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/

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:

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

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, κ)
$(\vec{F}, s, t) \leftarrow$ Normalize(F)
$(k, C, S) \leftarrow$ Cluster $(T, \vec{F}, \kappa)$
for all $i = 1, \ldots, k$ **do**
  $P_i \leftarrow GGA(A, S_i)$
end for Return (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	1. **ISAC+2013**	1. ckmax-small	1. **ISAC+2013**
	2. **ISAC+2013**	2. WMaxSatz09	2. **ISAC+2013**	2. WMaxSatz09
	3. ckmax-small	3. WMaxSatz+	3. Maxsatz2013f	3. WMaxSatz+
Crafted	1. **ISAC+2013**	1. **ISAC+2013**	1. **ISAC+2013**	1. MaxHS
	2. Maxsatz2013f	2. ILP-2013	2. WMaxSatz+	2. **ISAC+2013**
	3. ckmax-small	3. scip-maxsat	3. WMaxSatz09	3. ILP-2013
Industrial	1. pmifumax	1. **ISAC+2013**	1. –	1. WPM1-2013
	2. WPM1-2011	2. QMaxSAT2-mt	2. –	2. **ISAC+2013**
	3. **ISAC+2013**	3. MSUnCore	3. –	3. WPM2-2013

Table 12
MSE-2013 three best solvers per subcategory (ordered by mean family ratio).

	MS	PMS	WMS	WPMS
Random	1. MaxSatz2013f	1. **ISAC+2013**	1. ckmax-small	1. **ISAC+2013**
	2. **ISAC+2013**	2. WMaxSatz09	2. **ISAC+2013**	2. WMaxSatz09
	3. ckmax-small	3. WMaxSatz+	3. Maxsatz2013f	3. WMaxSatz+
Crafted	1. ahmaxsat	1. **ISAC+2013**	1. **ISAC+2013**	1. MaxHS
	2. **ISAC+2013**	2. QMaxSAT-m	2. WMaxSatz+	2. **ISAC+2013**
	3. Maxsatz2013f	3. QMaxSAT2-mt	3. WMaxSatz09	3. ILP-2013
Industrial	1. pmifumax	1. **ISAC+2013**	1. –	1. **ISAC+2013**
	2. WPM1-2011	2. QMaxSAT2-mt	2. –	2. WPM1-2013
	3. optimax	3. WPM2-2013	3. –	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	1. ahmaxsat	1. CCLS2akms
	2. ahmaxsat	2. ahmaxsat_ls	2. ahmaxsat_ls
	3. CCLS2akms	3. **ISAC+2014**	3. **ISAC+2014**
Crafted	1. ahmaxsat_ls	1. **ISAC+2014**	1. **ISAC+2014**
	2. ahmaxsat	2. scip-maxsat	2. ILP-2013
	3. **ISAC+2014**	3. ILP-2013	3. MaxHS
Industrial	1. Open-WBO-In	1. **ISAC+2014**	1. Eva500aW
	2. clasp	2. Open-WBO-In	2. **ISAC+2014**
	3. Eva500a	3. Eva500a	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.