Instance-Specific Algorithm Configuration as a Method for Non-Model-Based Portfolio Generation

Yuri Malitsky[1] and Meinolf Sellmann[2]

[1] Brown University, Department of Computer Science
ynm@cs.brown.edu
[2] IBM Research Watson
meinolf@us.ibm.com

Abstract. Instance-specific algorithm configuration generalizes both instance-oblivious algorithm tuning as well as algorithm portfolio generation. ISAC is a recently proposed non-model-based approach for tuning solver parameters dependent on the specific instance that needs to be solved. While ISAC has been compared with instance-oblivious algorithm tuning systems before, to date a comparison with portfolio generators and other instance-specific algorithm configurators is crucially missing. In this paper, among others, we provide a comparison with SATzilla, as well as three other algorithm configurators: Hydra, DCM and ArgoSmart. Our experimental comparison shows that non-model-based ISAC significantly outperforms prior state-of-the-art algorithm selectors and configurators. The following study was the foundation for the best sequential portfolio at the 2011 SAT Competition.

1 Introduction

The constraint programming (CP) and satisfiability (SAT) communities have a long tradition in research on algorithm selection and algorithm portfolios [8,18,21,35,27]. Portfolio-based solution approaches have been shown to boost solver performance when tackling hard satisfaction problems. Both in SAT and CP, portfolio-based solvers such as SATzilla [35] and CP-Hydra [21] have dominated solver competitions in the past. For an overview of the state-of-the-art in portfolio generation, see the thorough survey by [26].

In an orthogonal effort to develop ever more efficient solvers, both CP and SAT have also forced the development of new algorithm configurators, meta-algorithms that tune the parameters of SAT and CP solvers [15,2,20,16]. Most recently, general-purpose algorithm configurators have been developed that choose the parameters of a solver based on the particular instance that needs solving. It is a fair assessment that these instance-specific algorithm configurators unify both the ideas of algorithm tuning and algorithm portfolios.

The most recent configurators published are Hydra [33] (not to be mistaken with CP-Hydra [21]!) and ISAC [16]. Since Hydra and ISAC have been invented at the same time, hitherto a comparison of the two methods is crucially missing. Hydra is explicitly based on the SATzilla solver selection methodology, whereas ISAC uses a proprietary

N. Beldiceanu, N. Jussien, and É. Pinson (Eds.): CPAIOR 2012, LNCS 7298, pp. 244–259, 2012.

ISAC-Learn(A, T, F)
$(\bar{F}, s, t) \leftarrow$ Normalize(F)
$(k, C, S) \leftarrow$ Cluster $(T, \bar{F})$
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, s, t)
$f \leftarrow$ Features(x)
$\bar{f}_i \leftarrow (f_i - t_i)/s_i \; \forall \; i$
$Q \leftarrow P_1, m \leftarrow ||\bar{f} - C_1||$
for all $j = 2, \ldots, k$ **do**
    if $||\bar{f} - C_j|| < m$ **then**
        $Q \leftarrow P_j, m \leftarrow ||\bar{f} - C_j||$
    end if
end for
return $A(x, Q)$

Algorithm 1: Instance-Specific Algorithm Configuration

non-model-based selection approach that is based on a combination of supervised and un-supervised machine learning.

Our objective is to provide an empirical comparison of the ISAC methodology with existing algorithm tuners for SAT. We will first review ISAC and then provide an extensive empirical comparison with other approaches. We will show that, although ISAC was not developed to be used as an algorithm selector, it is able to significantly outperform highly efficient SAT solver selectors. Moreover, ISAC also improves the performance of parameterized solvers that were tuned with Hydra.

2 ISAC

Let us begin by reviewing instance-specific algorithm configuration. Most algorithms, and especially most combinatorial solvers, have parameters. These can be implicit parameters (thresholds, specific choices of subroutines, etc) that the programmer set when implementing the algorithm. Or, they can be explicit parameters, whose semantics are explained to the user in a user manual or other documentation, and which are left for the user to set when invoking the algorithm.

Both types of parameters are problematic. Implicit parameters are not tailored for the particular inputs that the algorithm is commonly facing since the algorithm designer often has no knowledge of the particular instance distributions that his program will be used for later. The user, on the other hand, should not have to become an expert on the internals of an algorithm in order to use it efficiently. Moreover, even for experts it is a very hard and time-consuming task to tune an algorithm effectively.

Consequently, it has been proposed that algorithms ought to be tuned automatically (see, e.g., [1,15,2]). An automatic system for tuning algorithms allows the user to provide a representative set of inputs to the algorithm and thereby to adapt the algorithm to his or her individual needs.

Recently, an even more ambitious configuration method was proposed, namely to tune an algorithm so that it can classify an input by itself and choose the most promising parameters for that particular instance automatically. This is the motivation behind instance-specific algorithm configuration (ISAC) [16]. Methods with the same objective had been proposed earlier (see, e.g., [13]). ISAC addressed issues like the interdependence of parameters and, by design, addresses the fact that parameter settings need to be pre-stabilized to work well in practice.

Interestingly, ISAC is not based on a model-based machine learning technique. Instead, it works with a low learning bias by invoking an approach that is based on unsupervised learning, in particular the clustering of inputs. Particularly, ISAC works in two phases, the learning phase, and the runtime phase (see Algorithm 1). In the learning phase, we are provided with the parameterized solver A, a list of training instances T, and their corresponding feature vectors F. First, we normalize the features in the training set so that the values for each feature span the interval $[-1, 1]$. We memorize the corresponding scaling and translation values (s, t) for each feature.

Then, we use the g-means [9] algorithm to cluster the training instances based on the normalized feature vectors. The final result of the clustering is a number of k clusters S_i, and a list of cluster centers C_i. For each cluster of instances S_i we compute favorable parameters P_i by employing the instance-oblivious tuning algorithm. While any parameter tuner can be used, we employ GGA [2], which can handle categorical, discreet and continuous parameters, can be parallelized, and employs racing to further reduce tuning time. In this scenario, unlike alternate approaches like k-nearest neighbor, clustering allows us to tune solvers offline.

In the second phase, when running algorithm A on an input instance x, we first compute the features of the input and normalize them using the previously stored scaling and translation values for each feature. Then, we determine the cluster with the nearest center to the normalized feature vector. Finally, we run algorithm A on x using the parameters for this cluster.

The important aspect to note here is that ISAC works without a learning model and has very low learning bias. The clustering of training instances is unsupervised, and the assignment of a test instance to a cluster, i.e. the selection of a set of parameters, is purely based on the shortest normalized Euclidean distance to the nearest cluster center.

3 Algorithm Configuration for Algorithm Selection

ISAC is a generalization of instance-oblivious configurators such as ParamILS [15] or GGA [2]. Interestingly, as we just noted, it can also be viewed as a generalization of algorithm selectors such as SATzilla [35].

Provided with a set of (usually very few, five to fifteen) solvers, SATzilla predicts the (expected/median, penalized, logarithmic) runtime of all its constituent solvers on a given instance as characterized by some pre-defined set of input features. At runtime, it then chooses the solver that has the lowest predicted (penalized) runtime. The core problem that SATzilla needs to address is therefore how to predict the runtime of a solver for a given instance. To make this prediction, SATzilla trains an empirical hardness model of all its constituent solvers based on the times that these solvers take on instances in a given training set.

3.1 Model-Based Solver Selection

This prediction model assumes that the (expected/median, logarithmic, penalized) runtime of a solver is a linear function over the instance features. That is, to facilitate the learning process, SATzilla introduces a learning bias, an assumption that limits the parameters of the function that needs to be learned, at the cost of being able to express more complex functions between instance features and runtime.

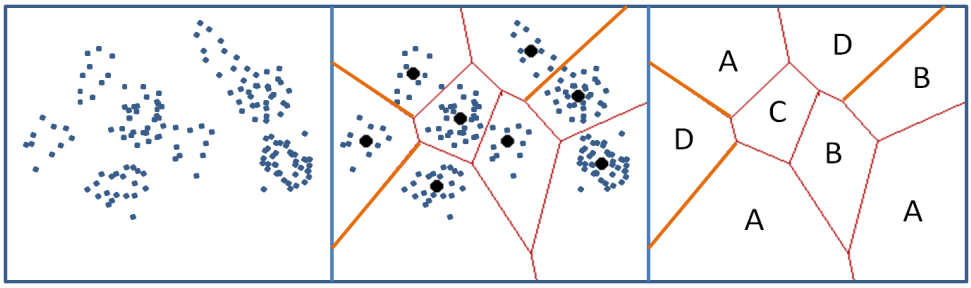

Fig. 1. Clustering of Instances in a 2-Dimensional Feature Space. Un-clustered instances on the left. In the middle we show the clustered instances with cluster centers and the corresponding partitioning of the feature space into Voronoi cells. On the right we show a hypothetical assignment of algorithms A–D to the clusters.

The problem with such a learning bias is that it forces us to learn functions that can be arbitrarily far from reality. Note that, as a direct consequence of this bias, along *any* line in the feature space no solver can define the minimum runtime in two disconnected intervals! This problem could be alleviated by adding more dimensions (i.e., features), or by redefining features. For example, using the clause over variable ratio as a feature for linear regression means that we must predict an increasing or decreasing runtime as this ratio grows. We could use the distance from the critical threshold of 4.27 instead, which is more likely to have a monotonic relation with runtime.

SATzilla addresses this problem by starting with 48 core SAT features. Then, using feedforward selection, it incrementally selects the features that are most important for the runtime prediction of a solver. After a base set of features is chosen, a binomial combination of the features is created and subsequently filtered. This process of feature selection and combination is iterated multiple times. SATzilla shows that this works well in practice. However, due to the greedy nature of the employed filtering, it is possible that some features are prematurely discarded in order to fit the linear prediction function – especially since the "right" features may very well be solver-dependent. We conjecture that this may in fact be the reason why, despite its success in SAT, the SATzilla methodology has, to our knowledge, not been applied to any other domains.

Consider the following Gedankenexperiment. Assume we have a number of algorithms A_i, and each has a runtime that can be accurately predicted as $\text{time}_{A_i}(F) = e^{\beta_{A_i}^T F}$ for algorithm A_i and an input with features F. Now, we build a portfolio P of these solvers. The portfolio P is, of course, itself a solver, and its runtime distribution is $\text{time}_P(F) = \alpha + \min_i e^{\beta_{A_i}^T F}$, where α is the time needed to decide which solver should be invoked. Now, it is easy to see that in general there is no β_P such that $e^{\beta_P^T F} = \alpha + \min_i e^{\beta_{A_i}^T F}$ – simply because a piecewise linear concave function cannot be approximated well with a linear function. This shows that, in general, we cannot assume that the logarithmic runtime distribution of an arbitrary solver can be expressed accurately as a linear function over input features.

3.2 Non-model Based Solver Selection

On the other hand, consider the idea of ISAC to cluster instances and to handle similar instances with the same solver. In fact, the clusters define Voronoi cells and each

instance whose normalized feature vector falls into a cell is solved with the corresponding solver. In the example in Figure 1 note that solver A is assigned to multiple disconnected clusters that intersect the same line. Such an assignment of instances to solvers is impossible for the linear regression-based approach. In fact, clustering allows us, at least in principle, to handle *all* continuous runtime distributions of algorithms as the continuity of runtime will result in the same solver to define the minimum runtime in an entire neighborhood. Moreover, assuming continuity of the runtime distributions, Analysis 101 tells us that, at the borders where one solver begins to outperform another, we may assume that the optimal solver only slightly outperforms the other. Consequently, the ISAC approach is somewhat fault-tolerant.

3.3 Using ISAC as Portfolio Generator

With this motivation, we intend to use ISAC to generate a portfolio of SAT solvers. We consider three ways, of differing complexity, of how we can use the ISAC methodology to obtain a portfolio of algorithms. Assume, we are given a set of (potentially parameterized) algorithms $A_1, \ldots, A_n$, a set of training inputs T, the set of associated feature vectors F, and a function $\mathrm{Features}$ that returns a feature vector for any given input x.

- **Pure Solver Portfolio:** Cluster the training instances according to their normalized features. For each cluster, determine the overall best algorithm. At runtime, determine the closest cluster and tackle the input with the corresponding solver.
- **Optimized Solver Portfolio:** Proceed as before. For each cluster, instance-obliviously tune the preferred algorithm for the instances within that cluster.
- **Instance-specific Meta-Solver Configuration:** Define a parameterized meta-algorithm where the first parameter determines which solver is invoked, and the remaining parameters determine the parameters for the underlying solvers. Use ISAC to tune this solver.

The difference between the pure solver portfolio and the other two approaches is that the first is limited to using the solvers with their default parameters. This means that the performance that can be achieved maximally is limited by that of the "virtually best solver" (a term used in the SAT competition) which gives the runtime of the best solver (with default parameters) for each respective instance. The difference between the optimized solver portfolio and the instance-specific meta-solver configuration is that the latter may find that a solver that is specifically tuned for a particular cluster of instances may work better overall than the best default solver for that cluster, even if the latter is tuned. Therefore, note that the potential for performance gains strictly increases from stage-to-stage. For the optimized solver portfolio as well as instance-specific meta-solver configuration it is therefore possible to outperform the virtually best solver.

In the following, we will test these three approaches and compare them with several portfolio generators for SAT.

4 Algorithm Configuration vs. Algorithm Selection of SAT Solvers

We begin our experimental study by comparing ISAC with the SATzilla_R portfolio. To this end we generate portfolios based on the following solvers: Ag2wsat0 [30], Ag2wsat+ [31], gnovelty+ [23], Kcnfs04 [7], March_dl04 [10], Picosat 8.46 [4], and SATenstein [17]. Note that these solvers are *identical* to the ones that the SATzilla09_R [34] solver was based on when it was entered in the 2009 SAT solver competition.[1] To make the comparisons as fair as possible, we use the same feature computation program made public by the developers of SATzilla to get the 48 core features to characterize a SAT input instance (see [35] for a detailed list of features).

Our training set is comprised of the random instances from the 2002-2007 SAT Competitions, whereby we remove instances that are solved in under a second by all the solvers in our portfolio or that cannot be solved with any of the solvers within a time limit of 1,200 seconds (this is the same time-out as used in phase 1 of the SAT Competition). This leaves 1,582 training instances. As test set, we use the 570 instances from the 2009 SAT Solver Competition [25] where SATzilla_R won gold. SATzilla_R is the version of SATzilla tuned specifically for random SAT instances. We choose the random instances because this was the category where SATzilla showed the most marked improvements over the competing approaches. We trained our cluster-based approaches on dual Intel Xeon 5540 (2.53 GHz) quad-core Nehalem processors and 24 GB of DDR-3 memory (1333 GHz).

Like SATzilla, we train for the PAR10 score, a penalized average of the runtimes: For each instance that is solved within 1,200 seconds, the actual runtime in seconds defines the penalty for that instance. For each instance that is not solved within the time limit, the penalty is set to 12,000, which is 10 times the original timeout. Note that, for our pure solver portfolio, we require much less time than SATzilla to generate the portfolio. Clustering takes negligible time compared to running the instances on the various solvers. However, when determining the best solver for each cluster, we can race them against each other, which means that the total CPU time for each cluster is the number of solvers times the time taken by the fastest solver (as opposed to the total time of running all solvers on all instances).

For the optimized solver portfolio generator and our meta-solver configurator, on each cluster of training instances we employ the instance-oblivious configurator GGA. In the following we give details on the parameters used for GGA to ensure reproducibility of our results, please refer to [2] for details: We use GGA with the standard population size of 100 genomes, split evenly between the competitive and noncompetitive groups. Initial tournaments consider five randomly chosen training instances. The size of this random subset grows linearly with each iteration until the entire training set is included by iteration 75. GGA then proceeds tuning until the 100th generation or until no further improvement in performance is observed.

For the meta-solver, parameters needed to be trained for 16 clusters. These clusters were built to have at least 50 instances each, which resulted in the average cluster having 99 instances and the largest cluster having 253 instances. In total, building the MSC required 260 CPU days of computation time. However, since each of the clusters could be tuned independently in parallel, only 14 days of tuning were required.

[1] Note that the benchmark for a portfolio generator consists in both the train and test sets of problem instances as well as the solvers used to build the portfolio!

4.1 Pure Solver Portfolio vs. SATzilla

We report the results of our experiments in Table 1. As a pure solver, gnovelty+ performs best in our experiments, it solves 51% of all test instances in time, whereby a number of the other solvers exhibits similar performance. Even though no individual solver can do better, when they join forces in a portfolio we can significantly boost performance further, as the pioneers of this research thread had pointed out [8,18]. SATzilla_R, for example, solves 71% of the test instances within the given captime of 1,200 seconds. Seeing that the virtually best solver (VBS) sets a hard limit of 80% of instances that can be solved in time by these solvers, this performance improvement is very significant: more than two thirds of the gap between the best pure solver and the VBS is closed by the SATzilla portfolio. Here, VBS assumes an oracle based portfolio approach that for each instance, always chooses the fastest solver.

The table also shows the performance of the various portfolios based on ISAC. We observe that on the data SATzilla_R was trained on, it performs very well, but on the actual test data even the simple pure solver portfolio generated by ISAC manages to outperform SATzilla_R. On the test set, the pure solver portfolio has a slightly better PAR10 score (the measure that SATzilla was trained for), and it solves a few more instances (408 compared to 405) within the given time limit.

In Table 1, under 'Cluster' we also give the best PAR10 score, average runtime, and number of solved instances *when we commit ourselves to using the same solver for each cluster*. We observe that the clustering itself already incurs some cost in performance when compared to the VBS.

Table 1. Comparison of SATzilla, the pure solver portfolio (PSP), the instance-specific meta-solver configuration (MSC), and the virtually best solver (VBS). We also show the best possible performance that can be achieved if we are forced to use the same solver for all instances in the same cluster (Cluster). The last columns show the performance of the meta-solver configuration with a pre-solver (MSC+pre). For the penalized and regular average of the time, we also present σ, the standard deviation.

Solver	gnovelty+	SATzilla_R	PSP	Cluster	MSC	VBS	MSC+pre
	Training Data Set						
PAR10	4828	685	1234	1234	505	58.6	456
σ	5846	2584	3504	3504	2189	156	2051
Ave	520	153	203	203	129	58.6	128
σ	574	301	501	501	267	156	261
Solved	951	1504	1431	1431	1527	1582	1534
%	60.1	95.1	90.5	90.5	96.5	100	97.0
	Testing Data Set						
PAR10	5874	3578	3569	3482	3258	2482	2480
σ	5963	5384	5322	5307	5187	4742	4717
Ave	626	452	500	470	452	341	357
σ	578	522	501	508	496	474	465
Solved	293	405	408	411	422	457	458
%	51.4	71.1	71.6	72.1	74.0	80.2	80.4

4.2 Meta-solver Configuration vs. SATzilla

Now we are interested in testing whether parameter tuning can help improve performance further. When considering the optimized solver portfolio, where we tune the best solver for each cluster, we found that none of the best solvers chosen for each cluster had parameters. Therefore, the performance of the OSP is identical to that of the PSP. The situation changes when we use the meta-solver configuration approach.

As Table 1 shows, the MSC provides a significant additional boost in test performance. This portfolio manages to solve 74% of all instances within the time limit, 17 instances more than SATzilla. When analyzing this result, we found that this improvement over the VBS is due to the introduction of two new configurations of SATenstein that MSC tuned and assigned to two clusters.

Studying SATzilla again, we found that the portfolio is not actually a pure algorithm selector. In the first minute, SATzilla employs both the mxc-sr08 [5] SAT solver and a specific parameterization of SATenstein. That is, SATzilla runs a schedule of three different solvers for each instance. We were curious to see whether we could boost the performance of our MSC portfolio further by running the same two solvers as SATzilla for the first minute. In Table 1, the corresponding portfolio is denoted by MSC+pre. We observe that the new portfolio now even outperforms the VBS. This is possible because the MSC has added new parameterizations of SATenstein, and also because mxc-sr08 is not one of our pure solvers. As a result, the new portfolio now solves 80% of all competition instances, 9% more than SATzilla. At the same time, runtime variance is also greatly reduced: Not only does the new portfolio run more efficiently, it also works more robustly. Seeing that ISAC was originally not developed with the intent to craft solver portfolios, this performance improvement over a portfolio approach that has dominated SAT competitions for half a decade is significant. Based on these results, we entered the 3S Solver Portfolio in the 2011 SAT Competition. 3S is just one portfolio (no subversions _R or _I) for all different categories which comprises 36 different SAT solvers. 3S was the first sequential portfolio that won gold in more than one main category (SAT+UNSAT instances).

Although we do not explicitly show it, all the results are significant as per the Wilcoxon signed rank test with continuity correction. MSC is faster than SATzilla_R with $p \leq 0.1\%$.

4.3 Improved Algorithm Selection

When comparing with the PSP solution, we noted that the MSC replaces some default solvers for some clusters with other default solvers and, while lowering the training performance, this resulted in an improved test performance. To explain this effect we need to understand how GGA tunes this meta-solver. GGA is a genetic algorithm with a specific mating scheme. Namely, some individuals need to compete against each other to gain the right of mating. This competition is executed by racing several individual parameter settings against one another *on a random subset of training instances*. That means that GGA will likely favor that solver for a cluster that has the greatest chance of winning the tournament on a random subset of instances.

Note that the latter is different from choosing the solver that achieves the best score on the entire cluster as we did in our pure solver portfolio (PSP). What we are seeing here

Table 2. Comparison of alternate strategies for selecting a solver for each cluster

Solver	SATzilla	PSP+pre	PSP-Pref+pre	PSP-Bag+pre
	Training Data Set			
PAR10	685	476	2324	531
σ	2584	2070	4666	2226
Ave	153	141	289	142
σ	301	489	465	280
Solved	1,504	1,533	1,284	1,525
%	95.1	97.0	81.2	96.4
	Testing Data Set			
PAR10	3578	2955	5032	2827
σ	5384	5024	5865	4946
Ave	452	416	560	402
σ	522	489	562	484
Solved	405	436	334	442
%	71.1	76.5	58.6	77.5

is that the PSP over-fits the training data. GGA implicitly performs a type of bagging [6] which results in solver assignments that generalize better.

Motivated by this insight we tested two more methods for the generation of a pure solver portfolio. The two alternative methods for generating cluster-based portfolios are:

- **Most Preferred Instances Portfolio (PSP-Pref):** Here, for each each cluster, for each instance in that cluster we determine which algorithms solves it fastest. We associate the cluster with the solver that most instances prefer.
- **Bagged Portfolio: (PSP-Bag)** For each cluster, we do the following: We choose a random subset of our training instances in that cluster and determine the fastest (in terms of PAR10 score) solver (note that we need to run each solver only once for each instance). This solver is the winner of this "tournament." We repeat this process 100 times and associate the solver that wins the most tournaments with this cluster.

In Table 2 we compare these three cluster-based algorithm selectors with SATzilla_R (whereby we augmented these portfolios again by running SATenstein and mxc-sr08 for the first minute, which we indicate by adding '+pre' to the portfolio name). We observe that PSP-Pref+pre is clearly not resulting in good performance. This is likely because it is important to note not only which solver is best, but also how much better is it than its contenders. PSP+pre works much better, but it does not generalize as well on the test set as PSP-Bag+pre. Therefore, when the base solvers of a portfolio have no parameters, we recommend to use the PSP-Bag approach to develop a high performance algorithm selector.

4.4 Latent-Class Model-Based Algorithm Selection

In [27] an alternative model-based portfolio approach was presented. The paper addresses the problem of computing the prediction of a solver's performance on a given

instance using natural generative models of solver behavior. Specifically the authors use a Dirichlet Compound Multinomial (DCM) distribution to create a schedule of solvers, that is, instead of choosing just one solver they give each solver a reduced time limit and run this schedule until the instance is solved or the time-limit is reached. For their experiments, the authors used the 570 instances from the 2009 SAT Competition in the Random category, along with the 40 additional random instances from the same competition originally used for a tie breaking round. This data set of 610 instances was then used to train a latent-class model using random sub-sampling.

In [27] the authors found that this portfolio leads to a slight improvement over SATzilla_R. However, the authors of DCM also mentioned that the comparison is not fully adequate because the latent-class model scheduler uses newer solvers than SATzilla and also the 610 instances were used for both training and testing.

Our thanks go to Bryan Silverthorn who provided the 610 instances used in the experiments in [27], as well as the runtime of the constituent solvers on his hardware, and also the final schedule of solvers that the latent class model found (see [27] for details). These times were run on Intel quad core Xeon X5355 (2.66 GHz) with 32GB of RAM. As competitors, we trained our algorithm selection portfolios based on the previously mentioned 1,582 instances from the Random category of the 2002-2007 SAT Competitions.

Table 3 shows the performance of SATzilla_R, DCM, and our PSP and PSP-Bag (without the '-pre' option!) using a 5,000 second timeout. To account for the random nature of the underlying solvers we repeated the evaluation of the DCM schedule and our portfolios ten times. The table shows mean and median statistics. Even though, as mentioned earlier, the comparison with SATzilla_R is problematic, we include it here to make sure that our comparison is consistent with the findings in [27] where it was found that DCM works slightly better than SATzilla. We observe the same. However, the PSP and PSP-Bag portfolios can do much better and boost the performance from 76% of all instances solved by the DCM to 87% solved by PSP-Bag. Keeping in mind the simplicity of clustering and solver assignment, this improvement in performance is noteworthy.

Table 3. Comparison with the DCM Portfolio developed by Silverthorn and Miikkulainen [27] (results presented here were reproduced by Silverthorn and sent to us in personal commmunication). The table presents mean run-times and median number of solved instances for 10 independent experiments.

Solver	SATzilla	DCM	PSP	PSP-Bag
PAR10	12794	12265	7092	7129
σ	182	314	180	293
Ave	1588	1546	1242	1250
σ	16.6	21.7	14.8	19.5
Solved	458	465	531	530
σ	2.38	4.03	2.36	3.83
%	75.1	76.2	87.0	86.9
σ	0.39	0.66	0.39	0.63
Solved (median)	458	464	531	531
% (median)	75.1	76.0	87.0	87.1

5 Comparison with Other Algorithm Configurators

When the solvers used have parameters, our optimized solver portfolio and our meta-solver configuration approach offer more potential than the pure solver portfolios PSP and PSP-Bag which serve merely as algorithm selectors. In this section, we compare ISAC with two other approaches which tune their solvers, ArgoSmart [20] and Hydra [33].

5.1 ISAC vs. ArgoSmart

The idea that parameterized solvers can be used in a portfolio has been considered recently in ArgoSmart [20]. The authors parameterize and tune the ArgoSAT solver [3] using a partition of the 2002 SAT Competition instances into training and testing set.

Using a supervised clustering approach, the authors build families of instances based on the directory structure in which the SAT Competition has placed these instances. The authors enumerate all possible parameterization of ArgoSAT (60 in total) and find the best parameterization for each family. For a test instance, ArgoSmart then computes the 33 of the 48 core SATzilla features that do not involve runtime measurements [35] and then assigns the instance to one of the instance families based on majority k-nearest-neighbor classification based on a non-Euclidean distance metric. The best parameterization for that family is then used to tackle the given instance.

ISAC enjoys wider applicability as it builds families of instances (our clusters) in an unsupervised fashion. Moreover, ISAC employs GGA to find good solver parameters. Therefore, if the parameter space was much bigger, the ArgoSmart approach would need to be augmented with an instance-oblivious parameter tuner to find good parameters for each of the instances families that it inferred from the directory structure. Despite these current limitations of the ArgoSmart methodology, we were curious to

Table 4. Comparison with ArgoSmart [20] (results presented here were reproduced by Nikolic and sent to us in personal commmunication)

Solver	ArgoSat	ArgoSmart	Unsupervised Clustering			Supervised Clustering			VBS
			PSP	PSP-Bag	Cluster	PSP	PSP-Bag	Cluster	
	Training Data Set								
PAR10	2704	-	2515	2527	2515	2464	2473	2464	2343
σ	2961	-	2935	2967	2935	2927	2959	2927	2906
Ave	294	-	276	276	276	270	271	270	255
σ	285	-	283	284	283	283	283	283	283
Solved	736	-	778	775	778	789	787	789	815
%	55.4	-	58.5	58.3	58.5	59.4	59.2	59.4	61.3
	Testing Data Set								
PAR10	2840	2650	2714	2705	2650	2650	2650	2628	2506
σ	2975	2959	2968	2967	2959	2959	2959	2959	2941
Ave	306	286	291	290	286	286	286	281	269
σ	286	286	287	287	286	286	286	287	286
Solved	337	357	350	351	357	357	357	359	372
%	53.1	56.2	55.1	55.3	56.2	56.2	56.2	56.5	58.6

Table 5. Comparison of Local-Search SAT Solvers and Portfolios Thereof on BM Data

Solver	saps	stein (FACT)	Hydra	MSC-stein	PSP-Bag 11	PSP-Bag 17	MSC-12
	Training Data Set						
PAR10	102	26.8	-	1.78	18.03	1.41	1.41
σ	197	109	-	13.6	87.9	4.09	4.16
Ave	13.5	4.25	-	1.48	3.63	1.11	1.41
σ	19.6	11.3	-	4.41	10.4	3.05	4.16
Solved	1206	1425	-	1499	1452	1499	1500
%	80.4	95.0	-	99.9	96.8	99.9	100
	Testing Data Set						
PAR10	861	220	1.43	1.27	73.5	1.21	1.21
σ	2086	1118	5.27	3.73	635	4.42	3.27
Ave	97.8	26.0	1.43	1.27	12.3	1.20	1.21
σ	210	114	5.27	3.73	69.0	4.42	3.27
Solved	1288	1446	1500	1500	1483	1500	1500
%	85.9	96.4	100	100	98.9	100	100

see whether our assignment of test instances to clusters based on the Euclidean distance to the nearest cluster center is competitive with more elaborate machine learning techniques.

To make this assessment, we generated a PSP and a PSP-Bag based on the time of each of ArgoSAT's parameterizations on each instance, information that was generously provided by Mladen Nikolic. These times were computed on Intel Xeon processors at 2 GHz with 2GB RAM. In Table 4 we compare ArgoSmart and two versions of PSP and PSP-Bag, respectively. Both use the same 33 features of ArgoSmart to classify a given test instance. In one version we also use unsupervised clustering of the training instances, based on the same 33 features. In the other version we consider a supervised clustering gained from the directory structure of the training instances which ArgoSmart used as part of its input. For both variants we also give the best possible cluster-based performance. We observe that the supervised clustering offers more potential. Moreover, when PSP-Bag has access to this clustering, despite its simple classification approach it performs equally well as the machine learning approach approach from [20]. However, even when no supervised clustering is available as part of the input, ISAC can still tune ArgoSAT effectively.

Note that the times of ArgoSmart are different from those reported in [20] because the authors only had the times for all parameterizations for the 2002 SAT data and not the 2007 SAT data they originally used for evaluation. The authors generously retuned their solver for a new partitioning of the 2002 dataset, to give the presented results.

5.2 ISAC vs. Hydra

The methodology behind our final competitor, Hydra [33], enjoys equal generality as ISAC. Hydra consists of a portfolio of various configurations of the highly parameterized local search SAT solver SATenstein. In Hydra, a SATzilla-like approach is used to determine whether a new configuration of SATenstein has the potential of improving a portfolio of parameterizations of SATenstein, and a ParamILS-inspired procedure is used to propose new instantiations of SATenstein.

We consider different approaches for building a portfolio of local search SAT solvers and compare these with Hydra[2] in Tables 5 and 6. The respective benchmarks BM and INDU were introduced in [33]. Both instance sets appear particularly hard for algorithm configuration: In [33], Hydra was not able to outperform an algorithm selection portfolio with 17 constituent solvers. The BM and INDU benchmarks consist of 1,500 train and 1,500 test instances, and 500 train and 500 test instances, respectively. The INDU dataset is comprised of only satisfiable industrial instances, while BM is composed of a mix of satisfiable crafted and industrial instances. We used dual Intel Xeon 5540 (2.53 GHz) quad-core Nehalem processors and 24 GB of DDR-3 memory (1333 GHz) to compute the runtimes.

The training of our portfolios was conducted using a 50 second timeout, for testing we used a 600 second timeout. It is important to point out that, despite the fact that we use a tenfold longer training timeout than [33], the total training time for each portfolio was about 72 CPU days, which is comparable with the 70 CPU days reported in [33] (note also that we used a significantly slower machine for tuning). The reason for this is that we use GGA instead of ParamILS to train the solvers on each cluster. GGA is population-based and races parameter sets against each other, which means that runs can be interrupted prematurely when a better parameter set has already won the race. It is an inherent strength of ISAC that it can handle longer timeouts than Hydra. In the presented results we compare the two approaches assuming they are given the same number of CPU days during which to tune.

The portfolio closest to Hydra is denoted MSC-stein. Here, like Hydra, we limit ourselves to instance-specifically tuning only one solver, SATenstein. As usual, this approach clusters our training instances, and for each cluster we tune SATenstein using GGA. For evaluation, like for the original Hydra experiments, we run each solver three times and use the median time. We observe again that the clustering approach to portfolio generation offers advantages. While Hydra uses a SATzilla-type algorithm selector to decide which tuned version of SATenstein an instance should be assigned to, we use our clusters for this task. As a result, we have a 12% reduction in runtime over Hydra on the BM data-set and more than 40% reduction on INDU. There is also a significant reduction in runtime variance over Hydra: Again, not only does the new portfolio work faster, it also works more robustly across various instances.

Next we use our methodology to build portfolios with more constituent solvers. Following the same setting as in [33], we build an algorithm selector of 11 local search solvers (PSP-Bag 11): paws [28], rsaps [12], saps [29], agwsat0 [30], agwsat+ [31], agwsatp [32], gnovelty+ [23], g2wsat [19], ranov [22], vw [24], and anov09 [11]. In this setting, saps' performance is the best. We next augment the number of constituent solvers by adding six fixed parameterizations of SATenstein to this set, giving us a total of 17 constituent solvers. The respective portfolio is denoted PSP-Bag 17. Finally, we build an MSC-12 based on the (un-parameterized) 11 original solvers plus the (highly parameterized) SATenstein.

Consistent with [33] we find the following:

[2] We are grateful to Lin Xu who provided the Hydra-tuned SATensteins as well as the mapping of test instances to solvers.

Table 6. Comparison of Local-Search SAT Solvers and Portfolios Thereof on INDU Data

Solver	saps	stein (CMBC)	Hydra	MSC-stein	PSP-Bag 11	PSP-Bag 17	MSC-12
	Training Data Set						
PAR10	54.6	6.40	-	2.99	51.7	3.97	3.00
σ	147	23.5	-	3.94	143	22.6	4.47
Ave	10.5	5.50	-	2.99	10.3	3.07	3.00
σ	15.5	8.07	-	3.94	15.3	6.54	4.47
Solved	451	499	-	500	454	499	500
%	90.2	99.8	-	100	90.8	99.8	100
	Testing Data Set						
PAR10	208	5.35	5.11	2.97	209	3.34	2.84
σ	1055	8.54	9.41	4.08	1055	7.05	4.07
Ave	35.7	5.35	5.11	2.97	36.4	3.34	2.84
σ	116	8.54	9.41	4.08	116	7.05	4.07
Solved	484	500	500	500	484	500	500
%	96.8	100	100	100	96.8	100	100

- Apart from the INDU data-set, where the portfolio of 11 solvers cannot improve the performance of the best constituent solver, the portfolios boost significantly the performance compared to the best constituent solver (saps for the 11 solvers on both benchmarks and, for the 17 solvers, SATenstein-FACT on the BM data-set and SATenstein-CMBC on the INDU data-set).
- The portfolio of 17 solvers dramatically improves performance over the portfolio of 11 solvers. Obviously the variants of SATenstein work very well and, on the INDU benchmark, also provide some much needed variance so that the portfolio is now able to outperform the best solver.

In [33] it was found that Hydra, based on only the SATenstein solver, can match the performance of the portfolio of 17 on both benchmarks. While this may be true when the portfolios are built using the SATzilla methodology, this is not true when using our algorithm selector PSP-Bag 17. On BM, PSP-Bag 17 works more than 15% faster than Hydra, on the INDU benchmark set it runs even more than 33% faster.

The full potential of our approach is of course only realized when we build a portfolio of parameterized and un-parameterized solvers. The result is MSC-12 which clearly outperforms all others, working on average almost 18% faster than Hydra on BM and more than 45% faster than Hydra on INDU.

6 Conclusion

We presented the idea of using instance-specific algorithm configuration (ISAC) for the construction of SAT solver portfolios. The approach works by clustering training instances according to normalized feature vectors. Then, for each cluster we determine the best solver or compute a high performance parameterization for a solver. At runtime, for each instance the nearest cluster is determined and the corresponding solver/parameterization is invoked. For the case where solvers are not parameterized, we studied different ways of determining the best solver for a cluster and found that an

approach inspired by bagging works very well in practice. In all experiments, to compare competing approaches we took every precaution to make sure that the conditions under which they were developed were as close as possible. This included using the same solvers in the portfolio, the same tuning times, and same training and testing sets.

We showed that this very simple approach results in portfolios that perform extremely well in practice, clearly outperforming the SAT portfolio generator SATzilla [35], a recent SAT solver scheduler based on a latent-class model, and the algorithm configuration method Hydra [33]. At the same time, ISAC enjoys wide applicability and works completely unsupervised.

Based on this study, we developed the 3S algorithm portfolio which won seven medals, two of them gold, at the 2011 SAT Competition. Even more importantly, this study shows that instance-specific algorithm tuning by means of clustering instances and tuning parameters for the individual clusters is highly efficient even as an algorithm portfolio generator. The fact that, when tuning instance-specifically, we consider portfolios of a potentially infinite number of solvers does not mean that we need to revert to sub-standard portfolio selection. On the contrary: Unsupervised clustering, which originally was a mere concession to tuning portfolios with extremely large numbers of solvers, has resulted in a new state-of-the-art in portfolio generation.

References

1. Adenso-Diaz, B., Laguna, M.: Fine-tuning of Algorithms using Fractional Experimental Design and Local Search. Operations Research 54(1), 99–114 (2006)
2. Ansótegui, C., Sellmann, M., Tierney, K.: A Gender-Based Genetic Algorithm for the Automatic Configuration of Algorithms. In: Gent, I.P. (ed.) CP 2009. LNCS, vol. 5732, pp. 142–157. Springer, Heidelberg (2009)
3. ArgoSAT, `http://argo.matf.bg.ac.rs/software/`
4. Biere, A.: Picosat version 535. Solver description. SAT Competition (2007)
5. Bregman, D.R., Mitchell, D.G.: The SAT Solver MXC, version 0.75. Solver description. SAT Race Competition (2008)
6. Breiman, L.: Bagging Predictors. Machine Learning 24(2), 123–140 (1996)
7. Dequen, G., Dubois, O.: Dubois. kcnfs. Solver description. SAT Competition (2007)
8. Gomes, C.P., Selman, B.: Algorithm Portfolios. Artificial Intelligence 126(1-2), 43–62 (2001)
9. Hamerly, G., Elkan, C.: Learning the K in K-Means. In: NIPS (2003)
10. Heule, M., Dufour, M., van Zwieten, J.E., van Maaren, H.: March_eq: Implementing Additional Reasoning into an Efficient Look-Ahead SAT Solver. In: H. Hoos, H., Mitchell, D.G. (eds.) SAT 2004. LNCS, vol. 3542, pp. 345–359. Springer, Heidelberg (2005)
11. Hoos, H.H.: Adaptive Novelty+: Novelty+ with adaptive noise. In: AAAI (2002)
12. Hutter, F., Tompkins, D., Hoos, H.H.: RSAPS: Reactive Scaling And Probabilistic Smoothing. In: CP (2002)
13. Hutter, F., Hamadi, Y.: Parameter Adjustment Based on Performance Prediction: Towards an Instance-Aware Problem Solver. Technical Report, MSR-TR-2005-125, Microsoft Research Cambridge (2005)
14. Hutter, F., Hamadi, Y., Hoos, H.H., Leyton-Brown, K.: Performance Prediction and Automated Tuning of Randomized and Parametric Algorithms. In: Benhamou, F. (ed.) CP 2006. LNCS, vol. 4204, pp. 213–228. Springer, Heidelberg (2006)
15. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stuetzle, T.: ParamILS: An Automatic Algorithm Configuration Framework. JAIR 36, 267–306 (2009)

16. Kadioglu, S., Malitsky, Y., Sellmann, M., Tierney, K.: ISAC – Instance-Specific Algorithm Configuration. In: ECAI, pp. 751–756 (2010)
17. KhudaBukhsh, A.R., Xu, L., Hoos, H.H., Leyton-Brown, K.: SATenstein: Automatically Building Local Search SAT Solvers From Components. In: IJCAI (2009)
18. Leyton-Brown, K., Nudelman, E., Andrew, G., McFadden, J., Shoham, Y.: A Portfolio Approach to Algorithm Selection. In: IJCAI, pp. 1542–1543 (2003)
19. Li, C.M., Huang, W.Q.: G2WSAT: Gradient-based Greedy WalkSAT. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 158–172. Springer, Heidelberg (2005)
20. Nikolić, M., Marić, F., Janičić, P.: Instance-Based Selection of Policies for SAT Solvers. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 326–340. Springer, Heidelberg (2009)
21. O'Mahony, E., Hebrard, E., Holland, A., Nugent, C., O'Sullivan, B.: Using Case-based Reasoning in an Algorithm Portfolio for Constraint Solving. In: Irish Conference on Artificial Intelligence and Cognitive Science (2008)
22. Pham, D.N., Anbulagan: ranov. Solver description. SAT Competition (2007)
23. Pham, D.N., Gretton, C.: gnovelty+. Solver description. SAT Competition (2007)
24. Prestwich, S.: VW: Variable Weighting Scheme. In: SAT (2005)
25. SAT Competition, `http://www.satcomptition.org`
26. Smith-Miles, K.A.: Cross-disciplinary perspectives on meta-learning for algorithm selection. ACM Comput. Surv. 41(1), 6:1–6:25 (2009)
27. Silverthorn, B., Miikkulainen, R.: Latent Class Models for Algorithm Portfolio Methods. In: AAAI (2010)
28. Thornton, J., Pham, D.N., Bain, S., Ferreira, V.: Additive versus multiplicative clause weighting for SAT. In: PRICAI, pp. 405–416 (2008)
29. Tompkins, D.A.D., Hutter, F., Hoos, H.H.: saps. Solver description. SAT Competition (2007)
30. Wei, W., Li, C.M., Zhang, H.: Combining adaptive noise and promising decreasing variables in local search for SAT. Solver description. SAT Competition (2007)
31. Wei, W., Li, C.M., Zhang, H.: Deterministic and random selection of variables in local search for SAT. Solver description. SAT Competition (2007)
32. Wei, W., Li, C.M., Zhang, H.: adaptg2wsatp. Solver description. SAT Competition (2007)
33. Xu, L., Hoos, H.H., Leyton-Brown, K.: Hydra: Automatically Configuring Algorithms for Portfolio-Based Selection. In: AAAI (2010)
34. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: SATzilla2009: an Automatic Algorithm Portfolio for SAT. Solver description. SAT Competition (2009)
35. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: SATzilla: Portfolio-based Algorithm Selection for SAT. JAIR 32(1), 565–606 (2008)