MaxSAT by Improved Instance-Specific Algorithm Configuration

Carlos Ansótegui
Department of Computer Science,
University of Lleida, Spain
carlos@diei.udl.es

Yuri Malitsky
Insight Centre for Data Analytics,
University College Cork, Ireland
yuri.malitsky@insight-centre.org

Meinolf Sellmann
IBM Watson Research Center,
Yorktown Heights, NY 10598, USA
meinolf@us.ibm.com

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 MaxSAT Evaluation where they won six of the eleven categories.

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 very recent revelation that traditional MIP solvers like Cplex can be extremely well suited for solving some families of partial MaxSAT instances (Ansotegui and Gabas 2013). Given that different solution approaches work well on different families of instances, (Matos et al. 2008) 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 (Argelich et al. 2012).

We describe the methodology that led to a MaxSAT portfolio that won six out of eleven categories at the 2013 MaxSAT Evaluation. 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 of our work on MaxSAT, we found a way to improve instance-specific algorithm configuration (ISAC) (Kadioglu et al. 2010) 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.

MaxSAT

Problem Definition Let us begin by formally stating the problem:

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* (WPMS) 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*. Here, a *hard* clause is one that must be satisfied, while also satisfying the maximum combined weight of *soft* clauses. A *Partial MaxSAT formula* (PMS) is a WPMS formula where the weights of soft clauses are equal. 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 $I : \text{var}(\varphi) \rightarrow \{0, 1\}$, 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 $\text{cost}(\varphi) = \min\{I(\varphi) \mid I : \text{var}(\varphi) \rightarrow \{0, 1\}\}$ 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.

Solvers Among the state-of-the-art MaxSAT solvers, we find two main approaches: branch-and-bound-based algorithms (Heras, Larrosa, and Oliveras 2007; Argelich and Manyà 2007; Darras et al. 2007; Lin, Su, and Li 2008; Ansotegui and Gabas 2013) and SAT-based solvers (Fu and Malik 2006; Marques-Silva and Planes 2007; Ansotegui, Bonet, and Levy 2009; Manquinho, Marques-Silva, and Planes 2009). From the annual results of the international MaxSAT Evaluation (Argelich et al. 2012), 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 (Matos et al. 2008) a MaxSAT portfolio was devised and tested in a promising but limited experimental evaluation. In particular, (Matos et al. 2008) 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., formulae 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, (Matos et al. 2008) 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 best knowledge this is the *only* study of MaxSAT portfolios. In particular, there has been no dominant algorithm portfolio in the annual MaxSAT Evaluations (Argelich et al. 2012).

We will devise instance-specific solvers for each of the MaxSAT domains. These will be based on algorithm tuning and algorithm portfolios. Therefore, in the next two sections we develop the technology that we will then later use to devise a novel solver for MaxSAT.

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 parametrisations of the same solver excel on different instance families. This is the underlying reason why algorithm portfolios have been so successful in SAT (Xu et al. 2008; Kadioglu et al. 2011), CP (O'Mahony et al. 2008), and QBF (Pulina and Tacchella 2007). 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 really stood-out was SATzilla-2007 (Xu et al. 2008). 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, more performant algorithm portfolio builders have been developed. For a while the trend was towards more highly biased regression and classification models (Silverthorn and Miikkulainen 2010). Then, the simple k-nearest neighbor (k-NN)-based portfolio 3S (Balint et al. 2012) 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 ((Kadioglu et al. 2011) 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 (Xu et al. 2012) 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 decisions of all trees are then aggregated and the solver with the highest score is used to solve the given instance. This portfolio clearly dominated the 2012 SAT Challenge (Competition 2012) where 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 (Malitsky et al. 2013). 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). CSHC was also shown to outperform both 3S and SATzilla-2012 and 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 the plethora of high-performance solvers. Often, though, there exists at least one solver that is highly parameterized. 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

Algorithm 1: Instance-Specific Algorithm Configuration

ISAC-Learn(A, T, F, κ)
$(\bar{F}, s, t) \leftarrow$ Normalize(F)
$(k, C, S) \leftarrow$ Cluster
$(T, \bar{F}, \kappa)$
for all $i = 1, \ldots, k$ **do**
  $P_i \leftarrow GGA(A, S_i)$
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 $A(x, P_i)$

number of automated algorithm configuration and parameter tuning approaches have been proposed over the last decade. These approaches range from gradient-free numerical optimization (Audet and Orban 2006), to gradient-based optimization (Coy et al. 2001), to iterative improvement techniques (Adenso-Diaz and Laguna 2006), to iterated local search techniques like ParamILS (Hutter et al. 2009), and to population-based local search approaches like the Gender-based Genetic Algorithm (GGA) (Ansotegui, Sellmann, and Tierney 2009).

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 (Xu, Hoos, and Leyton-Brown 2010), for example, uses the parameter tuner ParamILS (Hutter et al. 2009) to iteratively tune the solver and add parameterizations to a SATzilla portfolio that optimizes the final performance.

The ISAC Method

With the objective to boost performance in MaxSAT, we exploit an approach called Instance-Specific Algorithm Configuration (ISAC) (Kadioglu et al. 2010) 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 (Xu, Hoos, and Leyton-Brown 2010) on several standard benchmarks.

ISAC is an example of a low-bias approach. Unlike similar approaches, such as Hydra (Xu, Hoos, and Leyton-Brown 2010) and ArgoSmart (Nikolic, Maric, and Janici 2009), 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 data is therefore clustered into non-overlapping groups and a single solver is selected for each group based on some performance characteristic. Given a new instance, its features are computed and it is assigned to the nearest cluster. The instance is then solved by the solver assigned to that cluster.

More specifically, ISAC works as follows (see Algorithm 1). In the learning phase, ISAC is provided with a parameterized 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.

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*.

ISAC uses g-means (Hamerly and Elkan 2003) for clustering. 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 data set. In our case, we restrict clusters to have at least 50 instances. Beginning with the smallest cluster, the corresponding instances are redistributed 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.

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, this approach 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 (Kadioglu et al. 2010): 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 (Malitsky and Sellmann 2012) showed that cluster-based solver selection outperforms SATzilla-2009, this alone does not fully explain why ISAC often outperforms 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, hop-

ing 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 parametrisations, *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+).

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 (Xu, Hoos, and Leyton-Brown 2010) 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 (Thornton et al. 2008), rsaps (Hutter, Tompkins, and Hoos 2002), saps (Tompkins, Hutter, and Hoos 2007), agwsat0 (Wei, Li, and Zhang 2007b), agwsat+ (Wei, Li, and Zhang 2007c), agwsatp (Wei, Li, and Zhang 2007a), gnovelty+ (Pham and Gretton 2007), g2wsat (Li and Huang 2005), ranov (Pham and Anbulagan 2007), vw (Prestwich 2005), and anov09 (Hoos 2002). We augment these solvers by adding six fixed parameterizations of SATenstein 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, 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 1a 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 ISAC-GGA (using only SATenstein) solves only 52%. Using the whole set of solvers for tuning, ISAC-MSC solves about 56% of all instances, which is worse than always selecting the best base solver. Of course, we only know a posteriori that this parameterization of SATenstein is the best solver for this test set. However, ISAC's performance is still not convincing. By augmenting the approach using a final portfolio selection stage, we can boost performance. ISAC+ solves $\sim 60\%$ 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 for each instance which marks an upper bound on the performance we may realistically hope for.

Table 1: SAT Experiments

(a) 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

(b) 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

The second benchmark we present here is RAND. There are 581 test and 1141 train instances in this benchmark. In Table 1b we see that the best single solver (BS – gnovelty+) solves $\sim 84\%$ of the 581 instances in this test set. Hydra improves this to $\sim 91\%$, roughly equal in performance to ISAC-MSC. ISAC+ improves performance again and leads to almost 92% of all instances solved within the timelimit. 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 (Xu, Hoos, and Leyton-Brown 2010) and (Kadioglu et al. 2010). To verify this finding we also ran a comparison using untuned base solvers only. On the SAT RAND data set, for example, we find that CSHC using only 17 base solvers can only solve 520 instances, which is not competitive.

Maximum Satisfiability

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 cannot rely directly on the instance features used in (Matos et al. 2008) 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: avg, min, max, stdev). The remaining 32 features we use are a subset of the standard SAT features based on the entire formula, ignoring the 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. We also experimented with plain SAT features and found that this was not competitive compared to the proposed MaxSAT features above.

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 (Kuegel 2012), 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 (Max 2012). The solver also scored second place for crafted partial MaxSAT instances. Alternatively, solvers like ShinMaxSAT (Honjyo and Tanjo 2012) and sat4j (Berre 2006) 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 (Ansotegui, Bonet, and Levy 2009) or wbo1.6 (Manquinho, Marques-Silva, and Planes 2009) 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 (Koshimura et al. 2012) which is based on SAT. qMaxSAT searches for the optimum $\text{cost}(\varphi)$ from $k = \sum_{i=1}^{m} w_i$ to some value smaller than $\text{cost}(\varphi)$. 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 (Audemard and Simon 2012). The particular version of *qMaxSAT*, *qMaxSATg2*, that we use in our evaluation was the winner for the industrial partial MaxSAT category 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 (Argelich et al. 2012): 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.

Table 2: Fixed-Split MaxSAT

(a) 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

(b) 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

(c) 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+	38.3	126.4	895.9	285	95.0
VBS	43.3	43.3	43.3	300	100

We split our experiments into two parts. We first show the performance of ISAC+ on partial MaxSAT instances, crafted instances, industrial instances, and finally combining both. 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 awere 170 testing and 334 training. Our third dataset merged Crafted and Industrial instances and had 300 testing and 576 training instances.

The solvers we run on the the partial MaxSAT industrial and crafted instances were: 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 can be found in (Argelich et al. 2012).

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 2a 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 (Malitsky and Sellmann 2012) we incorporate also other high-performance MaxSAT solvers. Performance jumps, ISAC-MSC solves over 92% of all instances within our timelimit of 1,800 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 category, at the 2012 MaxSAT Evaluation the top five solvers were ranked just 20 instances apart, this improvement is very significant. In Table 2b 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 2c 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 1,800 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. However, the best performing approach is once more ISAC+ which solves 95% 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%.

Partial / Weighted MaxSAT The previous experiments were conducted on the particular train/test splits. In this section we conduct a 10-fold cross validation on the four categories of the 2012 MaxSAT Evaluation (Argelich et al. 2012). These are plain MaxSAT instances, weighted MaxSAT, partial MaxSAT, and weighted partial MaxSAT. The results of the cross validation are presented in Tables 3a – 3d. Specifically, each data set 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. Furthermore, all experiments were run with a 2,100 second timeout, on the same machines we used in the previous section. We use the 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 parameterized solver QMaxSat-g2.

The MS data set has 600 instances, split among random, crafted and industrial. Each fold has 60 test instances. Results in Table 3a confirm the findings observed in previous experiments. In this case, ISAC-MSC struggles to improve over the best single solver. At the same time ISAC+ 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 problems number 718.

Table 3: MaxSAT Cross-Validation

(a) 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

(b) 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

(c) 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

(d) 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

All in all, we observe 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 massively 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 of course differ from fold to fold) in the row indexed BS. Note this value is better than what the previous best single MaxSAT solver had to offer. Still, 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. This is a significant improvement in our ability to solve MaxSAT instances in practice.

These results were independently confirmed at the 2013 MaxSAT Evaluation where our portfolios, built based on the methodology described in this paper, won six out of eleven categories and came in second in another three.

Conclusion

We have introduced an improved instance-specific algorithm configurator by adding a portfolio stage to the existing ISAC approach. Extensive tests revealed that the new method consistently outperforms the best instance-specific configurators to date. The new method was then applied to partial MaxSAT, a domain where portfolios had never been used in a competitive setting. We devised a method to extend features originally designed for SAT to be exploited to help 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. Moreover, we conducted 10-fold cross validation in the four categories of the 2012 MaxSAT evaluation. Based on this work we entered our solvers in the 2013 MaxSAT Competition, where they won six out of eleven categories

and came second in another three. These results independently confirm that our solvers mark a significant step in solving weighted/partial MaxSAT instances efficiently.

Acknowledgments

Insight Centre is supported by SFI Grant SFI/12/RC/2289.

References

Adenso-Diaz, B., and Laguna, M. 2006. Fine-tuning of algorithms using fractional experimental design and local search. *Operations Research* 54(1):99–114.

Ansotegui, C., and Gabas, J. 2013. Solving (weighted) partial maxsat with ilp. *CPAIOR*.

Ansotegui, C.; Bonet, M.; and Levy, J. 2009. Solving (weighted) partial maxsat through satisfiability testing. *SAT* 427–440.

Ansotegui, C.; Sellmann, M.; and Tierney, K. 2009. A gender-based genetic algorithm for the automatic configuration of algorithms. *CP* 142–157.

Argelich, J., and Manyà, F. 2007. Partial Max-SAT solvers with clause learning. *SAT* 28–40.

Argelich, J.; Li, C.; Manyà, F.; and Planes, J. 2012. Maxsat evaluations. www.maxsat.udl.cat.

Audemard, G., and Simon, L. 2012. glucose.

Audet, C., and Orban, D. 2006. Finding optimal algorithmic parameters using derivative-free optimization. *SIAM J. on Optimization* 17(3):642–664.

Balint, A.; Belov, A.; Diepold, D.; Gerber, S.; Järvisalo, M.; and Sinz, C., eds. 2012. *Proceedings of SAT Challenge 2012: Solver and Benchmark Descriptions*, volume B-2012-2 of *Department of Computer Science Series of Publications B*. University of Helsinki.

Berre, D. 2006. Sat4j, a satisfiability library for Java. www.sat4j.org.

Competition, S. 2012. www.satcomptition.org.

Coy, S.; Golden, B.; Runger, G.; and Wasil, E. 2001. Using experimental design to find effective parameter settings for heuristics. *Journal of Heuristics* 7:77–97.

Darras, S.; Dequen, G.; Devendeville, L.; and Li, C. 2007. On inconsistent clause-subsets for Max-SAT solving. *CP* 225–240.

Fu, Z., and Malik, S. 2006. On solving the partial max-sat problem. *SAT* 252–265.

Hamerly, G., and Elkan, C. 2003. Learning the k in k-means. *NIPS*.

Heras, F.; Larrosa, J.; and Oliveras, A. 2007. Minimaxsat: A new weighted max-sat solver. *SAT* 41–55.

Honjyo, K., and Tanjo, T. 2012. Shinmaxsat.

Hoos, H. 2002. Adaptive novelty+: Novelty+ with adaptive noise. *AAAI*.

Hoos, H. 2012. *Autonomous Search*. Springer Verlag.

Hutter, F.; Hoos, H.; Leyton-Brown, K.; and Stuetzle, T. 2009. Paramils: An automatic algorithm configuration framework. *JAIR* 36:267–306.

Hutter, F.; Tompkins, D.; and Hoos, H. 2002. Rsaps: Reactive scaling and probabilistic smoothing. *CP*.

Kadioglu, S.; Malitsky, Y.; Sellmann, M.; and Tierney, K. 2010. ISAC – instance-specific algorithm configuration. *ECAI* 751–756.

Kadioglu, S.; Malitsky, Y.; Sabharwal, A.; Samulowitz, H.; and Sellmann, M. 2011. Algorithm selection and scheduling. *CP* 454–469.

Koshimura, M.; Zhang, T.; Fujita, H.; and Hasegawa, R. 2012. Qmaxsat: A partial max-sat solver. *JSAT* 8(1/2):95–100.

Kuegel, A. 2012. Improved exact solver for the weighted max-sat problem. *POS-10* 8:15–27.

Li, C., and Huang, W. 2005. G2wsat: Gradient-based greedy walksat. *SAT* 3569:158–172.

Lin, H.; Su, K.; and Li, C. 2008. Within-problem learning for efficient lower bound computation in Max-SAT solving. *AAAI* 351–356.

Malitsky, Y., and Sellmann, M. 2012. Instance-specific algorithm configuration as a method for non-model-based portfolio generation. *CPAIOR* 244–259.

Malitsky, Y.; Sabharwal, A.; Samulowitz, H.; and Sellmann, M. 2013. Algorithm portfolios based on cost-sensitive hierarchical clustering. *IJCAI*.

Manquinho, V.; Marques-Silva, J.; Planes, J.; and Martins, R. 2012. wbo1.6.

Manquinho, V.; Marques-Silva, J.; and Planes, J. 2009. Algorithms for weighted boolean optimization. *SAT* 495–508.

Marques-Silva, J., and Planes, J. 2007. On using unsatisfiability for solving maximum satisfiability. *CoRR* abs/0712.1097.

Matos, P.; Planes, J.; Letombe, F.; and Marques-Silva, J. 2008. A max-sat algorithm portfolio. *ECAI* 911–912.

2012. Maxsat evaluation.

Nikolic, M.; Maric, F.; and Janici, P. 2009. Instance based selection of policies for sat solvers. *SAT* 326–340.

O'Mahony, E.; Hebrard, E.; Holland, A.; Nugent, C.; and O'Sullivan, B. 2008. Using case-based reasoning in an algorithm portfolio for constraint solving. *AICS*.

Pham, D., and Anbulagan. 2007. ranov. solver description. SAT Competition.

Pham, D., and Gretton, C. 2007. gnovelty+. solver description. SAT Competition.

Prestwich, S. 2005. Vw: Variable weighting scheme. *SAT*.

Pulina, L., and Tacchella, A. 2007. A multi-engine solver for quantified boolean formulas. *CP* 574–589.

Silverthorn, B., and Miikkulainen, R. 2010. Latent class models for algorithm portfolio methods. *AAAI*.

Thornton, J.; Pham, D.; Bain, S.; and Ferreira, V. 2008. Additive versus multiplicative clause weighting for sat. *PRICAI* 405–416.

Tompkins, D.; Hutter, F.; and Hoos, H. 2007. saps. solver description. SAT Competition.

Wei, W.; Li, C.; and Zhang, H. 2007a. adaptg2wsatp. solver description. SAT Competition.

Wei, W.; Li, C.; and Zhang, H. 2007b. Combining adaptive noise and promising decreasing variables in local search for sat. solver description. SAT Competition.

Wei, W.; Li, C.; and Zhang, H. 2007c. Deterministic and random selection of variables in local search for sat. solver description. SAT Competition.

Xu, L.; Hutter, F.; Hoos, H.; and Leyton-Brown, K. 2008. Satzilla: portfolio-based algorithm selection for sat. *JAIR* 32(1):565–606.

Xu, L.; Hutter, F.; Shen, J.; Hoos, H.; and Leyton-Brown, K. 2012. Satzilla2012: Improved algorithm selection based on cost-sensitive classification models. SAT Competition.

Xu, L.; Hoos, H.; and Leyton-Brown, K. 2010. Hydra: Automatically configuring algorithms for portfolio-based selection. *AAAI*.