILP, is also not competitive on industrial instances. On crafted instances it performs similar to the other subproblem optimization variations. It solves 169 more crafted instances than $wpm2_{sh}$, but it is not the best performing variation. Notice, however, that on MS and WPMS industrial subcategories solves more instances than ilp-13.

Optimizing subproblems by refining the lower bound ($wpm2_{shla}$), gives us some additional solved instances in all categories, having the highest increases on WPMS industrial subcategory (50) and on WPMS crafted subcategory (101). It is important to highlight that optimizing subproblems with subset sum, instead of applying the subset sum as in the original WPM2 algorithm, leads to a total increase of 173 solved instances compared to $wpm2_{sh}$.

Optimizing subproblems by refining the upper bound ($wpm2_{shua}*$), gives us an additional boost with respect to $wpm2_{shla}$. We get the highest increases in solved instances, on PMS industrial subcategory (22), and on WPMS crafted subcategory (56). Compared to $wpm2_{sh}$, we have a total increase of 273 solved instances. Notice that this variation is the one that competed in the MSE13 and is referred as $wpm2$-13$*$ in Tables 1, 2 and 4. Optimizing with binary search ($wpm2_{shba}$) has almost the same global performance as $wpm2_{shua}*$.

By optimizing only subproblems that do contain clauses that were already extended in previous iterations, we have an increase in family ratio and number of solved instances on all subproblem optimization variations. The upper bound refinement variation ($wpm2_{shuc}$) is the one with best performance, with 64.3 % (1423) family ratio (number) of solved instances. It increases in performance by 11.5 % (296) compared to $wpm2_{sh}$.

Finally, $wpm2_{shucg}$ is the result of extending the previous best variation ($wpm2_{shuc}$) by guiding the search with the optimal assignments from subproblems (see Sect. 4.4). This way, the number of solved instances increases by 28. By guiding the search also with the satisfying assignments within the subproblem optimization ($wpm2_{shucgo}$), the number of solved instances increases by 22 more, solving 50 more instances than $wpm2_{shuc}$. This last variation ($wpm2_{shucgo}$), with 65.7 % (1473) family ratio (number) of solved instances, increases in performance by 18.1 % (514) compared to $wpm2$. This is in percentage 138.0 % (153.6 %). Actually, if we take into account the timeout of 7200 s used in our experiments, we obtain an overall speed-up of 1573 (three orders of magnitude) with respect to $wpm2$. Basically, we compare the total run time of the solvers on all the instances. Not solved instances are assumed to contribute only with the timeout.

Table 5 $wpm2_{shucgo}$, $certify\text{-}opt$ and $sat4j$ on industrial instances

				$wpm2_{shucgo}$			$certify\text{-}opt$			$sat4j$		
Subcategory	#	$\#H$	$\#S$	$\#*$	$\%b$	$\#AM$	$\#*$	$\%b$	$\#AM$	$\#*$	$\%b$	$\#AM$
MS	55	0.0	$1.8{\cdot}10^6$	20	0.8	41.2	0	100	1	0	100	1
PMS	627	$3.3{\cdot}10^5$	$1.3{\cdot}10^4$	528	63.6	87.8	267	100	1	247	100	1
WPMS	396	$4.9{\cdot}10^5$	$8.6{\cdot}10^3$	333	33.9	426	59	100	1	55	100	1
Total industrial	1078	$3.7{\cdot}10^5$	$9.4{\cdot}10^4$	881	49.6	212	326	100	1	302	100	1

Table 6 $wpm2_{shucgo}$, $certify\text{-}opt$ and $sat4j$ on crafted instances

				$wpm2_{shucgo}$			$certify\text{-}opt$			$sat4j$		
Subcategory	#	$\#H$	$\#S$	$\#*$	$\%b$	$\#AM$	$\#*$	$\%b$	$\#AM$	$\#*$	$\%b$	$\#AM$
MS	167	0.0	$1.2{\cdot}10^3$	14	96.5	98.8	6	100	1	6	100	1
PMS	377	$2.9{\cdot}10^4$	$3.8{\cdot}10^2$	272	92.0	24.3	222	100	1	224	100	1
WMS	116	0.0	$5.3{\cdot}10^3$	18	57.2	5.5	15	100	1	14	100	1
WPMS	340	$2.9{\cdot}10^4$	$5.8{\cdot}10^2$	288	84.4	6.9	205	100	1	203	100	1
Total crafted	1000	$2.1{\cdot}10^4$	$1.2{\cdot}10^3$	592	86.2	28.7	448	100	1	447	100	1

Table 4 shows our second experiment, where we compare the best variation of the improved WPM2 solver ($wpm2_{shucgo}$) with the best solvers of the MSE13 (with a timeout of 7200 s). In particular, we selected the best ground overall performing solvers presented in Table 2 and the best solvers for each subcategory in Table 1.

We see that $wpm2_{shucgo}$ is the best solver on the industrial set, both in family ratio and number of solved instances. On crafted instances, it is only the fifth in family ratio of solved instances. Although, on crafted instances, ilp is the first in family ratio of solved instances, as we have seen in Table 3, the variation which optimizes the subproblems through an ILP translation ($wpm2_{shia}$) does not improve the performance of the upper bound refinement variation ($wpm2_{shua*}$). We can conclude, however, that $wpm2_{shucgo}$ is the best in family ratio and number of solved instances across all industrial and crafted instances, and so the most robust followed by $maxhs$ and $qms2$.

There can be several explanations for the good performance of $wpm2_{shucgo}$. In the following we extend the study presented in Ansotegui (2013b). One of this explanations is that SAT-based MaxSAT solvers are supposed to take advantage by exploiting the information (learned clauses, At-Least and At-Most constraints, etc) obtained from each SAT instance (φ^k) into which the WPMS instance φ is reformulated.

The conjecture is that this information makes easier the resolution of the SAT instances where k is closer to $cost(\varphi)$. In particular, those ones that are unsatisfiable which tend to be harder to solve. In order to test the plausibility of this conjecture, we introduce a new complete algorithm which takes as input the WPMS formula φ and a cost that we initially set to $cost(\varphi)$. Therefore, this algorithm only needs to certify that the initial cost indeed corresponds to the cost of an optimal assignment. In particular, it just checks that $\varphi^{cost(\varphi)-1}$ is unsatisfiable and $\varphi^{cost(\varphi)}$ is satisfiable. We will refer to this algorithm as $certify\text{-}opt$.

In Tables 5 and 6, we compare $certify\text{-}opt$ with the two basic search schemes of SAT-based MaxSAT solvers: (i) those that focus the search on refining the lower bound, and exploit the information of unsatisfiable cores (solver $wpm2_{shucgo}$) and, (ii) those that focus the search on refining the upper bound, and exploit the information of satisfying assignments (solver $sat4j$ (Berre 2006)). All these approaches were implemented on top of the Yices SMT solver.

We experimented with the whole set of industrial and crafted instances from the MS, PMS, WMS and WPMS categories at the MSE13. Tables 5 and 6 show the results on industrial and crafted instances, respectively. In the tables, # stands for the total number of instances, $\#H$ and $\#S$ stand for the mean number of hard and soft clauses, respectively, and $\#*$ stands for the number of solved instances, within a timeout of 7200 s, for each solver.

The results of our experimentation show that $sat4j$ does not have a better performance than $certify\text{-}opt$. Although $certify\text{-}opt$ solves 25 more instances than $sat4j$, both solvers have almost the same overall performance. This can be explained because the upper bound refinement converges very quickly to the last satisfiable instance $\varphi^{cost(\varphi)}$, but it does not take advantage from any information of the previous satisfiable instances (φ^k with $k \in [cost(\varphi) + 1, W(\varphi)]$) in order to solve more efficiently $\varphi^{cost(\varphi)}$ and $\varphi^{cost(\varphi)-1}$. The structure of these instances can be seen in example 7.

Regarding $wpm2_{shucgo}$, it performs much better than $certify\text{-}opt$. For crafted instances, it solves 144 more instances than $certify\text{-}opt$. The difference is more dramatic for industrial instances where it solves 555 more. One of the keys of its success seems to be that it only needs to extend with auxiliary variables those soft clauses that have appeared into an unsatisfiable core. In contrast, $certify\text{-}opt$ or $sat4j$ need to extend all the soft clauses. In the tables, $\%b$ shows the percentage of extended soft clauses during the search. As we can see, both $certify\text{-}opt$ and $sat4j$ always extend 100 % of the soft clauses, while $wpm2_{shucgo}$ only extends a part of them. In particular, on industrial instances, where the difference in performance is higher, it only extend 49.6 % of the soft clauses.

Another key point in the good performance of $wpm2_{shucgo}$ is that, the extended soft clauses in the last query are covered by various At-Most constraints instead of a single and larger one as in $certify\text{-}opt$ or $sat4j$. This was proven to be more efficient in Ansótegui et al. (2009). In the tables, $\#AM$ stands for the number of At-Most constraints added during the search. When we go into detail in Table 5, in the last query of $wpm2_{shucgo}$ on industrial instances, the mean number of At-Most constraints is 212. In contrast, in Table 6, in the last query on crafted instances, the mean number of At-Most constraints is 28.7. This is also consistent with the better performance of $wpm2_{shucgo}$ on industrial instances.

6.2 Incomplete solvers

In this subsection, we analyze the performance of the improved WPM2 incomplete solver. We show the results that it would have obtained on the track for incomplete solvers at the MSE13 in Table 7 (full detailed information in Tables 15 and 16). We

Table 7 $wpm2_{shucgo}$ incomplete compared to MSE13 incomplete solvers on the industrial and crafted instances of MSE13 (number and mean ratio of solved instances)

Solvers	MS	PMS	WMS	WPMS	Ind.	MS	PMS	WMS	WPMS	Cra.	Total
$wpm2_{shucgo}$	18	**511**	–	**336**	**865**	15	260	20	224	519	**1384**
	32.0 %	**80.0 %**	–	**77.0 %**	**76.0 %**	15.0 %	61.0 %	19.0 %	62.0 %	47.0 %	**62.0 %**
ccls	**46**	501	–	198	745	129	205	83	93	510	1255
	91.0 %	79.0 %	–	52.0 %	73.0 %	76.0 %	37.0 %	60.0 %	17.0 %	42.0 %	58.0 %
optim	2	60	–	13	75	**167**	**303**	**115**	**257**	**842**	917
	1.0 %	10.0 %	–	8.0 %	9.0 %	**100.0 %**	**67.0 %**	**99.0 %**	**60.0 %**	**76.0 %**	40.0 %
ira-nov	3	103	–	63	169	9	225	30	167	431	600
	2.0 %	20.0 %	–	28.0 %	21.0 %	8.0 %	61.0 %	20.0 %	45.0 %	40.0 %	29.0 %

Best results are in bold

also discuss the quality of the upper bounds obtained during its execution on industrial and crafted instances in Figs. 2 and 3.

Table 7 shows our first experiment, where we compare the incomplete solver based on the improved WPM2 algorithm ($wpm2_{shucgo}$), with the best incomplete solvers of the MSE13. Results are presented following the same classification criteria as in the MSE13. For each instance, it is computed which are the solvers that obtain the best upper bound within 300 s (5 min). For each solver and subcategory we present the number of instances where the solver reported the best upper bound and the mean family ratio according to this number.

We can see that $wpm2_{shucgo}$ dominates on industrial instances, being the one that reaches the best upper bound 865 times. This is 120 more times than $ccls$, the second one. On crafted instances, both are dominated by $optim$, that in contrast performs not well on industrial instances. When we consider the whole set of industrial and crafted instances, $wpm2_{shucgo}$ is the best one 1384 times, 129 more times than the second one $ccls$.

In the following, we analyze the quality of upper bounds provided by $wpm2_{shucgo}$ on the whole set of industrial and crafted instances. In Fig. 2, we show the mean upper bound quality obtained during the resolution of the instances. Those instances not solved in 7200 s or solved in less than 60 s are discarded. Therefore, only 358 industrial and 129 crafted instances are taken into account. In x-axis we have the relative elapsed running time (100 % corresponds to the total time to solve the instance), and in y-axis we have the relative distance of the upper bounds to the optimum (an upper bound equal to the top weight or to the optimum, has a relative distance of 100 or 0 %, respectively). We refer to the relative distance to the optimum as the *error* in an upper bound (a small error means a high quality).

In the graphic at the top of Fig. 2, we have the experiments on the whole set of industrial instances (MS, PMS and WPMS categories), and on the two industrial families, with the best (WPMS *upgradeability* family) and the worst (PMS *close solutions* family) mean upper bound quality. In one third of the resolution time, the mean error in the upper bounds on the whole set of industrial instances is less than 10 % and in two thirds it is less than 6 %. In particular, for the WPMS *upgradeability* family, in

10 % of the resolution time the mean error is less than 1 %. Also, on some instances (PMS *pbo-mcq* family) the optimum (0 % error) is reached in one second while more than 1000 s are needed to certify it. On the other hand, for the PMS *close solution* family, in 90 % of the resolution time, the mean error is about 37 %.

In the graphic at the bottom of Fig. 2, we have the experiments on the whole set of crafted instances (MS, PMS, WMS and WPMS categories), and on the two crafted families, with the best (WPMS *random-net* family) and the worst (WMS *CSG* family) mean upper bound quality. In one third of the resolution time, the mean error in the upper bounds on the whole set of crafted instances is less than 4 % and in two thirds it is less than 2 %. In particular, for the WPMS *random-net* family, in 5 % of the resolution time the mean error is less than 1 %. On the other hand, for the WMS *CSG* family, in 10 % of the resolution time the mean error is 100 and in 90 % of the resolution time it is about 14 %.

It may seem that we get better results on crafted instances, however we should take into account that, the number of crafted instances solved by $wpm2_{shucgo}$ within 60 and 7200 run time seconds, is lower than the number of industrial instances. For some instances unsolved by $wpm2_{shucgo}$, we can consult the optimal cost found by other MaxSAT solvers (none dominates completely at the MSE13). This allows us to know which is the quality achieved by $wpm2_{shucgo}$ even if it was not able to solve the instance exactly.

In Fig. 3, we experimented with the industrial and crafted instances of MSE13 where any of the solvers in Table 4 was able to find the optimum.[2] We show the number of industrial and crafted instances, where $wpm2_{shucgo}$ reached an upper bound with a relative distance to the optimum (error) of less than 20, 5 and 0 % in a given elapsed run time. In x-axis we show the elapsed time from 0 to 7200 s, and in y-axis we show the number of instances.

In the graphic at the top of Fig. 3, we have the experiments on the industrial instances. We can see that, a high quality of upper bound is reached on a great number of instances in a relative short run time. From 20 to 5 % error, there is almost no difference. In 60 s, an upper bound with an error of less than 5 % is reached on 874 out of 1012 instances. In 300 s it is reached on 945 instances and in 7200 s on 973 instances. With respect to a 0 % error in the upper bound (optimum is not necessarily certified), in 60 s it is reached on 596 instances, in 300 s on 804 instances and in 7200 s on 881 instances.

In the graphic at the bottom of Fig. 3, we have the experiments on the crafted instances. Compared to what happens on the industrial set, there is a greater difference in number of instances, depending on the maximum error that we consider. We can see that, an upper bound with an error of less than 20 % is reached on 912 out of 952 instances in 60 s. However, the number of instances with less than a 5 % error in the upper bound is significantly lower, 704 instances in 60 s, 753 instances in 300 s and 822 instances in 7200 s. With respect to a 0 % error in the upper bound, the difference is even more important. In 60 s it is reached only on 464 instances, in 300 s only on 502 instances and in 7200 s only on 592 instances.

[2] There are only 66 industrial and 48 crafted instances for which was not found.

As a concluding remark, in 300 s (the timeout of the track for incomplete solvers at MSE13), $wpm2_{shucgo}$ has reached an upper bound with an error of less than 5 % on 945 out of 1012 industrial instances. This is consistent with the results in Table 7 where we have shown that $wpm2_{shucgo}$ has the best performance on industrial instances.

6.3 Results at MaxSAT evaluation 2014

In the previous subsections, we have analyzed and described in detail the impact of every improvement incorporated in $wpm2$. Here, we study the performance of the $wpm2014$ solver that took part in MSE14. This solver behaves exactly as $wpm2_{shuc}$ for (Partial) MaxSAT instances, and includes some efficiencies for Weighted (Partial) MaxSAT instances. In particular, we show that although this solver is not as competitive as the newest complete solvers, its incomplete version (as described in this article) ranked the first for the incomplete track.

The sets of instances at MSE14 are almost the same as the ones at MSE13. The main difference is that WMS families were integrated into the WPMS set. The classification criteria remains the same.

Table 8 summarizes the results for the best complete solvers on the whole set of industrial and crafted instances according to MSE14. We have excluded the portfolio

Table 8 Best complete solvers on the industrial and crafted instances at MSE14 (number and mean ratio of solved instances)

Solvers	MS	PMS	WPMS	Ind.	MS	PMS	WPMS	Cra.	Total
maxhs	27	417	280	724	7	328	219	554	1278
	73.0 %	70.5 %	52.6 %	62.6 %	11.8 %	69.2 %	**75.8 %**	**62.2 %**	**64.2 %**
eva500a	41	472	**368**	**881**	9	302	149	460	**1341**
	86.5 %	79.7 %	72.8 %	**78.4 %**	8.5 %	71.3 %	47.8 %	47.8 %	62.8 %
mscg	40	468	363	871	5	310	127	442	1313
	85.5 %	80.0 %	70.4 %	77.9 %	4.3 %	67.6 %	42.8 %	43.3 %	60.4 %
qms-g3	14	454	303	771	11	318	181	510	1281
	29.0 %	78.5 %	67.3 %	72.6 %	9.0 %	**73.4 %**	52.5 %	50.9 %	61.6 %
wpm2014	29	428	359	816	12	297	151	460	1276
	75.0 %	75.5 %	**73.9 %**	75.1 %	9.3 %	69.7 %	43.4 %	45.2 %	59.9 %
open-wbo	**42**	**473**	315	830	11	317	130	458	1288
	87.5 %	**81.1 %**	59.5 %	76.1 %	9.0 %	73.4 %	37.0 %	42.9 %	59.3 %
ilp	1	265	249	515	30	339	**224**	**593**	1018
	0.5 %	40.0 %	45.9 %	39.0 %	20.5 %	62.6 %	75.2 %	61.4 %	50.3 %
scip-ms	0	211	242	453	24	**342**	200	566	1019
	0.0 %	31.5 %	44.8 %	32.9 %	12.5 %	63.4 %	70.4 %	57.8 %	45.5 %
ahmaxsat-ls	0	32	25	57	**156**	294	128	578	635
	0.0 %	5.5 %	7.5 %	5.7 %	**60.5 %**	48.0 %	32.1 %	42.1 %	24.1 %

Best results are in bold

$ISAC+$, since our main aim here is to compare algorithms and ground solvers. We have also included ilp, $scip$-ms and $ahmaxsat$-ls, which win in mean family ratio or number of solved instances on some crafted categories. On crafted instances, the best solver in mean family ratio (number) of solved instances is $maxhs$ (ilp), and on the whole set, it is $maxhs$ ($eva500a$).

On industrial instances, we can see that $wpm2014$ is the fourth one. The best three solvers on industrial instances were $eva500a$ (Narodytska and Bacchus 2014), $mscg$ (Morgado et al. 2014) and $open$-wbo (Martins et al. 2014). For $open$-wbo, we selected the best version for each category. To our best knowledge $eva500a$ automatically detects the category and applies a predefined user parametrization. These three solvers are SAT-based MaxSAT solvers too. Without going into detail, we could say that the approaches of $eva500a$ and $mscg$ allow them to generate simpler PB constraints. Moreover, all three new solvers build incrementally these PB constraints, instead of generating them from scratch. In the case of $open$-wbo, PB constraints are explicitly built incrementally. In the case of $eva500a$ and $mscg$ this incrementality comes naturally as a result of the nature of the algorithm. These improvements are complementary to the approach of $wpm2014$ and therefore they could be incorporated. We can yet see that $wpm2014$ achieves the best ratio in WPMS instances.

Table 9 summarizes the results for the best incomplete solvers on the whole set of industrial and crafted instances according to MSE14. Clearly, $wpm2014$ dominates on industrial instances, and it is the best overall solver on industrial and crafted instances. For $optimax$ (implementing the BCD algorithm (Morgado et al. 2012)), the second best solver for industrial instances, we also selected the best version for each category.

Since we are showing a partial order, we have also included the solvers that take part in all categories $dist$, $ccls2014$, $ccmpa$ and $ahmaxsat$-ls, from which $dist$ and

Table 9 Best incomplete solvers on the industrial and crafted instances at MSE13 (number and mean ratio of solved instances)

Solvers	MS	PMS	WPMS	Ind.	MS	PMS	WPMS	Cra.	Total
$wpm2014$	30	**407**	**365**	**802**	15	301	180	496	**1298**
	75.5 %	**71.9 %**	**73.2 %**	**73.2 %**	7.0 %	67.4 %	**51.3 %**	51.3 %	**62.1 %**
$optimax2$	**33**	354	247	634	15	295	149	459	1093
	78.5 %	59.7 %	59.8 %	59.8 %	7.0 %	**68.8 %**	47.2 %	47.2 %	53.4 %
$dist$	0	177	38	215	171	**354**	**186**	**711**	926
	0.0 %	31.7 %	10.8 %	24.5 %	82.7 %	66.3 %	50.8 %	**61.3 %**	43.2 %
$ccls2014$	0	61	39	100	**176**	299	167	642	742
	0.0 %	10.5 %	15.1 %	11.0 %	**97.2 %**	58.4 %	48.7 %	60.5 %	36.1 %
$ccmpa$	5	91	52	148	173	281	154	608	756
	4.5 %	17.5 %	22.4 %	17.9 %	87.5 %	54.5 %	38.8 %	52.4 %	35.4 %
$sat4j$	5	117	66	188	2	200	82	284	472
	20.5 %	23.6 %	19.3 %	22.3 %	0.5 %	40.3 %	28.6 %	27.1 %	24.7 %

Best results are in bold

Table 10 Comparison as incomplete solvers of *wpm*2014, *open-wbo* and *qms*

	*wpm*2014	*open-wbo*	*qms*
*wpm*2014		**939** (**51** **498** **390**)	**850** (**52** 423 **375**)
open-wbo	857 (43 473 341)		**864** (**45** 458 **361**)
qms	820 (13 **528** 279)	827 (19 **522** 286)	

Best results are in bold

*ccls*2014 win in mean family ratio or number of solved instances on some crafted categories.

Finally, we have decided to extend the results of the incomplete track of the MaxSAT Evaluation by comparing to *wpm*2014 the best complete solvers than can provide upper bounds but did not take part at the incomplete track: *qms*2 and *open-wbo*. We performed the comparison on industrial instances, where all these solvers were competitive.

In Table 10, we present the dominance relation between pairs of solvers on the the total (MS PMS WPMS) set of industrial instances. For example, *wpm*2014 (*open-wbo*) is able to obtain a better or equal upper bound than *open-wbo* (*wpm*2014) on 939 (857) industrial instances of which 51 (43) are MS, 498 (473) are PMS and 390 (341) are WPMS.

On the whole set of industrial instances *wpm*2014 outperforms both *open-wbo* and *qms*2. Even though *wpm*2014 was not the best complete solver at MSE14, it dominates the other solvers as an incomplete approach. For PMS, we can see that *qms*2 is the best performing approach, but for MS and WPMS, *wpm*2014 performs better. Notice that *qms*2 is an efficient implementation of a SAT-based MaxSAT algorithm that mainly follows and upper-bound refinement strategy but unlike *wpm*2014 needs to place PB constraints on the whole set of soft clauses. So, if the subproblems represent most of the original instance, *qms*2 should be better, but if the subproblems cover only a small fraction, *wpm*2014 has advantages even with a less efficient approach for subproblems. Precisely, thanks to the insights of Table 5 in Sect. 6.1, we know that for PMS instances, $wpm2_{shucgo}$ (*wpm*2014) covers in average a 63.6 % of the soft clauses, while for WPMS, it covers in average only a 33.9 %. This suggests that incorporating the efficiencies of *qms*2 into *wpm*2014 will certainly improve its performance on PMS instances and perhaps on WPMS instances too.

These results confirm that by applying the subproblem optimization technique and exploiting the satisfying assignments from subproblems, a complete MaxSAT solver can be used as an effective incomplete MaxSAT solver.

7 Conclusions and future work

True innovation in heuristic-search research is not achieved from yet another method that performs better than its competitors if there is no understanding as to why the method performs well (Sörensen 2015). Following this principle, we have provided a detailed analysis on how the improved WPM2 solver has experienced a significant boost on solving industrial instances, which is our ultimate goal.

In particular, the subproblem optimization approach and the exploitation of the satisfying assignments from subproblems seem to have the most clear impact on efficiency. Therefore, solving incrementally an optimization problem by solving some of its subproblems is a promising avenue. This way we can focus on a reduced portion of the instance, allowing us to use less complex PB constraints to solve the whole problem.

We have seen that SAT-based MaxSAT solvers are able to exploit the information derived from refining lower bounds and upper bounds. We have confirmed that this is not only crucial to locate quickly better upper bounds, also to certify efficiently that a given upper bound is the optimum.

Moreover, from a more practical point of view, we know that although many NP-hard problems can not be solved exactly, in industry they are mostly focused on obtaining better upper bounds. Therefore, completeness is not always a mandatory requirement to guarantee practical success. In this sense, we have shown how we can turn a complete solver into an efficient incomplete solver by extending the satisfying assignments from the subproblem optimization phases to the whole problem.

As shown in the experimental evaluation, the incomplete version of the improved WPM2 solver would have dominated on industrial instances the track for incomplete solvers at the MSE13. Furthermore, the solver $wpm2014$, which is just a more efficient implementation of the improved WPM2, was the best performing solver on industrial instances on the track for incomplete solvers at the MSE14.

As future work, we will study how to improve the interaction with the optimization of the subproblems. A portfolio that selects the most suitable optimization approach depending on the structure of the subproblems seems another way of achieving additional speed-ups.

From the point of view managing even more efficiently PB constraints, following recent works (Narodytska and Bacchus 2014; Morgado et al. 2014; Martins et al. 2014), it is also quite recommendable to use less complex PB constraints and reuse as much as possible of them when they are modified.

Finally, we have also shown that the SMT technology is an underlying efficient technology for solving the MaxSAT problem. A positive side effect is that our algorithm can be naturally extended to solve the MaxSMT problem.

Acknowledgments Research partially supported by the Ministerio de Economía y Competividad research project TASSAT2: TIN2013-48031-C4-4-P and Google Faculty Research Award program.

Appendix

See Tables 11, 12, 13, 14, 15 and 16.

Table 11 Impact of WPM2 improvements, compared on the industrial instances of MSE13 (number and mean ratio of solved instances)

Family	#	$wpm2$	$wpm2_s$	$wpm2_{sh}$	$wpm2_{shia}$	$wpm2_{shic}$	$wpm2_{shla}$	$wpm2_{shlc}$	$wpm2_{shba}$	$wpm2_{shbc}$	$wpm2_{shua}$*	$wpm2_{shuc}$	$wpm2_{shucg}$	$wpm2_{shucgo}$
MS														
cir-dp	3	402(2)	75.3(2)	70.4(2)	77.7(2)	**55.3(2)**	319(2)	70.1(2)	446(2)	63.2(2)	489(2)	58.5(2)	251(2)	248(2)
sean-s	52	496(18)	681(19)	690(19)	625(16)	397(17)	592(16)	**548(19)**	435(15)	649(18)	356(14)	397(16)	591(18)	493(18)
Total	55	20	**21**	**21**	18	19	18	**21**	17	20	16	18	20	20
		50.6 %	**51.6 %**	**51.6 %**	48.7 %	49.7 %	48.7 %	**51.6 %**	47.8 %	50.6 %	46.8 %	48.7 %	50.6 %	50.6 %
PMS														
aes	7	0.0(0)	0.0(0)	0.0(0)	2857(1)	92.8(1)	0.0(0)	0.0(0)	**11.6(1)**	0.0(0)	524(1)	1975(1)	0.0(0)	0.0(0)
bcp-fir	50	58.5(49)	**19.2(49)**	21.2(49)	103(49)	92.6(49)	131(49)	36.6(49)	50.9(49)	48.2(48)	19.8(49)	30.6(49)	107(49)	52.6(49)
bcp-hysi	17	325(13)	584(13)	607(13)	635(7)	590(7)	175(13)	524(13)	602(16)	646(14)	156(16)	**99.7(16)**	148(16)	120(16)
bcp-hysu	38	229(19)	513(17)	524(17)	76.3(1)	13.8(1)	679(20)	153(18)	398(28)	185(25)	**256(34)**	453(33)	405(33)	511(34)
bcp-msp	50	897(19)	1100(21)	841(20)	570(17)	880(19)	739(22)	322(22)	927(26)	551(26)	804(26)	902(29)	671(28)	**591(29)**
bcp-mtg	40	1538(21)	1607(28)	1221(26)	816(22)	742(20)	916(36)	1487(31)	810(38)	977(33)	**441(39)**	844(36)	809(36)	746(36)
bcp-syn	50	323(24)	417(23)	664(24)	85.4(24)	128(24)	146(22)	834(25)	337(26)	501(25)	121(24)	320(25)	**591(30)**	121(28)
cir-tc	4	273(3)	1064(4)	816(4)	2537(1)	447(1)	163(4)	300(3)	114(4)	129(4)	94.4(4)	95.2(4)	99.1(4)	**90.3(4)**
clo-s	50	489(34)	819(33)	833(33)	4246(3)	748(23)	1049(29)	**771(39)**	1358(22)	1032(32)	1310(17)	1508(27)	1426(34)	629(35)
des	50	405(19)	601(23)	678(23)	0.0(0)	34.7(1)	918(24)	782(23)	1004(28)	598(22)	843(26)	833(23)	837(24)	**1188(29)**
hap-a	6	**1.7(5)**	5.2(5)	5.2(5)	275(5)	182(5)	39.0(5)	7.8(5)	42.0(5)	12.4(5)	51.3(5)	909(5)	33.2(5)	11.7(5)
pac-pms	40	49.2(12)	21.9(38)	21.6(38)	854(40)	62.2(40)	291(36)	29.8(40)	286(36)	**29.0(40)**	287(36)	30.1(40)	40.4(40)	51.0(40)
pbo-mne	50	611(50)	640(50)	644(50)	990(1)	0.0(0)	94.2(50)	120(50)	116(50)	124(50)	279(50)	273(50)	104(50)	**51.2(50)**
pbo-mnl	50	276(50)	284(50)	309(50)	5926(1)	6360(2)	17.4(50)	26.4(50)	65.9(50)	64.5(50)	153(50)	141(50)	47.5(50)	**15.7(50)**
pbo-rou	15	**0.4(15)**	2.2(15)	2.2(15)	1278(6)	677(8)	6.1(15)	3.8(15)	6.7(15)	4.8(15)	6.3(15)	5.4(15)	4.3(15)	4.3(15)
pro-ins	12	2769(11)	2753(11)	2456(10)	0.7(1)	0.6(1)	500(12)	470(12)	251(12)	350(12)	351(12)	236(12)	241(12)	**230(12)**
tpr-Mp	48	476(36)	452(37)	432(37)	294(12)	271(13)	567(43)	505(41)	327(47)	706(46)	**189(48)**	238(48)	300(48)	363(46)
tpr-Op	50	433(49)	468(50)	485(50)	1463(48)	928(44)	111(50)	166(50)	12.9(50)	12.1(50)	9.1(50)	9.5(50)	**7.7(50)**	94.4(50)

Table 11 continued

Family	#	$wpm2$	$wpm2_s$	$wpm2_{sh}$	$wpm2_{shia}$	$wpm2_{shic}$	$wpm2_{shla}$	$wpm2_{shlc}$	$wpm2_{shba}$	$wpm2_{shbc}$	$wpm2_{shua}*$	$wpm2_{shuc}$	$wpm2_{shucg}$	$wpm2_{shucgo}$
Total	627	429	467	464	239	259	480	486	503	497	502	513	524	**528**
		67.9 %	74.2 %	73.5 %	37.8 %	40.5 %	76.3 %	75.6 %	80.6 %	78.7 %	80.7 %	81.9 %	82.5 %	**82.9 %**
WPMS														
hap-ped	100	57.0(24)	201(25)	201(90)	337(51)	470(57)	135(92)	80.0(91)	76.0(95)	156(96)	142(98)	**240(99)**	152(97)	196(98)
pac-wpms	99	143(28)	163(23)	165(23)	351(72)	192(70)	688(62)	367(69)	422(67)	**312(77)**	347(62)	224(69)	200(73)	401(69)
pre-pla	29	35.9(28)	158(27)	151(27)	404(13)	464(13)	34.3(29)	26.9(29)	15.0(29)	14.2(29)	**13.2(29)**	17.9(29)	13.4(29)	14.6(29)
time	26	706(8)	597(8)	560(8)	0.0(0)	476(1)	**448(9)**	820(9)	730(8)	1205(9)	1045(9)	1407(9)	1055(9)	552(9)
upg-pro	100	16.4(100)	14.4(100)	**11.6(100)**	214(100)	57.5(100)	301(100)	79.2(100)	295(100)	77.2(100)	297(100)	78.1(100)	104(100)	105(100)
wcsp-s5d	21	1032(8)	232(10)	254(12)	**135(15)**	390(15)	433(14)	364(14)	318(14)	322(14)	122(14)	8.4(14)	34.2(14)	17.4(14)
wcsp-s5l	21	0.3(6)	408(10)	125(9)	572(10)	364(11)	58.4(13)	603(14)	50.5(13)	135(14)	17.0(14)	23.3(14)	29.8(14)	**11.7(14)**
Total	396	202	203	269	261	267	319	326	326	**339**	326	334	336	333
		49.5 %	52.5 %	62.4 %	55.4 %	57.2 %	74.0 %	75.5 %	74.6 %	**77.4 %**	75.5 %	76.7 %	77.0 %	76.5 %
Total Ind.	1078	651	691	754	518	545	817	833	846	856	844	865	880	**881**
		61.8 %	66.9 %	69.0 %	43.2 %	45.5 %	73.7 %	73.8 %	76.6 %	76.3 %	76.8 %	78.1 %	78.7 %	**78.9 %**

Best results are in bold

Table 12 Impact of WPM2 improvements, compared on the crafted instances of MSE13 (number and mean ratio of solved instances)

Family	#	$wpm2$	$wpm2_s$	$wpm2_{sh}$	$wpm2_{shia}$	$wpm2_{shic}$	$wpm2_{shla}$	$wpm2_{shlc}$	$wpm2_{shba}$	$wpm2_{shbc}$	$wpm2_{shua}*$	$wpm2_{shuc}$	$wpm2_{shucg}$	$wpm2_{shucgo}$
MS														
bip-mc.7	50	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)
bip-mc.8	50	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)
mc-dm	62	844(10)	797(10)	992(10)	**1390(16)**	1138(15)	688(10)	498(10)	1005(13)	691(12)	359(11)	567(12)	320(11)	854(12)
mc-sg	5	**0.2(2)**	0.2(2)	0.2(2)	1.3(2)	1.7(2)	0.8(2)	0.2(2)	39.4(2)	49.9(2)	14.4(2)	2.6(2)	2.4(2)	17.7(2)
Total	167	12	12	12	**18**	17	12	12	15	14	13	14	13	14
		14.0 %	14.0 %	14.0 %	**16.5 %**	16.0 %	14.0 %	14.0 %	15.2 %	14.8 %	14.4 %	14.8 %	14.4 %	14.8 %
PMS														
frb	25	0.0(0)	0.0(0)	0.0(0)	**1222(1)**	3656(1)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)
job-sh	3	60.6(3)	52.3(3)	65.3(3)	0.0(0)	0.0(0)	54.0(3)	47.2(3)	**41.6(3)**	51.9(3)	55.8(3)	67.8(3)	62.0(3)	65.9(3)
mcq-ran	96	445(76)	450(76)	459(76)	**365(93)**	305(90)	517(78)	497(78)	362(82)	484(84)	551(90)	750(90)	377(87)	569(90)
mcq-str	62	727(21)	569(21)	577(21)	**644(32)**	710(28)	806(24)	755(23)	810(26)	1087(27)	816(27)	1008(27)	680(26)	752(28)
mo-3s	80	15.7(80)	16.7(80)	17.4(80)	81.8(80)	33.3(80)	8.8(80)	9.9(80)	5.0(80)	5.1(80)	6.7(80)	6.6(80)	5.8(80)	**4.4(80)**
mo-str	60	145(58)	81.5(59)	83.9(59)	1300(31)	549(47)	30.1(60)	15.4(60)	45.5(60)	12.9(60)	45.0(60)	**9.0(60)**	14.0(60)	19.2(60)
mine-kbt	42	1325(4)	1872(5)	1623(5)	**1065(7)**	2398(7)	2241(6)	2512(6)	1356(5)	1498(6)	1577(5)	847(6)	1276(6)	1373(6)
pse-ml	4	106(4)	97.9(4)	94.8(4)	26.3(4)	**25.3(4)**	63.1(4)	91.0(4)	28.9(4)	28.3(4)	34.0(4)	47.4(4)	32.5(4)	38.1(4)
sch	5	5652(1)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	3203(1)	3845(1)	1439(1)	3652(1)	804(1)	1210(1)	1276(1)	**688(1)**
Total	377	247	248	248	248	257	256	255	261	265	270	271	267	**272**
		59.9 %	58.1 %	58.1 %	46.8 %	48.7 %	61.6 %	61.4 %	62.1 %	62.8 %	63.2 %	63.5 %	63.0 %	**63.7 %**

Table 12 continued

Family	#	$wpm2$	$wpm2_s$	$wpm2_{sh}$	$wpm2_{shia}$	$wpm2_{shic}$	$wpm2_{shla}$	$wpm2_{shlc}$	$wpm2_{shba}$	$wpm2_{shbc}$	$wpm2_{shua}*$	$wpm2_{shuc}$	$wpm2_{shucg}$	$wpm2_{shucgo}$
WMS														
frb	34	128(4)	129(4)	1439(9)	611(10)	**84.7(10)**	615(9)	769(9)	336(9)	253(9)	182(9)	195(9)	87.2(9)	125(9)
ram	15	0.7(1)	0.4(1)	0.4(1)	474(3)	477(3)	684(2)	556(2)	**404(4)**	624(4)	93.0(3)	84.2(3)	1181(4)	117(3)
wmc-dm	62	6.7(5)	63.0(5)	61.6(5)	**1469(9)**	466(8)	1.4(5)	1.4(5)	2.2(5)	5.1(5)	18.3(5)	6.7(5)	11.4(5)	2.3(5)
wmc-sg	5	0.0(0)	0.0(0)	0.0(0)	**44.5(1)**	45.4(1)	0.0(0)	0.0(0)	216(1)	154(1)	348(1)	435(1)	364(1)	409(1)
Total	116	10	10	15	**23**	22	16	16	19	19	18	18	19	18
		6.6 %	6.6 %	10.3 %	**21.0 %**	20.6 %	12.0 %	12.0 %	20.3 %	20.3 %	18.6 %	18.6 %	20.3 %	18.6 %
WPMS														
CSG	10	866(5)	717(5)	731(5)	69.7(4)	168(4)	34.1(6)	56.8(6)	62.2(10)	66.1(10)	71.2(10)	66.8(10)	**39.3(10)**	45.3(10)
auc-pat	86	0.0(0)	344(1)	349(1)	871(76)	808(75)	525(33)	688(33)	952(75)	957(74)	**276(82)**	426(82)	366(82)	316(82)
auc-sch	84	0.0(0)	140(50)	144(50)	2.7(84)	1.8(84)	71.3(84)	71.8(84)	1.6(84)	2.6(84)	1.0(84)	1.2(84)	**0.9(84)**	0.9(84)
mine-pl	56	420(30)	139(38)	144(38)	696(50)	485(50)	2.6(56)	3.2(56)	0.7(56)	0.7(56)	0.8(56)	0.7(56)	**0.6(56)**	0.7(56)
mine-wa	18	2.9(1)	3.6(1)	3.6(1)	0.1(1)	0.0(1)	0.1(1)	0.1(1)	0.0(1)	0.0(1)	2802(2)	1136(2)	0.1(1)	**2304(4)**
pse-ml	12	371(3)	1488(3)	1570(3)	3.9(2)	2.8(2)	12.9(3)	14.3(3)	767(5)	409(4)	**756(5)**	287(4)	242(4)	260(4)
ran-net	74	0.0(0)	0.0(0)	0.0(0)	1084(36)	1364(42)	1968(16)	977(15)	1510(33)	1704(34)	3275(16)	3292(17)	1409(35)	**2593(48)**
Total	340	39	98	98	253	258	199	198	264	263	255	255	272	**288**
		19.2 %	29.9 %	29.9 %	55.5 %	56.5 %	50.1 %	49.9 %	68.4 %	67.3 %	67.1 %	66.1 %	68.8 %	**73.7 %**
Total Cra.	1000	308	368	373	542	554	483	481	559	561	556	558	571	**592**
		31.5 %	34.0 %	34.6 %	40.0 %	40.8 %	42.0 %	41.9 %	49.2 %	49.0 %	48.8 %	48.7 %	49.5 %	**51.0 %**

Best results are in bold

Table 13 $wpm2_{shucgo}$ compared to MSE13 best complete solvers on the industrial instances of MSE13 (number and mean ratio of solved instances)

Family	#	$wpm2_{shucgo}$	$wpm2$-13∗	$msunc$	$optim$-ni	$qms2$-$g2$	$wpm1$-13	$pwbo2.33$	$maxhs13$	$ilp13$	$w/pmifu$	$wmsz09$	$msz13f$	ckm-s	$ahms$
MS															
cir-dp	3	248(2)	489(2)	21.9(2)	10.2(3)	349(1)	413(2)	3359(1)	737(1)	680(1)	**5.6(3)**	0.0(0)	0.0(0)	0.0(0)	0.0(0)
sean-s	52	493(18)	356(14)	139(24)	1025(35)	939(18)	296(19)	1123(6)	492(6)	294(6)	**250(39)**	0.0(0)	0.0(0)	0.0(0)	0.0(0)
Total	55	20	16	26	38	19	21	7	7	7	**42**	0	0	0	0
		50.6 %	46.8 %	56.4 %	83.7 %	34.0 %	51.6 %	22.4 %	22.4 %	22.4 %	**87.5 %**	0.0 %	0.0 %	0.0 %	0.0 %
PMS															
aes	7	0.0(0)	524(1)	163(1)	112(1)	0.0(0)	2721(1)	0.0(0)	**480(3)**	493(3)	0.0(0)	2123(1)	1661(1)	3.2(1)	2.9(1)
bcp-fir	50	52.6(49)	19.8(49)	10.5(49)	176(48)	39.6(47)	16.9(49)	276(48)	659(37)	**12.2(50)**	77.3(48)	809(35)	746(36)	170(17)	32.5(7)
bcp-hysi	17	120(16)	156(16)	230(16)	148(16)	**300(17)**	80.8(16)	139(15)	360(10)	1.8(5)	123(11)	135(6)	48.1(6)	623(2)	134(7)
bcp-hysu	38	511(34)	256(34)	251(32)	482(31)	**141(35)**	287(28)	435(29)	1114(26)	0.0(0)	113(15)	0.0(0)	0.0(0)	0.0(0)	0.0(0)
bcp-msp	50	591(29)	804(26)	216(30)	514(15)	299(20)	693(8)	219(18)	**317(33)**	572(28)	2.8(3)	982(22)	567(19)	1625(14)	372(5)
bcp-mtg	40	746(36)	441(39)	1.2(40)	0.6(40)	**0.2(40)**	6.0(40)	0.9(40)	8.1(40)	592(28)	251(29)	418(15)	10.1(8)	140(10)	0.0(0)
bcp-syn	50	121(28)	121(24)	57.2(27)	51.2(25)	94.9(19)	154(28)	286(23)	18.0(48)	**14.7(48)**	640(21)	420(18)	319(18)	196(29)	139(14)
cir-tc	4	90.3(4)	94.4(4)	84.3(4)	58.2(4)	**37.5(4)**	145(4)	190(4)	375(1)	473(1)	376(1)	552(1)	0.0(0)	0.0(0)	0.0(0)
clo-s	50	629(35)	1310(17)	256(29)	139(35)	**741(43)**	566(35)	410(44)	968(22)	175(30)	56.5(32)	0.0(0)	188(2)	0.0(0)	0.0(0)
des	50	1188(29)	843(26)	497(33)	733(28)	**429(48)**	580(22)	378(19)	2005(17)	818(18)	402(24)	0.0(0)	0.0(0)	0.0(0)	0.0(0)
hap-a	6	11.7(5)	51.3(5)	44.2(4)	**1.1(5)**	7.2(5)	3.1(5)	13.9(5)	58.9(5)	1209(5)	2.4(5)	0.0(0)	0.0(0)	0.0(0)	0.0(0)
pac-pms	40	51.0(40)	287(36)	254(40)	17.8(40)	7.3(40)	6.2(40)	67.7(40)	18.4(34)	**1.0(40)**	11.4(37)	2833(1)	0.0(0)	0.0(0)	0.0(0)
pbo-mne	50	51.2(50)	279(50)	287(50)	494(50)	188(50)	780(35)	199(50)	**107(50)**	3148(2)	1224(17)	1665(29)	3509(7)	4951(4)	0.0(0)
pbo-mnl	50	15.7(50)	153(50)	85.4(50)	138(50)	79.0(50)	278(35)	129(50)	**15.2(50)**	370(2)	403(24)	1046(37)	2353(21)	3863(12)	0.0(0)
pbo-rou	15	4.3(15)	6.3(15)	1.1(15)	0.6(15)	2.7(15)	4.4(15)	13.6(15)	229(11)	33.6(15)	**0.6(15)**	2.9(5)	8.9(5)	252(1)	0.0(0)

Table 13 continued

Family	#	$wpm2_{shucgo}$	$wpm2$-13∗	$msunc$	$optim$-ni	$qms2$-$g2$	$wpm1$-13	$pwbo2.33$	$maxhs13$	$ilp13$	$w/pmifu$	$wmsz09$	$msz13f$	ckm-s	$ahms$
pro-ins	12	230(12)	351(12)	350(3)	350(12)	**121(12)**	2027(3)	2398(8)	1536(3)	0.3(1)	0.9(1)	3082(5)	480(7)	4.4(1)	0.0(0)
tpr-Mp	48	636(46)	**189(48)**	551(39)	853(38)	621(48)	224(16)	2424(24)	772(46)	2675(28)	1226(4)	0.0(0)	0.0(0)	498(3)	0.0(0)
tpr-Op	50	94.4(50)	**9.1(50)**	41.5(50)	220(50)	153(50)	1082(42)	5214(13)	18.0(50)	36(50)	0.0(0)	395(25)	1143(22)	363(25)	0.0(0)
Total	627	528	502	512	503	**543**	422	445	486	354	287	200	152	119	34
		82.9 %	80.7 %	77.9 %	80.8 %	**85.0 %**	68.4 %	71.9 %	70.6 %	54.1 %	46.2 %	29.0 %	22.8 %	15.2 %	6.0 %
WPMS															
hap-ped	100	196(98)	**142(98)**	267(93)	437(72)	1391(35)	126(93)	264(77)	857(33)	1495(22)	81.7(85)	0.0(0)	0.0(0)	0.0(0)	0.0(0)
pac-wpms	99	401(69)	347(62)	544(12)	116(22)	3065(17)	36.0(91)	31.3(43)	13.1(85)	**0.5(99)**	9.2(45)	2249(23)	1032(27)	0.0(0)	209(2)
pre-pla	29	14.6(29)	**13.2(29)**	218(29)	63.9(29)	19.0(29)	28.0(28)	154(29)	84.6(28)	1013(12)	2.5(11)	1649(7)	6.3(5)	414(5)	49.0(1)
time	26	552(9)	1045(9)	191(8)	393(10)	789(8)	**900(11)**	1078(7)	79.0(1)	0.0(0)	90.0(8)	0.0(0)	0.0(0)	0.0(0)	0.0(0)
upg-pro	100	105(100)	297(100)	70.0(100)	15.9(8)	0.0(0)	4.0(100)	56.3(100)	18.6(100)	**1.0(100)**	26.5(100)	311(47)	2231(90)	0.0(0)	0.0(0)
wcsp-s5d	21	17.4(14)	122(14)	13.9(14)	84.9(13)	550(15)	73.2(14)	110(7)	2.4(6)	**19.4(17)**	17.8(6)	777(5)	1278(6)	0.5(3)	293(4)
wcsp-s5l	21	11.7(14)	17.0(14)	23.7(14)	488(11)	**693(15)**	291(14)	1.0(6)	2.6(6)	899(9)	0.4(6)	1.0(3)	0.3(4)	0.9(4)	0.5(3)
Total	396	333	326	270	165	119	**351**	269	259	259	261	85	132	12	10
		76.5 %	75.5 %	67.0 %	50.7 %	46.5 %	**79.6 %**	58.5 %	53.8 %	55.3 %	50.9 %	18.9 %	26.0 %	7.2 %	5.5 %
Total Ind.	1078	**881**	844	808	706	681	794	721	752	620	590	285	284	131	44
		78.9 %	76.8 %	73.5 %	73.2 %	71.2 %	70.1 %	64.8 %	62.7 %	52.0 %	50.5 %	24.2 %	22.0 %	12.0 %	5.4 %

Best results are in bold

Table 14 $wpm2_{shucgo}$ compared to MSE13 best complete solvers on the crafted instances of MSE13 (number and mean ratio of solved instances)

Family	#	*ilp*13	*wmsz*09	*msz*13*f*	*maxhs*13	$wpm2_{shucgo}$	*qms*2-*g*2	*ahms*	*wpm*2-13∗	*pwbo*2.33	*ckm-s*	*wpm*1-13	*optim-ni*	*msunc*	*w/pmifu*
MS															
mc.7	50	5160(2)	376(50)	304(50)	0.0(0)	0.0(0)	0.0(0)	637(47)	0.0(0)	0.0(0)	**293(50)**	0.0(0)	0.0(0)	0.0(0)	0.0(0)
mc.8	50	5132(6)	290(50)	230(50)	0.0(0)	0.0(0)	0.0(0)	469(48)	0.0(0)	0.0(0)	**206(50)**	0.0(0)	0.0(0)	0.0(0)	0.0(0)
mc-dm	62	1014(34)	193(53)	**115(53)**	745(5)	854(12)	509(8)	58.7(47)	359(11)	856(6)	63.4(52)	962(10)	996(6)	238(7)	8.4(4)
mc-sg	5	**318(4)**	3.0(3)	10.3(3)	2.5(2)	17.7(2)	179(2)	153(4)	14.4(2)	20.1(2)	554(4)	0.7(2)	0.1(1)	0.4(1)	0.1(1)
Total	167	46	**156**	**156**	7	14	10	146	13	8	**156**	12	7	8	5
		37.7 %	86.4 %	86.4 %	12.0 %	14.8 %	13.2 %	86.5 %	14.4 %	12.4 %	**91.0 %**	14.0 %	7.4 %	7.8 %	6.6 %
PMS															
frb	25	821(14)	316(5)	163(5)	704(15)	0.0(0)	**164(25)**	310(5)	0.0(0)	1265(20)	1067(5)	0.0(0)	1577(15)	0.0(0)	111(24)
job-sh	3	0.0(0)	0.0(0)	0.0(0)	217(3)	65.9(3)	**24.9(3)**	0.0(0)	55.8(3)	164(3)	0.0(0)	31.7(2)	102(3)	89.9(3)	34.3(3)
mcq-ran	96	34.3(96)	2.3(96)	**1.7(96)**	26.2(96)	569(90)	308(79)	3.1(96)	551(90)	555(71)	9.3(96)	800(59)	336(78)	191(74)	39.7(2)
mcq-str	62	349(38)	374(38)	**131(38)**	144(38)	752(28)	778(28)	67.4(31)	816(27)	57.8(19)	176(31)	225(14)	602(26)	225(22)	310(15)
mo-3s	80	9.1(80)	0.5(80)	**0.4(80)**	2.6(80)	4.4(80)	622(78)	34.8(80)	6.7(80)	398(72)	5.4(80)	144(80)	389(75)	8.3(80)	557(12)
mo-str	60	278(59)	86.8(58)	124(57)	41.3(60)	19.2(60)	22.8(60)	657(5)	45.0(60)	**9.0(60)**	1108(50)	855(42)	143(54)	60.2(60)	0.2(1)
mine-kbt	42	**212(42)**	2985(38)	2316(38)	1065(8)	1373(6)	138(6)	1202(4)	1577(5)	2010(5)	2300(28)	2147(6)	774(6)	418(6)	2482(1)
pse-ml	4	25.3(4)	583(3)	594(3)	3.9(4)	38.1(4)	**3.0(4)**	508(3)	34.0(4)	148(4)	725(3)	158(4)	7.3(4)	49.3(4)	1.4(3)
sch	5	0.0(0)	0.0(0)	0.0(0)	0.0(0)	688(1)	777(1)	0.0(0)	**804(1)**	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)	0.0(0)
Total	377	**333**	318	317	304	272	284	224	270	254	293	207	261	249	61
		68.4 %	60.4 %	60.2 %	71.1 %	63.7 %	**73.2 %**	40.3 %	63.2 %	65.2 %	55.0 %	48.3 %	64.6 %	58.5 %	35.1 %

Table 14 continued

Family	#	$ilp13$	$wmsz09$	$msz13f$	$maxhs13$	$wpm2_{shucgo}$	$qms2$-$g2$	$ahms$	$wpm2$-$13*$	$pwbo2.33$	ckm-s	$wpm1$-13	$optim$-ni	$msunc$	$w/pmifu$
WMS															
frb	34	649(22)	119(14)	58.0(14)	858(25)	125(9)	2316(19)	130(14)	182(9)	1.7(4)	356(14)	446(5)	539(25)	155(9)	**72.5(33)**
ram	15	221(3)	103(4)	**29.2(4)**	2.4(1)	117(3)	231(3)	16.9(3)	93.0(3)	220(3)	883(4)	0.6(1)	167(3)	115(3)	1.6(1)
wmc-dm	62	1279(46)	277(60)	115(60)	1523(13)	2.3(5)	2052(8)	125(55)	18.5(5)	0.1(4)	**50.0(60)**	99.8(5)	1.0(4)	693(5)	0.1(3)
wmc-sg	5	**658(5)**	32.0(4)	1082(5)	1096(4)	409(1)	0.0(0)	0.1(1)	348(1)	20.4(1)	0.5(1)	0.0(0)	0.0(0)	0.0(0)	0.0(0)
Total	116	76	82	**83**	43	18	30	73	18	12	79	11	32	17	37
		64.7 %	61.2 %	**66.2 %**	45.3 %	18.6 %	22.2 %	42.5 %	18.6 %	14.6 %	46.2 %	7.4 %	25.0 %	13.6 %	27.1 %
WPMS															
CSG	10	172(4)	806(2)	340(1)	**8.0(10)**	45.3(10)	1189(10)	242(1)	71.2(10)	188(10)	0.0(0)	1.5(4)	15.1(6)	35.9(6)	3.3(5)
auc-pat	86	**0.1(86)**	15.1(86)	6.9(86)	**0.1(86)**	316(82)	1444(47)	325(42)	276(82)	1359(30)	33.5(10)	558(72)	657(17)	840(17)	0.1(2)
auc-sch	84	0.1(84)	365(81)	216(84)	**0.1(84)**	0.9(84)	848(84)	134(68)	1.0(84)	71.5(82)	0.0(0)	1.7(84)	240(84)	8.5(84)	0.4(84)
mine-pl	56	156(56)	335(51)	66(41)	8.1(56)	0.7(56)	34.3(56)	95.1(6)	0.8(56)	**0.7(56)**	0.5(10)	4.5(53)	6.3(53)	14.7(53)	145.1(25)
mine-wa	18	**0.1(18)**	1945(9)	2354(9)	0.8(18)	2304(4)	0.3(1)	0.1(17)	2802(2)	4.0(17)	0.2(1)	19.1(14)	0.5(1)	2.8(1)	0.0(0)
pse-ml	12	142(3)	205(4)	490(3)	158(3)	260(4)	**548(5)**	636(1)	756(5)	729(4)	335(2)	316(4)	1954(4)	242(3)	0.1(1)
ran-net	74	225(60)	4267(1)	4240(8)	**11.3(73)**	2593(48)	4678(4)	389(44)	3275(16)	49.7(5)	0.0(0)	6.2(70)	0.0(0)	0.0(0)	65.8(8)
Total	340	311	234	232	**330**	288	207	179	255	204	23	301	165	164	125
		78.0 %	56.1 %	52.7 %	**89.1 %**	73.7 %	58.2 %	44.7 %	67.1 %	66.7 %	7.4 %	74.9 %	44.8 %	43.6 %	30.9 %
Total Cra.	1000	766	**790**	788	684	592	531	622	556	478	451	531	465	438	228
		65.5 %	63.6 %	63.4 %	62.2 %	51.0 %	50.3 %	49.6 %	48.8 %	48.4 %	45.6 %	43.5 %	42.7 %	38.2 %	27.8 %

Best results are in bold

Table 15 *wpm2*$_{shucgo}$ incomplete compared to MSE13 incomplete solvers on the industrial instances of MSE13 (number and mean ratio of solved instances)

Family	#	*wpm2*$_{shucgo}$	*ccls*	*ira-nov*	*optim*
MS					
cir-dp	3	6.80(1)	**4.26(3)**	0.00(0)	0.00(0)
sean-s	52	11.77(17)	**34.56(43)**	9.62(3)	9.97(2)
Total	55	18	**46**	3	2
		32.0 %	**91.0 %**	2.0 %	1.0 %
PMS					
aes	7	15.26(1)	33.09(1)	**35.81(6)**	18.54(2)
bcp-fir	50	**11.22(47)**	17.51(47)	35.25(33)	10.06(11)
bcp-hysi	17	**18.07(17)**	3.09(16)	2.86(7)	5.60(9)
bcp-hysu	38	39.74(32)	**10.34(32)**	0.00(0)	4.88(1)
bcp-msp	50	**26.50(39)**	9.02(14)	16.48(7)	10.79(6)
bcp-mtg	40	2.92(40)	**0.23(40)**	0.00(0)	11.62(2)
bcp-syn	50	5.10(25)	12.48(28)	**13.56(30)**	24.07(28)
cir-tc	4	105.67(4)	**79.21(4)**	0.00(0)	0.00(0)
clo-s	50	43.21(19)	**20.61(46)**	7.14(9)	0.00(0)
des	50	59.75(32)	**31.56(38)**	0.00(0)	0.00(0)
hap-a	6	11.09(5)	**0.41(6)**	0.00(0)	0.00(0)
pac-pms	40	34.45(40)	**1.43(40)**	0.00(0)	0.00(0)
pbo-mne	50	**57.27(47)**	17.81(41)	0.00(0)	0.00(0)
pbo-mnl	50	**20.04(49)**	16.68(48)	0.00(0)	0.00(0)
pbo-rou	15	4.14(15)	**0.33(15)**	5.45(1)	0.00(0)
pro-ins	12	96.55(9)	60.30(7)	**17.61(10)**	0.24(1)
tpr-Mp	48	**64.36(40)**	38.53(36)	0.00(0)	0.00(0)
tpr-Op	50	**6.90(50)**	47.91(42)	0.00(0)	0.00(0)
Total	627	**511**	501	103	60
		80.0 %	79.0 %	20.0 %	10.0 %
WPMS					
hap-ped	100	**38.26(89)**	28.18(64)	13.65(23)	0.00(0)
pac-wpms	99	**104.53(99)**	23.52(23)	0.00(0)	0.00(0)
pre-pla	29	**8.16(24)**	5.83(23)	13.78(10)	1.92(2)
time	26	37.22(10)	**45.85(18)**	22.34(3)	0.00(0)
upg-pro	100	**67.99(82)**	51.55(54)	0.00(0)	0.00(0)
wcsp-s5d	21	**21.20(18)**	1.98(10)	14.10(11)	1.08(5)
wcsp-s5l	21	4.66(14)	0.02(6)	**37.07(16)**	0.86(6)
Total	396	**336**	198	63	13
		77.0 %	52.0 %	28.0 %	8.0 %
Total Ind.	1078	**865**	745	169	75
		76.0 %	73.0 %	21.0 %	9.0 %

Best results are in bold

Table 16 $wpm2_{shucgo}$ incomplete compared to MSE13 incomplete solvers on the crafted instances of MSE13 (number and mean ratio of solved instances)

Family	#	*optim*	$wpm2_{shucgo}$	*ccls*	*ira-nov*
MS					
bip-mc-.7	50	**1.92(50)**	0.00(0)	40.25(31)	0.00(0)
bip-mc-.8	50	**2.61(50)**	0.00(0)	39.30(33)	0.00(0)
mc-dm	62	**1.14(62)**	2.98(13)	2.79(61)	0.61(8)
mc-sg	5	**1.63(5)**	1.00(2)	26.86(4)	0.01(1)
Total	167	**167**	15	129	9
		100.0 %	15.0 %	76.0 %	8.0 %
PMS					
frb	25	**14.77(25)**	0.00(0)	5.51(1)	7.14(2)
job-sh	3	0.00(0)	**37.52(3)**	0.00(0)	53.01(3)
mcq-ran	96	**2.10(96)**	27.26(82)	35.60(74)	22.59(71)
mcq-str	62	**13.82(55)**	7.36(28)	5.52(23)	5.79(20)
mo-3s	80	2.80(77)	**4.98(80)**	6.55(77)	31.66(62)
mo-str	60	0.87(3)	**4.04(59)**	17.74(18)	2.30(56)
mine-kbt	42	**7.24(42)**	9.27(3)	19.97(9)	6.88(4)
pse-ml	4	2.31(4)	6.03(4)	0.62(3)	**1.09(4)**
sch	5	14.51(1)	45.15(1)	0.00(0)	**19.60(3)**
Total	377	**303**	260	205	225
		67.0 %	61.0 %	37.0 %	61.0 %
WMS					
frb	34	**8.11(33)**	10.70(10)	14.46(10)	26.68(26)
ram	15	**44.77(15)**	13.33(3)	22.87(8)	0.00(0)
wmc-dm	62	**1.09(62)**	0.69(6)	2.20(62)	0.04(4)
wmc-sg	5	**52.95(5)**	5.16(1)	8.07(3)	0.00(0)
Total	116	**115**	20	83	30
		99.0 %	19.0 %	60.0 %	20.0 %
WPMS					s
CSG	10	0.00(0)	**54.41(10)**	0.00(0)	1.73(6)
auc-pat	86	**2.17(86)**	35.55(63)	30.08(46)	18.33(14)
auc-sch	84	2.72(84)	**0.98(84)**	0.71(35)	39.42(76)
mine-pl	56	1.12(7)	**0.62(56)**	0.79(10)	0.52(53)
mine-wa	18	**155.70(18)**	0.00(1)	0.04(1)	0.02(1)
pse-ml	12	32.71(4)	**26.21(7)**	1.18(1)	2.13(4)
ran-net	74	**115.17(58)**	8.02(3)	0.00(0)	27.33(13)
Total	340	**257**	224	93	167
		60.0 %	62.0 %	17.0 %	45.0 %
Total Cra.	1000	**842**	519	510	431
		76.0 %	47.0 %	42.0 %	40.0 %

Best results are in bold

References

Andres, B., Kaufmann, B., Matheis, O., Schaub, T.: Unsatisfiability-based optimization in clasp. In: ICLP (Technical Communications), pp. 211–221 (2012)

Ansotegui, C.: Maxsat Latest Developments. Invited Tutorial at CP 2013 (2013a)

Ansotegui, C.: Tutorial: Maxsat Latest Developments (2013b)

Ansótegui, C., Gabàs, J.: Solving (weighted) partial maxsat with ilp. In: CPAIOR, pp. 403–409 (2013)

Ansótegui, C., Bonet, M.L., Levy, J.: On solving MaxSAT through SAT. In: Proceedings of the 12th International Conference of the Catalan Association for Artificial Intelligence (CCIA'09), pp. 284–292 (2009)

Ansotegui, C., Bonet, M.L., Levy, J.: Solving (weighted) partial maxsat through satisfiability testing. In: Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing (SAT'09), pp. 427–440 (2009)

Ansotegui, C., Bonet, M.L., Levy, J.: A new algorithm for weighted partial maxsat. In: Proceedings the 24th National Conference on Artificial Intelligence (AAAI'10), pp. 867–872 (2010)

Ansótegui, C., Bofill, M., Palahí, M., Suy, J., Villaret, M.: A proposal for solving weighted CSPs with SMT. In: Proceedings of the 10th International Workshop on Constraint Modelling and Reformulation (ModRef 2011), pp. 5–19 (2011)

Ansótegui, C., Bonet, M.L., Gabàs, J., Levy, J.: Improving sat-based weighted maxsat solvers. In: Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming (CP'12), pp. 86–101 (2012)

Ansótegui, C., Bonet, M.L., Gabàs, J., Levy, J.: Improving wpm2 for (weighted) partial maxsat. In: CP, pp. 117–132 (2013a)

Ansótegui, C., Bonet, M.L., Levy, J.: Sat-based maxsat algorithms. Artif. Intell. **196**, 77–105 (2013b)

Argelich, J., Li, C.M., Manyà, F., Planes, J.: Maxsat evaluation. http://www.maxsat.udl.cat (2006–2014)

Bailleux, O., Boufkhad, Y., Roussel, O.: New encodings of pseudo-boolean constraints into cnf. In: SAT, pp. 181–194 (2009)

Barrett, C., Stump, A., Tinelli, C.: The Satisfiability Modulo Theories Library (SMT-LIB). http://www.SMT-LIB.org (2010)

Berre, D.L.: Sat4j, a satisfiability library for java. www.sat4j.org (2006)

Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press, Amsterdam (2009)

Bofill, M., Palahí, M., Suy, J., Villaret, M.: Boosting weighted csp resolution with shared bdds. Proceedings of the 12th International Workshop on Constraint Modelling and Reformulation (ModRef 2013), pp. 57–73. Uppsala, Sweden (2013)

Bonet, M.L., Levy, J., Manyà, F.: A complete calculus for Max-SAT. In: SAT, pp. 240–251 (2006)

Borchers, B., Furman, J.: A two-phase exact algorithm for max-sat and weighted max-sat problems. J. Comb. Optim. **2**(4), 299–306 (1998)

Cimatti, A., Franzén, A., Griggio, A., Sebastiani, R., Stenico, C.: Satisfiability modulo the theory of costs: foundations and applications. In: TACAS, pp. 99–113 (2010)

Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. MIT Press, Cambridge, MA (2009)

Davies, J., Bacchus, F.: Solving maxsat by solving a sequence of simpler sat instances. In: Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming (CP'11), pp. 225–239 (2011)

Davies, J., Bacchus, F.: Exploiting the power of mip solvers in maxsat. In: SAT, pp. 166–181 (2013)

Dutertre, B., de Moura, L.: The Yices SMT Solver. http://yices.csl.sri.com (2014)

Eén, N., Sörensson, N.: Translating pseudo-boolean constraints into SAT. JSAT **2**(1–4), 1–26 (2006)

Fu, Z., Malik, S.: On solving the partial max-sat problem. In: Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing (SAT'06), pp. 252–265 (2006)

Heras, F., Larrosa, J., Oliveras, A.: MiniMaxSat: A new weighted Max-SAT solver. In: Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing (SAT'07), pp. 41–55 (2007)

Heras, F., Larrosa, J., Oliveras, A.: Minimaxsat: an efficient weighted max-sat solver. J. Artif. Intell. Res. (JAIR) **31**, 1–32 (2008)

Heras, F., Morgado, A., Marques-Silva, J.: Core-guided binary search algorithms for maximum satisfiability. In: Proceedings of the 25th National Conference on Artificial Intelligence (AAAI'11), pp. 36–41 (2011)

Honjyo, K., Tanjo, T.: Shinmaxsat. A Weighted Partial Max-SAT Solver Inspired by MiniSat+. Information Science and Technology Center, Kobe University, Kobe (2012)

Koshimura, M., Zhang, T., Fujita, H., Hasegawa, R.: Qmaxsat: a partial max-sat solver. JSAT **8**(1/2), 95–100 (2012)

Kügel, A.: Improved exact solver for the weighted MAX-SAT problem. In: POS-10. Pragmatics of SAT, Edinburgh, UK, July 10, 2010, pp. 15–27 (2010)

Larrosa, J., Heras, F.: Resolution in max-sat and its relation to local consistency in weighted csps. In: IJCAI, pp. 193–198 (2005)

Larrosa, J., Heras, F., de Givry, S.: A logical approach to efficient max-sat solving. Artif. Intell. **172**(2–3), 204–233 (2008)

Li, C.M., Manyà, F.: Maxsat, hard and soft constraints. In: Biere, A., van Maaren, H., Walsh, H. (eds.) Handbook of Satisfiability. IOS Press, Amsterdam (2009)

Li, C.M., Manyà, F., Planes, J.: Detecting disjoint inconsistent subformulas for computing lower bounds for max-sat. In: AAAI, pp.86–91 (2006)

Li, C.M., Manyà, F., Planes, J.: New inference rules for Max-SAT. J. Artif. Intell. Res. (JAIR) **30**, 321–359 (2007)

Li, C.M., Manyà, F., Mohamedou, N.O., Planes, J.: Exploiting cycle structures in Max-SAT. In: Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing (SAT'09), pp. 467–480 (2009)

Lin, H., Su, K.: Exploiting inference rules to compute lower bounds for Max-SAT solving. In: IJCAI'07, pp. 2334–2339 (2007)

Lin, H., Su, K., Li, C.M.: Within-problem learning for efficient lower bound computation in Max-SAT solving. In: Proceedings of the 23th National Conference on Artificial Intelligence (AAAI'08), pp. 351–356 (2008)

Manquinho, V., Marques-Silva, J., Planes, J.: Algorithms for weighted boolean optimization. In: Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing (SAT'09), p.p 495–508 (2009)

Manquinho, V.M., Martins, R., Lynce, I.: Improving unsatisfiability-based algorithms for boolean optimization. In: Proceedings of the 13th International Conference on Theory and Applications of Satisfiability Testing (SAT'10), Lecture Notes in Computer Science, vol. 6175, pp. 181–193. Springer, (2010)

Marques-Silva, J., Argelich, J., Graça, A., Lynce, I.: Boolean lexicographic optimization: algorithms and applications. Ann. Math. Artif. Intell. **62**(3–4), 317–343 (2011)

Martins, R., Manquinho, V.M., Lynce, I.: Exploiting cardinality encodings in parallel maximum satisfiability. In: ICTAI, pp. 313–320 (2011)

Martins, R., Manquinho, V.M., Lynce, I.: Clause sharing in parallel maxsat. In: LION, pp. 455–460 (2012)

Martins, R., Joshi, S., Manquinho, V.M., Lynce, I.: ncremental cardinality constraints for maxsat. In: Principles and Practice of Constraint Programming—20th International Conference, CP 2014, Lyon, France, September 8–12, 2014. Proceedings, pp. 531–548 (2014)

Morgado, A., Heras, F., Marques-Silva, J.: Improvements to core-guided binary search for maxsat. In: Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing (SAT'12), pp. 284–297 (2012)

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 (2013a)

Morgado, A., Heras, F., Marques-Silva, J.: Model-guided approaches for maxsat solving. In: 2013 IEEE 25th International Conference on Tools with Artificial Intelligence, Herndon, VA, USA, November 4–6, 2013, pp. 931–938 (2013b)

Morgado, A., Dodaro, C., Marques-Silva, J.: Core-guided maxsat with soft cardinality constraints. In: Proceedings of the Principles and Practice of Constraint Programming—20th International Conference, CP 2014, Lyon, France, September 8–12, 2014. pp. 564–573 (2014)

Narodytska, N., Bacchus, F.: Maximum satisfiability using core-guided maxsat resolution. In: Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27–31, 2014, pp. 2717–2723. Québec City, Canada. (2014)

Nieuwenhuis, R., Oliveras, A.: On sat modulo theories and optimization problems. In: SAT, pp. 156–169 (2006)

Pipatsrisawat, K., Darwiche, A.: Clone: Solving weighted Max-SAT in a reduced search space. In: Australian Conference on Artificial Intelligence, pp. 223–233 (2007)

Raz, R.: Resolution lower bounds for the weak pigeonhole principle. In: Proceedings of the 17th Annual IEEE Conference on Computational Complexity, Montréal, Canada, May 21–24, 2002, p 3 (2002)

Razborov, A.A.: Improved resolution lower bounds for the weak pigeonhole principle. Electron. Colloquium Comput. Complex. (ECCC) **8**(55), (2001)

Rossi, F., van Beek, P., Walsh, T. (eds.): Handbook of Constraint Programming. Elsevier, Amsterdam (2006)

Sebastiani, R.: Lazy satisfiability modulo theories. J. Satisf. Boolean Model. Comput. **3**(3–4), 141–224 (2007)

Sörensen, K.: Metaheuristics the metaphor exposed. Int. Trans. Oper. Res. **22**(1), 3–18 (2015). doi:10.1111/itor.12001

F

MaxSAT by improved instance-specific algorithm configuration

C. Ansótegui, J. Gabàs,
Y. Malitsky, M. Sellmann

Artificial Intelligence 235

2016

Artificial Intelligence 235 (2016) 26–39

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

MaxSAT by improved instance-specific algorithm configuration[☆,☆☆]

Carlos Ansótegui[a], Joel Gabàs[a], Yuri Malitsky[b,*], Meinolf Sellmann[b,*]

[a] *Department of Computer Science, University of Lleida, Spain*
[b] *IBM Watson Research Center, Yorktown Heights, NY 10598, USA*

ARTICLE INFO

Article history:
Received 28 October 2014
Received in revised form 13 September 2015
Accepted 30 December 2015
Available online 8 February 2016

Keywords:
Algorithm configuration
Algorithm selection
MaxSAT

ABSTRACT

Our objective is to boost the state-of-the-art performance in MaxSAT solving. To this end, we employ the instance-specific algorithm configurator ISAC, and improve it with the latest in portfolio technology. Experimental results on SAT show that this combination marks a significant step forward in our ability to tune algorithms instance-specifically. We then apply the new methodology to a number of MaxSAT problem domains and show that the resulting solvers consistently outperform the best existing solvers on the respective problem families. In fact, the solvers presented here were independently evaluated at the 2013 and 2014 MaxSAT Evaluations where they won several categories.

1. Introduction

MaxSAT is the optimization version of the Satisfiability (SAT) problem. It can be used effectively to model problems in several domains, such as scheduling, timetabling, FPGA routing, design and circuit debugging, software package installation, bioinformatics, probabilistic reasoning, etc. From the research perspective, MaxSAT is also of particular interest as it requires the ability to reason about both optimality and feasibility. Depending on the particular problem instance being solved, it is more important to emphasize one or the other of these inherent aspects.

MaxSAT technology has significantly progressed in the last years, thanks to the development of several new core algorithms and the recent revelation that traditional MIP solvers like Cplex can be extremely well suited for solving some families of partial MaxSAT instances [3]. Given that different solution approaches work well on different families of instances, [40] used meta-algorithmic techniques developed in CP and SAT to devise a solver portfolio for MaxSAT. Surprisingly, and in contrast to SAT, until 2013 this idea had not led to the development of a highly efficient MaxSAT solver that would dominate, e.g., the yearly MaxSAT Evaluations [8].

We describe the methodology that led to a MaxSAT portfolio that won several categories at the 2013 and 2014 MaxSAT Evaluations. In particular, we develop an instance-specifically tuned solver for every version of MaxSAT that outperforms all existing solvers in their respective domains. We do this for regular MaxSAT (MS), Partial (PMS), Weighted (WMS), and Weighted Partial (WPMS) MaxSAT. The method we apply to obtain these solvers is a portfolio tuning approach ISAC++ which generalizes both tuning of individual solvers as well as combining multiple solvers into one solver portfolio. As a side-effect

☆ This paper was submitted to the Competition Section of the journal.

☆☆ Research partially supported by the Ministerio de Economía y Competividad research project TASSAT2: TIN2013-48031-C4-4-P.

* Corresponding authors.
E-mail addresses: carlos@diei.udl.es (C. Ansótegui), joel.gabas@diei.udl.es (J. Gabàs), ymalits@us.ibm.com (Y. Malitsky), meinolf@us.ibm.com (M. Sellmann).

http://dx.doi.org/10.1016/j.artint.2015.12.006
0004-3702/© 2016 Elsevier B.V. All rights reserved.

of our work on MaxSAT, we found a way to improve instance-specific algorithm configuration (ISAC) [28] by combining the original methodology with one of the latest and most efficient algorithm portfolio builders to date.

The next section formally introduces our target problem, MaxSAT. Then, we review the current state-of-the-art in instance-specific algorithm configuration and algorithm portfolios. We show how both techniques can be combined and empirically demonstrate on SAT that our improved method works notably better than the original method and other instance-specific algorithm tuners. We then apply the new technique to MaxSAT. Finally, in extensive experiments we show that the developed solvers significantly outperform the current state-of-the-art in every MaxSAT domain.

2. MaxSAT

Problem definition

Definition 1. A *literal* l is either a Boolean variable x or its negation $\bar{x}$. A *clause* c is a disjunction of literals. A *SAT formula* is a set of clauses that represents a Boolean formula in Conjunctive Normal Form (CNF), i.e. a conjunction of clauses.

Definition 2. A *weighted clause* is an ordered pair (c, w), where c is a clause and w is a natural number or infinity (indicating the cost of falsifying c, see Definitions 4 and 5). If w is infinite the clause is *hard*, otherwise it is *soft*. Infinite costs express that hard clauses must be satisfied while soft clauses may be falsified.

Definition 3. A *Weighted Partial MaxSAT (WPMS) formula* is an ordered multi-set of weighted clauses:

$$\varphi = \langle (c_1, w_1), \ldots, (c_s, w_s), (c_{s+1}, \infty), \ldots, (c_{s+h}, \infty) \rangle$$

where the first s clauses are soft and the last h clauses are hard. The presence of soft clauses with different weights makes the formula Weighted and the presence of hard clauses makes it Partial.

To make the definitions a bit less abstract, here are examples for MaxSAT (MS), Partial MaxSAT (PMS) and Weighted Partial MaxSAT (WPMS) formulas:
MS formula: $\langle (x_1, 1), (x_2, 1), (x_3, 1), (\bar{x}_1 \vee \bar{x}_2, 1), (\bar{x}_1 \vee \bar{x}_3, 1), (\bar{x}_2 \vee \bar{x}_3, 1) \rangle$.
PMS formula: $\langle (x_1, 1), (x_2, 1), (x_3, 1), (\bar{x}_1 \vee \bar{x}_2, \infty), (\bar{x}_1 \vee \bar{x}_3, \infty), (\bar{x}_2 \vee \bar{x}_3, \infty) \rangle$.
WPMS formula: $\langle (x_1, 5), (x_2, 3), (x_3, 3), (\bar{x}_1 \vee \bar{x}_2, \infty), (\bar{x}_1 \vee \bar{x}_3, \infty), (\bar{x}_2 \vee \bar{x}_3, \infty) \rangle$.

Definition 4. An *assignment* for a set of Boolean variables X is a function $\mathcal{I} : X \to \{0, 1\}$, that can be extended to literals, (weighted) clauses, SAT formulas and WPMS formulas as follows:

$$\mathcal{I}(\bar{x}) = 1 - \mathcal{I}(x)$$
$$\mathcal{I}(l_1 \vee \ldots \vee l_m) = \max\{\mathcal{I}(l_1), \ldots, \mathcal{I}(l_m)\}$$
$$\mathcal{I}(\{c_1, \ldots, c_n\}) = \min\{\mathcal{I}(c_1), \ldots, \mathcal{I}(c_n)\}$$
$$\mathcal{I}((c, w)) = w\,(1 - \mathcal{I}(c))$$
$$\mathcal{I}(\langle (c_1, w_1), \ldots, (c_{s+h}, w_{s+h}) \rangle) = \sum_{i=1}^{s+h} \mathcal{I}((c_i, w_i))$$

We will refer to the value returned by an assignment $\mathcal{I}$ on a weighted clause or a WPMS formula as the cost of $\mathcal{I}$.

Definition 5. We say that an assignment $\mathcal{I}$ satisfies a clause or a SAT formula if the value returned by $\mathcal{I}$ is equal to 1. In the case of SAT formulas, we will refer also to this assignment as a satisfying assignment or solution. Otherwise, if the value returned by $\mathcal{I}$ is equal to 0, we say that $\mathcal{I}$ falsifies the clause or the SAT formula.

Definition 6. The *SAT problem* for a SAT formula φ is the problem of finding a solution for φ. If a solution exists the formula is satisfiable, otherwise it is unsatisfiable.

Definition 7. Given an unsatisfiable SAT formula φ, an *unsatisfiable core* φ_C is a subset of clauses $\varphi_C \subseteq \varphi$ that is also unsatisfiable. A *minimal unsatisfiable core* is an unsatisfiable core such that any proper subset of it is satisfiable.

Given the SAT formula: $\varphi = \{(x_1), (x_2), (x_3), (\bar{x}_1 \vee \bar{x}_2), (\bar{x}_1 \vee \bar{x}_3), (\bar{x}_2 \vee \bar{x}_3)\}$ we have that $\{(x_1), (x_2), (x_3), (\bar{x}_1 \vee \bar{x}_2)\} \subseteq \varphi$ is an unsatisfiable core and $\{(x_1), (x_2), (\bar{x}_1 \vee \bar{x}_2)\} \subseteq \varphi$ is a minimal unsatisfiable core.

Definition 8. A SAT algorithm for the SAT problem, takes as input a SAT formula φ and returns an assignment $\mathcal{I}$ such that $\mathcal{I}(\varphi) = 1$ if the formula is satisfiable. Otherwise, it returns an unsatisfiable core φ_C.

Given unlimited resources of time and memory, we say that a SAT algorithm is *complete* if it terminates for any SAT formula. We say that it is *incomplete* if it only terminates for some formulas.

Definition 9. The *optimal cost (or optimum)* of a WPMS formula φ is $\text{cost}(\varphi) = \min\{\mathcal{I}(\varphi) \mid \mathcal{I} : \text{var}(\varphi) \to \{0, 1\}\}$ and an *optimal assignment* is an assignment $\mathcal{I}$ such that $\mathcal{I}(\varphi) = \text{cost}(\varphi)$. We will refer to this assignment as a solution for φ if $\mathcal{I}(\varphi) \neq \infty$. Any cost above (below) *cost*(φ) is called an upper (lower) bound for φ.

Definition 10. The *Weighted Partial MaxSAT problem* for a WPMS formula φ is the problem of finding a solution for φ. If a solution does not exist the formula is unsatisfiable.

Definition 11. A WPMS algorithm for the WPMS problem, takes as input a WPMS formula φ and returns an assignment $\mathcal{I}$, such that, $\mathcal{I}(\varphi) \geq cost(\varphi)$.

Given unlimited resources of time and memory, we say that a WPMS algorithm is *complete or exact* if for any input WPMS formula φ and returned $\mathcal{I}$, $\mathcal{I}(\varphi) = cost(\varphi)$. Otherwise, we say it is *incomplete*.

Solvers

Among the state-of-the-art MaxSAT solvers, we find two main approaches: branch-and-bound-based algorithms [21,9,18,35,3] and SAT-based solvers [19,39,2,38]. Branch-and-bound-based solvers extend SAT search algorithms with a branch-and-bound scheme. Each *branch* of the search tree is checked against the upper and lower *bound* estimates on the cost of an optimal solution, and it is discarded if the current lower bound exceeds the upper bound, i.e., the partial assignment represented by the current branch cannot be extended to a better solution. Efficient lower-bound refinement and estimation techniques and incomplete MaxSAT inference rules have been developed to boost the search in this context (see, e.g., [10,33]).

On the other hand, SAT-based MaxSAT algorithms basically reformulate a MaxSAT instance into a sequence of SAT instances (see [6,41] for further information). From the annual results of the international MaxSAT Evaluation [8], we can see that SAT-based solvers clearly dominate on industrial and some crafted instances, while branch-and-bound solvers dominate on random and some families of crafted instances.

In this setting, employing multiple solution techniques is well motivated. Consequently, in [40] a MaxSAT portfolio was devised and tested in a promising but limited experimental evaluation. In particular, [40] used SATzilla's 2009 approach to build a portfolio of solvers for MaxSAT. The proposed portfolio was then applied on some pure MaxSAT instances, i.e., formulas where all clauses have weight 1 (which implies that there are no hard clauses). The format of these instances is exactly the DIMACS CNF format. Therefore, [40] could use existing SAT instance-features to characterize the given MaxSAT instances. The particular features used were problem size features, balance features, and local search probe features. The extension to partial and weighted MaxSAT instances, which would have required the definition of new features, was left for future work. To our knowledge this is the only existing prior study of MaxSAT portfolios. In particular, there had been no dominant algorithm portfolio in the annual MaxSAT Evaluations [8] before we conducted the work summarized in this paper.

We devise instance-specific solvers for each MaxSAT domain. They are based on algorithm tuning and algorithm portfolios. In the next two sections we develop the technology that we will then later use to build a novel solver for MaxSAT.

3. Meta-algorithms

Just as we observed for MaxSAT, in the practice of combinatorial search algorithms there is oftentimes no single solver that performs best on every single instance family. Rather, different algorithms and even different parameterizations of the same solver excel on different instance families. This is the underlying reason why algorithm portfolios have been so successful in SAT [55,27], CP [43], and QBF [47]. Namely, all these portfolio builders select and schedule solvers *instance-specifically*.

In the literature, we find two meta-algorithmic approaches for making solvers instance-specific. The first are algorithm portfolio builders for given sets of solvers, the second are instance-specific tuners for parametrized solvers. In the following, we will review the state-of-the-art in both related areas.

Algorithm portfolios

The first approach on algorithm selection that stood-out was SATzilla-2007 [55]. In this approach, a regression function is trained to predict the performance of every solver in the given set of solvers based on the features of an instance. When faced with a new instance, the solver with the best predicted runtime is run on the given instance. The resulting SAT portfolios excelled in the SAT Competitions in 2007 and in 2009 and pushed the state-of-the-art in SAT solving.

Meanwhile, better performing algorithm portfolio builders have been developed. For a while the trend was towards more highly biased regression and classification models [48]. Then, the simple k-nearest neighbor (k-NN)-based portfolio 3S [13] won in the 2011 SAT Competition. 3S is notable because it was the first portfolio that excelled in different categories while using the *same* solver and training base for all categories: random, combinatorial, and industrial (SATzilla had won multiple categories earlier, but by entering a different portfolio tailored for each instance category). This was achieved by using a low-bias ([27] call it "non-model based") machine learning approach for selecting the primary solver used to tackle the given instance. Namely, 3S uses a cost-sensitive k-NN approach for this purpose.

The latest SATzilla [56] now also uses a low-bias machine learning approach, that relies on cost-sensitive decision forests and voting. For every pair of solvers in its portfolio, a forest of binary decision trees is trained to choose what is the better choice for the instance at hand. The predictions are then aggregated and the solver with the most number of favorable pairwise predictions is used to solve the given instance. This portfolio clearly dominated the 2012 SAT Challenge [16] where

Algorithm 1 Instance-Specific Algorithm Configuration.

```
ISAC-Learn($A, T, F, \kappa$)
$(\bar{F}, s, t) \leftarrow$ Normalize($F$)
$(k, C, S) \leftarrow$ Cluster $(T, \bar{F}, \kappa)$
for all $i = 1, \ldots, k$ do
    $P_i \leftarrow GGA(A, S_i)$
end forReturn $(k, P, C, s, t)$
ISAC-Run($A, x, k, P, C, d, s, t$)
$f \leftarrow$ Features($x$)
$\bar{f}_i \leftarrow 2(f_i/s_i) - t_i \ \forall\, i$
$i \leftarrow \min_i(||\bar{f} - C_i||)$ Return $A(x, P_i)$
```

it performed best on both industrial and combinatorial instances. Moreover, the SATzilla-all portfolio, which is *identical* for all three categories, came in second in both categories.

In 2013, a new portfolio builder was introduced [37]. This tool, named CSHC, is based on cost-sensitive hierarchical clustering of training instances. CSHC combines the ability of SATzilla-2012 to handle large and partly uninformative feature sets (difficult for 3S as the distance metric is corrupted) with 3S' ability to handle large sets of base solvers (difficult for SATzilla-2012 as it trains a random forest for each pair of solvers).

Specifically, this method introduced a novel splitting criterion for building tree classifiers. The split feature and value are chosen such that the cumulative performance when solving all training instances in each partition with the same solver is minimized. That is, sub-partitions are selected such that the instances in each partition can best agree on one compromise solver to handle them all (this solver is then associated with the corresponding node). In CSHC, a forest of such trees is trained, whereby for each tree only a subset of features is available for splitting and the training set is a random sub-selection with replacement of the original full training set.

At the time of classification, the features of the test instance are used to determine, for each tree, which leaf the instance falls into. The solver associated with that leaf then receives a vote in favor. Overall, the solver with the most votes is selected to solve the given instance. CSHC was shown to achieve state-of-the-art performance when it won two categories in the 2013 SAT Competition.

Algorithm tuning

Portfolio approaches are very powerful in practice, but there are many domains that do not have a plethora of diverse high-performance solvers. Often, though, there exists at least one solver that is highly parametrized. In such cases, it may be possible to configure the parameters of the solver to gain the most benefit on a particular benchmark.

The fact that there are often subtle non-linear interactions between parameters of sophisticated state-of-the-art algorithms makes manual tuning very difficult. Consequently, a number of automated algorithm configuration and parameter tuning approaches have been proposed over the last decade. These approaches range from gradient-free numerical optimization [12], to gradient-based optimization [17], to iterative improvement techniques [1], and to iterated local search techniques like ParamILS [24]. As of this writing, arguably the two most successful techniques are the model-based predictions of SMAC [25] and the population-based local search approach Gender-based Genetic Algorithm (GGA) [4].

In light of the success of these (one-configuration-fits-all) tuning methods, a number of studies explored how to use them to effectively create instance-specific tuners. Hydra [54], for example, uses the parameter tuner ParamILS [24] to iteratively tune the solver and add parameterizations to a SATzilla portfolio that optimizes the final performance.

4. The ISAC method

With the objective to boost performance in MaxSAT, we exploit an approach called Instance-Specific Algorithm Configuration (ISAC) [28] that we recap in detail in this section. ISAC has been previously shown to outperform the regression-based SATzilla-2009 approach and, when coupled with the parameter configurator GGA, ISAC outperformed Hydra [54] on several standard benchmarks [36].

ISAC is an example of a low-bias approach. Unlike similar approaches, such as Hydra [54] and ArgoSmart [42], ISAC does not use regression-based analysis. Instead, it computes a representative feature vector that characterizes the given input instance in order to identify clusters of similar instances. The training instances are therefore partitioned and a single solver parametrization is searched for each partition to optimize the desired performance characteristic. Given a new instance, its features are computed and it is then solved by the parametrization that was found for the nearest cluster.

More specifically, ISAC works as follows (see Algorithm 1). In the learning phase, ISAC is provided with a parametrized solver A, a list of training instances T, their corresponding feature vectors F, and the minimum cluster size κ. First, the gathered features are normalized so that every feature ranges from $[-1, 1]$, and the scaling and translation values for each feature (s, t) are memorized. This normalization helps keep all the features at the same order of magnitude, and thereby keeps the larger range values from being given more weight than the lower ranging values. Somewhat surprisingly, while there have been some works exploring the application of feature filtering to this process, no significant improvements were demonstrated [31,15].

Next, the instances are clustered based on the normalized feature vectors. Clustering is advantageous for several reasons. First, training parameters on a collection of instances generally provides more robust parameters than one could obtain when tuning on individual instances. That is, tuning on a collection of instances helps prevent over-tuning and allows parameters to generalize to similar instances. Secondly, the parameters found are “pre-stabilized,” meaning they are shown to work well *together*.

While alternatives have been investigated in [15], ISAC uses g-means [20] for clustering. This clustering methodology assumes a good cluster to be Gaussian distributed, and iteratively splits clusters in two until they all pass the Anderson–Darling statistic, a statistical test for “Gaussianness.” Robust parameter sets are obtained by not allowing clusters to contain fewer than a manually chosen threshold, a value which depends on the size of the dataset. In our case, we restrict clusters to have at least 50 instances. Beginning with the smallest cluster, the corresponding instances are re-distributed to the nearest clusters, where proximity is measured by the Euclidean distance of each instance to the cluster’s center. The final result of the clustering is a number of k clusters S_i, and a list of cluster centers C_i. Then, for each cluster of instances S_i, favorable parameters P_i are computed using the instance-oblivious tuning algorithm GGA.

When running algorithm A on an input instance x, ISAC first computes the features of the input and normalizes them using the previously stored scaling and translation values for each feature. Then, the instance is assigned to the nearest cluster. Finally, ISAC runs A on x using the parameters for this cluster.

5. Portfolio tuner

Note how ISAC solves a core problem of instance-specific algorithm tuning, namely the selection of a parametrization out of a very large and possibly even infinite pool of possible parameter settings. In algorithm portfolios we are dealing with a small set of solvers, and all methods devised for algorithm selection make heavy use of that fact. Clearly, these portfolio approaches will not work when the number of solvers explodes.

ISAC overcomes this problem by *clustering* the training instances. This is a key step in the ISAC methodology as described in [28]: Training instances are first clustered into groups and then a high-performance parametrization is computed for each of the clusters. That is, in ISAC clustering is used *both* for the *generation* of high-quality solver parameterizations, and then for the subsequent *selection* of the parametrization for a given test instance.

Beyond cluster-based algorithm selection

While [36] showed that cluster-based solver selection could outperform the original SATzilla-2009 approach in a setting where informative features were available, this alone does not fully explain why ISAC often competed well against other instance-specific algorithm configurators like Hydra. Clustering instances upfront appears to give us an advantage when tuning individual parameterizations. Not only do we save a lot of tuning time with this methodology, since the training set for the instance-oblivious tuner is much smaller than the whole set. We also bundle instances together, hoping that they are somewhat similar and thus amenable for being solved efficiently with just one parametrization.

Consequently, we want to keep clustering in ISAC. However, and this is the core observation in this paper, *once the parameterizations for each cluster have been computed, there is no reason why we would need to stick to these clusters for selecting the best parametrization for a given test instance*. Consequently, we propose to use an alternate state-of-the-art algorithm selector to choose the best parametrization for the instance we are to solve.

To this end, after ISAC finishes clustering and tuning the parameters of existing solvers on each cluster, we can then use any algorithm selector to choose one of the parameterizations, *independent of the cluster* an instance belongs to! For this final stage, we can use any efficient algorithm selector. In our experiments, we will use CSHC. We name the resulting approach Portfolio Tuner (ISAC++).

To summarize the new approach, here is the process as a whole. First, we compute features for each training instance and normalize these. We then cluster the training instances as represented by their normalized features. For each cluster, we then use an instance-oblivious tuner to compute a good parametrization that works well for all instances in the same cluster. At runtime, we compute the features of the instance to be solved. We normalize these features in the same way we normalized the training instances. Up to this point, the original ISAC and the new ISAC++ work exactly the same. ISAC now computes the cluster nearest to the given instance, as measured by the distance of the cluster center to the normalized feature vector of the given instance. ISAC++, on the other hand, uses an algorithm selector to determine which parametrization should be used for the given instance. Throughout this paper, we use GGA to tune instance-obliviously, and CSHC for algorithm selection.

Comparison of ISAC++ with ISAC and Hydra

Before we return to our goal of devising new cutting-edge solvers for MaxSAT, we want to test the ISAC++ methodology in practice and compare it with the best instance-specific algorithm configurators to date, ISAC and Hydra.

We use the benchmark set from [54] where Hydra was first introduced. In particular, there are two non-trivial sets of instances: Random (RAND) and Crafted (HAND).

Following the previously established methodology, we start our portfolio construction with 11 local search solvers: paws [49], rsaps [26], saps [50], agwsat0 [52], agwsat+ [53], agwsatp [51], gnovelty+ [45], g2wsat [34], ranov [44], vw [46], and anov09 [23]. We augment these solvers by adding six fixed parameterizations of SATenstein [29] to this set, giving us a total of 17 constituent solvers.

We clustered the training instances of each dataset and added GGA trained versions of SATenstein for each cluster, resulting in 11 new solvers for Random and 8 for Crafted. We used a timeout of 50 seconds when training these solvers,

Table 1
HAND.

	Average	PAR1	PAR10	Solved	%Solved
BS	28.71	289.3	2753	93	54.39
Hydra	19.80	260.7	2503	100	58.48
ISAC-GGA	18.79	297.5	2887	89	52.05
ISAC-MSC	18.24	273.4	2642	96	56.14
ISAC++	22.09	251.9	2395	103	60.23
VBS	16.40	228.0	2186	109	64.33

Table 2
RAND.

	Average	PAR1	PAR10	Solved	%Solved
BS	27.37	121.0	1004	486	83.64
Hydra	20.88	75.7	586.9	526	90.53
ISAC-GGA	22.11	154.4	1390	448	77.11
ISAC-MSC	27.47	79.7	572.3	528	90.88
ISAC++	24.77	71.1	506.3	534	91.91
VBS	15.96	61.2	479.5	536	92.25

but employed a 600 seconds timeout to evaluate the solvers on each respective dataset. The times were measured on dual Intel Xeon 5540 (2.53 GHz) quad-core Nehalem processors and 24 GB of DDR-3 memory (1333 GHz).

In Table 1 we show the test performance of various solvers on the HAND benchmark set (342 train and 171 test instances). We conduct 5 runs on each instance for each solver. When referring to a value as 'Average', we give the mean time it takes to solve only those instances that do not timeout. The value 'PAR1' includes the timeout instances when computing the average. 'PAR10', then gives a penalized average, where every instance that times out is treated as having taken 10 times the timeout to complete. Finally, we present the number of instances solved and the corresponding percentage of solved instances in the test set.

The best single solver (BS) is one of the SATenstein parameterizations tuned by GGA and is able to solve about 54% of all instances. Hydra solves 58% while a portfolio consisting only of SATenstein parameterizations (ISAC-GGA) solves only 52%. Using the whole set of solvers for tuning (ISAC-MSC) solves about 56% of all instances, which is better than ISAC-GGA but still not overly convincing performance. By augmenting the approach using a final portfolio selection stage, we can boost performance. ISAC++ solves ∼60% of all test instances, outperforming all other approaches and closing almost 30% of the GAP between Hydra and the Virtual Best Solver (VBS), an imaginary perfect oracle that always correctly picks the best solver and parametrization (among all parameterizations generated by all approaches considered here) for each instance. The VBS marks a good estimate on the maximum performance we may realistically hope for.

The second benchmark we present here is RAND. There are 581 test and 1141 train instances in this benchmark. In Table 2 we see that the best single solver (BS – gnovelty+) solves ∼84% of the 581 instances in this test set. Hydra improves this to ∼91%, roughly equal in performance to ISAC-MSC. ISAC++ improves performance again and leads to almost 92% of all instances solved within the time-limit. The improved approach outperforms all other methods, and ISAC++ closes over 37% of the gap between the original ISAC and the VBS.

Note that using portfolios of the untuned SAT solvers only is in general not competitive as shown in [54] and [28]. To verify this finding we also ran a comparison using untuned base solvers only. On the SAT RAND dataset, for example, we find that CSHC using only 17 base solvers can only solve 520 instances, which is not competitive.

Note also, that the new version of ISAC++ is able to achieve a much higher performance than ISAC, while still requiring significantly less wall-clock training time than Hydra. Recall that Hydra works by iteratively tuning a new configuration of SATenstein to improve the current portfolio. This means that, in order to tune its five or so new configurations, it must do so sequentially. Alternatively, the cluster-based methodology embraced by ISAC allows for all the configuration steps to be run in parallel.

6. ISAC++ for MaxSAT

In the preceding section we demonstrated the potential effectiveness of the new ISAC++ approach on SAT problems. We now apply this methodology to our main target, the MaxSAT problem. In order to apply the ISAC++ methodology, we obviously first need to address how MaxSAT instances can be characterized.

Feature computation

As we aim to tackle the variety of existing MaxSAT problems, we should not rely directly on the instance features used in [40] which considered instances where all clauses are soft with identical weights. We therefore compute the percentage of clauses that are soft, and the statistics of the distribution of weights: mean, minimum, maximum, and standard deviation. The remaining 32 features we use are a subset of the standard SAT features based on the entire formula, ignoring the

Table 3
PMS Crafted.

	Average	PAR1	PAR10	Solved	%Solved
BS	187.9	473.1	3339	107	82.31
ISAC-GGA	115.2	478.1	3967	102	78.46
ISAC-MSC	56.2	190.3	1436	120	92.31
ISAC++	60.7	87.5	332.9	128	98.46
VBS	40.7	40.7	40.7	130	100

weights. Specifically, these features cover statistics like the number of variables, number of clauses, proportion of positive to negative literals, the number of clauses a variable appears in on average, etc.

Solvers

To apply ISAC++ we also need a parametrized MaxSAT solver that we can tune. In the past three years, SAT-based MaxSAT solvers have become very efficient at solving industrial MaxSAT instances, and perform well on most crafted instances. Also, with annual MaxSAT Evaluations since 2006, there have been a number of diverse methodologies and solvers proposed. akmaxsat_ls [32], for example, is a branch-and-bound algorithm with lazy deletion and a local search for an initial upper bound. This solver dominated the randomly generated partial MaxSAT problems in the 2012 MaxSAT Evaluations [8]. The solver also scored second place for crafted partial MaxSAT instances. Alternatively, solvers like ShinMaxSAT [22] and sat4j [14] tackle weighted partial MaxSAT problems by encoding them to SAT and then resolving them using a dedicated SAT solver. Finally, there are solvers like WPM1 [2] or wbo1.6 [38] that are based on iterative identification of unsatisfiable cores and are well suited for unweighted Industrial MaxSAT.

One of the few parametrized highly efficient partial MaxSAT solvers is qMaxSAT [30] which is based on SAT. QMaxSat searches for the optimum cost(φ) from $k = \sum_{i=1}^{m} w_i$ to some value smaller than cost(φ). Each subproblem is solved by employing the underlying SAT solver *glucose*. QMaxSat inherits its parameters from glucose: rnd-init, -luby, -rnd-freq, -var-dec, -cla-decay, -rinc and -rfirst [11]. The particular version of *QMaxSat*, *QMaxSatg2*, that we use in our evaluation was the winner for the industrial partial MaxSAT subcategory at the MaxSAT 2012 Evaluation.

Numerical results

Now, we have everything in place to run the ISAC++ methodology and devise a new MaxSAT solver; the primary objective of this study. We conducted our experimentation on the same environment as the MaxSAT Evaluation 2012 [8]: operating system Rocks Cluster 4.0.0 Linux 2.6.9, processor AMD Opteron 248 Processor 2 GHz, memory 0.5 GB and compilers GCC 3.4.3 and javac JDK 1.5.0.

We split our experiments into two parts. We first show the performance of ISAC++ on partial MaxSAT instances: a benchmark set of crafted instances, one consisting of industrial instances, and finally a set that contains both crafted and industrial instances. In the second set of experiments we train solvers for MaxSAT (MS), Weighted MaxSAT (WMS), Partial MaxSAT (PMS), and Weighted Partial MaxSAT (WPMS). In these datasets we will combine instances from the crafted, industrial and random subcategories.

Partial MaxSAT

We used three benchmarks in our numeric analysis obtained from the 2012 MaxSAT Evaluation: (i) the 8 families of partial MaxSAT crafted instances with a total of 372 instances, (ii) the 13 families of partial MaxSAT industrial instances with a total of 504 instances, and the mixture of both sets. This data was split into training and testing sets. Crafted had 130 testing and 242 training, while Industrial instances were split so there were 170 testing and 334 training instances. Our third dataset merged Crafted and Industrial instances and had 300 testing and 576 training instances.

The solvers we run on the partial MaxSAT industrial and crafted instances are: QMaxSat-g2 (this is the solver we tune), pwbo2.0, QMaxSat, PM2, ShinMaxSat, Sat4j, WPM1, wbo1.6, WMaxSatz+, WMaxSatz09, akmaxsat, akmaxsat_ls, iut_rr_rv and iut_rr_ls. More details on solvers and competition results can be found in [8].

For each of these benchmark sets we built an instance-specifically tuned MaxSAT solver by applying the ISAC++ methodology. We use a training set (which is always distinct from the test set on which we report results) of instances which we cluster. For each cluster we tune a parametrization of QMaxSat-g2. Then we combine these parameterizations with the other MaxSAT solvers described above. For this set of algorithms, we train an algorithm selector using CSHC. Finally, we evaluate the performance of the resulting solver on the corresponding test set.

In Table 3 we show the test performance of various solvers. BS shows the performance of the single best untuned solver from our base set. It solves 82% of all 130 instances in this set. ISAC-GGA, which instance-specifically tunes only QMaxSat without using other solvers, solves 78%. In ISAC-MSC [36] we incorporate also other high-performance MaxSAT solvers. Performance jumps, ISAC-MSC solves over 92% of all instances within our time-limit of 1800 seconds.

ISAC++ does even better. It solves over 98% of all instances, closing the gap between ISAC-MSC and VBS by almost 80%! Compared to the previous state of the art (BS), we increase the number of solved instances from 107 to 128. Seeing that, in this subcategory, at the 2012 MaxSAT Evaluation the top five solvers were ranked just 20 instances apart, this improvement

Table 4
PMS Industrial.

	Average	PAR1	PAR10	Solved	%Solved
BS	64.0	186.5	1327	158	92.94
ISAC-GGA	64.0	186.6	1330	158	92.94
ISAC-MSC	108.9	208.4	1161	160	94.12
ISAC++	56.7	138.7	865.2	162	95.29
VBS	45.4	45.4	45.4	170	100

Table 5
PMS Crafted + Industrial.

	Average	PAR1	PAR10	Solved	%Solved
BS	88.2	316.4	2476	260	86.7
ISAC-GGA	90.5	312.7	2418	261	87.0
ISAC-MSC	100.5	242.1	1592	275	91.7
ISAC++	45.0	115.2	1453	288	96.0
VBS	43.3	43.3	43.3	300	100

Table 6
PMS Crafted + Industrial using only QMaxSat.

	Average	PAR1	PAR10	Solved	%Solved
QMaxSat	134.3	378.6	2754	256	85.3
GGA	78.4	308.0	2468	260	86.7
ISAC-GGA	90.5	312.7	2418	261	87.0
ISAC++	85.2	291.0	2585	264	88.0
VBS	82.2	254.0	1873	270	90.0

is significant. In Table 4 we see exactly the same trend, albeit a bit less pronounced: ISAC++ closes about 20% of the gap between ISAC and the VBS.

In the subsequent experiment, we built a MaxSAT solver that excels on both crafted and industrial MaxSAT instances. Table 5 shows the results. The single best solver for this mixed set of instances is the default QMaxSat-g2, and it solves about 87% of all instances within 1800 seconds. It is worth noting that this was the state-of-the-art in partial MaxSAT before we conducted this work. Tuning QMaxSat-g2 instance-specifically (ISAC-GGA), we improve performance only slightly. ISAC-MSC works clearly better and is able to solve almost 92% of all instances. The best performing approach is ISAC++ which solves 96% of all instances in time, closing the gap between perfect performance and the state-of-the-art in partial MaxSAT before we conducted this study by over 60%.

Given the performance of ISAC++ on the combined PMS dataset, a logical question is what impact using CSHC at the portfolio stage has when tuning one solver rather than an entire portfolio of solvers. Table 6 aims to answer this question by showing the performances if only the QMaxSat solver was available to the user. Immediately, we observe that it is beneficial to tune the solver even when doing so instance-obliviously. After tuning with GGA, we solve an additional four instances. If we tune QMaxSat instance-specifically using the original ISAC methodology, we only slightly improve performance.

However, the improved instance-specific tuner ISAC++ solves yet another four instances more than the instance-obliviously tuned QMaxSat. This demonstrates that, through tuning, we are able to find parameterizations that solve instances that could not be solved before. In fact, when analyzing MaxSAT Evaluation results in greater detail, we find that our method generated parameterizations that no other solver could solve before, within the competition settings. For example, at the 2013 MaxSAT Evaluation MSE13, ISAC+2013 solved instances that no other solver submitted was able to solve, using one of the parameterizations that ISAC++ had generated automatically.

The new portfolio stage in ISAC++ helps invoke these automatically generated parameterizations when they are best suited for a given instance. As Table 6 shows, a more effective way of choosing the right parametrization is quite important and leads to a better realization of the potential that the parameterizations that were generated offline offer.

Partial/weighted MaxSAT

The previous experiments were conducted on particular train/test splits. To harden these results, we conduct a 10-fold cross validation on the four categories of the 2012 MaxSAT Evaluation [8]. These are plain MaxSAT instances, weighted MaxSAT, partial MaxSAT, and weighted partial MaxSAT. The results of the cross validation are presented in Tables 7–10. Specifically, each dataset is broken uniformly at random into non-overlapping subsets. Each of these subsets is then used as the test set (one at a time) while the instances from all other folds are used as training data. The tables present the average performance over 10-folds. All experiments were run in the same machines as in the previous section. We use the following solvers: akmaxsat_ls, akmaxsat, bincd2, WPM1-2012, pwbo2.1, wbo1.6-cnf, QMaxSat-g2, ShinMaxSat, WMaxSatz09, and WMaxSatz+. We also employ the highly parametrized solver QMaxSat-g2.

Table 7
MS MIX has 60 test instances per fold.

	Average	PAR1	PAR10	Solved	% Solved
BS	117.0	600.5	5199	45.4	75.7
ISAC-MSC	146.3	603.3	4887	47.2	78.7
ISAC++	134.5	487.7	3952	49.0	81.7
VBS	115.9	473.8	3876	49.2	82.0

Table 8
PMS MIX has 108 test instances per fold.

	Average	PAR1	PAR10	Solved	% Solved
BS	68.0	822.3	7834	68.0	63.0
ISAC-MSC	100.1	328.3	2398	96.1	89.0
ISAC++	98.4	232.7	1713	99.6	92.2
VBS	69.9	206.2	1476	100.8	93.3

Table 9
WMS MIX has 27 test instances per fold.

	Average	PAR1	PAR10	Solved	% Solved
BS	50.2	302.7	2633	23.7	87.9
ISAC-MSC	65.6	323.5	2653	23.7	87.9
ISAC++	58.8	184.3	1349	25.3	93.8
VBS	58.6	184.3	1349	25.3	93.8

Table 10
WPMS MIX has 71 test instances per fold.

	Average	PAR1	PAR10	Solved	% Solved
BS	56.3	632.1	5949	51.1	72.0
ISAC-MSC	47.1	229.0	1914	64.7	91.1
ISAC++	54.6	168.6	1511	66.0	92.9
VBS	15.5	131.8	1185	67.1	94.5

The MS dataset has 600 instances, split among random, crafted and industrial. Each fold has 60 instances, which means that, ten times, we train on 540 instances and test on 60. Results in Table 7 confirm the findings observed in previous experiments. The combination of tuning and using an algorithm portfolio realized by ISAC++ improves over all other methods and, in this case, nearly completely closes the gap between BS and VBS.

The partial MaxSAT dataset is similar to the one used in the previous section, but in this case we also augment it with randomly generated instances bringing the count up to 1086 instances. The Weighted MaxSAT problems consist of only crafted and random instances creating a dataset of size 277. Finally, the weighted partial MaxSAT instances number 718.

All in all, these cross-validation experiments on all MaxSAT families show clearly that ISAC++ always outperforms the original ISAC methodology significantly, closing the gap between ISAC-MSC and the VBS by 90%, 74%, 100%, and 52%. More importantly for the objective of this study, we improve the prior state-of-the-art in MaxSAT. The tables give the average performance of the single best solver for each fold (which may differ from fold to fold!) in the row indexed BS. Note this value is never worse than what the previous best single MaxSAT solver had to offer. The BS values thus provide an upper bound on the state-of-the-art in MaxSAT before this study, in terms of best single solver performance. On plain MaxSAT, ISAC++ solves 8% more instances, 58% more on partial MaxSAT, 6% more on weighted MaxSAT, and 29% more instances on weighted partial MaxSAT instances within the timeout.

Results at the international MaxSAT Evaluations

Our results were independently confirmed at the 2013 and 2014 editions of the international MaxSAT Evaluation (MSE13 and MSE14) where our portfolios, built based on the methodology described in this paper, placed in all but two subcategories while winning a total of nine.

We present and discuss the results at these competitions of ISAC+2013 (submitted to MSE13) and ISAC+2014 (submitted to MSE14). The MSE13 was run on a cluster featured with Intel Xeon CPU E7-8837 @ 2.67 GHz processors, and the MSE14 was run on a cluster features with Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00 GHz processors. A memory limit of 3.5 GB and a timeout of 1800 seconds was used for both evaluations.

The MSE13 evaluation had four categories depending on the variant of the MaxSAT problem: MaxSAT (MS), Partial MaxSAT (PMS), Weighted MaxSAT (WMS) and Weighted Partial MaxSAT (WPMS). In MSE14 there were three categories, MS, PMS, and WPMS (which also contained some WMS instances but these did not form their own category anymore). Within each category, instances were classified as either random, crafted, or industrial.

Table 11
MSE-2013 three best solvers per subcategory (ordered by num. of solved instances).

	MS	PMS	WMS	WPMS
Random	1. MaxSatz2013f 2. **ISAC+2013** 3. ckmax-small	1. **ISAC+2013** 2. WMaxSatz09 3. WMaxSatz+	1. ckmax-small 2. **ISAC+2013** 3. Maxsatz2013f	1. **ISAC+2013** 2. WMaxSatz09 3. WMaxSatz+
Crafted	1. **ISAC+2013** 2. Maxsatz2013f 3. ckmax-small	1. **ISAC+2013** 2. ILP-2013 3. scip-maxsat	1. **ISAC+2013** 2. WMaxSatz+ 3. WMaxSatz09	1. MaxHS 2. **ISAC+2013** 3. ILP-2013
Industrial	1. pmifumax 2. WPM1-2011 3. **ISAC+2013**	1. **ISAC+2013** 2. QMaxSAT2-mt 3. MSUnCore	1. – 2. – 3. –	1. WPM1-2013 2. **ISAC+2013** 3. WPM2-2013

Table 12
MSE-2013 three best solvers per subcategory (ordered by mean family ratio).

	MS	PMS	WMS	WPMS
Random	1. MaxSatz2013f 2. **ISAC+2013** 3. ckmax-small	1. **ISAC+2013** 2. WMaxSatz09 3. WMaxSatz+	1. ckmax-small 2. **ISAC+2013** 3. Maxsatz2013f	1. **ISAC+2013** 2. WMaxSatz09 3. WMaxSatz+
Crafted	1. ahmaxsat 2. **ISAC+2013** 3. Maxsatz2013f	1. **ISAC+2013** 2. QMaxSAT-m 3. QMaxSAT2-mt	1. **ISAC+2013** 2. WMaxSatz+ 3. WMaxSatz09	1. MaxHS 2. **ISAC+2013** 3. ILP-2013
Industrial	1. pmifumax 2. WPM1-2011 3. optimax	1. **ISAC+2013** 2. QMaxSAT2-mt 3. WPM2-2013	1. – 2. – 3. –	1. **ISAC+2013** 2. WPM1-2013 3. WPM2-2013

Table 13
MSE-2014 three best solvers per subcategory (ordered by num. of solved instances).

	MS	PMS	WPMS
Random	1. ahmaxsat_ls 2. ahmaxsat 3. CCLS2akms	1. ahmaxsat 2. ahmaxsat_ls 3. **ISAC+2014**	1. CCLS2akms 2. ahmaxsat_ls 3. **ISAC+2014**
Crafted	1. ahmaxsat_ls 2. ahmaxsat 3. **ISAC+2014**	1. **ISAC+2014** 2. scip-maxsat 3. ILP-2013	1. **ISAC+2014** 2. ILP-2013 3. MaxHS
Industrial	1. Open-WBO-In 2. clasp 3. Eva500a	1. **ISAC+2014** 2. Open-WBO-In 3. Eva500a	1. Eva500aW 2. **ISAC+2014** 3. MSCG

ISAC+2013 incorporated the following single solvers: akmaxsat, akmaxsat_ls, WMaxSatz09, WMaxSatz+, ILP-2013, QMaxSAT, QMaxSAT-g2, ShinMaxSat, MSUnCore, pwbo2.1, wbo1.6-cnf, WPM1-2013, WPM1-CP12 [5] and WPM2-2013.

Table 11 summarizes the results of MSE13. Here, we present the three best performing solvers in each subcategory. In the evaluation, solvers are ranked by the number of solved instances. However, within each category/type there are varying numbers of instances that stem from various instance "families" where instances cluster together. We therefore also present the ranking according to the "mean family ratio" in Table 12. We can see that ISAC+2013 did very well in this competition. Out of eleven subcategories, ISAC+2103 was the best performing solver in six subcategories, and it placed second on four.

In Tables 13 and 14, we show the results of ISAC+2014 at MSE14. ISAC+2014 includes the following single solvers: ahmaxsat, akmaxsat_ls, WMaxSatz09, WMaxSatz+, Maxsatz2013f, ckmax_small, ILP-2013, scip-maxsat, antom_seq1, antom_seq2, iraNovelty++, QMaxSAT-m, QMaxSAT2-m, QMaxSAT2-mt, ShinMaxSat, MSUnCore, pwbo2.33, pmifumax, WPM1-2011, and WPM1-2013. For this competition, in accordance to the ISAC++ methodology, we also included automatically generated parameterizations of the new WPM2-2013 solver. As we can see, ISAC+2014 excelled particularly on the PMS and WPMS categories for crafted and industrial instances which is easily explained by the fact that the solver tuned within the portfolio, WPM2-2013, is particularly well suited for crafted and industrial PMS and WPMS instances.

Detailed results of MSE14 are presented in Table 15, grouped by the three categories (MS, PMS, and WPMS) and instance types (random, crafted, and industrial). We also add cumulative results for each category as well as type of instances. Absolute numbers show the number of instances solved, the percentages give mean family ratios.

The main strength of portfolio approaches is robust performance across categories. The total numbers across multiple categories (with the only exception of random instances) and also across types show clearly that ISAC+2014 outperformed all competitors in this general setting. When neither instance category nor instance type are known, out of 2809 instances,

Table 14
MSE-2014 three best solvers per subcategory (ordered by mean family ratio).

	MS	PMS	WPMS
Random	1. ahmaxsat_ls 2. ahmaxsat 3. CCLS2akms	1. ahmaxsat_ls 2. ahmaxsat 3. **ISAC+2014**	1. CCLS2akms 2. ahmaxsat_ls 3. **ISAC+2014**
Crafted	1. ahmaxsat_ls 2. ahmaxsat 3. **ISAC+2014**	1. **ISAC+2014** 2. QMS-g3-auto 3. Open-WBO-SU	1. MaxHS 2. ILP-2013 3. scip-maxsat
Industrial	1. Open-WBO-In 2. clasp 3. Eva500a	1. **ISAC+2014** 2. Open-WBO-In 3. MSCG	1. WPM-2014-co 2. **ISAC+2014** 3. Eva500a

Table 15
ISAC+2014 at MSE-2014.

Solvers	Random		Crafted		Industrial		Total	
MS	378		177		55		610	
MSE14-VBS	304	78.0%	160	65.7%	47	92.3%	511	75.2%
ISAC+2014	301	76.9%	156	60.5%	39	84.5%	496	71.7%
ahmaxsat-ls	303	77.6%	156	60.5%	0	0.0%	459	61.5%
Open-WBO-In	0	0.0%	11	9.0%	42	87.5%	53	14.3%
PMS	210		421		568		1199	
MSE14-VBS	209	99.4%	389	86.9%	522	88.0%	1120	89.8%
ISAC+2014	193	91.9%	387	86.4%	510	86.3%	1090	87.3%
Open-WBO-In	11	5.1%	290	68.3%	473	81.1%	774	64.2%
QMS-g3-auto	3	1.4%	318	73.4%	454	78.5%	775	63.4%
ahmaxsat-ls	208	99.0%	294	48.0%	32	5.5%	534	33.2%
WPMS	280		310		410		1000	
MSE14-VBS	280	100.0%	280	88.1%	379	79.8%	939	89.0%
ISAC+2014	280	100.0%	241	66.7%	367	73.5%	888	76.4%
ILP-2013	57	22.6%	224	75.2%	249	45.9%	530	55.3%
MaxHS	5	2.0%	219	75.8%	280	52.6%	404	52.3%
CCLS2akms	280	100.0%	152	44.3%	25	8.9%	457	49.2%
WPM-2014-co	0	0.0%	151	43.4%	359	73.9%	510	40.3%
Total	868		908		1033		2809	
MSE14-VBS	793	92.2%	829	83.7%	948	86.2%	2570	86.8%
ISAC+2014	774	89.5%	784	71.5%	916	83.0%	2474	80.4%
MaxHS	24	2.8%	554	62.2%	724	66.2%	1302	48.1%
Eva500a	1	0.1%	460	47.8%	881	78.4%	1342	46.4%
ahmaxsat-ls	791	91.9%	578	42.1%	57	5.7%	1426	40.9%

ISAC+2014 solves 2474 within the time limit, a mere 96 instances less than a perfect oracle of all solvers submitted to the 2014 evaluation. (Note that ISAC+2014 did not have access to all these solvers. We will show results when ISAC++ is given access to all latest solvers next).

On top of this, the ISAC+2014 solver also improved performance within the PMS and WPMS categories on crafted and industrial instances. Out of the 421 crafted PMS instances, e.g., ISAC+2014 solves 387, just two less than a perfect oracle based on the latest 2014 solvers. The best competitor from that year solves 318 instances, over 17% less than ISAC+2014.

As stated previously, ISAC+2014 did not have access to all solvers submitted to that evaluation but had to rely, in part, on outdated solvers from earlier years. In preparation for the 2015 evaluation, we trained two additional solvers (i) ISAC+MSE14 that incorporates the best solvers of MSE14 and (ii) ISAC+MSE14* that also incorporates configurations of the new solver WPM3 [7].

In Table 16 we compare the results of ISAC+2014 (the version submitted to MSE14) and the VBS of MSE14 with these new solvers. Overall, we now reach almost the same performance as the 2014 VBS and, by finding new configurations of WPM3, ISAC+MSE14* is able to solve more industrial instances than the 2014 VBS.

We close our numerical results section by giving a detailed insight in the inner workings of the ISAC+2014 solver that was submitted to the 2014 MaxSAT Evaluation. In Table 17 we show which solver/parametrization ISAC+2014 invoked for how many MaxSAT instances within each industrial problem family. Recall that, in ISAC+2014, we parameterized the SMT-based MaxSat solver WPM2-2013. At MSE13, this program had solved the most industrial instances (844) and had the highest mean family ratio of solved instances on industrial families of 76.8%.

Table 16
Variations of ISAC+ on MSE-2014 instances.

Solvers	Random		Crafted		Industrial		Total	
Total	868		908		1033		2809	
MSE14-VBS	793	**92.2%**	829	**83.7%**	948	86.2%	2569	**86.8%**
ISAC+MSE14*	792	92.0%	820	81.6%	950	**86.3%**	2562	86.0%
ISAC+MSE14	792	92.0%	820	81.6%	935	83.4%	2547	85.0%
ISAC+2014	774	89.5%	784	71.5%	916	83.0%	2474	80.4%

Table 17
Solvers selected by ISAC+2014 on MSE14 industrial instances.

Family	#	WPM2													
		1	2	3	4	5	6	7	8	9	10	11	12	13	T
MS															
cir-dp	3	–	–	–	–	–	–	–	–	3	–	–	–	–	3
sean-s	52	–	–	–	–	–	–	–	–	34	–	–	2	–	36
Total	55			–				–	–	37	–	–	2	–	39
PMS															
aes	7	–	–	–	–	–	–	3	–	–	–	–	–	–	3
at-mes	18	–	–	–	–	–	–	–	11	–	–	–	–	–	11
at-sug	19	–	–	–	–	–	–	–	11	–	–	–	–	–	11
bcp-fir	32	1	–	–	–	–	–	9	21	–	–	1	–	–	32
bcp-hysi	10	–	–	–	–	–	–	–	9	–	–	–	–	–	9
bcp-hysu	38	–	–	–	–	–	–	–	35	–	–	–	–	–	35
bcp-msp	40	2	–	–	–	–	–	12	7	–	–	1	–	–	22
bcp-mtg	30	–	–	–	–	–	–	–	22	–	8	–	–	–	30
bcp-syn	38	–	–	–	–	–	–	36	–	–	–	–	–	–	36
cir-tc	4	–	–	–	–	–	–	–	4	–	–	–	–	–	4
clo-sol	50	1	–	–	–	–	–	–	36	12	–	–	–	–	49
des	50	–	–	–	–	–	–	–	45	–	–	–	–	–	45
hap-a	6	–	–	–	–	–	–	–	5	–	–	–	–	–	5
hs-tim	2	–	–	–	–	–	–	–	1	–	–	–	–	–	1
mbd	46	–	–	–	–	–		–	39	–	–	–	–	–	39
pac-pms	40	–	–	–	–	–	–	35	5	–	–	–	–	–	40
pbo-mqcne	25	–	10	–	–	–	–	–	13	–	2	–	–	–	25
pbo-mqcnl	25	–	9	–	–	–	–	–	11	–	5	–	–	–	25
pbo-rou	15	–	–	–	–	–	–	3	12	–	–	–	–	–	15
pro-ins	12	–	–	–	–	–	–	–	12	–	–	–	–	–	12
tpr-Mp	36	–	7	–	–	10	6	–	6	–	7	–	–	–	36
tpr-Op	25	7	–	–	3	–	15	–	–	–	–	–	–	–	25
Total	568			71				98	305	12	22	2	–	–	510
WPMS															
hap-ped	100	82	–	–	–	15	–	–	–	–	–	–	–	–	97
hs-tim	14	–	–	–	–	–	–	–	–	–	–	–	–	–	0
pac-wpms	99	–	–	–	–	–	–	99	–	–	–	–	–	–	99
pre-pla	29	3	–	–	–	25	1	–	–	–	–	–	–	–	29
tim	26	1	–	4	–	3	–	–	–	–	–	1	–	–	9
upg-pro	100	–	–	–	–	–	–	100	–	–	–	–	–	–	100
wcsp-s5d	21	–	–	–	–	–	–	17	–	–	–	–	–	–	17
wcsp-s5l	21	2	–	–	–	4	7	1	–	–	–	–	–	2	16
Total	410			147				217	–	–	–	1	–	2	367
Total Ind.	1033			218				315	305	49	22	3	2	2	916

In Table 17, columns (1)–(6) show the five WPM2 parameterizations that were generated by ISAC++. The remaining columns are for the following solvers: (7) ILP-2013, (8) QMaxSAT2-mt, (9) pmifumax, (10) QMaxSAT-m, (11) MSUnCore, (12) WPM1-2011 and (13) ShinMaxSat. Details about solvers and authors can be found in [8]. There are other solvers underlying ISAC+2014, but they are not selected for the industrial instances and consequently not listed here. The rows in the table are the families of industrial instances at the MSE14, with the column marked "#" specifying the total number of instances in the class.

Table 17 reveals the number of times a solver was used to solve instances of a particular type of industrial instance. As we can see, configurations of WPM2 are selected on 218 out of 916 solved instances, whereby some parameterizations are category specific (such as (3) which excels only on WPMS instances, or (4) which excels only on PMS instances), while others cover a greater range of problem instances. Overall, none of the trained parameterizations is clearly dominant. We also observe that, even within single instance families, ISAC+2014 selects different solvers.

7. Conclusion

We introduced an improved instance-specific algorithm configurator by adding a non-cluster based portfolio stage to the existing ISAC approach. Extensive tests showed that the new method consistently outperforms the best instance-specific configurators to date.

We applied the new method to partial MaxSAT, a domain where portfolios had never been used in a competitive setting before we conducted this work. We devised a method to extend features originally designed for SAT to characterize weighted partial MaxSAT instances. Then, we built three instance-specific partial MaxSAT solvers for crafted and industrial instances, as well as a combination of those. Results clearly showed the improvements of the new instance-specific tuner over the prior state-of-the-art in tuning. A 10-fold cross validation in the four categories of the 2012 MaxSAT evaluation confirmed these findings.

Based on this work we entered our solvers in the 2013 MaxSAT Competition, where they won six out of eleven categories and came in second in another four. We subsequently updated the portfolio with new solvers and submitted it to the 2014 MaxSAT Competition, where it placed in seven of the nine categories, winning three of them. These results independently confirm that our solvers mark a significant step forward in solving MaxSAT instances efficiently.

References

[1] B. Adenso-Diaz, M. Laguna, Fine-tuning of algorithms using fractional experimental design and local search, Oper. Res. 54 (1) (2006) 99–114.
[2] C. Ansotegui, M.L. Bonet, J. Levy, Solving (weighted) partial maxsat through satisfiability testing, in: SAT, 2009, pp. 427–440.
[3] C. Ansotegui, J. Gabas, Solving (weighted) partial maxsat with ilp, in: CPAIOR, 2013, pp. 403–409.
[4] C. Ansotegui, M. Sellmann, K. Tierney, A gender-based genetic algorithm for the automatic configuration of algorithms, in: CP, 2009, pp. 142–157.
[5] Carlos Ansótegui, Maria Luisa Bonet, Joel Gabàs, Jordi Levy, Improving sat-based weighted maxsat solvers, in: CP, 2012, pp. 86–101.
[6] Carlos Ansótegui, Maria Luisa Bonet, Jordi Levy, Sat-based maxsat algorithms, Artif. Intell. 196 (2013) 77–105.
[7] Carlos Ansótegui, Frédéric Didier, Joel Gabàs, Exploiting the structure of unsatisfiable cores in maxsat, in: IJCAI, 2015, pp. 283–289.
[8] J. Argelich, C.M. Li, F. Manyà, J. Planes, Max-sat evaluation, 2006–2014.
[9] J. Argelich, F. Manyà, Partial Max-SAT solvers with clause learning, in: SAT, 2007, pp. 28–40.
[10] Josep Argelich, Felip Manyà, Exact max-sat solvers for over-constrained problems, J. Heuristics 12 (4–5) (2006) 375–392.
[11] Gilles Audemard, Laurent Simon, Predicting learnt clauses quality in modern SAT solvers, in: IJCAI, 2009, pp. 399–404.
[12] C. Audet, D. Orban, Finding optimal algorithmic parameters using derivative-free optimization, SIAM J. Optim. 17 (3) (2006) 642–664.
[13] Adrian Balint, Anton Belov, Daniel Diepold, Simon Gerber, Matti Järvisalo, Carsten Sinz (Eds.), Proceedings of SAT Challenge 2012: Solver and Benchmark Descriptions, in: Department of Computer Science Series of Publications B, vol. B-2012-2, University of Helsinki, 2012.
[14] Daniel Le Berre, Anne Parrain, The sat4j library, release 2.2, JSAT 7 (2–3) (2010) 59–64.
[15] Marco Collautti, Yuri Malitsky, Deepak Mehta, Barry O'Sullivan, Snnap: solver-based nearest neighbor for algorithm portfolios, in: ECML, 2013, pp. 435–450.
[16] SAT Competition, www.satcompetition.org, 2012.
[17] S.P. Coy, B.L. Golden, G.C. Runger, E.A. Wasil, Using experimental design to find effective parameter settings for heuristics, J. Heuristics 7 (2001) 77–97.
[18] S. Darras, G. Dequen, L. Devendeville, C.M. Li, On inconsistent clause-subsets for Max-SAT solving, in: CP, 2007, pp. 225–240.
[19] Z. Fu, S. Malik, On solving the partial max-sat problem, in: SAT, 2006, pp. 252–265.
[20] G. Hamerly, C. Elkan, Learning the k in k-means, in: NIPS, 2003, pp. 281–288.
[21] F. Heras, J. Larrosa, A. Oliveras, Minimaxsat: a new weighted max-sat solver, in: SAT, 2007, pp. 41–55.
[22] K. Honjyo, T. Tanjo, Shinmaxsat, 2012.
[23] Holger H. Hoos, An adaptive noise mechanism for walksat, in: AAAI, 2002, pp. 655–660.
[24] F. Hutter, H.H. Hoos, K. Leyton-Brown, T. Stuetzle, Paramils: an automatic algorithm configuration framework, J. Artif. Intell. Res. 36 (2009) 267–306.
[25] Frank Hutter, Holger Hoos, Kevin Leyton-Brown, Sequential model-based optimization for general algorithm configuration, in: LION, 2011, pp. 507–523.
[26] Frank Hutter, Dave A.D. Tompkins, Holger H. Hoos, Scaling and probabilistic smoothing: efficient dynamic local search for SAT, in: CP, 2002, pp. 233–248.
[27] S. Kadioglu, Y. Malitsky, A. Sabharwal, H. Samulowitz, M. Sellmann, Algorithm selection and scheduling, in: CP, 2011, pp. 454–469.
[28] S. Kadioglu, Y. Malitsky, M. Sellmann, K. Tierney, ISAC – instance-specific algorithm configuration, in: ECAI, 2010, pp. 751–756.
[29] A.R. KhudaBukhsh, L. Xu, H.H. Hoos, K. Leyton-Brown, Satenstein: automatically building local search sat solvers from components, in: IJCAI, 2009.
[30] M. Koshimura, T. Zhang, H. Fujita, R. Hasegawa, Qmaxsat: a partial max-sat solver, JSAT 8 (1/2) (2012) 95–100.
[31] Christian Kroer, Yuri Malitsky, Feature filtering for instance-specific algorithm configuration, in: ICTAI, 2011, pp. 849–855.
[32] Adrian Kuegel, Improved exact solver for the weighted max-sat problem, POS-10 8 (2012) 15–27.
[33] Chu Min Li, Felip Manyà, Jordi Planes, New inference rules for Max-SAT, J. Artif. Intell. Res. 30 (2007).
[34] C.M. Li, W.Q. Huang, G2wsat: gradient-based greedy walksat, in: SAT, vol. 3569, 2005, pp. 158–172.
[35] H. Lin, K. Su, C.M. Li, Within-problem learning for efficient lower bound computation in Max-SAT solving, in: AAAI, 2008, pp. 351–356.
[36] Y. Malitsky, M. Sellmann, Instance-specific algorithm configuration as a method for non-model-based portfolio generation, in: CPAIOR, 2012, pp. 244–259.
[37] Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, Meinolf Sellmann, Algorithm portfolios based on cost-sensitive hierarchical clustering, in: IJCAI, 2013, pp. 608–614.
[38] V. Manquinho, J. Marques-Silva, J. Planes, Algorithms for weighted boolean optimization, in: SAT, 2009, pp. 495–508.
[39] J. Marques-Silva, J. Planes, Algorithms for maximum satisfiability using unsatisfiable cores, in: DATE, 2008, pp. 408–413.
[40] P.J. Matos, J. Planes, F. Letombe, J. Marques-Silva, A max-sat algorithm portfolio, in: ECAI, 2008, pp. 911–912.
[41] António Morgado, Federico Heras, Mark H. Liffiton, Jordi Planes, Joao Marques-Silva, Iterative and core-guided maxsat solving: a survey and assessment, Constraints 18 (4) (2013) 478–534.
[42] M. Nikolic, F. Maric, P. Janici, Instance based selection of policies for sat solvers, in: SAT, 2009, pp. 326–340.
[43] E. O'Mahony, E. Hebrard, A. Holland, C. Nugent, B. O'Sullivan, Using case-based reasoning in an algorithm portfolio for constraint solving, in: AICS, 2008, pp. 210–216.

[44] D.N. Pham, Anbulagan, ranov. Solver description, SAT Competition, 2007.
[45] D.N. Pham, C. Gretton, gnovelty+. Solver description, SAT Competition, 2007.
[46] Steven David Prestwich, Random walk with continuously smoothed variable weights, in: SAT, 2005, pp. 203–215.
[47] L. Pulina, A. Tacchella, A multi-engine solver for quantified boolean formulas, in: CP, 2007, pp. 574–589.
[48] Bryan Silverthorn, Risto Miikkulainen, Latent class models for algorithm portfolio methods, in: AAAI, 2010, pp. 167–172.
[49] J. Thornton, D.N. Pham, S. Bain, V. Ferreira, Additive versus multiplicative clause weighting for sat, in: PRICAI, 2008, pp. 405–416.
[50] D.A.D. Tompkins, F. Hutter, H.H. Hoos, saps. Solver description, SAT Competition, 2007.
[51] W. Wei, C.M. Li, H. Zhang, adaptg2wsatp. Solver description, SAT Competition, 2007.
[52] W. Wei, C.M. Li, H. Zhang, Combining adaptive noise and promising decreasing variables in local search for sat. Solver description, SAT Competition, 2007.
[53] W. Wei, C.M. Li, H. Zhang, Deterministic and random selection of variables in local search for sat. Solver description, SAT Competition, 2007.
[54] L. Xu, H.H. Hoos, K. Leyton-Brown, Hydra: automatically configuring algorithms for portfolio-based selection, in: AAAI, 2010, pp. 210–216.
[55] L. Xu, F. Hutter, H.H. Hoos, K. Leyton-Brown, Satzilla: portfolio-based algorithm selection for sat, J. Artif. Intell. Res. 32 (1) (2008) 565–606.
[56] L. Xu, F. Hutter, J. Shen, H.H. Hoos, K. Leyton-Brown, Satzilla2012: improved algorithm selection based on cost-sensitive classification models, SAT Competition, 2012.

G

WPM3: an (in)complete algorithm for Weighted Partial MaxSAT

C. Ansótegui, J. Gabàs

submitted to Artificial Intelligence

2016

WPM3: an (in)complete algorithm for Weighted Partial MaxSAT

Carlos Ansótegui, Joel Gabàs

DIEI, University of Lleida, Spain

Abstract

Maximum Satisfiability (MaxSAT) has been used to solve efficiently many combinatorial optimization problems. At recent editions of international MaxSAT Evaluation (MSE), the best performing solvers for real world (industrial) problems were those implementing SAT-based algorithms. These algorithms reformulate the MaxSAT optimization problem into a sequence of SAT decision problems where Pseudo-Boolean (PB) constraints may be introduced. In order to identify the most suitable PB constraints, some algorithms (core-guided) analyze the unsatisfiable cores retrieved from the previous SAT problems in the sequence while refining the lower bound. In this paper, we first conduct a comprehensive study on the complete core-guided algorithms Eva and OLL, that inspired the best performing solvers on industrial instances at MSE-2014. Despite of its apparently different foundations, we show how they are intimately related and identify how to improve them. In this sense, we present our complete core-guided algorithm WPM3. We show how to further exploit the analysis of unsatisfiable cores by being aware of their global structure, i.e., how the cores are interrelated. This is used to encode more efficient PB constraints and enables the algorithm to obtain assignments and refine also the upper bound. Therefore, WPM3 can also work as an incomplete algorithm. At MSE-2015, it showed a competitive performance on industrial instances. It got one out of three gold medals at the complete track and dominated at the incomplete track.

Email addresses: `carlos@diei.udl.es` (Carlos Ansótegui), `joel.gabas@udl.cat` (Joel Gabàs)

Research partially supported by the Ministerio de Economía y Competividad research project TASSAT2: TIN2013-48031-C4-4-P and Google Faculty Research Award program.

Keywords:
optimization, MaxSAT, SAT, Pseudo-Boolean constraint

Maximum Satisfiability (MaxSAT) has been used to solve efficiently many combinatorial optimization problems that appear in real world (industrial) domains Among these domains, we can find software package upgrade [1], debugging of hardware designs [2, 3], bioinformatics [4, 5], fault localization in C code [6], course timetabling [7], planing [8, 9], scheduling [10], routing [11], electronic markets [12], combinatorial auctions [13] and many others [14, 15].

MaxSAT is the natural optimization variant of Satisfiability (SAT). The main idea is that sometimes not all restrictions of a problem can be satisfied, and we try to satisfy the maximum number of them. The basic MaxSAT problem can be further generalized to the Weighted Partial MaxSAT (WPMS) problem. In this case, we can divide the restrictions into two groups: the clauses that must be satisfied (hard), and the ones that may or may not be satisfied (soft). We may put different weights to the soft clauses, where the weight is the cost of falsifying the clause. The presence of soft clauses with different weights makes a MaxSAT instance Weighted and the presence of hard clauses makes it Partial. To solve the MaxSAT problem we have to find an assignment that satisfies all hard clauses and minimizes the aggregated cost of falsified soft clauses.

Solving exactly (completely) the MaxSAT problem, i.e. finding and certifying an optimal assignment, can be NP-hard from a computational point of view. Anyway, there are many industrial problems slightly beyond the reach of state-of-the-art techniques and often the goal is not finding an optimal assignment but an assignment of a good quality in a reasonable time. For some domains, even a small gain in the quality can lead to important practical consequences. To this end, many incomplete algorithms have been developed for industrial problems. However, the experience achieved from the international MaxSAT Evaluation (MSE) [16], shows us that a reasonable strategy is to improve complete algorithms and modify them so that they become also incomplete (i.e., return assignments when improved) [17, 18]. This is also the case of the WPM3 algorithm that we present in this paper.

There are two main types of complete algorithms to solve the MaxSAT problem: (i) branch and bound [19, 20, 21, 22, 23] and (ii) SAT-based [14, 15]. At recent editions of MSE, we have seen that solvers implementing SAT-based algorithms clearly dominate on industrial instances. These algorithms proceed by reformulating the MaxSAT optimization problem into a sequence of

SAT decision problems. Each SAT instance of the sequence encodes whether it exists an assignment to the MaxSAT instance with a cost less than or equal to a certain k. Those SAT instances with a k less than the optimal cost are unsatisfiable, and the others satisfiable. Therefore, when SAT-based algorithms find the phase transition point, they find the optimum.

Among SAT-based MaxSAT algorithms, there are also two main types: (i) core-guided [24, 25, 26, 27, 28, 29, 30] and (ii) model-guided [31, 32, 33, 34]. The first ones refine (increase) the lower bound and guide the search with unsatisfiable subproblems (cores) obtained from unsatisfiable SAT instances. The second ones refine (decrease) the upper bound and guide the search with satisfying assignments (models) obtained from satisfiable SAT instances. Both have strengths and weaknesses and some hybrid approaches have been proposed [35, 36, 17, 18].

SAT-based MaxSAT algorithms use Pseudo-Boolean (PB) constraints to create the SAT instances in the sequence. The PB constraints are used to express the arithmetic and comparison needed to only allow satisfying assignments with a cost less than or equal to the k of the instance. The size, the management and the complexity of these PB constraints are crucial for SAT-based algorithms. With respect to the size, the naive approach uses a unique PB constraint that involves all soft clauses. Fortunately, SAT-based algorithms can use a set of smaller PB constraints that do not necessarily cover all soft clauses. With respect to the management, the default option is that PB constraints are translated to SAT since SAT-based algorithms use internally a SAT solver. However, there are other options, like modeling PB constraints with the Linear Integer Arithmetic theory and using internally an SMT solver [18]. With respect to the complexity, depending on the particular SAT-based algorithm, we may only need to use a simpler form of PB constraints with all coefficients equal to 1 i.e., Cardinality constraints. In case PB constraints are managed through their translation to SAT, the best encoding that preserve arc-consistency has a quadratic size with respect to the size of the constraint [37]. For Cardinality constraints, the size of the best SAT encoding is quasilinear with respect to the size of the constraint [38].

PB constraints are defined on auxiliary variables which express whether a particular soft clause is falsified or not. However, some SAT-based algorithms may require more than one auxiliary variable per clause introducing additional complexity. For WPMS instances in particular, the WPM1/WBO algorithm presented in [24, 25] only needed 1-Cardinality constraints (with independent term equal to 1) but it needed multiple auxiliary variables per clause. The WPM2 algorithm presented in [26] needed only a unique auxiliary variable for each clause but it needed general PB constraints.

To our best knowledge, the OLL algorithm, presented in [39] originally for ASP (Answer Set Programming), was the first one that only needed a unique auxiliary variable per clause while using Cardinality constraints. Later, this algorithm was applied to MaxSAT and shown to be competitive in [40]. Previous to this last work, the Eva algorithm was presented and also shown to be competitive in [41]. It applies the MaxSAT resolution rule to create the next instance of the sequence. The solvers inspired by Eva and OLL were the best on industrial instances at MSE-2014. Although both algorithms were core-guided, they had apparently different foundations.

In this paper, we demonstrate that the transformation applied by Eva at each iteration using the MaxSAT resolution rule corresponds to the incremental construction of a Cardinality constraint. In particular, it finally creates the regular encoding for 1-Cardinality constraints described in [42]. We have analyzed in detail the connections between Eva and OLL and we present a first study showing that they are in fact very similar.

With the aim of improving the performance in MaxSAT, we have made a step forward and developed a new algorithm that takes profit of our study. The WPM3 algorithm only needs Cardinality constraints like Eva and OLL, but additionally, it is aware of the global structure of the unsatisfiable cores identified in the problem. Thus, it can exploit this structure to make encodings for the Cardinality constraints that improve the efficiency of the solvers. Moreover, WPM3 is able to exploit subproblem optimization [18] efficiently, which as subproduct provides assignments that can be used to refine the upper bound for the whole problem. In this way, WPM3 can also work as an incomplete algorithm, given limited time and memory. The assignments can be further exploited to extend to MaxSAT a very effective technique used in SAT solvers called phase saving [43].

Finally, we have conducted an extensive experimental investigation on industrial instances showing the impact of every technique. With respect to Cardinality constraints, we show that the improvement on the performance is mostly due to the exploitation of the core structure and not only to incrementality as suggested in previous works [30]. We also analyze the results of the MSE-2015. Of the six industrial subcategories, including complete and incomplete track, WPM3 got a total of five medals. At the complete track, it got a gold (PMS) and a bronze (MS) medals. At the incomplete track, it dominated with two gold (PMS, WPMS) and a silver (MS) medals.

This paper proceeds as follows. Section 1 introduces some preliminary concepts. Section 2 analyzes how Eva and OLL algorithms are related. Section 3 presents the WPM3 complete MaxSAT algorithm. Section 4 discusses how to encode the Cardinality constraints generated by the algorithm. Section 5 compares WPM3 with Eva and OLL. Section 6 explains how to exploit the assignments and upper bounds generated by the algorithm. Section 7 shows the experimental evaluation. Finally, Section 8 concludes.

1. Preliminaries

Definition 1. *A* literal l *is either a Boolean variable* x *or its negation* $\overline{x}$. *A* clause c *is a disjunction of literals. An* empty clause $\Box$ *is a clause without literals. A* SAT formula *is a set or conjunction of clauses, i.e. a Boolean formula in Conjunctive Normal Form (CNF). We note the conversion of a Boolean formula* β *that is not in CNF into a SAT formula* ϕ *as* $\phi = \mathit{CNF}(\beta)$. *We note the union of two SAT formulas* ϕ_1 *and* ϕ_2 *as* $\phi_1 \cup \phi_2$, *e.g.:*

$$\{x_1 \vee x_2\} \cup \{\overline{x}_1 \vee \overline{x}_2\} = \{x_1 \vee x_2, \overline{x}_1 \vee \overline{x}_2\}$$

Definition 2. *A* weighted clause *is an ordered pair* $\langle c, w\rangle$, *where* c *is a clause and* w *is a natural number or infinity (indicating the cost of falsifying* c, *see Definitions 4 and 8). If* w *is infinite the clause is* hard, *otherwise it is* soft.

Definition 3. *A* Weighted Partial MaxSAT (WPMS) formula *is an ordered multiset of weighted clauses:*

$$\varphi = \langle\langle c_1, w_1\rangle, \ldots, \langle c_s, w_s\rangle, \langle c_{s+1}, \infty\rangle, \ldots, \langle c_{s+h}, \infty\rangle\rangle$$

The presence of soft clauses with different weights makes the formula Weighted and the presence of hard clauses makes it Partial. The ordered multiset of weights of the soft clauses in the formula is noted as $w(\varphi)$. *The hard clauses are occasionally noted as* $\langle \phi_H, \infty\rangle, \phi_H = \{c_{s+1}, \ldots, c_{s+h}\}$, *when their order can be ignored. The set of variables occurring in the formula is noted as* $\mathit{var}(\varphi)$. *We note the concatenation of two WPMS formulas* φ_1 *and* φ_2 *as* $\varphi_1 + \varphi_2$ *e.g.:*

$$\langle\langle c_1, w_1\rangle\rangle + \langle\langle c_2, w_2\rangle\rangle = \langle\langle c_1, w_1\rangle, \langle c_1, w_1\rangle\rangle$$

Definition 4. *An* assignment *for a set of Boolean variables X is a function $\mathcal{I} : X \to \{0,1\}$, that can be extended to literals, (weighted) clauses, SAT formulas and WPMS formulas as follows:*

$$\begin{array}{l}
\mathcal{I}(\overline{x}) = 1 - \mathcal{I}(x) \\
\mathcal{I}(l_1 \vee \ldots \vee l_m) = \max\{\mathcal{I}(l_1), \ldots, \mathcal{I}(l_m)\} \\
\mathcal{I}(\{c_1, \ldots, c_n\}) = \min\{\mathcal{I}(c_1), \ldots, \mathcal{I}(c_n)\} \\
\mathcal{I}(\langle c, w\rangle) = w\,(1 - \mathcal{I}(c)) \\
\mathcal{I}(\langle \Box, w\rangle) = w \\
\mathcal{I}(\langle\langle c_1, w_1\rangle, \ldots, \langle c_{s+h}, w_{s+h}\rangle\rangle) = \sum_{i=1}^{s+h} \mathcal{I}(\langle c_i, w_i\rangle)
\end{array}$$

We will refer to the value returned by an assignment $\mathcal{I}$ on a weighted clause or a WPMS formula as the cost of $\mathcal{I}$.

Definition 5. *Given the WPMS formulas φ and φ', we say that φ is MaxSAT reducible to φ' if, for any assignment $\mathcal{I} : var(\varphi) \to \{0,1\}$, we have that $\mathcal{I}(\varphi) = \min\{\mathcal{I}'(\varphi') \mid \mathcal{I}'(x) = \mathcal{I}(x)\ \forall x \in var(\varphi)\}$.*

Definition 6. *Given a weighted clause $\langle c, w\rangle$ and an integer $w' \leq w$, the* split rule *replaces $\langle c, w\rangle$ by two weighted clauses as follows:*

$$\frac{\langle c, w\rangle}{\begin{array}{c}\langle c, w'\rangle \\ \langle c, w - w'\rangle\end{array}}$$

It is trivial to see MaxSat reducibility is guaranteed.

Definition 7. *The* MaxSAT resolution rule *[44] replaces two premises, $\langle x \vee A, 1\rangle$ and $\langle \overline{x} \vee B, 1\rangle$, by a set of three conclusions, as follows:*

$$\frac{\begin{array}{c}\langle x \vee A, 1\rangle \\ \langle \overline{x} \vee B, 1\rangle\end{array}}{\begin{array}{c}\langle A \vee B, 1\rangle \\ \langle x \vee A \vee \overline{B}, 1\rangle \\ \langle \overline{x} \vee \overline{A} \vee B, 1\rangle\end{array}}$$

Where A and B are disjunctions of literals. The first conclusion, $\langle A \vee B, 1\rangle$, is the resolvant and the others the compensation clauses. The set of premises is MaxSAT reducible to the set of conclusions.

It is easy to see that we can extend the MaxSAT resolution rule to the weighted case as follows:

$$\begin{array}{c}
\langle x \vee A, w_1 \rangle \\
\langle \overline{x} \vee B, w_2 \rangle \\
\hline
\langle A \vee B, w_{min} \rangle \\
\langle x \vee A \vee \overline{B}, w_{min} \rangle \\
\langle \overline{x} \vee \overline{A} \vee B, w_{min} \rangle \\
\langle x \vee A, w_1 - w_{min} \rangle \\
\langle \overline{x} \vee B, w_2 - w_{min} \rangle
\end{array}$$

Where A and B are disjunctions of literals and $w_{min} = \min(w_1, w_2)$.

Notice that we can apply the split rule on each of the premises till we get w_{min} copies of $\langle x \vee A, 1 \rangle$ and $\langle x \vee B, 1 \rangle$ as well as the last two conclusions $\langle x \vee A, w_1 - w_{min} \rangle$ and $\langle x \vee B, w_2 - w_{min} \rangle$. Then, we can apply the MaxSAT resolution rule on the w_{min} pairs of premises $\langle x \vee A, 1 \rangle$ and $\langle x \vee B, 1 \rangle$. This will generate w_{min} copies of $\langle A \vee B, 1 \rangle$, $\langle x \vee A \vee \overline{B}, 1 \rangle$ and $\langle \overline{x} \vee \overline{A} \vee B, 1 \rangle$, which can be collapsed into the first three conclusions by applying the inversion of the split rule.

The second (third) conclusion is not in CNF if B (A) contains more that one literal. In [45], the version in CNF is provided. If we allow hard clauses in the conclusions, there is a simple trick by reifying $\overline{B}$ ($\overline{A}$) into a new fresh variable b. For example, $\langle x \vee A \vee \overline{B}, w_{min} \rangle$ can be replaced by $\langle x \vee A \vee b, w_{min} \rangle$ and $\langle CNF(b \leftrightarrow \overline{B}), \infty \rangle$.

These are special cases of the MaxSAT resolution rule:

$$\begin{array}{c}
\langle x \vee A, \infty \rangle \\
\langle \overline{x}, 1 \rangle \\
\hline
\langle A, 1 \rangle \\
\langle x \vee A, \infty \rangle \\
\langle \overline{x} \vee \overline{A}, 1 \rangle \rangle
\end{array}
\qquad
\begin{array}{c}
\langle x \vee A, 1 \rangle \\
\langle \overline{x}, 1 \rangle \\
\hline
\langle \langle A, 1 \rangle \\
\langle \overline{x} \vee \overline{A}, 1 \rangle
\end{array}
\qquad
\begin{array}{c}
\langle x, 1 \rangle \\
\langle \overline{x}, 1 \rangle \\
\hline
\langle \langle \square, 1 \rangle
\end{array}$$

Definition 8. *We say that an assignment $\mathcal{I}$ satisfies a clause or a SAT formula if the value returned by $\mathcal{I}$ is equal to 1. In the case of SAT formulas, we will refer also to this assignment as a satisfying assignment or solution. Otherwise, if the value returned by $\mathcal{I}$ is equal to 0, we say that $\mathcal{I}$ falsifies the clause or the SAT formula.*

Definition 9. *The* SAT problem *for a SAT formula ϕ is the problem of finding a solution for ϕ. If a solution exists the formula is satisfiable, otherwise it is unsatisfiable.*

Definition 10. *Given an unsatisfiable SAT formula ϕ, an* unsatisfiable core *ϕ_C is a subset of clauses $\phi_C \subseteq \phi$ that is also unsatisfiable.*

Definition 11. *A SAT algorithm for the SAT problem, takes as input a SAT formula ϕ and returns an assignment $\mathcal{I}$ such that $\mathcal{I}(\phi) = 1$ if the formula is satisfiable. Otherwise, it returns an unsatisfiable core ϕ_C.*

Given unlimited resources of time and memory, we say that a SAT algorithm is complete *if it terminates for any SAT formula. Otherwise, it is* incomplete*.*

Definition 12. *The* optimal cost (or optimum) *of a WPMS formula φ is* $\mathrm{cost}(\varphi) = \min\{\mathcal{I}(\varphi) \mid \mathcal{I} : var(\varphi) \to \{0,1\}\}$ *and an* optimal assignment *is an assignment $\mathcal{I}$ such that $\mathcal{I}(\varphi) = \mathrm{cost}(\varphi)$. We will refer to this assignment as a* solution *for φ if $\mathcal{I}(\varphi) \neq \infty$. Any cost above (below) cost(φ) is called an upper (lower) bound for φ.*

Example 1. *Given the WPMS formula $\varphi = \langle\langle x_1, 5\rangle, \langle x_2, 3\rangle, \langle x_3, 3\rangle, \langle \overline{x}_1 \vee \overline{x}_2, \infty\rangle, \langle \overline{x}_1 \vee \overline{x}_3, \infty\rangle, \langle \overline{x}_2 \vee \overline{x}_3, \infty\rangle\rangle$, we have that* $\mathrm{cost}(\varphi) = \min\{6, 8, 11, \infty\} = 6$ *and the solution $\mathcal{I}$ maps $\langle x_1, x_2, x_3\rangle$ to $\langle 1, 0, 0\rangle$.*

Definition 13. *The* Weighted Partial MaxSAT problem *for a WPMS formula φ is the problem of finding a solution for φ. If a solution does not exist the formula is unsatisfiable.*

Definition 14. *A WPMS algorithm for the WPMS problem, takes as input a WPMS formula φ and returns an assignment $\mathcal{I}$, such that, $\mathcal{I}(\varphi) \geq cost(\varphi)$.*

Given unlimited resources of time and memory, we say that a WPMS algorithm is complete or exact *if for any input WPMS formula φ and returned $\mathcal{I}$, $\mathcal{I}(\varphi) = cost(\varphi)$. Otherwise, we say it is* incomplete.

Definition 15. *Given a WPMS formula φ such that cost(φ) ≥ 0, an* unsatisfiable core *φ_C is a subset of clauses $\varphi_C \subseteq \varphi$ such that cost(φ_C) ≥ 0.*

Definition 16. *An integer linear* Pseudo-Boolean (PB) constraint *is an inequality of the form $w_1x_1 + \cdots + w_nx_n$ op k, where op $\in \{\leq, \geq, =, >, <\}$, the independent term k and the coefficients w_i are integers, and the variables x_i are Boolean. A* Cardinality constraint *is a PB constraint where the coefficients w_i are equal to* 1*. A* 1-Cardinality constraint *is a Cardinality constraint where the independent term k is equal to* 1*. A PB* at-most constraint *is a PB constraint where op is $\leq$.*

2. Comparison of Eva and OLL algorithms

In this section, we compare Eva [41] and OLL [39] MaxSAT algorithms, that inspired the best performing solvers on industrial instances at MSE-2014, *eva*500*a* and *mscg* [40] respectively. To simplify the analysis, we will only consider the non-weighted case. The analysis holds true for the weighted case since both algorithms adapt it to the non-weighted one by applying the idea described in the WPM1/WBO MaxSAT algorithms [24, 25] (i.e. by using the split rule for weighted clauses described in Definition 6). Let us recall that Eva and OLL are core-guided algorithms that iteratively solve a sequence of SAT instances until the first satisfiable instance is found. Indeed they work on a sequence of φ^k MaxSAT instances such that $cost(\varphi^k) = cost(\varphi) - k$. If $cost(\varphi^k) = 0$, they stop and k is the optimum. To test this, they remove the weights from φ^k and check if the corresponding SAT instance ϕ^k is satisfiable.

The key point is how to transform φ^k into φ^{k+1}. After analyzing both algorithms, we have observed that the underlying idea comes from the following observation: a MaxSAT formula φ_n with n soft clauses and $cost(\varphi_n) \geq e$, $1 \leq e \leq n$ is MaxSAT reducible (Definition 5) to a formula $\varphi'_n + \langle \Box, e \rangle$ with $n - e$ soft clauses and e empty clauses:

$$\varphi_n = \langle \langle c_1, 1 \rangle, \ldots, \langle c_n, 1 \rangle, \langle \phi_H, \infty \rangle \rangle$$

$$\varphi'_n = \langle \langle c'_1, 1 \rangle, \ldots, \langle c'_{n-e}, 1 \rangle, \langle \phi'_H, \infty \rangle, \langle \phi_H, \infty \rangle \rangle$$

Where $cost(\varphi'_n) = cost(\varphi_n) - cost(\langle \Box, e \rangle)$. The SAT formula $\{c'_1, \ldots, c'_{n-k}\} \cup \phi'_H \cup \phi_H$ only accepts solutions such that $\overline{c}_1 + \ldots + \overline{c}_n = e$. The at-least constraint $\overline{c}_1 + \ldots + \overline{c}_n \geq e$ is implicitly encoded in the formula since $cost(\varphi_n) \geq e$, and the at-most constraint $\overline{c}_1 + \ldots + \overline{c}_n \leq e$ is encoded in $\{c'_1, \ldots, c'_{n-k}\} \cup \phi'_H$. This way, if the at-most-constraint is too restrictive (i.e., the lower bound has to be further refined), the soft clauses involved will appear in an unsatisfiable core, and can be relaxed again.

Back to how to transform φ^k into φ^{k+1}, Eva and OLL identify a $\varphi_n \subseteq \varphi^k$ subformula and apply the previous reduction. Then, $\varphi^{k+1} = (\varphi^k / \varphi_n) \cup \varphi'_n$. Both algorithms use as φ_n an unsatisfiable core of φ^k. This guarantees that $cost(\varphi_n) \geq e \geq 1$ and hopefully n is less than the number of soft clauses in φ^k. We may want n to be as small as possible, this can be achieved by minimizing the unsatisfiable core, and e as large as possible, this can be achieved by solving to optimality φ_n. However, Eva and OLL only apply the transformations with $e = 1$ at each iteration, i.e., at each iteration they introduce a new 1-Cardinality constraint. It is interesting to see that the incremental addition of 1-Cardinality constraints end up producing k-Cardinality constraints. Therefore, if we could identify a φ_n whose optimal cost is greater than 1, we could directly write the $cost(\varphi_n)$-Cardinality constraint and skip all the intermediate iterations performed by Eva and OLL.

In Section 3, we present the WPM3 algorithm, that identifies φ_n with e greater than 1 and solve them to optimality. In Section 5, we provide an example of how the three algorithms build a 3-Cardinality constraint. In this section, we focus our attention on the transformation applied by Eva and OLL at each iteration. In the following, we will initially analyze the transformation applied by Eva and then explain how to adapt it such that it becomes very similar to OLL.

In the case of Eva, the generation of φ^{k+1} is performed by applying the MaxSAT resolution rule (Definition 7) to φ^k. The idea is to apply successively this rule to an unsatisfiable core φ_n obtained from φ^k and MaxSAT reduce it to $\varphi'_n + \langle\Box, 1\rangle$. Let us see in detail how φ'_n is generated. Previous to applying the MaxSAT resolution rule, each soft clause of φ_n is reified with a fresh auxiliary variable b_i. Notice that $\langle\langle c_i, 1\rangle\rangle$ is MaxSAT reducible to $\langle\langle \overline{b}_i, 1\rangle, \langle \overline{b}_i \leftrightarrow c_i, \infty\rangle\rangle$. The n hard clauses $\langle \overline{b}_i \leftrightarrow c_i, \infty\rangle$ are directly introduced into φ'_n as a part of $\langle \phi'_H, \infty\rangle$. The $n-1$ soft clauses $\langle\langle c'_1, 1\rangle, \ldots, \langle c'_{n-1}, 1\rangle\rangle$ are obtained by applying the MaxSAT resolution rule on the φ_b formula which is composed of the n soft clauses $\langle \overline{b}_i, 1\rangle$ and the hard clause $\langle b_1 \vee \ldots \vee b_n, \infty\rangle$ (the core implies that at least one of the soft clauses c_i (variables b_i) must be false (true)).

$$\varphi_b = \langle\langle \overline{b}_1, 1\rangle, \ldots, \langle \overline{b}_n, 1\rangle, \langle b_1 \vee \ldots \vee b_n, \infty\rangle\rangle$$

In the first step, the MaxSAT resolution rule replaces the first soft clause and the hard clause by the resolvant clause $\langle b_2 \vee \ldots \vee b_n, 1\rangle$ and the compensation clauses $\langle b_1 \vee \ldots \vee b_n, \infty\rangle$ and $\langle \overline{b}_1 \vee \overline{b_2 \vee \ldots \vee b_n}, 1\rangle$. Then, it is applied successively to each one of the remaining original soft clauses and the resolvant clause obtained in the previous step, resulting:

$$\begin{array}{l} 1: \ \overline{b}_1 \quad \vee \overline{b_2 \vee \ldots \vee b_n} \\ \quad \ldots \\ 1: \ \overline{b}_{n-2} \vee \overline{b_{n-1} \vee b_n} \\ 1: \ \overline{b}_{n-1} \vee \overline{b}_n \\ 1: \ \Box \\ \infty : b_1 \vee \ldots \vee b_n \end{array}$$

The above forumla is not in CNF. The naive translation into *CNF*, by reifying the negation of the disjunctions into new fresh variables (see Definition 7) , generates a quadratic number of clauses with respect to n. To make it lineal, while preserving the MaxSAT reducibility, the new fresh variables are reused recursively in the next reification as follows:

$$\begin{array}{lll} 1: & \overline{b}_1 & \vee \ \overline{r}_2 \\ & \ldots & \\ 1: & \overline{b}_{n-2} & \vee \ \overline{r}_{n-1} \\ 1: & \overline{b}_{n-1} & \vee \ \overline{r}_n \\ 1: & \Box & \\ \infty : & r_2 & \leftrightarrow (b_2 \vee r_3) \\ & \ldots & \\ \infty : & r_{n-1} & \leftrightarrow (b_{n-1} \vee r_n) \\ \infty : & r_n & \leftrightarrow b_n \\ \infty : & b_1 \vee \ldots \vee b_n & \end{array}$$

At the end, the working MaxSAT formula φ^k is transformed into:

$$\begin{aligned} \varphi^{k+1} &= (\varphi^k / \varphi_n) + \langle \langle \overline{b}_1 \vee \overline{r}_2, 1 \rangle, \ldots, \langle \overline{b}_{n-1} \vee \overline{r}_n, 1 \rangle, \langle \phi'_H, \infty \rangle, \langle \phi_H, \infty \rangle \rangle \\ \phi'_H &= \mathit{CNF}(\{ \overline{b}_1 \leftrightarrow c_1, \ldots, \overline{b}_n \leftrightarrow c_n \}) \cup \\ & \quad \mathit{CNF}(\{ r_2 \leftrightarrow (b_2 \vee r_3), \ldots, r_{n-1} \leftrightarrow (b_{n-1} \vee r_n), r_n \leftrightarrow b_n \}) \end{aligned}$$

As a curiosity, we have observed that the transformation made to φ_b using the MaxSAT resolution rule corresponds to the regular encoding in [42]1-Cardinality constraint as we can see in Example 2.

Example 2. *Applying successively the MaxSAT resolution rule like Eva to* $\varphi = \langle\langle \bar{b}_1, 1\rangle, \ldots, \langle \bar{b}_4, 1\rangle, \langle b_1 \vee \ldots \vee b_4, \infty\rangle\rangle$, *we get the MaxSAT formula* φ^e*:*

	CNF	
$1 : \bar{b}_1 \vee \bar{r}_2$	$1 : \bar{b}_1 \vee \bar{r}_2$	(1)
$1 : \bar{b}_2 \vee \bar{r}_3$	$1 : \bar{b}_2 \vee \bar{r}_3$	(2)
$1 : \bar{b}_3 \vee \bar{r}_4$	$1 : \bar{b}_3 \vee \bar{r}_4$	(3)
$1 : \square$	$1 : \square$	
$\infty : r_2 \leftrightarrow b_2 \vee r_3$	$\infty : \bar{r}_2 \vee b_2 \vee r_3$	(4)
	$\infty : \bar{b}_2 \vee r_2$	(5)
	$\infty : \bar{r}_3 \vee r_2$	(6)
$\infty : r_3 \leftrightarrow b_3 \vee r_4$	$\infty : \bar{r}_3 \vee b_3 \vee r_4$	(7)
	$\infty : \bar{b}_3 \vee r_3$	(8)
	$\infty : \bar{r}_4 \vee r_3$	(9)
$\infty : r_4 \leftrightarrow b_4$	$\infty : \bar{r}_4 \vee b_4$	(10)
	$\infty : \bar{b}_4 \vee r_4$	(11)
$\infty : b_1 \vee b_2 \vee b_3 \vee b_4$	$\infty : b_1 \vee b_2 \vee b_3 \vee b_4$	(12*)

The regular encoding of $b_1 + b_2 + b_3 + b_4 = 1$ *is the SAT formula* φ^r *where* r_i *is true iff* $(b_i \vee \ldots \vee b_n)$ *is true:*

	CNF	
$b_1 \leftrightarrow \bar{r}_2$	$\bar{b}_1 \vee \bar{r}_2$	(1)
	$b1 \vee r_2$	(12)
$b_2 \leftrightarrow r_2 \wedge \bar{r}_3$	$\bar{b}_2 \vee r_2$	(5)
	$\bar{b}_2 \vee \bar{r}_3$	(2)
	$\bar{r}_2 \vee b_2 \vee r_3$	(4)
$b_3 \leftrightarrow r_3 \wedge \bar{r}_4$	$\bar{b}_3 \vee r_3$	(8)
	$\bar{b}_3 \vee \bar{r}_4$	(3)
	$\bar{r}_3 \vee b_3 \vee r_4$	(7)
$b_4 \leftrightarrow r_4$	$\bar{b}_4 \vee r_4$	(11)
	$\bar{r}_4 \vee b_4$	(10)
$r_4 \rightarrow r_3$	$\bar{r}_4 \vee r_3$	(9)
$r_3 \rightarrow r_2$	$\bar{r}_3 \vee r_2$	(6)

Clauses $1 - 11$ *are exactly the same for* φ^e *and* φ^r*, and correspond to* $b_1 + b_2 + b_3 + b_4 \leq 1$*. Since* $r_2 \equiv (b_2 \vee b_3 \vee b_4)$*, from clause* 12*, we can get clause* 12^**, that corresponds to* $b_1 + b_2 + b_3 + b_4 \geq 1$*.*

Another interesting observation comes from the fact that the regular encoding of a 1-Cardinality constraint is equivalent to the Sequential counter encoding [46] for k-Cardinality constraints when $k = 1$. Then, if we get a core φ_n with $cost(\varphi_n) = e$ and apply recursively e times the Eva transformation we end up with the Sequential counter encoding for an e-Cardinality constraint.

Back to the algorithms, we have seen the particular transformation that Eva applies to φ_n to get φ'_n. Actually, we can use any other transformation if we ensure that φ_n is MaxSAT reducible to $\varphi'_n + \langle \square, 1\rangle$. For example, when we have already reified the soft clauses $\langle \overline{b}_i \leftrightarrow c_i, \infty\rangle$ and we want to derive the empty clause from φ_b, we can transform it into $\varphi'_o + \langle \square, 1\rangle$ as follows. We can replace the n soft clauses $\langle b_i, 1\rangle$ with $n-1$ new pairs of clauses $\langle \overline{o}_k, 1\rangle, \langle \overline{o}_k \rightarrow (b_1+, \ldots, +b_n \leq k), \infty\rangle$ for $k \in \{1, \ldots, n-1\}$ and an empty clause $\langle \square, 1\rangle$. We explain in detail this transformation in Example 3.

Example 3. *Given $\varphi_b = \langle \langle \overline{b}_1, 1\rangle, \ldots, \langle \overline{b}_4, 1\rangle, \langle b_1 \vee \ldots \vee b_4, \infty\rangle\rangle$, let us show an alternative transformation to derive the empty clause $\langle \square, 1\rangle$ and guarantee MaxSAT reducibility.*

Let us first notice that the formula on the left, φ_b, is MaxSAT reducible to the formula on the right φ_o. If the assignment $\mathcal{I} : var(\varphi_b) \rightarrow \{0,1\}$ sets m b_i variables to true: (i) the cost $\mathcal{I}(\varphi_b)$ will be m and (ii) the cost $\mathcal{I}(\varphi_o)$ will also be m since o_k will be true for $k \in \{0, \ldots, m-1\}$.

$1 : \overline{b}_1$	$1 : \overline{o}_3$
$1 : \overline{b}_2$	$1 : \overline{o}_2$
$1 : \overline{b}_3$	$1 : \overline{o}_1$
$1 : \overline{b}_4$	$1 : \overline{o}_0$
$\infty : b_1 \vee b_2 \vee b_3 \vee b_4$	$\infty : CNF(b_1 + b_2 + b_3 + b_4 \geq 1)$
	$\infty : CNF(\overline{o}_3 \rightarrow (b_1+b_2+b_3+b_4 \leq 3))$
	$\infty : CNF(\overline{o}_2 \rightarrow (b_1+b_2+b_3+b_4 \leq 2))$
	$\infty : CNF(\overline{o}_1 \rightarrow (b_1+b_2+b_3+b_4 \leq 1))$
	$\infty : CNF(\overline{o}_0 \rightarrow (b_1+b_2+b_3+b_4 \leq 0))$

In the case of φ_o, it is trivial to see that $\langle o_0, \infty\rangle$ is a logical consequence of the hard clauses. Applying the MaxSAT resolution rule to $\langle o_0, \infty\rangle, \langle \overline{o}_0, 1\rangle$, we can replace $\langle \overline{o}_0, 1\rangle$ with $\langle \square, 1\rangle$, resulting $\varphi'_o + \langle \square, 1\rangle$:

$$
\begin{aligned}
&1 : \overline{o}_3\\
&1 : \overline{o}_2\\
&1 : \overline{o}_1\\
&1 : \Box\\
&\infty : \mathit{CNF}(b_1 + b_2 + b_3 + b_4 \geq 1)\\
&\infty : \mathit{CNF}(\overline{o}_3 \rightarrow (b_1 + b_2 + b_3 + b_4 \leq 3))\\
&\infty : \mathit{CNF}(\overline{o}_2 \rightarrow (b_1 + b_2 + b_3 + b_4 \leq 2))\\
&\infty : \mathit{CNF}(\overline{o}_1 \rightarrow (b_1 + b_2 + b_3 + b_4 \leq 1))\\
&\infty : \mathit{CNF}(\overline{o}_0 \rightarrow (b_1 + b_2 + b_3 + b_4 \leq 0))
\end{aligned}
$$

For $(b_1 + b_2 + b_3 + b_4 \geq k)$*, we could replace* $\langle o_i, 1\rangle \mid i < k$ *with* $\langle \Box, k\rangle$*.*

Applying this alternative transformation to Eva is almost what OLL does. The only difference is that when OLL converts the working formula φ'_o to SAT (removes the weights) and sends it to the SAT solver to check if $\mathit{cost}(\varphi'_o) = 0$, it does not to include the clauses $(\overline{o}_2)$ and $(\overline{o}_3)$. Notice that all assignments to b_i variables that satisfy $(\overline{o}_1)$ also satisfy $(\overline{o}_2)$ and $(\overline{o}_3)$. OLL only sends to the SAT solver the clause $(\overline{o}_{i+1})$ if $(\overline{o}_i)$ appeared previously in a unsatisfiable core. Therefore, we can conclude that both algorithms are in fact very similar.

Finally, for the non-weighted case, Eva and OLL increase at each iteration the lower bound only by 1 (i.e. $\mathit{cost}(\varphi^k) = \mathit{cost}(\varphi^{k+1}) + 1$). We know however that there are algorithms like WPM2 that can increase at each iteration the lower bound by more than 1 by optimizing subproblems [18]. In the next section, we will present the WPM3 algorithm that is able to increase the lower bound by more than 1 at each iteration like WPM2 while preserving some of the strengths of Eva and OLL.

3. The WPM3 Complete Algorithm

In this section, we present the WPM3 complete algorithm for the MaxSAT problem. Like Eva and OLL, it is a core-guided algorithm that only handles Cardinality constraints. In addition, WPM3 is able to increase the lower bound to higher values at each iteration and obtain also upper bounds and assignments. The basic schema of WPM3 consists in, given an input WPMS formula φ, testing the satisfiability of a sequence of SAT instances ϕ^k where $0 \leq k \leq \sum w(\varphi)$. Each SAT instance ϕ^k encodes whether there is an assignment to φ with a cost $\leq k$. Notice that SAT instances with $k < \mathit{cost}(\varphi)$ are unsatisfiable, while the others are satisfiable. WPM3 increases the lower bound by testing unsatisfiable ϕ^k SAT instances.

Algorithm 1: WPM3

```
    Input: $\varphi = \langle\langle c_1, w_1\rangle, ..., \langle c_s, w_s\rangle, \langle c_{s+1}, \infty\rangle, ..., \langle c_{s+h}, \infty\rangle\rangle$
1  $\langle AM, w_{strat}\rangle := \langle\langle\langle\langle 1\rangle, 0, w_1\rangle, \ldots, \langle\langle s\rangle, 0, w_s\rangle\rangle, \infty\rangle$
2  while $true$ do
3      $\langle st, C, \mathcal{I}\rangle := sat(\varphi, AM, w_{strat})$
4      if $st =$ SAT then
5          if $w_{strat} = \min_{w_j \in w(AM), w_j \neq 0}(\{w_j\})$ then return $\langle \mathcal{I}, \mathcal{I}(\varphi)\rangle$
6          $w_{strat} = decrease(AM, w_{strat})$
7      else
8          if $w_{strat} = \infty$ then return $\langle \emptyset, \infty\rangle$
9          $w_{min} := min(w(AM_C))$
10         $AM := AM[\underset{j \in C}{\langle A_j, k_j, w_j\rangle} / \underset{j \in C}{\langle A_j, k_j, w_j - w_{min}\rangle}]$
11         $\langle A, k_A\rangle := optimize(\varphi, AM_C)$
12         $AM := AM + \langle\langle A, k_A, w_{min}\rangle\rangle$
```

Function sat(φ, AM, w_{strat})

```
1  $\phi^k := \bigcup_{\langle c_i, w_i\rangle \in \varphi, w_i = \infty} \{c_i\} \cup \bigcup_{\langle c_i, w_i\rangle \in \varphi, w_i \neq \infty} \{c_i \vee b_i\} \cup \bigcup_{\langle A_j, k_j, w_j\rangle \in AM, w_j \geq w_{strat}} \{CNF(A_j, k_j) \vee a_j, \overline{a}_j\}$
2  $\langle st, \phi^k_C, \mathcal{I}\rangle := satsolver(\phi^k)$
3  $C := \{j \mid \{\overline{a}_j\} \cap \phi^k_C \neq \emptyset\}$
4  return $\langle st, C, \mathcal{I}\rangle$
```

Function optimize(φ, AM)

```
1  $A := \underset{\langle A_j, K_j, w_j\rangle \in AM}{+A_j}$
2  $ub := \sum_{\langle A_j, k_j, w_j\rangle \in AM} \mid A_j \mid$
3  while $true$ do
4      $k := ub - 1$
5      $\langle st, \_, \mathcal{I}\rangle := sat(\varphi, \langle\langle A, k, \_\rangle\rangle, \_)$
6      if $st =$ SAT then $ub := \mathcal{I}(\langle\underset{i \in A}{\langle c_i, 1\rangle \mid \langle c_i, w_i\rangle} \in \varphi\rangle)$
7      else return $\langle A, ub\rangle$
```

Roughly speaking, from every unsatisfiable SAT instance WPM3 finds and keeps an unsatisfiable core. The algorithm is designed to be aware of the global structure of theses cores. This is used both for producing more efficient Cardinality constraint encodings (see Section 4) and focus the search on subproblems of the input WPMS instance.

The algorithm maintains a set of soft at-most Cardinality constraints AM. We note these constraints as $\langle A, k, w\rangle$, where A is an ordered multiset of indexes of the original soft clauses, k indicates at most how many clauses from A can be falsified, and w is the cost for falsifying this soft constraint. The at-most constraints are used to do not accept those assignments whose cost exceeds the current k, where $k = \sum_{\langle A_j, k_j, w_j\rangle \in AM} k_j \cdot w_j$. Moreover, the algorithm guarantees that $k \leq \mathit{cost}(\varphi)$. The idea of maintaining multiple at-most Cardinality constraints instead of a single one was originally introduced in [47] for PM2 algorithm. Notice that from the AM set the global core structure can be obtained.

We start (line 1) by adding to AM a soft at-most constraint for each original soft clause. Then, the algorithm will iterate (line 2) till it is able to determine $\mathit{cost}(\varphi)$. This will happen if it detects that the set of hard clauses is unsatisfiable (line 8, $\mathit{cost}(\varphi) = \infty$) or when it is able to generate the first satisfiable instance (line 5, $\mathit{cost}(\varphi) = k = \mathcal{I}(\varphi)$). We obviate for the moment the role of w_{stat} and we consider it is ∞ for the first iteration and $\underset{w_j \in w(AM), w_j \neq 0}{min}(\{w_j\})$ for the rest.

Function *sat* (line 3) builds the SAT instance ϕ^k at the current iteration and sends it to the SAT solver. The SAT instance ϕ^k is constructed through the union of the following sets expressed as SAT clauses: (i) the set of hard clauses, (ii) the reification to variables b_i of soft clauses, (iii) the reification to variables a_j of the translation into *CNF* of the at-most constraints in AM, and finally (iv) the unit clauses $\overline{a}_j$. The new b_i variables are true if the respective original soft clause becomes false. They are used to encode the at-most constraints which restrict the number of falsified soft clauses. The new a_j variables are true if the respective at-most constraint becomes false. The unit clauses $\overline{a}_j$ encode that we would like to satisfy all the at-most constraints. If this is not possible, some of them will appear into the unsatisfiable core ϕ^k_C.

Example 4. *Given* $\varphi = \langle\langle x_1, 1\rangle, \langle x_2, 1\rangle, \langle x_3, 1\rangle, \langle \overline{x}_1 \vee \overline{x}_2, \infty\rangle, \langle \overline{x}_1 \vee \overline{x}_3, \infty\rangle, \langle \overline{x}_2 \vee \overline{x}_3, \infty\rangle\rangle$ *and* $AM = \langle\langle\langle 1, 2\rangle, 1, 1\rangle, \langle\langle 3\rangle, 0, 1\rangle\rangle$*, then:*

$$\phi^k = \{\overline{x}_1 \vee \overline{x}_2,\ \overline{x}_1 \vee \overline{x}_3,\ \overline{x}_2 \vee \overline{x}_3\} \cup\ \{x_1 \vee b_1,\ x_2 \vee b_2,\ x_3 \vee b_3\}$$
$$\cup\ \{CNF(b_1 + b_2 \leq 1) \vee a_1, \overline{a}_1, CNF(b_3 \leq 0) \vee a_2, \overline{a}_2\}$$

Following the same idea of the transformation in Example 3 of Section 2, we can consider that WPM3 maintains in AM the information to build a WPMS formula φ^k such that $cost(\varphi) = cost(\varphi^k + \langle \Box, k \rangle)$. For each $\langle A_j, k_j, w_j \rangle$ in AM, the clause $\overline{a}_j$ of Example 4 corresponds to $\langle \overline{o}^j_{k_j}, w_j \rangle$ according to the notation of Example 3. The formula ϕ^k, sent to the SAT solver to check if $(\varphi^k) = 0$, does not include the clauses $\overline{o}^j_i \mid i > k_j$ and the corresponding Cardinality constraints since they are satisfied for all assignments to b_i variables that satisfy the clauses $\overline{o}^j_{k_j}$ (i.e. $\overline{a}_j$).

Back to the WPM3 algorithm, if function *sat* returns satisfiable ($st = SAT$) (line 4), then we return the optimal assignment $\mathcal{I}$ and its cost (line 5). Otherwise (line 7), C is the set of indexes of at-most constraints involved into the last unsatisfiable core. If the core only involves original hard clauses, we can return unsatisfiable (line 8). If there are at-most constraints involved in the core, then, we need to relax some of them since they only allow assignments with a cost strictly less than $cost(\varphi)$.

At this stage (lines 9-12), we need to relax the set of AM constraints, but guaranteeing we do not exceed $cost(\varphi)$. Basically, we need to replace the set of at-most constraints AM_C involved in the last core with a new set of at-most constraints which allows assignments with a higher cost.

Since we will only use Cardinality constraints, we first apply the idea described in the WPM1/WBO MaxSAT algorithms [24, 25] to deal with weighted instances (lines 9-10). It basically prevents the algorithm to introduce weighted PB constraints instead of Cardinality constraints when the at-most constraints involved in the core have different weights. In this case, we compute the minimum weight w_{min} of the constraints in AM_C (line 9), and replace every soft constraint $\langle A_j, k_j, w_j \rangle$ by two copies with weights $w_j - w_{min}$ and w_{min}. The first set of copies will remain in AM (line 10) while the second will be replaced by the new at-most constraint $\langle A, k_A, w_{min} \rangle$ (line 12). Notice we guarantee that $\sum_{\langle A_j, k_j, w_j \rangle \in AM} \mid A_j \mid \cdot w_j = \sum w(\varphi)$.

Function *optimize* (line 11) allows to determine which is the new ϕ_k we will test. This function, basically, solves exactly the subproblem that involves the new at-most constraint we will generate on the original soft clauses, and the hard clauses. The result is the set of indexes of original soft clauses A of the new at-most constraint (notice that any index can be repeated) and the number of clauses k_A that we will at most allow to be falsified. To our best knowledge, the idea of solving a subproblem of the original optimization instance φ was originally applied for MaxSAT in WPM2 algorithm [26]. In [48] a similar approach is applied calling a MIP solver to solve the subproblem. Recently, in [17], this process is extended and named as *cover optimization*. The best strategy reported consists in refining iteratively the upper bound on the subproblem using the model-guided MaxSAT algorithm in [32]. We apply it within function *optimize*, although depending on the particular family of instances other strategies or algorithms can have better performance. At this point we have increased k by $(k_A - \sum_{\langle A_j, k_j, w_j\rangle \in AM_C} k_j) \cdot w_{min}$ towards $cost(\varphi)$. Without the *optimize* step, we can only increase k by $1 \cdot w_{min}$.

Treating explicitly the new at-most constraint on the original soft clauses (line 12) and its relation to the constraints it replaces is fundamental, not only for function *optimize*, but also to encode more efficient Cardinality constraints (see Section 4). In contrast, this information (the global core structure) is not explicitly present in Eva and OLL algorithms (see Section 2) and therefore harder to be exploited efficiently (see Section 5).

During this description, we have obviated the role of w_{strat} (lines 5 and 6). It corresponds to the application of the stratified approach introduced in [49]. The stratified approach consists in sending to the SAT solver only a subset of the soft clauses, i.e., those with a weight $\geq w_{strat}$. Function *decrease* updates conveniently w_{strat}. This can help to reduce the size of the unsatisfiable cores, produce simpler at-most constraints and progress faster to the optimum. We also apply *hardening* techniques like the ones described in [29, 36, 17].

4. Efficient Cardinality Constraints for MaxSAT

In the last decade, we have seen many contributions on how to encode efficiently PB and Cardinality constraints into SAT [50, 46, 31, 51, 38]. The goal is to achieve an arc-consistent encoding (i.e., with good propagation properties) as small as possible.

Since WPM3 only uses Cardinality constraints, let us consider the Cardinality constraint: $b_1 + \cdots + b_n \leq k$. From the sake of clarity, the encoder is split into two black boxes: the *sum* and the operator *op* (in our case representing $\leq$). The *sum* takes as input a list of n variables $[b_1, \ldots, b_n]$ and returns a set of SAT clauses S and a list of $m+1$ variables $[o_0, \ldots, o_m]$. The operator takes as input the o variables and integer k and returns a set of SAT clauses OP. The encoding of the Cardinality constraint into SAT corresponds to the union $S \cup OP$.

$$\langle S, [o_0, \ldots, o_m] \rangle := sum([b_1, \ldots, b_n])$$
$$OP := op(k, [o_0, \ldots, o_m])$$

In our case, we use the Cardinality Networks encoding in [38]. There, $m = k$ and the *sum* builds a SAT formula such that if i of the input b variables are set to true then the first i of the output o variables are set to true. Therefore, *op* returns the unit clause $\overline{o}_k$. [2]

Our first observation is that it is crucial for the efficiency of the MaxSAT solver in which order the b variables are fed into the *sum*. In previous MaxSAT solvers, the order of the b variables was not taken into account. They were just added in the same order of their respective soft clauses.

However, the b variables should be added taking into account the structure of the unsatisfiable cores. In particular, variables belonging to the same core should be as close as possible. In our algorithm the set A in an at-most constraint $\langle A, k, w \rangle$ is ordered. In line 1 of function *optimize* (see Section 3) when we generate the set A of the new at most constraint, we concatenate the sets of b variables of the at-most constraints that it replaces. Respect to latest advances in MaxSAT [40, 41] a deeper explanation of their efficiency could be that these algorithms implicitly preserve the order.

Our second observation has to do again with the structure of the unsatisfiable cores we have detected so far. As we have just commented, the new at-most constraint $\langle A, k_A, w_{min} \rangle$ (line 12 of WPM3) replaces/merges other at-most constraints. In the end, we can consider that there is a tree structure that represents how we have merged the unsatisfiable cores and where the root node is the new at-most constraint. We can store this tree structure by replacing in the new at-most constraint, the multiset A by the set C of indexes of the merged at-most constraints in AM. Going over the tree, we find in the leaf nodes exactly the same indexes we would have in A. Instead of creating a Cardinality Network for the new constraint we can reproduce this tree structure. We basically reproduce the totalizer encoding proposed originally in [50]. The leaf nodes join the at-most constraints related with a single soft clause of the input formula (i.e., with $k = 0$) that appeared in the same core. The leaf nodes are encoded with Cardinality Networks.

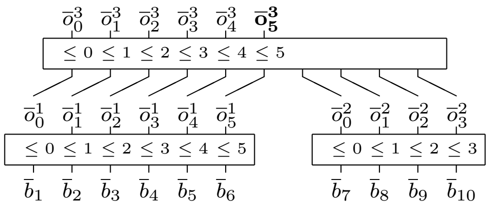
[2]In this paper, we have adapted the indexes so that $\overline{o}_i \rightarrow b_1 + \ldots + b_n \leq i$.

Example 5. *Imagine at-most constraint $\langle A_3, 5, 1\rangle$ (root node) replaces at-most constraints $\langle A_1, 2, 1\rangle$ and $\langle A_2, 1, 1\rangle$ (child 1 and child 2). Let us see how we start constructing the tree. Children are processed recursively in the same way:* [3]

$$\langle S', \overset{\overset{|A_3|=10}{sum\leq 5}}{[o^3_0, \ldots, o^3_5]}\rangle := merge(\overset{\overset{|A_1|=6}{sum\leq 5}}{[o^1_0, ..., o^1_5]}, \overset{\overset{|A_2|=4}{sum\leq 4}}{[o^2_0, ..., o^2_3]})$$

$$S^3 := S' \cup S^1 \cup S^2 \qquad OP^3 := \{\overline{o}^3_5\} \cup OP^1 \cup OP^2$$

We can represent the tree structure of the constraint as follows (the big boxes represent the merge function): [4]



$\overline{o}^3_0$ $\overline{o}^3_1$ $\overline{o}^3_2$ $\overline{o}^3_3$ $\overline{o}^3_4$ $\mathbf{\overline{o}^3_5}$
$\leq 0 \leq 1 \leq 2 \leq 3 \leq 4 \leq 5$
$\overline{o}^1_0$ $\overline{o}^1_1$ $\overline{o}^1_2$ $\overline{o}^1_3$ $\overline{o}^1_4$ $\overline{o}^1_5$
$\leq 0 \leq 1 \leq 2 \leq 3 \leq 4 \leq 5$
$\overline{o}^2_0$ $\overline{o}^2_1$ $\overline{o}^2_2$ $\overline{o}^2_3$
$\leq 0 \leq 1 \leq 2 \leq 3$
$\overline{b}_1$ $\overline{b}_2$ $\overline{b}_3$ $\overline{b}_4$ $\overline{b}_5$ $\overline{b}_6$ $\overline{b}_7$ $\overline{b}_8$ $\overline{b}_9$ $\overline{b}_{10}$

Using the totalizer encoding, the set S' would be as follows:

$$\begin{array}{l}
\overline{o}^3_0 \rightarrow ((\overline{o}^1_0) \wedge (\overline{o}^2_0)) \\
\overline{o}^3_1 \rightarrow ((\overline{o}^1_1) \wedge (\overline{o}^1_0 \vee \overline{o}^2_0) \wedge (\overline{o}^2_1)) \\
\overline{o}^3_2 \rightarrow ((\overline{o}^1_2) \wedge (\overline{o}^1_1 \vee \overline{o}^2_0) \wedge (\overline{o}^1_0 \vee \overline{o}^2_1) \wedge (\overline{o}^2_2)) \\
\overline{o}^3_3 \rightarrow ((\overline{o}^1_3) \wedge (\overline{o}^1_2 \vee \overline{o}^2_0) \wedge (\overline{o}^1_1 \vee \overline{o}^2_1) \wedge (\overline{o}^1_0 \vee \overline{o}^2_1) \wedge (\overline{o}^2_3)) \\
\overline{o}^3_4 \rightarrow ((\overline{o}^1_4) \wedge (\overline{o}^1_3 \vee \overline{o}^2_0) \wedge (\overline{o}^1_2 \vee \overline{o}^2_1) \wedge (\overline{o}^1_1 \vee \overline{o}^2_2) \wedge (\overline{o}^1_0 \vee \overline{o}^2_3)) \\
\overline{o}^3_5 \rightarrow ((\overline{o}^1_5) \wedge (\overline{o}^1_4 \vee \overline{o}^2_0) \wedge (\overline{o}^1_3 \vee \overline{o}^2_1) \wedge (\overline{o}^1_2 \vee \overline{o}^2_2) \wedge (\overline{o}^1_1 \vee \overline{o}^2_3))
\end{array}$$

The advantage of preserving this structure, in contrast to having a single Cardinality constraint, is that we can again exploit it to derive smaller encodings. In particular, we can use the lower bounds of each node to skip parts of the encoding. Moreover, we can restrict the upper bounds of the nodes using the lower bounds of their siblings. The upper bound for a non root node is set to the difference between the upper bound of its parent and the sum of the lower bounds of its siblings. We apply this in a top-down update process.

[3]Since $b_1 + \ldots + b_n \leq n$ is always true, the variable o_n is not needed.

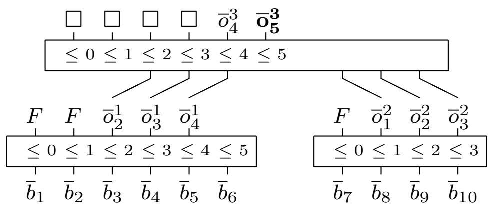
[4]In the case of $b_i \leq 0$, we can use directly $\overline{b}_i$ instead of $\overline{o}_0$.

Example 6. *At Example 5, the upper bound for the root node is* 5*. Child 1 (child 2) already contributes with a lower bound of* 2 (1)*, therefore the new upper bound for child 1 (child 2) is* $5-1=4$ ($5-2=3$)*:* [5]

$$\langle S', \underbrace{\overset{\substack{|A_3|=10\\ sum\leq 5}}{[o^3_0,\ldots,o^3_5]}}_{lb=4}\rangle := merge(\underbrace{\overset{\substack{|A_1|=6\\ sum\leq 4}}{[o^1_0,...,o^1_4]}}_{lb=2}, \underbrace{\overset{\substack{|A_2|=4\\ sum\leq 3}}{[o^2_0,...,o^2_3]}}_{lb=1})$$

$$S^3 := S' \cup S^1 \cup S^2 \qquad OP^3 := \{\overline{o}^3_5\} \cup OP^1 \cup OP^2$$

The tree structure can be represented as follows (the lower bounds are represented: by empty clauses $\Box$ *at the root node for the new constraint, and, by the value false* F *at the intermediate nodes for the replaced constraints):*



□ □ □ □ $\overline{o}^3_4$ $\mathbf{o^3_5}$
≤ 0 ≤ 1 ≤ 2 ≤ 3 ≤ 4 ≤ 5
F F $\overline{o}^1_2$ $\overline{o}^1_3$ $\overline{o}^1_4$
≤ 0 ≤ 1 ≤ 2 ≤ 3 ≤ 4 ≤ 5
$\overline{b}_1$ $\overline{b}_2$ $\overline{b}_3$ $\overline{b}_4$ $\overline{b}_5$ $\overline{b}_6$
F $\overline{o}^2_1$ $\overline{o}^2_2$ $\overline{o}^2_3$
≤ 0 ≤ 1 ≤ 2 ≤ 3
$\overline{b}_7$ $\overline{b}_8$ $\overline{b}_9$ $\overline{b}_{10}$

Taking into account the lower and upper bounds of all constraints, the following parts of the set S' of Example 5 can be skipped:

~~$\overline{o}^3_0 \rightarrow ((\overline{o}^1_0) \wedge (\overline{o}^2_0))$~~
~~$\overline{o}^3_1 \rightarrow ((\overline{o}^1_1) \wedge (\overline{o}^1_0 \vee \overline{o}^2_0) \wedge (\overline{o}^2_1))$~~
~~$\overline{o}^3_2 \rightarrow ((\overline{o}^1_2) \wedge (\overline{o}^1_1 \vee \overline{o}^2_0) \wedge (\overline{o}^1_0 \vee \overline{o}^2_1) \wedge (\overline{o}^2_2))$~~
~~$\overline{o}^3_3 \rightarrow ((\overline{o}^1_3) \wedge (\overline{o}^1_2 \vee \overline{o}^2_0) \wedge (\overline{o}^1_1 \vee \overline{o}^2_1) \wedge (\overline{o}^1_0 \vee \overline{o}^2_1) \wedge (\overline{o}^2_3))$~~
$\overline{o}^3_4 \rightarrow (\cancel{(\overline{o}^1_4)} \wedge (\overline{o}^1_3 \vee \cancel{\overline{o}^2_0}) \wedge (\overline{o}^1_2 \vee \overline{o}^2_1) \wedge (\cancel{\overline{o}^1_1} \vee \overline{o}^2_2) \wedge \cancel{(\overline{o}^1_0 \vee \overline{o}^2_3)})$
$\overline{o}^3_5 \rightarrow (\cancel{(\overline{o}^1_5)} \wedge (\overline{o}^1_4 \vee \cancel{\overline{o}^2_0}) \wedge (\overline{o}^1_3 \vee \overline{o}^2_1) \wedge (\overline{o}^1_2 \vee \overline{o}^2_2) \wedge (\cancel{\overline{o}^1_1} \vee \overline{o}^2_3))$

Finally, the upper bound of the root node can be initialized taking into account the information retrieved in previous calls to *optimize* function. In particular, the assignments that this function provides can be used to get upper bounds for the whole formula (see Section 6). The upper bound of the root node can be initialized to the cost of the best global assignment on the set of soft clauses related to it.

[5]Notice that the lower bound of the new constraint is 4 since it must be higher than the addition of the lower bounds of the replaced constraints (2 + 1). After applying the *o*ptimize function we may learn that the lower bound is increased in more than 1.

5. Comparison of WPM3 with EVA and OLL

According to Sections 2 and 3, WPM3, OLL and Eva maintain a working WPMS formula φ^k and transform it at each iteration. These transformations guarantee that $cost(\varphi) = \ldots = cost(\varphi^{k^i} + \langle\Box, k^i\rangle) = cost(\varphi^{k^{i+1}} + \langle\Box, k^i + n.w^i_{min}\rangle) = \ldots$. Each φ^k without weights is sent to a SAT solver and if it is satisfiable ($cost(\varphi^k) = 0$), k is the optimum. One of the advantage of WPM3 is that, unlike Eva and OLL, n can be more than 1 at each iteration. It achieves this by applying *cover optimization* efficiently thanks to being aware of the global structure of the cores. One of the key points is how WPM3 can explicitly build constraints on the original soft clauses like the others do implicitly. We can see this in Example 7.

Example 7. *To illustrate how WPM3 builds explicitly Cardinality constraints on the original soft clauses like Eva and OLL do implicitly, let us compare the three iterations they make to solve the following pigeon-hole formula:* [6]

$$\varphi = \langle\langle\overline{b}_1, 1\rangle, \langle\overline{b}_2, 1\rangle, \langle\overline{b}_3, 1\rangle, \langle\overline{b}_4, 1\rangle, \langle\overline{b}_5, 1\rangle, \langle\phi_H, \infty\rangle\rangle$$

$$\phi_H = CNF(\{\underset{i\in\{1,\ldots,5\}}{x_i \leftrightarrow \overline{b}_i}\}) \cup CNF(x_1 + \ldots + x_5 \leq 2)$$

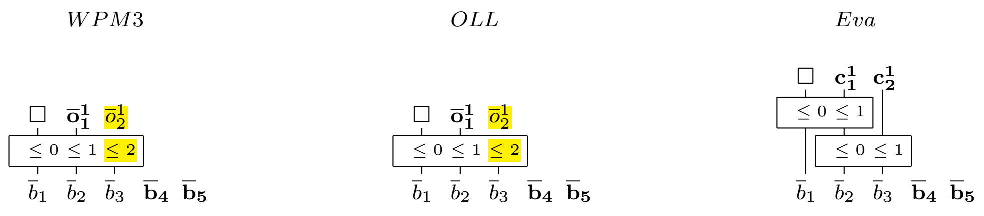
Each x_i (b_i) variable corresponds to a pigeon. If it is true (false), the pigeon is in a hole. According to hard clauses, there are only two holes so only two pigeons can be in a hole (at most 2 true x_i variables) and the rest must be out (at least 3 true b_i variables). Therefore, the optimum is 3.

In the first iteration, the three algorithms send the input MaxSAT formula without weights to a SAT solver to check if $cost(\varphi) = 0$, and get the unsatisfiable core $\{\overline{b}_1, \overline{b}_2, \overline{b}_3\} \cup \phi_H$. Then, they transform φ to derive the empty clause and generate φ^1. WPM3 and OLL replace the soft clauses of the core with two new soft clauses, $\langle\overline{o}^1_1, 1\rangle$ and $\langle\overline{o}^1_2, 1\rangle$, and corresponding Cardinality constraints allowing respectively 1 and 2 of the original soft clauses (b_i variables) of the core to be false (true). The soft clause $\langle\overline{o}^1_2, 1\rangle$ and corresponding Cardinality constraint are highlighted, meaning they are not needed to check if $cost(\varphi^1) = 0$ in the next iteration (($\overline{o}^1_2$) is satisfied for all assignments to b_i variables that satisfy ($\overline{o}^1_1$)). Eva replaces the soft clauses of the core with two new soft clauses obtained by applying successively the MaxSAT resolution rule. The three resulting φ^1 allow one pigeons to be out (1 true b_i variable):

[6]Soft clauses are already reified ($x_i \leftrightarrow \overline{b}_i$) to simplify the analysis (only $\leftarrow$ is needed).

WPM3
$\langle\langle\square,1\rangle\rangle + \langle\langle\overline{o}^1_1,1\rangle, \langle\overline{o}^1_2,1\rangle, \langle\overline{b}_4,1\rangle,\langle\overline{b}_5,1\rangle,$ $\langle\overline{o}^1_1 \rightarrow (b_1+b_2+b_3\le 1),\infty\rangle, \langle\overline{o}^1_2 \rightarrow (b_1+b_2+b_3\le 2),\infty\rangle,$ $\langle\phi_H,\infty\rangle\rangle$
OLL
$\langle\langle\square,1\rangle\rangle + \langle\langle\overline{o}^1_1,1\rangle, \langle\overline{o}^1_2,1\rangle, \langle\overline{b}_4,1\rangle,\langle\overline{b}_5,1\rangle,$ $\langle\overline{o}^1_1 \rightarrow (b_1+b_2+b_3\le 1),\infty\rangle, \langle\overline{o}^1_2 \rightarrow (b_1+b_2+b_3\le 2),\infty\rangle,$ $\langle\phi_H,\infty\rangle\rangle$
Eva
$\langle\langle\square,1\rangle\rangle + \langle\langle\overline{b_1}\vee\overline{(b_2\vee b_3)},1\rangle,\langle\overline{b_2}\vee\overline{b_3},1\rangle,\langle\overline{b}_4,1\rangle,\langle\overline{b}_5,1\rangle,$ $\langle\phi_H,\infty\rangle\rangle$

We can see a representation of the transformations below these lines. The original soft clauses are at the bottom ($\overline{b}_i$) and the new soft clauses at the top (in the case of Eva, c^1_1 and c^2_1 represent $\overline{b_1}\vee\overline{(b_2\vee b_3)}$ and $\overline{b_2}\vee\overline{b_3}$ respectively). The big boxes represent the Cardinality constraints (see Section 4):



WPM3
OLL
Eva
$\square$ $\overline{\mathbf{o}}^1_1$ $\overline{\mathbf{o}}^1_2$
≤ 0 ≤ 1 ≤ 2
$\overline{b}_1$ $\overline{b}_2$ $\overline{b}_3$ $\overline{\mathbf{b}}_4$ $\overline{\mathbf{b}}_5$
$\mathbf{c}^1_1$ $\mathbf{c}^1_2$

In the second iteration, the three algorithms retrieve an unsatisfiable core from φ^1 and proceed to generate φ^2. WPM3 and OLL get a core containing the soft clauses $\{\overline{b}_4,\overline{b}_5,\overline{o}^1_1\}$ and then, the differences between them begin. WPM3 replaces the soft clauses $\langle\overline{b}_4,1\rangle$ $\langle\overline{b}_5,1\rangle$ and $\langle\overline{o}^1_1,1\rangle$ (Cardinality constraints $\langle\{4\},0,1\rangle$, $\langle\{5\},0,1\rangle$ and $\langle\{1,2,3\},1,1\rangle$ using the notation of Section 3) with a new one $\langle\overline{o}^2_2,1\rangle$ that merges them ($\langle\{1,2,3,4,5\},2,1\rangle$). OLL just handles the soft clause $\langle\overline{o}^1_1,1\rangle$ the same way as the original ones, $\langle\overline{b}_5,1\rangle$ and $\langle\overline{b}_5,1\rangle$, building the new Cardinality constraint on the current soft clauses instead of on the original ones. This is why, when it checks if cost(φ^2) $= 0$, it needs to take into account $\langle\overline{o}^1_2,1\rangle$ and the corresponding Cardinality constraint. Eva gets a core containing the soft clauses $\{\overline{b}_4,\overline{b}_5,c^1_1\}$ and replaces them with two new ones. Notice that the three φ^2 allow two pigeons out (2 true b_i variables). However, the only algorithm that has explicitly introduced a Cardinality constraint on the original soft clauses (b_i variables) is WPM3:

WPM3

$$\langle\langle\Box,2\rangle\rangle + \langle\langle\overline{o}_2^2,1\rangle,\ \langle\overline{o}_3^2,1\rangle,\langle\overline{o}_4^2,1\rangle,$$
$$\langle\overline{o}_2^2 \rightarrow (b_1+\ldots+b_5 \le 2),\infty\rangle,$$
$$\langle\overline{o}_3^2 \rightarrow (b_1+\ldots+b_5 \le 3),\infty\rangle,$$
$$\langle\overline{o}_4^2 \rightarrow (b_1+\ldots+b_5 \le 4),\infty\rangle,$$
$$\langle\phi_H,\infty\rangle\rangle$$

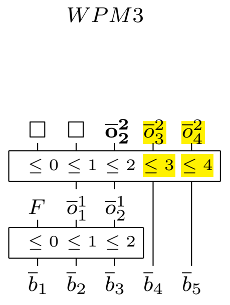
OLL

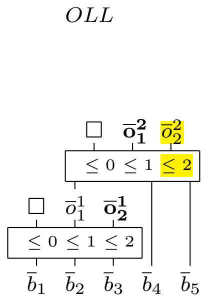
$$\langle\langle\Box,2\rangle\rangle + \langle\langle\overline{o}_1^2,1\rangle,\ \langle\overline{o}_2^2,1\rangle,\ \langle\overline{o}_2^1,1\rangle,$$
$$\langle\overline{o}_1^2 \rightarrow (b_4+b_5+o_1^1 \le 1),\infty\rangle,\ \langle\overline{o}_2^2 \rightarrow (b_4+b_5+o_1^1 \le 2),\infty\rangle,$$
$$\langle\overline{o}_1^1 \rightarrow (b_1+b_2+b_3 \le 1),\infty\rangle,\ \langle\overline{o}_2^1 \rightarrow (b_1+b_2+b_3 \le 2),\infty\rangle,$$
$$\langle\phi_H,\infty\rangle\rangle$$

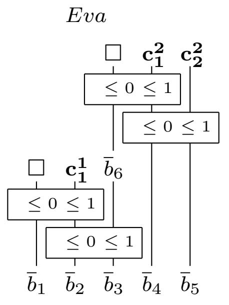
Eva

$$\langle\langle\Box,2\rangle\rangle + \langle\langle\overline{b_6} \vee \overline{(b_4 \vee b_5)},1\rangle,\langle\overline{b_4} \vee \overline{b_5},1\rangle,\langle\overline{b_1} \vee \overline{(b_2 \vee b_3)},1\rangle,$$
$$\langle(\overline{b_2} \vee \overline{b_3}) \leftrightarrow \overline{b}_6,\infty\rangle,$$
$$\langle\phi_H,\infty\rangle\rangle$$

In the representation of the transformations, we can see how WPM3 builds explicitly a Cardinality constraint on the original soft clauses (b_i variables). Thus, it is the only one able to apply cover optimization *efficiently, find out that $\langle o_2^2,1\rangle$ can also be replaced with $\langle\Box,1\rangle$, increase the lower bound by* 2 *and get the optimum before the third iteration. The Cardinality constraints introduced by OLL and Eva are not aware of the global structure:*



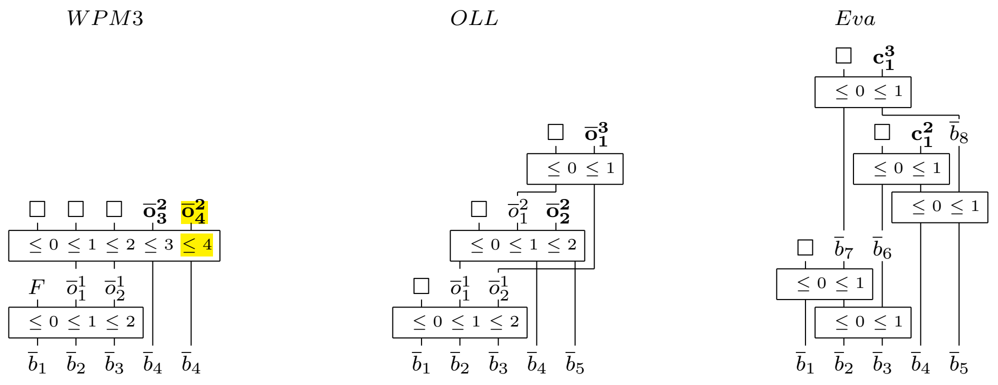




In the third and last iteration, the third empty clause is derived and φ^3 is generated. Without cover optimization, *WPM3 would just get a core with the soft clause $\{\overline{o}_2^2\}$ and replace it with an empty set. OLL and Eva, both get a core with two soft clauses and replaced them with one. After these transformations, all three φ^3 allow three pigeons out* (3 *true b_i variables). This is the condition imposed by the hard clauses, so we will find that cost$(\varphi^3) = 0$ and know that cost$(\varphi) =$ cost$(\varphi^3 + \langle\Box,3\rangle) = 3$:*

$WPM3$
$\langle\langle\square,3\rangle\rangle + \langle\langle\overline{o}_3^2,1\rangle, \langle\overline{o}_4^2,1\rangle,$
$\langle\overline{o}_3^2 \rightarrow (b_1+\ldots+b_5 \leq 3),\infty\rangle,$
$\langle\overline{o}_4^2 \rightarrow (b_1+\ldots+b_5 \leq 4),\infty\rangle,$
$\langle\phi_H,\infty\rangle\rangle$
OLL
$\langle\langle\square,3\rangle\rangle + \langle\ \langle\overline{o}_1^3,1\rangle, \langle\overline{o}_2^2,1\rangle,$
$\langle\overline{o}_1^3 \rightarrow (o_2^1+o_1^2 \leq 1),\infty\rangle,$
$\langle\overline{o}_1^2 \rightarrow (b_4+b_5+o_1^1 \leq 1),\infty\rangle, \langle\overline{o}_2^2 \rightarrow (b_4+b_5+o_1^1 \leq 2),\infty\rangle,$
$\langle\overline{o}_1^1 \rightarrow (b_1+b_2+b_3 \leq 1),\infty\rangle, \langle\overline{o}_2^1 \rightarrow (b_1+b_2+b_3 \leq 2),\infty\rangle,$
$\langle\phi_H,\infty\rangle\rangle$
Eva
$\langle\langle\square,3\rangle\rangle + \langle\langle\overline{b_7}\vee\overline{b_8},1\rangle, \langle\langle\overline{b_6}\vee\overline{(b_4\vee b_5)},1\rangle,$
$\langle(\overline{b_4}\vee\overline{b_5})\leftrightarrow\overline{b}_8,\infty\rangle, \langle(\overline{b_1}\vee\overline{(b_2\vee b_3)})\leftrightarrow\overline{b}_7,\infty\rangle,$
$\langle(\overline{b_2}\vee\overline{b_3})\leftrightarrow\overline{b}_6,\infty\rangle,$
$\langle\phi_H,\infty\rangle\rangle$

In the representation of the transformations, we can see how the Cardinality constraints $b_1+\ldots+b_5 \leq 3$ is built explicitly in the case of WPM3 and implicitly in the case of the other two algorithms:


WPM3
□ □ □ $\overline{o}_3^2$ $\overline{o}_4^2$
$\leq 0 \leq 1 \leq 2 \leq 3 \leq 4$
F $\overline{o}_1^1$ $\overline{o}_2^1$
$\leq 0 \leq 1 \leq 2$
$\overline{b}_1$ $\overline{b}_2$ $\overline{b}_3$ $\overline{b}_4$ $\overline{b}_4$
OLL
□ $\overline{o}_1^3$
$\leq 0 \leq 1$
□ $\overline{o}_1^2$ $\overline{o}_2^2$
$\leq 0 \leq 1 \leq 2$
□ $\overline{o}_1^1$ $\overline{o}_2^1$
$\leq 0 \leq 1 \leq 2$
$\overline{b}_1$ $\overline{b}_2$ $\overline{b}_3$ $\overline{b}_4$ $\overline{b}_5$
Eva
□ c_1^3
$\leq 0 \leq 1$
□ c_1^2 $\overline{b}_8$
$\leq 0 \leq 1$
$\leq 0 \leq 1$
□ $\overline{b}_7$ $\overline{b}_6$
$\leq 0 \leq 1$
$\leq 0 \leq 1$
$\overline{b}_1$ $\overline{b}_2$ $\overline{b}_3$ $\overline{b}_4$ $\overline{b}_5$

Example 7 shows how WPM3 builds explicitly Cardinality constraints on the original soft clauses while OLL and Eva build them implicitly. The advantage of doing this is that, by maintaining the global structure, WPM3 is able to perform *cover optimization* efficiently. If we look at the representations of the second iteration, we find a situation where WPM3 is able to increase the lower bound by more than 1 using *cover optimization*, while others are not. In the case of OLL for example, it does not fully exploit that $\overline{o}_1^1$ is a constraint on three original soft clauses and therefore the lower bound of the new constraint can not be increased by more than 1. This example also brings some light on why *cover optimization* was experimentally shown to be specially effective after merging constraints in [18].

Finally, we will show in our experimentation that building incrementally Cardinality constraints helps but preserving the structure of the cores is the key of the boost on the performance of the solvers. A version of WPM3 feeding from scratch the new Cardinality constraint at each iteration but preserving the structure of the cores is already competitive compared with OLL and EVA. By applying cover optimization, we further improve the performance. Building incrementally Cardinality constraints is just a final step to improve the performance a little bit more.

6. WPM3 Incomplete: Upper Bounds and Phase Saving

In Section 3, we have described the complete core-guided algorithm WPM3 that keeps increasing a lower bound towards the optimum. However, its design, in particular function *optimize* which implements a model-guided MaxSAT algorithm, allows to produce upper bounds during its execution. Whenever function *optimize* finds an assignment to the subproblem, we can just extend this assignment to the rest of the formula and check its cost, obtaining this way naturally an incomplete approach. We do this by considering that, those clauses in an undefined state under the assignment to the subproblem, are falsified [7]. Function *optimize* keeps track of the assignment to the subproblem whose extension to the rest of the formula had the lowest cost. The cost of this assignment is an upper bound for the whole problem. We will refer to it as the best global assignment found by function *optimize*.

We can further exploit the best global assignment we get from function *optimize*. State-of-the-art SAT solvers apply a technique called *phase saving* introduced in [43]. In SAT solvers, when non-chronological backtracking is applied due to a conflict, lots of variable assignments get lost and have to be revealed again during search. The idea is to avoid redoing this work by storing the phase of the variables when we find a conflict. Then, we assign to the next decision variables the stored polarity till we find a new conflict

[7]This can be improved looking at the structure of the undefined clauses.

and update the polarities again. This technique has been shown to be quite effective. We can extend this idea to MaxSAT in the following way. Within function *optimize* we let the SAT solver apply phase saving in the regular way (line 5). Then, if the best global assignment found so far has the same cost for the subproblem as the solution found by function *optimize*, we store the polarity of the variables in the best global assignment. We use these polarities to guide the search of the SAT solver in line 3 of WPM3, disabling the regular phase saving that would be applied. In particular, we only store the polarities of the original variables in φ.

7. Experimental Results

We have evaluated our approach on the industrial instances from the MSE-2014. We run our experiments on a cluster featured with 2.6GHz processors and a memory limit of 3.5 GB. The instance set of MSE-2014 is divided into three categories depending on the variant of the MaxSAT problem: MaxSAT (MS), Partial MaxSAT (PMS) or Weighted Partial MaxSAT (WPMS). In each category, instances are grouped by families: 2 for MS, 22 for PMS and 8 for WPMS. Since families have different number of instances, we considered more fair to present the solvers ranked by mean family ratio of solved instances.

We provide results for the new $wpm3$ MaxSAT solver and the best solvers of the MSE-2014. We have excluded the MaxSAT solver $ISAC$+ [52], since it is a portfolio and our intention here is to compare ground solvers. The ground solvers with the best overall performance at MSE-2014 for industrial instances were: $eva500a$ [41], *mscg* [40] and *open-wbo* [30].

Table 1 shows our first experiment, where we evaluate the impact of each improvement on WPM3 (with a timeout of 1800 seconds). All the variations on WPM3 are implemented on top of the glucose SAT solver (version 3.0) [53]. The different variations and corresponding implementations are named $wpm3$ with different subindexes. Subindex $_o$ stands for *cover optimization* (see Section 3). Regarding how Cardinality constraints are encoded, $_t$ stands for core based tree, $_{tk}$ stands for core based tree with refinement of the k upper bound in sub-sums (see Section 4). Finally, $_p$ stands for phase saving extended to MaxSAT (see Section 6).

At the table, we present for each category and solver the mean ratio of solved instances per family and the total number of instances. We introduce

	MS Ind. 100% 55	PMS Ind. 100% 568	WPMS Ind. 100% 410	Total Ind. 100% 1033
$wpm3_{tkop}$	**88,5% 43**	**84,0% 499**	**74,0% 367**	**81,7% 909**
$wpm3_{tko}$	87,5% 42	83,4% 494	73,9% 366	81,3% 902
$wpm3_{tk}$	87,5% 42	82,5% 483	73,0% 359	80,4% 884
$wpm3_{to}$	87,5% 42	83,4% 494	72,9% 358	81,0% 894
$wpm3_{t}$	87,5% 42	81,9% 480	72,9% 358	80,0% 880
$wpm3_{o}$	87,5% 42	82,9% 489	71,3% 349	80,3% 880
$wpm3$	87,5% 42	80,7% 470	70,4% 346	78,6% 858
$pm2_{o}$	84,0% 39	78,9% 458	-	-
$pm2$	84,0% 39	77,5% 445	-	-
$wpm1/wbo$	80,0% 36	61,8% 349	67,4% 348	64,4% 733

Table 1: WPM3 and improvements.

results for $wpm1/wbo$ [24, 25] and $pm2$ algorithms [8]. $wpm3$ can be considered as a hybridization of these two algorithms (see Section 3). As we can see, $wpm3$ clearly outperforms $pm2$. This is because, as described in Section 4, we build Cardinality constraints taking into account the order imposed by the unsatisfiable cores. If we add the *cover optimization* technique, see $wpm3_o$, then we get a version that would have ranked the first at MSE-2014 for industrial instances in terms of the total mean family ratio. The next two versions, $wpm3_t$ and $wpm3_{tk}$, that further exploit the structure of the cores and improve the construction of the Cardinality constraint, provide a total of 13 additional solved instances for PMS and 13 for WPMS. As we can see, the *cover optimization* technique always improves, in particular for weighted instances at version $wpm3_{tko}$. Finally, the extension of phase saving for MaxSAT improves the average running time, and it helps to solve 7 additional instances within the timeout.

	MS Ind. 100% 55	PMS Ind. 100% 568	WPMS Ind. 100% 410	Total Ind. 100% 1033
$wpm3_{tkop}$	**88.5% 43**	**84.0% 499**	**74.0%** 367	**81.7% 909**
eva500a	86.5% 41	80.0% 476	72.8% **368**	78.7% 885
mscg	86.5% 41	80.2% 468	68.5% 361	77.7% 870
open-wbo	87.5% 42	81.0% 472	64.9% 335	77.3% 849

Table 2: WPM3 compared to best MSE-2014 solvers

[8] $pm2$ algorithm is only designed for Partial MaxSAT instances

Table 2 compares our best version $wpm3_{tkop}$ with the best performing complete solvers at MSE-2014 for industrial instances. Clearly, $wpm3_{tkop}$ dominates both on mean family ratio and total number of solved instances. A deeper analysis reveals that $wpm3_{tkop}$ has best ratio on 30 out of the 32 families that compose the categories, while *eva*500*a* (the second one) has best ratio on 20.

	$wpm3_{tkop}$	*open-wbo*	*qms*	*wpm*2014	*optimax*
$wpm3_{tkop}$		**50** **556** **394**	**50** 502 **379**	**49** **547** **386**	**53** **524** **390**
open-wbo	45 466 344		**45** 458 **361**	44 **477** 341	**45** **466** **378**
qms	15 **511** 281	19 **522** 286		17 **534** 279	18 **530** **302**
*wpm*2014	48 427 369	**48** **477** **387**	**48** 402 **379**		**51** 420 **367**
optimax	39 368 259	43 387 270	**47** 337 284	39 **423** 260	

Table 3: WPM3 compared to MSE-2014 best incomplete solvers.

Our last experiment is presented in Table 3. Since $wpm3_{tkop}$ is able to produce upper bounds we also compared it with the best performing solvers *wpm*2014 and *optimax* [9] for industrial instances at the incomplete track of the MSE-2014. We also compared with other MaxSAT solvers that did not take part in the incomplete track but they are able to produce upper bounds: *open-wbo* [30] and *qms* [33]. We do not include *eva*500*a* and *mscg* since they can not produce upper bounds.

The timeout for the incomplete track at MSE is set to 300 seconds. For a given instance, the winners are the solvers that produce the best upper bound. The best solver is the one that won on more instances. Since these results give us a partial order, it can be misleading to report an overall winner. In table 3, we present the dominance relation between pairs of solvers on the three categories. For example, $wpm3_{tkop}$ (*open-wbo*) is able to obtain a better or equal upper bound than *open-wbo* ($wpm3_{tkop}$) on 50 (45) ms instances, 556 (466) PMS instances and 394 (344) WPMS instances. As we can see, $wpm3_{tkop}$ practically dominates the rest of the solvers. The exception is *qms* on PMS where *qms* outperforms by 9 instances $wpm3_{tkop}$. For this case, we

[9]*optimax* builds on top on the bincd2 algorithm [36]

extended the timeout to 1800 seconds. We found that $wpm3_{tkop}$ outperformed qms by 5 instances. This is somehow expected since $wpm3_{tkop}$, within this timeout, solves to optimality 40 instances more than qms.

	MS Ind. 100% 55	PMS Ind. 100% 568	WPMS Ind. 100% 410	Total Ind. 100% 1033
$wpm3_{tkop}*$	**89.5% 44**	**84.5% 505**	**75.0% 373**	**82.4% 922**
$wpm3_{tkop}$	88.5% 43	84.0% 499	74.0% 367	81.7% 909

Table 4: WPM3 incremental version.

Finally, the WPM3 variants described so far build from scratch the new Cardinality constraint at each iteration. To know the impact of building them incrementally, like Eva and OLL, we implemented another WPM3 variant ($wpm3_{tkop}*$). This variant reuses as much as possible the encodings of the replaced constraints, already introduced to the SAT solver, to build the encoding of the new constraint. We show the results of $wpm3_{tkop}*$ in Table 4 following the same criteria of Table 1. With respect to $wpm3_{tkop}$, it increases the number (mean family ratio) of solved instances in 13 (0,7%). This increase is not as much as the 51 (3,1%) of $wpm3_{tkop}$ with respect to $wpm3$ in Table 1. We can conclude that what makes the difference is taking into account the structure of unsatisfiable cores to build the Cardinality constraints and apply *cover optimization*.

7.1. Results at the MaxSAT Evaluation 2015

We submitted the best variation of WPM3 ($wpm3_{tkop}*$) to the last edition of the international MaxSAT Evaluation (MSE-2015) both at the complete and incomplete track, with the names $WPM3$-2015-*co* and $WPM3$-2015-*in* respectively. The results for industrial instances of all ground solvers, excluding the portfolio approach $ISAC+$-2015 [54] that also incorporated $WPM3$-2015-*co*, are summarized in Tables 5 and 6. At the complete track, $WPM3$-2015-*co* showed a good performance being first and third for PMS and MS industrial subcategories, respectively. At the incomplete track, $WPM3$-2015-*in* won two of the three industrial subcategories, being second in the other and the best solver overall. The other best performing solvers that took part at the complete track of the evaluation were: the new versions of *mscg* and *Open-WBO*, already submitted to MSE-2014, *maxino*, based on the core-guided algorithm described in [55], *MaxHS* and *LMHS*, both based on core-guided algorithm as described in [48, 56, 57, 58]. The two last solvers are interesting alternatives that do not need PB constraints and excelled in few families with a particular structure of weights. They placed in number but not in mean family ratio of solved instances. However, they do not produce upper bounds and can not work as incomplete approaches. At the incomplete track, the other best performing solvers were: the new version of *optiriss*, already submitted to MSE-2014, and ILP, that translated the instance into ILP and applied an ILP solver as described in [59]. Details about all the solvers and authors can be found in [60].

	MS	PMS	WPMS
Ordered by	1. mscg2015a	1. WPM3-2015-co	1. LMHS-I
number	2. mscg2015b	2. Open-WBO-R	2. MaxHS
of solved instances	3. WPM3-2015-co	3. maxino-k16	3. mscg2015b
Ordered by	1. mscg2015a	1. WPM3-2015-co	1. mscg2015b
mean family ratio	2. mscg2015b	2. Open-WBO-R	2. maxino-kdyn
of solved instances	3. WPM3-2015-co	3. maxino-k16	3. mscg2015a

Table 5: MSE-2015 three best complete solvers on industrial subcategories.

	MS	PMS	WPMS
Ordered by	1. optiriss-def-i	1. WPM3-2015-in	1. WPM3-2015-in
number	2. WPM3-2015-in	2. optiriss-def-i	2. optiriss-def-i
of solved instances	3. optiriss-sel-i	3. optiriss-sel-i	3. ILP-2015-in
Ordered by	1. optiriss-def-i	1. WPM3-2015-in	1. WPM3-2015-in
mean family ratio	2. WPM3-2015-in	2. optiriss-def-i	2. optiriss-def-i
of solved instances	3. optiriss-sel-i	3. optiriss-sel-i	3. optiriss-sel-i

Table 6: MSE-2015 three best incomplete solvers on industrial subcategories.

In Table 7, we analyze in detail the results of $WPM3$-2015-*in* on the industrial instances at the incomplete track of MSE-2015. The rows correspond to the subcategories, with the column marked "#" specifying the total number of instances in the class. The columns correspond to solvers, being the first one the virtual best solver at the complete track (VBS-*co*). For VBS-*co* and each class, we present the number and mean family ratio of solved instances in 1800 seconds. The rest of the columns correspond to the virtual best solvers at the incomplete track (VBS-*in*) and the incomplete solvers that took part in the evaluation. For each incomplete solver and class, we present the number of instances where the solver reported the best upper

Family	#	$VBS-co$	$VBS-in$	*wpm3*	*opti*	*ilp*	*dist*	*cc*
MS	55	48	55	39	43	7	-	28
		93.0%	100.0%	84.5%	88.0%	22.0%	-	58.0%
PMS	601	555	587	515	475	238	218	183
		89.0%	97.7%	83.3%	77.4%	34.6%	39.8%	29.3%
WPMS	610	532	606	432	356	253	72	82
		76.1%	98.3%	67.8%	54.5%	33.2%	17.2%	20.0%
Total Ind.	1266	1135	1248	986	874	498	290	293
		85.3%	98.0%	78.7%	71.0%	33.4%	30.7%	28.1%

Table 7: MSE-2015 virtual best complete and incomplete solvers compared to the best incomplete solvers on the industrial instances (number and mean family ratio).

bound in 300 seconds and the mean family ratio according to this number. For the solvers with different versions, *dist* and *cc*, the best upper bound on each instance is considered.

The mean family ratio of solved instances in 1800 seconds for VBS-*co* is 85,3%. However, for some industrial domains the objective is not finding the optimum (optimal assignment) but an upper bound (assignment) of a good quality in a reasonable time. In this sense, the mean family ratio of instances where VBS-*in* is able to obtain an upper bounds in 300 seconds is 98,0%. To measure the quality of these upper bounds we can use the complementary of their relative distance to the optimum, i.e. an upper bound equal to the aggregated cost of all soft clauses or to the optimal cost, has a quality of 0% or 100%, respectively. For those instances with know optimums, those solved by VBS-*co*, the upper bound obtained by VBS-*in* in 300 seconds has always a quality of more than 75%. Moreover for all except a couple of instances the quality is more than 90%. For more information about the quality of optimums obtained thanks to *cover optimization* see [18].

Finally, to place a solver in the first positions of the evaluation, it has to be as robust as possible across all families of a category. However, the best overall performing solvers on a category can be outperformed on some families by others that are less competitive overall. It happens the same with different parameter configurations of a given solver [54]. In Tables 8, 9 and 10 we show the results (best upper bound in 300 seconds) of some variants of our solver *wpm*3 that we have configured for the incomplete track. The parameters include some options of the MaxSAT solver as well as some others of the underlying SAT solver. We can see that several configurations are better than the solver we submitted to MSE-2015 (it corresponds to C_7^m, C_3^p and C_1^w on MS, PMS and WPMS instances respectively). For example, on *trail-tr*

WPMS family C_{10}^w gets the best upper bound in 300 seconds for 6 instances while C_1^w only for 2. Developing this configuration process and implementing new features we could improve even further the performance of *wpm*3.

Instance set	#	$\mathbf{C_1^m}$	C_2^m	C_3^m	C_4^m	C_5^m	C_6^m	C_7^m	C_8^m	C_9^m	C_{10}^m
cir-dp	3	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
sean-s	52	**49**	48	48	47	47	45	44	44	41	39
Total	55	**52**	51	51	50	50	48	47	47	44	42

Table 8: Different configurations of *wpm*3 incomplete on MS industrial instances.

Instance set	#	$\mathbf{C_1^p}$	C_2^p	C_3^p	C_4^p	C_5^p	C_6^p	C_7^p	C_8^p	C_9^p	C_{10}^p
aes	7	**7**	**7**	6	6	6	**7**	**7**	6	1	1
atc-mes	18	**16**	15	14	14	14	13	12	13	12	12
atc-sug	19	12	15	**16**	15	**16**	14	13	15	12	12
bcp-fir	32	**32**	**32**	**32**	**32**	**32**	**32**	**32**	**32**	31	31
bcp-hysi	10	**10**	**10**	9	9	9	**10**	9	9	9	9
bcp-hysu	38	34	**36**	32	31	31	34	34	31	33	33
bcp-msp	40	**32**	28	**32**	**32**	31	**32**	30	**32**	**32**	**32**
bcp-mtg	30	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
bcp-syn	38	24	24	27	27	27	23	25	27	**28**	**28**
cir-tc	4	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
clo-s	50	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	46	46
des	50	**48**	47	45	44	43	45	47	43	45	45
hap-a	6	**6**	5	5	5	5	5	5	5	5	5
hs-time	2	1	1	1	1	1	1	1	1	**2**	**2**
mbd	46	**46**	**46**	45	45	45	**46**	**46**	45	42	42
pac-pms	40	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**
pbo-mne	25	**25**	24	**25**	**25**	**25**	**25**	**25**	**25**	**25**	**25**
pbo-mnl	25	**25**	**25**	**25**	**25**	**25**	**25**	**25**	**25**	**25**	**25**
pbo-rou	15	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**
pro-ins	12	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**
tpr-Mp	36	34	**35**	34	34	**35**	34	33	34	**35**	34
tpr-Op	25	**25**	**25**	**25**	**25**	**25**	**25**	**25**	**25**	**25**	**25**
tr-comp	33	26	27	**28**	**28**	**28**	26	27	**28**	25	24
Total	601	**554**	553	552	549	549	548	547	547	534	532

Table 9: Different configurations of *wpm*3 incomplete on PMS industrial instances.

Instance set	#	$\mathbf{C_1^w}$	C_2^w	C_3^w	C_4^w	C_5^w	C_6^w	C_7^w	C_8^w	C_9^w	C_{10}^w
btbnsl	60	17	13	18	**18**	17	13	13	11	12	10
cor-clu	129	**53**	47	39	39	39	44	39	39	35	38
hap-ped	100	98	98	98	97	97	98	98	**99**	98	98
hs-time	14	**6**	**6**	4	5	5	**6**	**6**	3	5	2
pac-wpms	99	**99**	**99**	**99**	97	97	**99**	**99**	**99**	**99**	**99**
pre-pla	29	**29**	**29**	**29**	**29**	**29**	**29**	**29**	**29**	**29**	**29**
tail-tr	11	2	3	4	4	4	2	2	3	5	**6**
time	26	14	14	15	**18**	**18**	14	14	14	15	13
upg-pro	100	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**
wcsp-s5d	21	17	17	**18**	16	16	17	17	**18**	17	**18**
wcsp-s5l	21	15	15	16	16	16	15	15	16	15	**17**
Total	610	**450**	441	440	439	438	437	432	431	430	430

Table 10: Different configurations of *wpm*3 incomplete on WPMS industrial instances.

8. Conclusions

In this paper, we have first studied the core-guided algorithms Eva and OLL, that inspired the best performing solvers on industrial instances at MSE-2014. We have demonstrated that the transformation applied by Eva at each iteration using the MaxSAT resolution rule corresponds to the introduction of a Cardinality constraint. We have described an alternative transformation applicable to Eva and, after analyzing the connections with OLL, we have concluded that both algorithms are in fact very similar.

After analyzing Eva and OLL, we have also developed our new complete algorithm WPM3. The design of the algorithm allows us to combine several techniques to use only Cardinality constraints and perform the optimization of subproblems efficiently. We have also shown how these Cardinality constraints can be efficiently built by exploiting the structure of unsatisfiable cores in the MaxSAT instances. On the other hand, the optimization of subproblems allows the algorithm, not only to converge more quickly to the optimum, but also to get assignments and upper bounds, being able to work as an incomplete approach given limited time. We have further exploited these assignments to extend effectively the notion of phase saving to MaxSAT.

Finally, we have conducted an extensive experimental investigation on industrial instances, showing the impact of every technique. We have pointed out the importance of exploiting the structure of unsatisfiable cores to build the Cardinality constraints. In fact, we have shown that the improvement on the performance is mostly due to this exploitation and not only to building the Cardinality constraints incrementally. To evaluate independently WPM3, we have submitted it both to the complete and incomplete track of the MSE-2015, winning several categories and subcategories. At the complete track, it got a gold and a bronze medals on PMS and MS industrial subcategories respectively. At the incomplete track, it dominated on industrial instances getting medals in all industrial subcategories, two golds on WPMS and PMS and a silver on MS.

[1] J. Argelich, D. L. Berre, I. Lynce, J. P. M. Silva, P. Rapicault, Solving linux upgradeability problems using boolean optimization, in: Proceedings First International Workshop on Logics for Component Configuration (LoCoCo), 2010, pp. 11–22.

[2] S. Safarpour, H. Mangassarian, A. G. Veneris, M. H. Liffiton, K. A.

Sakallah, Improved design debugging using maximum satisfiability, in: FMCAD, IEEE Computer Society, 2007, pp. 13–19.

[3] Y. Chen, S. Safarpour, J. Marques-Silva, A. G. Veneris, Automated design debugging with maximum satisfiability, IEEE Trans. on CAD of Integrated Circuits and Systems 29 (11) (2010) 1804–1817.

[4] D. M. Strickland, E. R. Barnes, J. S. Sokol, Optimal protein structure alignment using maximum cliques, Operations Research 53 (3) (2005) 389–402.

[5] A. Graça, I. Lynce, J. Marques-Silva, A. L. Oliveira, Efficient and accurate haplotype inference by combining parsimony and pedigree information, in: Algebraic and Numeric Biology - 4th International Conference, ANB 2010, Hagenberg, Austria, July 31- August 2, 2010, Revised Selected Papers, 2010, pp. 38–56.

[6] M. Jose, R. Majumdar, Cause clue clauses: error localization using maximum satisfiability, in: Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2011, San Jose, CA, USA, June 4-8, 2011, 2011, pp. 437–446.

[7] R. J. A. Achá, R. Nieuwenhuis, Curriculum-based course timetabling with SAT and maxsat, Annals OR 218 (1) (2014) 71–91.

[8] M. C. Cooper, S. Cussat-Blanc, M. de Roquemaurel, P. Régnier, Soft arc consistency applied to optimal planning, in: Proc. of CP'06, 2006, pp. 680–684.

[9] L. Zhang, F. Bacchus, MAXSAT heuristics for cost optimal planning, in: Proc. of AAAI'12, 2012, pp. 1846–1852.

[10] M. Vasquez, J. Hao, A "logic-constrained" knapsack formulation and a tabu algorithm for the daily photograph scheduling of an earth observation satellite, Comp. Opt. and Appl. 20 (2) (2001) 137–157.

[11] H. Xu, R. A. Rutenbar, K. A. Sakallah, sub-sat: a formulation for relaxed boolean satisfiability with applications in routing, IEEE Trans. on CAD of Integrated Circuits and Systems 22 (6) (2003) 814–820.

[12] T. Sandholm, An algorithm for optimal winner determination in combinatorial auctions, in: Proc. of IJCAI'99, 1999, pp. 542–547.

[13] F. Heras, J. Larrosa, S. de Givry, T. Schiex, 2006 and 2007 max-sat evaluations: Contributed instances, JSAT 4 (2-4) (2008) 239–250.

[14] C. Ansótegui, M. L. Bonet, J. Levy, Sat-based maxsat algorithms, Artif. Intell. 196 (2013) 77–105.

[15] A. Morgado, F. Heras, M. H. Liffiton, J. Planes, J. Marques-Silva, Iterative and core-guided maxsat solving: A survey and assessment, Constraints 18 (4) (2013) 478–534.

[16] Maxsat evaluations (2015).
URL `www.maxsat.udl.cat`

[17] C. Ansótegui, M. L. Bonet, J. Gabàs, J. Levy, Improving wpm2 for (weighted) partial maxsat, in: Proc. of CP'13, 2013, pp. 117–132.

[18] C. Ansótegui, J. Gabàs, J. Levy, Exploiting subproblem optimization in sat-based maxsat algorithms, J. Heuristics 22 (1) (2016) 1–53.

[19] F. Heras, J. Larrosa, A. Oliveras, MiniMaxSat: A new weighted Max-SAT solver, in: Proc. of SAT'07, 2007, pp. 41–55.

[20] H. Lin, K. Su, Exploiting inference rules to compute lower bounds for Max-SAT solving, in: Proc. of IJCAI'07, 2007, pp. 2334–2339.

[21] H. Lin, K. Su, C. M. Li, Within-problem learning for efficient lower bound computation in Max-SAT solving, in: Proc. of AAAI'08, 2008, pp. 351–356.

[22] C. M. Li, F. Manyà, N. O. Mohamedou, J. Planes, Exploiting cycle structures in Max-SAT, in: Proc. of SAT'09, 2009, pp. 467–480.

[23] A. Kügel, Improved exact solver for the weighted MAX-SAT problem, in: POS-10. Pragmatics of SAT, Edinburgh, UK, July 10, 2010, 2010, pp. 15–27.

[24] C. Ansótegui, M. L. Bonet, J. Levy, Solving (weighted) partial maxsat through satisfiability testing, in: Proc. of SAT'09, 2009, pp. 427–440.

[25] V. Manquinho, J. Marques-Silva, J. Planes, Algorithms for weighted boolean optimization, in: Proc. of SAT'09, 2009, pp. 495–508.

[26] C. Ansotegui, M. L. Bonet, J. Levy, A new algorithm for weighted partial maxsat, in: Proc. of AAAI'10, 2010, pp. 867–872.

[27] R. Martins, V. M. Manquinho, I. Lynce, Exploiting cardinality encodings in parallel maximum satisfiability, in: ICTAI, 2011, pp. 313–320.

[28] R. Martins, V. M. Manquinho, I. Lynce, Clause sharing in parallel maxsat, in: LION 12, 2012, pp. 455–460.

[29] C. Ansótegui, M. L. Bonet, J. Gabàs, J. Levy, Improving sat-based weighted maxsat solvers, in: Proc. of CP'12, 2012, pp. 86–101.

[30] R. Martins, S. Joshi, V. M. Manquinho, I. Lynce, Incremental cardinality constraints for maxsat, in: Proc. of CP'14, 2014, pp. 531–548.

[31] N. Eén, N. Sörensson, Translating pseudo-boolean constraints into SAT, JSAT 2 (1-4) (2006) 1–26.

[32] D. L. Berre, Sat4j, a satisfiability library for java, www.sat4j.org (2006).

[33] M. Koshimura, T. Zhang, H. Fujita, R. Hasegawa, Qmaxsat: A partial max-sat solver, JSAT 8 (1/2) (2012) 95–100.

[34] K. Honjyo, T. Tanjo, Shinmaxsat, a Weighted Partial Max-SAT solver inspired by MiniSat+, Information Science and Technology Center, Kobe University (2012).

[35] F. Heras, A. Morgado, J. Marques-Silva, Core-guided binary search algorithms for maximum satisfiability, in: Proc. of AAAI'11, 2011, pp. 36–41.

[36] A. Morgado, F. Heras, J. Marques-Silva, Improvements to core-guided binary search for maxsat, in: Proc. of SAT'12, 2012, pp. 284–297.

[37] N. Manthey, T. Philipp, P. Steinke, A more compact translation of pseudo-boolean constraints into CNF such that generalized arc consistency is maintained, in: KI 2014: Advances in Artificial Intelligence - 37th Annual German Conference on AI, Stuttgart, Germany, September 22-26, 2014. Proceedings, 2014, pp. 123–134.

[38] R. Asín, R. Nieuwenhuis, A. Oliveras, E. Rodríguez-Carbonell, Cardinality networks: a theoretical and empirical study, Constraints 16 (2) (2011) 195–221.

[39] B. Andres, B. Kaufmann, O. Matheis, T. Schaub, Unsatisfiability-based optimization in clasp, in: ICLP (Technical Communications), 2012, pp. 211–221.

[40] A. Morgado, C. Dodaro, J. Marques-Silva, Core-guided maxsat with soft cardinality constraints, in: Proc. of CP'14, 2014, pp. 564–573.

[41] N. Narodytska, F. Bacchus, Maximum satisfiability using core-guided maxsat resolution, in: Proc. of AAAI'14, 2014, pp. 2717–2723.

[42] C. Ansótegui, F. Manyà, Mapping problems with finite-domain variables to problems with boolean variables, in: Proc. of SAT'04, 2004, pp. 1–15.

[43] K. Pipatsrisawat, A. Darwiche, A lightweight component caching scheme for satisfiability solvers, in: Proc. of SAT'07, 2007, pp. 294–299.

[44] J. Larrosa, F. Heras, Resolution in max-sat and its relation to local consistency in weighted csps, in: Proc. of IJCAI'05, 2005, pp. 193–198.

[45] M. L. Bonet, J. Levy, F. Manyà, Resolution for max-sat, Artif. Intell. 171 (8-9) (2007) 606–618.

[46] C. Sinz, Towards an optimal CNF encoding of boolean cardinality constraints, in: Proc. of CP'05, 2005, pp. 827–831.

[47] C. Ansótegui, M. L. Bonet, J. Levy, On solving MaxSAT through SAT, in: Proc. of CCIA'09, 2009, pp. 284–292.

[48] J. Davies, F. Bacchus, Solving maxsat by solving a sequence of simpler sat instances, in: Proc. of CP'11, 2011, pp. 225–239.

[49] C. Ansotegui, M. L. Bonet, J. Levy, Solving (weighted) partial maxsat through satisfiability testing, in: Proc. of SAT'09, 2009, pp. 427–440.

[50] O. Bailleux, Y. Boufkhad, Efficient CNF encoding of boolean cardinality constraints, in: Proc. of CP'03, 2003, pp. 108–122.

[51] O. Bailleux, Y. Boufkhad, O. Roussel, New encodings of pseudo-boolean constraints into cnf, in: Proc. of SAT'09, 2009, pp. 181–194.

[52] C. Ansótegui, Y. Malitsky, M. Sellmann, Maxsat by improved instance-specific algorithm configuration, in: Proc. of AAAI'14, 2014, pp. 2594–2600.

[53] G. Audemard, L. Simon, Predicting learnt clauses quality in modern SAT solvers, in: Proc. of IJCAI'09, 2009, pp. 399–404.

[54] C. Ansótegui, J. Gabàs, Y. Malitsky, M. Sellmann, Maxsat by improved instance-specific algorithm configuration, Artif. Intell. 235 (2016) 26–39.

[55] M. Alviano, C. Dodaro, F. Ricca, A maxsat algorithm using cardinality constraints of bounded size, in: Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015, 2015, pp. 2677–2683.

[56] J. Davies, F. Bacchus, Exploiting the power of mip solvers in maxsat, in: Proc. of SAT'13, 2013, pp. 166–181.

[57] J. Davies, F. Bacchus, Postponing optimization to speed up MAXSAT solving, in: Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings, 2013, pp. 247–262.

[58] J. Berg, P. Saikko, M. Järvisalo, Improving the effectiveness of sat-based preprocessing for maxsat, in: Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015, 2015, pp. 239–245.

[59] C. Ansotegui, J. Gabas, Solving (weighted) partial maxsat with ilp, CPAIOR (2013) 403–409.

[60] J. Argelich, C. M. Li, F. Manyà, J. Planes, Maxsat evaluation, http://www.maxsat.udl.cat (2006-2015).

H

A new SAT encoding for PB constraints allowing to adjust the balance between size and arc-consistency

C. Ansótegui, J. Gabàs

Internal Technical Report

2016

A new SAT encoding for PB constraints allowing to adjust the balance between size and arc-consistency

Carlos Ansótegui and Joel Gabàs

DIEI, Univ. de Lleida
carlos@diei.udl.cat
joel.gabas@udl.cat

Technical Report 05-06-2016

Abstract. Many decision and optimization problems can be modelled and solved using Pseudo-Boolean (PB) constraints, i.e. linear inequalities $w_1 \cdot b_1, \ldots, w_n \cdot b_n \leq k$ with variables $b_i \rightarrow \{0, 1\}$. Although there are special formalisms for PB constraints, some of the problems have a predominant Boolean nature and the more reasonable choice is to translate them into a Boolean formula (SAT), usually into Conjunctive Normal Form (CNF). Many SAT encodings for PB constraints have been presented in the last decades. All of them have different properties of size and arc-consistency, i.e. propagation of variables. In terms of size, the best encoding has a complexity of $O(n \cdot \log(k))$ (5), but does not guarantee general arc-consistency. The best encoding guaranteeing general arc-consistency has a complexity of $O(n^2 \cdot \log(n)^2 \cdot log(w_{max}))$ (3). In this report, we present a new SAT encoding for PB constraints that guarantees general arc-consistency but with an exponential size in some particular cases. The conjecture is that, the encoding will maintain a reasonable size for a wide range of PB constraints appearing in real world problems. For the particular cases where the size explodes exponentially, we also present a method to limit it, while preserving partially its arc-consistency properties. This limit can even be set to $O(n \cdot \log(k))$. Our new encoding is an extension of the regular encoding for 1-cardinality constraints (1). Cardinality constraints are a special case of PB constraints with all w_i equal to 1 (the 1 before cardinality indicates the value of k). We describe the regular encoding for 1-cardinality constraints in Section 1 and how to extend it for k-cardinality constraints in Section 2. Finally, we explain how to extend the regular encoding for PB constraints in Section 3, where we also explain how to limit its size.

1 Regular SAT encoding for 1-cardinality At-Most constraints

In this section, we give a concise explanation of the regular SAT encoding for the 1-cardinality constraint that was originally presented in (1). We do this, not only for the sake of clarity and selfcontainment, but also to introduce the notation we will use in the rest of the paper. For the 1-cardinality constraint $(b_1 + \ldots + b_n = 1)$, the regular encoding can be written this way:

$$\begin{array}{lll}
\overline{b}_1 & \vee & \overline{r}_2 \\
\ldots & & \\
\overline{b}_{n-2} & \vee & \overline{r}_{n-1} \\
\overline{b}_{n-1} & \vee & \overline{r}_n \\
\\
\mathit{CNF}\ (\ r_2 & \leftrightarrow & b_2 \vee r_3) \\
\ldots & & \\
\mathit{CNF}\ (\ r_{n-1} & \leftrightarrow & b_{n-1} \vee r_n) \\
\mathit{CNF}\ (\ r_n & \leftrightarrow & b_n) \\
\\
b_1 \vee \ldots \vee b_n & &
\end{array}$$

The last clause corresponds to the At-Least constraint and all the other clauses correspond to the At-Most constraint. If we analyze carefully the regular variables r_i, we can easily see that $r_i \leftrightarrow b_i \vee \ldots \vee b_n$ (just replace the r_i variables from bottom to top in the clauses that begin with r_i). Thus, if the variable r_i is false, all the variables $b_i, \ldots, b_n$ are false. In this way, the At-Most encoding can also we written as follows:

$$\begin{array}{lll}
\overline{b}_1 & \vee & \overline{r}_2 \\
\ldots & & \\
\overline{b}_{n-2} & \vee & \overline{r}_{n-1} \\
\overline{b}_{n-1} & \vee & \overline{r}_n \\
\mathit{CNF}\ (\ \overline{r}_2 & \leftrightarrow & (b_2 + \ldots + b_n \leq 0)) \\
\ldots & & \\
\mathit{CNF}\ (\ \overline{r}_{n-1} & \leftrightarrow & (b_{n-1} + b_n \leq 0)) \\
\mathit{CNF}\ (\ \overline{r}_n & \leftrightarrow & (b_n \leq 0))
\end{array} \tag{1.1}$$

Let us briefly check that this formula only allows solutions with at most 1 b_i variable set to true. Just imagine that a variable b_m is set to true, then: from the clause that begins with $\overline{b}_n$, we have that the regular variable r_{m+1} must be false, and, from the clauses that begin with $\overline{r}_i \mid i \leq m$, we have that the regular variables $\overline{r}_i \mid i \leq m$ must be false. Since the regular variables $\overline{r}_i \mid i \leq m+1$ must be false: from the clause that begins with $\overline{r}_{m+1}$, we have that all $b_i \mid i \geq m+1$ must be false, and, from the clauses that begin with $\overline{b}_i \mid i \leq m-1$, we have that the variables $\overline{b}_i \mid \leq m-1$ must be false. Thus, if we set 1 b_i variable to true the encoding sets all the other b_i variables to false. In the next section, we are going to see that we can generalize this idea for k-cardinality constraints.

2 Regular SAT encoding for k-cardinality At-Most constraints

In this section we are going to see a new SAT encoding for k-cardinality At-Most constraints that we have conceived developing the idea of the regular encoding for the 1-cardinality constraint. First of all, let us extend the the formula 1.1 to the k-cardinality constraint ($b_1 + \ldots + b_n = k$):

$$\begin{array}{lll}
\overline{b}_1 & \vee & \overline{r}_2 \\
\ldots & & \\
\overline{b}_{n-k} & \vee & \overline{r}_{n-k+1} \\
CNF\ (\ \overline{r}_2 & \leftrightarrow & \sum_{i\in\{2,\ldots,n\}} b_i \le k-1\ ) \\
\ldots & & \\
CNF\ (\ \overline{r}_{n-k+1} & \leftrightarrow & \sum_{i\in\{n-k+1,\ldots,n\}} b_i \le k-1\ )
\end{array} \qquad (2.1)$$

Let us demonstrate that this encoding only accepts solutions with at most k b_i variables set to true. Imagine we have a set B^k of k b_i variables set to true and let us prove that all the other b_i variables must be set to false. First of all, let us call m the lower index of the variables in B^k. For the clause that begins with $\overline{b}_{m-1}$, the regular variable r_m is true since $\sum_{i\in\{m,\ldots,n\}} b_i \le k-1$ is false. Therefore the variable b_{m-1} must be false. The same holds true for all the clauses above the one that begins with $\overline{b}_{m-1}$, i.e., all the variables $\{b_i \mid i \le m-1\}$ must be false. On the other hand, for the clause that begins with $\overline{b}_m$, the regular variable r_{m+1} must be false, so we have that $\sum_{i\in\{m+1,\ldots,n\}} b_i \le k-1$ must be true. We know that $k-1$ of these variables are set to true, so the others must be false. Thus, if we set k b_m variables to true the encoding sets all the other b_i variables to false.

The formula 1.1 encodes a k-cardinality constraint and consists partially of $n-k$ $(k-1)$-cardinality constraints each one with different number of b_i variables. We are going to see that we can use the formula 1.1 recursively to build a SAT encoding. First of all, notice that if we have the encoding of $\sum_{i\in\{1,\ldots,n\}} b_i \le k$ and we want to encode $\sum_{i\in\{x,\ldots,n\}} b_i \le k$, we only have to select the clause that begins with b_x and the ones below:

$$\begin{array}{l}
\overline{b}_1 \vee \overline{r}_2 \\
\ldots \\
\overline{b}_{x-1} \vee \overline{r}_x \\
\boxed{\begin{array}{lll}
\overline{b}_x & \vee & \overline{r}_{x+1} \\
\ldots & & \\
\overline{b}_{n-k} & \vee & \overline{r}_{n-k+1} \\
CNF\ (\ \overline{r}_2 & \leftrightarrow & \sum_{i\in\{2,\ldots,n\}} b_i \le k-1\ ) \\
\ldots & & \\
CNF\ (\ \overline{r}_{n-k+1} & \leftrightarrow & \sum_{i\in\{n-k+1,\ldots,n\}} b_i \le k-1\ )
\end{array}}
\end{array}$$

Using this idea, we can replace in formula 2.1 the following part:

$$
\begin{array}{ll}
\overline{r}_2 & \leftrightarrow \sum_{i\in\{2,\dots,n\}} b_i \leq k-1 \\
\dots & \\
\overline{r}_{n-k+1} & \leftrightarrow \sum_{i\in\{n-k+1,\dots,n\}} b_i \leq k-1
\end{array}
$$

For this one:

$$
\begin{array}{ll}
\overline{r}_2 & \rightarrow \overline{r}_3 \\
\dots & \\
\overline{r}_{n-k} & \rightarrow \overline{r}_{n-k+1} \\
\overline{r}_2 & \leftrightarrow (\bar{b}_2 \vee \overline{r}'_3) \\
\dots & \\
\overline{r}_{n-k+1} & \leftrightarrow (\bar{b}_{n-k+1} \vee \overline{r}'_{n-k+2}) \\
\overline{r}'_3 & \leftrightarrow \sum_{i\in\{3,\dots,n\}} b_i \leq k-2 \\
\dots & \\
\overline{r}'_{n-k+2} & \leftrightarrow \sum_{i\in\{n-k+2,\dots,n\}} b_i \leq k-2
\end{array}
$$

Repeating this process recursively until the independent term of the new constraints is 1 and then applying the regular encoding for the 1-cardinality constraints, we get the following formula:

$$
\begin{array}{lllclll}
\bar{b}_1 & \vee & \overline{r}_2^{k-1} & & & & \\
\dots & & & & & & \\
\bar{b}_{n-k} & \vee & \overline{r}_{n-k+1}^{k-1} & & & & \\
\cline{1-3} \cline{5-7}
\overline{r}_2^{k-1} & \rightarrow & \overline{r}_3^{k-1} & \dots & \overline{r}_k^1 & \rightarrow & \overline{r}_{k+1}^1 \\
\dots & & & \dots & \dots & & \\
\overline{r}_{n-k}^{k-1} & \rightarrow & \overline{r}_{n-k+1}^{k-1} & \dots & \overline{r}_{n-2}^1 & \rightarrow & \overline{r}_{n-1}^1 \\
\overline{r}_2^{k-1} & \leftrightarrow & (\bar{b}_2 \vee \overline{r}_3^{k-2}) & \dots & \overline{r}_k^1 & \leftrightarrow & (\bar{b}_k \vee \overline{r}_{k+1}^0) \\
\dots & & & \dots & \dots & & \\
\overline{r}_{n-k+1}^{k-1} & \leftrightarrow & (\bar{b}_{n-k+1} \vee \overline{r}_{n-k+2}^{k-2}) & \dots & \overline{r}_{n-1}^1 & \leftrightarrow & (\bar{b}_{n-1} \vee \overline{r}_n^0) \\
\cline{1-3} \cline{5-7}
 & & & & r_{k+1}^0 & \leftrightarrow & b_{k+1} \vee r_{k+2}^0 \\
 & & & & \dots & & \\
 & & & & r_{n-1}^0 & \leftrightarrow & b_{n-1} \vee r_n^0 \\
 & & & & r_n^0 & \leftrightarrow & b_n
\end{array}
$$

That we can rewrite compactly changing the occurrences of r_i for $\overline{r}_i$:

$$
\begin{array}{lll}
 & \bigwedge_{i\in\{1,\dots,n-k\}} & \bar{b}_i \vee \overline{r}_{i+1}^{k-1} \\
\bigwedge_{j\in\{k-1,\dots,1\}} & \bigwedge_{i\in\{k+1-j,\dots,n-j-1\}} & \overline{r}_i^j \rightarrow \overline{r}_{i+1}^j \\
\bigwedge_{j\in\{k-1,\dots,1\}} & \bigwedge_{i\in\{k+1-j,\dots,n-j\}} & \overline{r}_i^j \leftrightarrow (\bar{b}_i \vee \overline{r}_{i+1}^{j-1}) \\
 & \bigwedge_{i\in\{k+1,\dots,n-1\}} & \overline{r}_i^0 \leftrightarrow \bar{b}_i \wedge \overline{r}_{i+1}^0 \\
 & & \overline{r}_n^0 \leftrightarrow \bar{b}_n
\end{array}
$$

Then replacing all occurrences of $\leftrightarrow$ for $\rightarrow$ [1] and rewriting the clauses $\overline{r}_i^0 \rightarrow (\overline{b}_i \wedge \overline{r}_{i+1}^0)$ as follows: $(\overline{r}_i^0 \rightarrow \overline{r}_{i+1}^0) \wedge (\overline{r}_i^0 \rightarrow \overline{b}_i)$, we get:

$$\begin{array}{lll}
 & \bigwedge_{i\in\{1,\dots,n-k\}} & \overline{b}_i \vee \overline{r}_{i+1}^{k-1} \\
\bigwedge_{j\in\{k-1,\dots,1\}} & \bigwedge_{i\in\{k+1-j,\dots,n-j-1\}} & \overline{r}_i^j \rightarrow \overline{r}_{i+1}^j \\
\bigwedge_{j\in\{k-1,\dots,1\}} & \bigwedge_{i\in\{k+1-j,\dots,n-j\}} & \overline{r}_i^j \rightarrow (\overline{b}_i \vee \overline{r}_{i+1}^{j-1}) \\
 & \bigwedge_{i\in\{k+1,\dots,n-1\}} & \overline{r}_i^0 \rightarrow \overline{r}_{i+1}^0 \\
 & \bigwedge_{i\in\{k+1,\dots,n-1\}} & \overline{r}_i^0 \rightarrow \overline{b}_i \\
 & & \overline{r}_n^0 \rightarrow \overline{b}_n
\end{array}$$

Which can be rewritten into *CNF* merging lines 4 with 2 and 6 with 5:

$$\begin{array}{lll}
 & \bigwedge_{i\in\{1,\dots,n-k\}} & \overline{b}_i \vee \overline{r}_{i+1}^{k-1} \\
\bigwedge_{j\in\{k-1,\dots,0\}} & \bigwedge_{i\in\{k+1-j,\dots,n-j-1\}} & r_i^j \vee \overline{r}_{i+1}^j \\
\bigwedge_{j\in\{k-1,\dots,1\}} & \bigwedge_{i\in\{k+1-j,\dots,n-j\}} & r_i^j \vee \overline{b}_i \vee \overline{r}_{i+1}^{j-1} \\
 & \bigwedge_{i\in\{k+1,\dots,n\}} & r_i^0 \vee \overline{b}_i
\end{array} \quad (2.2)$$

In fact, this encoding for k-cardinality At-Most constraints is almost the sequential counter described in (4) that we write here adapting the notation:

$$\begin{array}{lll}
 & \bigwedge_{i\in\{1,\dots,n-1\}} & \overline{b}_i \vee \overline{r}_{i+1}^{k-1} \\
\bigwedge_{j\in\{k-1,\dots,0\}} & \bigwedge_{i\in\{2,\dots,n-1\}} & r_i^j \vee \overline{r}_{i+1}^j \\
\bigwedge_{j\in\{k-1,\dots,1\}} & \bigwedge_{i\in\{2,\dots,n-1\}} & r_i^j \vee \overline{b}_i \vee \overline{r}_{i+1}^{j-1} \\
 & \bigwedge_{i\in\{2,\dots,n\}} & r_i^0 \vee \overline{b}_i
\end{array} \quad (2.3)$$

Our encoding, that also preserves arc-consistency, is a little bit more efficient in terms of the size of the formula. The number of clauses of the sequential counter encoding (2.3) is $O(n{\cdot}k)$, while the number of clauses of our new encoding (2.2) is $O(n \cdot (n-k))$.

3 Regular SAT encoding for PB At-Most constraints

We have seen in the previous section how can we generalize the regular encoding for k-cardinality At-Most constraints. We are going to see in this section that we can further generalize it for PB constraints. First of all, we are going to generalize formula (2.1) for for PB constraints:

$$\begin{array}{l}
\overline{b}_1 \quad \vee \ \overline{r}_2 \\
\dots \\
\overline{b}_{n-1} \quad \vee \ \overline{r}_n \\
\mathit{CNF}\ (\ \overline{r}_2 \leftrightarrow \sum_{i\in\{2,\dots,n\}} w_i \cdot b_i \le k - w_1\ ) \\
\dots \\
\mathit{CNF}\ (\ \overline{r}_n \leftrightarrow \sum_{i\in\{n\}} w_i \cdot b_i \le k - w_{n-1}\ )
\end{array} \quad (3.1)$$

[1] By doing so formula 2.1 still encodes a k-cardinality constraint.

To construct the formula 3.1, we have first made a naive transformation of the PB constraint into a k-cardinality constraint adding the variable b_i as many times as its weight w_i. Then, we have applied the formula 2.1, that for each variable b_i can be written as follows grouping again the variables with weights:

$$\begin{array}{lll}
\overline{b}_i & \vee & \overline{r}^{1}_{i+1} \\
\dots & & \\
\overline{b}_i & \vee & \overline{r}^{w_i}_{i+1} \\
CNF\ (\ \overline{r}^{1}_{i+1} & \leftrightarrow & (w_i - 1)\cdot b_i + \sum_{j\in\{i+1,\dots,n\}} w_i \cdot b_j \le k-1\ ) \\
\dots & & \\
CNF\ (\ \overline{r}^{w_i}_{i+1} & \leftrightarrow & 0\cdot b_i + \sum_{j\in\{i+1,\dots,n\}} w_i \cdot b_j \le k-1\ )
\end{array}$$

The new At-Most constraints only play a role when b_i is true. Otherwise, they can be false. The formula can be rewritten as follows:

$$\begin{array}{lll}
\overline{b}_i & \vee & \overline{r}^{1}_{i+1} \\
\dots & & \\
\overline{b}_i & \vee & \overline{r}^{w_i}_{i+1} \\
CNF\ (\ \overline{r}^{1}_{i+1} & \leftrightarrow & \sum_{j\in\{i+1,\dots,n\}} w_i \cdot b_j \le k-w_i\ ) \\
\dots & & \\
CNF\ (\ \overline{r}^{w_i}_{i+1} & \leftrightarrow & \sum_{j\in\{i+1,\dots,n\}} w_i \cdot b_j \le k-1\ )
\end{array}$$

We also know that if the first At-Most constraint (regular variable r^{1}_{i+1}) is true (false) all the other At-Most constraints (regular variables r^{j}_{i+1}) are also true (false). In fact, the formula can be rewritten as follows:

$$\begin{array}{lll}
\overline{b}_i & \vee & \overline{r}_{i+1} \\
CNF\ (\ \overline{r}_{i+1} & \leftrightarrow & \sum_{j\in\{i+1,\dots,n\}} w_i \cdot b_j \le k-w_i\ )
\end{array}$$

For b_n, in the non-trivial cases where $k \ge w_n$ the At-Most constraints (regular variable r_{n+1}) will be true (false) and the corresponding clauses can be skipped.

In order to construct the SAT encoding using formula 3.1, we need to construct recursively all the new PB constraints like we have made in the previous section for k-cardinality constraints. The encoding for k-cardinality constraints (2.2) can be written as follows:

$$\begin{array}{lll}
 & & \overline{r}^{k}_{1} \\
\bigwedge_{j\in\{k,\dots,0\}} & \bigwedge_{i\in\{k+1-j,\dots,n-j-1\}} & r^{j}_{i} \vee \overline{r}^{j}_{i+1} \\
\bigwedge_{j\in\{k,\dots,0\}} & \bigwedge_{i\in\{k+1-j,\dots,n-j\}} & r^{j}_{i} \vee \overline{b}_i \vee \overline{r}^{j-1}_{i+1} \\
 & \bigwedge_{i\in\{k+2,\dots,n+1\}} & r^{-1}_{i}
\end{array} \qquad (3.2)$$

Notice that $\overline{r}^{k'}_{i'}$ encodes the constraint $\sum_{i\in\{i',\dots,n\}} b_i \le k'$. In the case of the encoding for PB constraints, the new PB constraints are generated according to the different values of $k - w_i$ of encoding (3.1). Therefore, we cannot describe the encoding using a simple formula with indexes and we need an algorithm.

Algorithm 1: Encoding.

```
    Input: $B = \{b_1, \ldots, b_n\}$, $W = \{w_1, \ldots, w_n\}$, $k$
1:  $K = \{\langle k, 1\rangle\}$
2:  $\varphi = \{\overline{r}_1^k\}$
3:  $PB(B, W, K, \varphi)$
4:  return $\varphi$
```

Function PB(B, W, K, φ)

```
1:  $k_{max} := max(\{k\}_{\langle k,i\rangle \in K})$
2:  $i_{min} := min(\{i\}_{\langle k_{max},i\rangle \in K})$
3:  $K := \{\langle k, i\rangle\}_{\langle k,i\rangle \in K \wedge k \neq k_{max}}$
4:  for $i \in \{i_{min}, \ldots, |B|\}$ do
5:      if $\sum_{w_j \in W \wedge j \geq i} w_j > k_{max}$ then
6:          $k := k_{max} - w_i \mid w_i \in W$
7:          $\varphi := \varphi \cup \{(r_i^{k_{max}} \vee \overline{r}_{i+1}^{k_{max}})\}$
8:          $\varphi := \varphi \cup \{(r_i^{k_{max}} \vee \overline{b}_i \vee \overline{r}_{i+1}^{k})\}$
9:          if $k \geq 0$ then
10:             $K := K \cup \{\langle k, i+1\rangle\}$
11:         else
12:             $\varphi := \varphi \cup \{r_{i+1}^k\}$
13: if $|K| > 0$ then $PB(B, W, K, \varphi)$
```

Algorithm 1 gets as input the variables B, weights W and independent term k of the PB constraint and it returns a SAT formula φ (line 4) with the encoding. The SAT formula φ (line 2) is generated by calling function PB (line 3) that constructs the encoding (3.1) and calls itself recursively to include the encodings of the new PB constraints. It is called as many times as the number of values of k appearing in the new PB constraints. Each value of k has an associated index i indicating the first variable b_i of the larger PB constraint that needs to be constructed. Both numbers are stored together in the set K. Initially K contains a pair with the input independent term k and the index 1 (line 1).

Function PB has a main loop (PB lines 4-12) where the encoding (3.1) for given values of k and i is introduced to φ and the values of k and i of the new PB constraints are stored in K. Before the main loop, the values of k and i of the current PB constraint are searched in the set K. They correspond to the highest k (k_{max} in PB line 1) with a smallest i (i_{min} in PB line 2). Then, all values of k_{max} are removed from K (PB line 3). In the first call to the function, k_{max} and i_{min} are the input k and 1 respectively. After the main loop, the function is applied recursively for the new values of k and i stored in K (PB line 13). This values are lower than k_{max} and greater than or equal to 0. Thus, the function will be called at most as many times as the value of the input k plus 1.

The main loop of function PB (lines 4-12) gets through the indexes i of the variables b_i and weights w_i of the PB constraints with the current k_{max}.

From each i on, the PB constraint $\sum_{j\in\{i,\ldots,n\}} w_j \cdot b_j \leq k_{max}$ will be encoded, corresponding to $\overline{r}_i^{k_{max}}$. First of all it is checked if the PB constraint is a tautology (line 5), i.e. it will always be true if k_{max} is higher than or equal to $\sum_{j\in\{i,\ldots,n\}} w_j$. If this is the case, nothing is done until the next call to the function PB, Otherwise, the clauses corresponding to the values k_{max} and i are introduced into φ. These clauses (PB lines 7 and 8)[2] correspond to the two lines in the middle of encoding (3.2), but slightly modified. Namely, since we are encoding a PB constraint, the new value k for the regular variable is not $k_{max}-1$ but $k_{max}-w_i$ (PB line 6). If $k \geq 0$ (PB lines 9 and 10), the values k and $i+1$ are added into K to encode the corresponding PB constraint in subsequent calls to the function PB. If $k < 0$ (PB lines 11 and 12), the new PB constraint is always false and the clause r_{i+1}^k is introduced into φ. Let us see some examples of the resultant encoding.

Example 1. Given the PB constraint $8\dot{b}_1 + 4\cdot b_2 + 2\cdot b_3 + 1\cdot b_4 \leq k$:

(a) the regular SAT encoding for $k = 14$ is:

$$\begin{array}{lllll}
\overline{r}_1^{14}, & & & & \\
 & r_1^{14} \vee \overline{b}_1 \vee \overline{r}_2^{6}, & & & \\
 & & r_2^{6} \vee \overline{b}_2 \vee \overline{r}_3^{2}, & & \\
 & & & r_3^{2} \vee \overline{b}_3 \vee \overline{r}_4^{0}, & \\
 & & & & r_4^{0} \vee \overline{b}_4 \vee \overline{r}_5^{-1}, \\
r_5^{-1} & & & &
\end{array}$$

(b) the regular SAT encoding for $k = 9$ is:

$$\begin{array}{lll}
\overline{r}_1^{9}, & & \\
 & & r_2^{1} \vee \overline{r}_3^{1}, \\
 & r_1^{9} \vee \overline{b}_1 \vee \overline{r}_2^{1}, & \\
 & & r_2^{1} \vee \overline{b}_2 \vee \overline{r}_3^{-3}, \\
 & & r_3^{1} \vee \overline{b}_3 \vee \overline{r}_4^{-1}, \\
r_3^{-3},\ r_4^{-1} & &
\end{array}$$

(c) the regular SAT encoding for $k = 6$ is:

$$\begin{array}{llll}
\overline{r}_1^{6}, & & & \\
 & r_1^{6} \vee \overline{r}_2^{6}, & & \\
 & r_1^{6} \vee \overline{b}_1 \vee \overline{r}_2^{-2}, & & \\
 & r_2^{6} \vee \overline{b}_2 \vee \overline{r}_3^{2}, & & \\
 & & r_3^{2} \vee \overline{b}_3 \vee \overline{r}_4^{-1}, & \\
 & & & r_4^{0} \vee \overline{b}_4 \vee \overline{r}_5^{-1}, \\
r_2^{-2},\ r_5^{-1} & & &
\end{array}$$

[2] In order not to have an excessively complex pseudo-code, we did not express that the clause of line 7 is only needed if the PB constraint corresponding to $\overline{r}_{i+1}^{k_{max}}$ is going to be encoded in the following iteration, i.e. if $\sum_{w_j\in W \wedge j\geq i+1} w_j > k_{max}$.

The weighted version of the sequential counter described in (2) has a size in clauses of $O(n \cdot k)$. The problem is that k can be exponential with respect to n. In our encoding, not all the k need to be encoded being this size dramatically reduced for many PB constraints, included those whose diversity of weights is low, the number of clauses can be dramatically reduced even with the highest k. In the worst case however, for some specific configurations of weights, the number of clauses can also be exponential with respect to n.

3.1 How to limit the size

We can limit the size of the regular SAT encoding for PB constraints while preserving partially its arc-consistency properties. Remember that the encoding is built by constructing recursively the formula 3.1 for a PB constraint, where each $\overline{r}_i^k$ corresponds to a new PB constraint. This process is explained in Algorithm 1, where all the new PB constraints that have to be build are stored in the set K in form of pairs $\langle k, i \rangle$. To limit the size of the encoding φ, we can end this process when a certain number of clauses in φ and new PB constraints in K is exceeded. At this point, we can build all the new constraints with any other encoding with a lower size. The point is that, if we build this complementary encoding in a certain way, we can reuse it for all the constraints. For example, we can use the encoding described in (5) but merging the nodes like this: $w_n \cdot b_n \leq a_n, w_{n-1} \cdot b_{n-1} + a_n \leq a_{n-1}, \ldots, w_1 \cdot b_1 + a_2 \leq a_1$. For the new PB constraints in K, we only have to introduce to our encoding $\mathit{CNF}(r_i^k \leftrightarrow (a_i = k))$. This complementary encoding has a size of $O(n \cdot \log(k))$ so, whenever we decide to stop Algorithm 1, we will only need to add $O((|K| + n) \cdot log(k))$ clauses. So, we can always limit the size of our encoding to $O(n \cdot log(k))$.

Bibliography

[1] Ansótegui, C., Manyà, F.: Mapping problems with finite-domain variables to problems with boolean variables. In: Proc. of SAT'04. pp. 1–15 (2004)

[2] Hölldobler, S., Manthey, N., Steinke, P.: A compact encoding of pseudo-boolean constraints into SAT. In: KI 2012: Advances in Artificial Intelligence - 35th Annual German Conference on AI, Saarbrücken, Germany, September 24-27, 2012. Proceedings. pp. 107–118 (2012)

[3] Manthey, N., Philipp, T., Steinke, P.: A more compact translation of pseudo-boolean constraints into CNF such that generalized arc consistency is maintained. In: KI 2014: Advances in Artificial Intelligence - 37th Annual German Conference on AI, Stuttgart, Germany, September 22-26, 2014. Proceedings. pp. 123–134 (2014)

[4] Sinz, C.: Towards an optimal CNF encoding of boolean cardinality constraints. In: Proc. of CP'05. pp. 827–831 (2005)

[5] Warners, J.P.: A linear-time transformation of linear inequalities into conjunctive normal form. Inf. Process. Lett. 68(2), 63–69 (1998)