Feature Filtering for Instance-Specific Algorithm Configuration

Christian Kroer
IT-University of Copenhagen
Copenhagen, Denmark
ckro@itu.dk

Yuri Malitsky
Dept. of Computer Science
Brown University
Providence, RI. USA
ynm@cs.brown.edu

***Abstract*—Instance-Specific Algorithm Configuration (ISAC) is a novel general technique for automatically generating and tuning algorithm portfolios. The approach has been very successful in practice, but up to now it has been committed to using all the features it was provided. However, traditional feature filtering techniques are not applicable, requiring multiple computationally expensive tuning steps during the evaluation stage. To this end, we show three new evaluation functions that use precomputed runtimes of a collection of untuned solvers to quickly evaluate subsets of features. One of our proposed functions even shows how to generate such an effective collection of solvers when only one highly parameterized solver is available. Using these new functions, we show that the number of features used by ISAC can be reduced to less than a quarter of the original number while often providing significant performance gains. We present numerical results on both SAT and CP domains.**

***Keywords*-feature selection; algorithm configuration; SAT; CP;**

I. Introduction

It has long been observed in the constraint programming (CP) and satisfiability (SAT) communities that certain problems are easier to solve with one algorithm, but performance then suffers on other problems. This observation has led to the pursuit of designing approaches that can adapt to the problem at hand. One such approach can be characterized as algorithm configuration, which adjusts the parameters defining the behavior of a solver to best suit the provided training set [1], [9], [2]. Alternatively, algorithm portfolios take advantage of the variety of solvers already developed and the inherent variance in their performance. When given a new instance, this approach predicts the performance quality of each solver, to then use the one with the best expected outcome [5], [13], [16], [33], [24], [15].

Instance-Specific Algorithm Configuration (ISAC) [11] is a recent example of an approach which creates its own portfolio of solvers. The approach first clusters the training instances and then tunes a separate solver for each cluster. When a new instance is provided it is assigned to a cluster and then solved with the tuned solver. The tuned solver can be a single parameterized solver like SATenstein [12], saps [8], or even Cplex [10]. The solver can also be a portfolio, where the tuner not only chooses the best solver for a cluster, but also the parameters for that solver.

ISAC was recently shown to be highly effective in the SAT domain. A novel solver tuned using the ISAC methodology won 2 gold medals in the 2011 SAT competition [22]. ISAC's application has also been shown for other domains like set covering (SCP) and mixed integer (MIP) problems [11].

One of the major drawbacks of ISAC is its dependence on the feature vector it uses to differentiate the problem instances. The success of the entire clustering hinges on the ability of these features to correctly group instances that are likely to behave similarly under the same solver. However, there is a disconnect between the clustering objective and the performance objective ISAC tries to improve. So far, research of ISAC was fortunate that the computed clusters yielded significant performance gains over the other approaches.

To tackle this problem we propose an approach that builds off our initial assumption that instances with similar features will behave comparably under the same parameter settings. We therefore design three new evaluation functions that can be computed without retuning solvers for each iteration. These functions are first evaluated on four standard SAT benchmarks, and then confirmed in the CP domain.

In the remainder of the paper we give an overview of related work in Section II and a description of ISAC in Section III. Section IV presents the new evaluation strategies we propose. Section V introduces the feature filtering algorithms that we employ. Section VI presents the numeric results on SAT and CP domains and Section VII concludes with a discussion of our contributions and results.

II. Related Work

Algorithm selection is a well studied approach in machine learning [20], [27], [23]. In this scenario, a large number of algorithms are run on a collection of training instances and the resulting data would be used to train a model mapping from (algorithm, problem) pairs to the expected performance. The problem is represented as a vector of feature values. When a new instance is provided, its feature vector and expected time for each solver is computed. The solver with the shortest expected runtime is then used to solve the instance.

In the optimization domain, these portfolios were first introduced in 2001 by Carla Gomes and Bart Selman [5]. Portfolio algorithms have become increasingly popular, especially after the success of SATzilla [32]. This SAT solver dominated the SAT Competitions in 2007 and 2009 [21] in the randomly generated problem category. The solver is based on basic machine learning techniques: sparse multinomial logistic regression to predict the log runtime of a solver and feedforward selection to filter superfluous features. In the case of SATzilla, feature filtering can be applied directly because a new prediction model can be trained quickly.

CPhydra [16] is another portfolio algorithm that dominated the recent CP competitions, creating a schedule of solvers in its portfolio that best utilizes the allotted time. In this approach, a k-nearest-neighbor approach is employed to gather the data used by a constraint program to create the schedule. This approach assumes that all the features are equally important and performs no feature filtering. This could possibly be due to the time requirements of evaluating the quality of a subset of features. Evaluation will need a large number of constraint programs to be computed to create a new schedule for each training instance before overall performance is computed.

For situations where there is only a limited supply of solvers available, a hybrid methodology has been proposed that aims to create a custom set of solvers for the algorithm portfolio. One such method is Hydra [31], which iteratively tunes a new version of a highly parameterized solver, Satenstein [12], adding it to the existing portfolio only if it helps to improve performance on a training set of instances. This methodology is based on SATzilla, once the solver is added the resulting portfolio is tuned just like SATzilla.

In ArgoSmart [15], the authors parametrize and tune the ArgoSAT solver [3] using a partition of the 2002 SAT Competition instances from the Random category 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 parametrization of ArgoSAT (60 in total) and find the best parametrization for each family. For a test instance, ArgoSmart then computes the 33 core SATzilla features that do not involve runtime measurements [33] and assigns the instance to one of the instance families based on majority k-nearest-neighbor classification utilizing a non-Euclidean distance metric. The best parametrization for that family is then used to tackle the given instance. Yet here also, the authors assume that all features are equally important.

In these and other approaches in the optimization community, algorithm portfolios have been shown to drastically improve performance over using a single solver. However, only a few of the approaches use feature filtering. This is partly because applying any standard feature selection methods would require very computationally expensive evaluation functions. In this paper we show how ISAC can be augmented using feature filtering with three newly proposed evaluation functions.

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, 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 $A(x, P_i)$

Algorithm 1: Instance-Specific Algorithm Configuration

III. ISAC

ISAC is a newly proposed approach for tuning algorithms for the problem instances they will be used for. The approach works as follows (see Algorithm 1). In the learning phase, ISAC is provided with a parameterized solver A, a list of training instances T, and their corresponding feature vectors F. 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) is memorized. This normalization helps keep all the features at the same order of magnitude, and thereby keeps the larger values from being given more weight than the lower values.

Then, 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 found parameters are "pre-stabilized," meaning they are shown to work well *together*.

To avoid specifying the desired number of clusters beforehand, the g-means [6] algorithm is used. 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. 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 [2].

1: **E_Dist**(C, R, I)
2: $N \leftarrow \emptyset$, $v^* \leftarrow 0$
3: **for** $i \in I$ **do**
4: $N_i \leftarrow$**Normalize**(R_i)
5: **end for**
6: **for** $c \in C$ **do**
7: $v^* \leftarrow v^* + |c| * \sum_{i \in I} \sum_{j>i} ||N_i - N_j||$
8: **end for**
9: **return** v^*

Algorithm 2: Evaluation functions used to measure the quality of a clustering of instances.

GGA races a large number of parameter settings on subsets of the training instances, with the best performing parameters getting the chance to crossover and continue to subsequent generations. Starting with a small number of instances, the subset of instances used for tuning grows with each iteration as the bad parameter settings get weeded out of consideration. In the final generation of GGA, when all training instances are used, the best parameter setting has been shown to work very well on these and similar instances.

When running algorithm A on an input instance x, ISAC first computes the features of the input and normalize 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.

IV. Cluster Evaluation

The effect of the filtering algorithms, such as the ones discussed in Section V, strongly depends on the quality of the evaluation function. In order to evaluate a set of features using standard techniques, the training instances would be clustered and a new solver tuned for each cluster. The quality of the feature would then be defined as the performance of the portfolio solver on some validation set of instances. However, because of the long time needed to tune the algorithms, evaluation based on this kind of performance is impractical. To sidestep this issue, we instead focus on the primary assumption behind ISAC; that a solver will have consistent performance on instances that are clustered together. Based on this assumption we introduce three possible evaluation functions that utilize a collection of untuned solvers to determine the quality of a cluster of instances.

The first evaluation criteria is presented in Algorithm 2 as E_Dist. Given the clustering of the instances C, the runtime of each untuned solver on each instance R, and the list of instances I, this algorithm tries to match the relative quality of solver runtimes on instances in the same cluster. Thus, the algorithm tries to make sure that the same solver works best on all instances in the cluster, and the same solver provides the worst performance. Because the runtimes

1: **E_Time**(C, R)
2: $v^* \leftarrow 0$
3: **for** $c \in C$ **do**
4: $v^* \leftarrow v^* + \min_{s \in R}$(**Runtime**($s$, c))
5: **end for**
6: **return** v^*

Algorithm 3: Evaluation functions used to measure the quality of a clustering of instances.

can vary significantly between instances, these times are normalized for each instance to be from 0 to 1, with 0 being the fastest runtime and 1 the longest. These normalized runtimes N can then be used to judge how similar two instances are, and a *good* cluster is one where the average euclidean distances between each instance within the cluster is minimized. The evaluation of the overall clustering v^* is then the summation of the quality of each cluster weighted by the number of instances in that cluster, $|c|$. Here we do not consider the distances between clusters because it is not necessarily the case that different clusters require different solvers. Only the uniformity of the instances within a cluster determine the success of a clustering.

An alternative evaluation function measures the quality of the clustering directly by computing the performance of a portfolio algorithm based on the available solvers. E_Time in Algorithm 3 creates a portfolio of untuned solvers and chooses which solver to assign to each cluster. The algorithm finds the best performing solver in R on the instances of each cluster. The clustering can then be evaluated by summing the score for each cluster when using the best solver. This evaluation approach benefits from being similar to how ISAC will be evaluated in practice, without having to tune each solver for the cluster.

For Algorithm 3 we use the exact runtimes of the best solvers to evaluate a clustering. We also experimented with an algorithm where we again select the best solver for each cluster, but the evaluation is done using a penalized version of the runtimes, called Par10. Each instance not solved gets penalized with a runtime that is ten times the cutoff time. Using penalized runtimes makes the algorithms focus on minimizing the number of instances not solved. However, we found that using the regular non-penalized runtimes provided better performance, both in terms of the average runtimes achieved and number of instances solved.

Using the performance of a portfolio for evaluating the clustering can yield very good results if the solvers in the available portfolio are numerous and have a lot of variance in their performance. This is true in the case of a well studied problem like SAT but is not necessarily the case in all problem domains. To circumvent this issue, we extend the evaluation criteria to generate the desired portfolio.

Given a single, highly parameterized solver, it is possible

```
1: FeedForwardSelection(F, I, R)
2: F* ← ∅ , F̂ ← F , s ← ∞ , s* ← ∞
3: while s ≤ s* do
4:    f* ← ∅ , s ← ∞
5:    for f ∈ F̂ do
6:       v = EVALUATE(CLUSTER(F* ∪ f, I), R)
7:       if v ≤ s then
8:          f* ← f
9:          s ← v
10:      end if
11:   end for
12:   if s ≤ s* then
13:      F* ← F* ∪ f*
14:      s* ← s
15:   end if
16: end while
17: return F*
```

Algorithm 4: Feed forward feature selection

to tune this solver using GGA. In our case however, we do not need the best performing parameter set, but many parameter sets that behave reasonably well and with a lot of variance. These parameter sets can be initialized randomly, but the resulting solvers are likely to perform very poorly. In a case were every solver times out, it is impossible to determine which solver is best. But, if we use the solvers from an intermediate generation of GGA, we will find that the very bad parameter sets have already been killed off by the tuner, and all that remains are parameters that work well on different subsets of our training data. Using these parameter settings, we create a large portfolio of solvers that we can use for the direct evaluation of a clustering. This evaluation approach works as Algorithm 3, using the best solver on each cluster to compute the performance score of a clustering, the difference being that the runtimes of the generated solvers are used in place of the regular solvers.

V. Filtering Algorithms

It is well established that the success of a machine learning algorithm depends on the quality of its features. Too few features might not be enough to properly differentiate between two or more classes of instances. Alternatively, too many features often results in some of the features being noisy and even damaging. Furthermore, as the feature space increases, more data is needed in order to make accurate predictions. In this paper, we work with the three standard feature selection algorithms: feed-forward selection, backward selection, and a hybrid of the two. All three of these algorithms can use any of the evaluation functions discussed in Section IV.

Feed-forward selection (Algorithm 4) starts with an empty set F^* and tries to add each of the available features $\hat{F}$. Using each of these new subsets of features, the training set I is clustered and evaluated. The feature f^* whose addition to the current set yields the best performance s is added to the set and the process is repeated until no more features can be added without the evaluation score deteriorating.

Alternatively, backward-selection starts with the full feature set and removes features one at a time in a manner that is analogous with how feed forward selection adds them. The algorithm terminates when such a removal leads to a decrease in performance according to the evaluation function.

A natural extension of the above two algorithms is a hybrid approach of the two. As in backward selection, the algorithm begins with the full feature set, and removes features one at a time while the solution doesn't deteriorate. The algorithm however also checks if adding any of the removed features improves the solution. If this is the case, the feature is added back into the set. This helps when the beneficial effects are being obfuscated by many noisy features. Once the troublesome features are removed, the benefits are observed and the mistake of removing the feature is rectified.

VI. Numerical Results

For our experiments we first focus on the SAT domain, a well studied problem that has numerous solvers, benchmarks, and well defined features. SAT is also a domain where ISAC has been shown to yield state-of-the-art performance. Showing the performance gains on SAT we then continue by switching to the CP domain. The timed experiments used dual Intel Xeon 5540 (2.53 GHz) quad-core Nehalem processors with 24 GB of DDR-3 memory (1333 GHz).

A. Benchmarks

For SAT, we choose to focus on local search solvers predominately due to the existence of SATenstein [12], a highly parameterized solver. SATenstein is used to explore our evaluation function that uses GGA to create a portfolio of solvers. Evaluation is based on the HAND and RAND datasets [31] that span a variety of problem types and were developed to test local search based solvers. Because local search can never prove that a solution doesn't exist, these four datasets only have satisfiable instances. These two data sets were chosen because they have been shown to be hard for portfolio algorithms in [31].

The HAND and RAND datasets are respectively composed of hand-crafted and randomly generated instances. The HAND dataset has 342 training and 171 testing instances. The RAND dataset has 1141 training and 581 testing instances. For features, we use the well established 48 features introduced in [32]. It is important to note that these features have been used for SAT portfolios for over a decade, so they have been thoroughly vetted as being important.

HAND	BS	All	E_Dist			E_Time			E_Time (GGA)		
	rsaps	Features	Forward	Backward	Hybrid	Forward	Backward	Hybrid	Forward	Backward	Hybrid
Par 10 - avg	3034	2789	2784	2823	2823	**2546**	2712	2711	2752	**2748**	**2748**
Par 10 - std	2977	2975	2980	2979	2979	**2945**	2975	2976	2974	**2978**	**2978**
Runtime - avg	296.9	294.8	289.5	296.5	296.5	**273.0**	280.8	280.8	289.6	**285.1**	**285.1**
Runtime - std	142.5	289.5	293.6	290.6	290.6	**288.5**	292	292.6	290.5	**291.6**	**291.6**
features	-	48	10	33	33	**4**	20	23	5	**39**	**39**
clusters	-	4	5	5	5	**5**	5	5	5	**5**	**5**
solved	93	92	92	91	91	**99**	94	94	93	**93**	**93**
% solved	54.39	53.8	53.8	53.2	53.2	**57.89**	54.97	54.97	54.39	**54.39**	**54.39**
RAND	BS	All	E_Dist			E_Time			E_Time (GGA)		
	gnovelty+	Features	Forward	Backward	Hybrid	Forward	Backward	Hybrid	Forward	Backward	Hybrid
Par 10 - avg	1138	755.5	780.1	745.2	745.2	**698.5**	729.9	729.9	762.7	**688.9**	709.4
Par 10 - std	2239	1958	1995	1946	1946	**1899**	1936	1936	1971	**1886**	1911
Runtime - avg	126.2	95.61	92.33	94.58	94.58	**85.1**	88.59	88.59	93.48	**84.8**	86.72
Runtime - std	229.9	92.33	204.3	202.0	202.0	**195.4**	199.1	199.1	201.9	**194.8**	197.1
features	-	48	5	37	37	**6**	38	38	5	**30**	33
clusters	-	11	12	14	14	**11**	12	12	11	**11**	11
solved	483	510	507	511	511	**515**	512	512	509	**516**	514
% solved	83.13	87.78	87.26	88	88	**88.64**	88.12	88.12	87.61	**88.81**	88.47

Table I
RESULTS ON THE SAT BENCHMARKS, COMPARING THE BEST PERFORMING INDIVIDUAL SOLVER "BS", THE ORIGINAL ISAC USING ALL FEATURES "ALL FEATURES", AND ALL THE COMBINATIONS OF EVALUATION FUNCTIONS AND FILTERING ALGORITHMS.

In these experiments we used an assortment of successful local search solvers: paws [25], rsaps [8], saps [26], agwsat0 [28], agwsat+ [29], agwsatp [30], gnovelty+ [18], g2wsat [14], ranov [17], vw [19], and anov09 [7]. We also use six additional fixed parameterizations of SATenstein, known as Fact, Cmbc, R3fix, Hgen, Swgcp, and Qcp. For evaluation, a 600 second timeout was used.

For the CP benchmarks we employ instances from CPAI08 [4]. We removed a small number of instances for which the cpHydra feature computation code [16] did not work. The remaining instances were split into 901 train instances and 930 test instances. Our portfolio consisted of a subset of the solvers that competed in the original competition: Abscon_112v4 (AC), Abscon_112v4 (ESAC), bpsolver (2008-06-27), casper (zao), casper (zito), choco2_dwdeg (2008-06-26), choco2_impwdeg (2008-06-26), cpHydra (k_10), cpHydra (k_40), MDG-noprobe (2008-06-27), MDG-probe (2008-06-27), Minion_Tailor (2008-07-04), Mistral-option (1.314), Mistral-prime (1.313), SAT4J_CSP (2008-06-13), Sugar (v1.13+minisat), Sugar (v1.13+picosat). For the runtimes, we used the runtimes from the competition results which had a 1800 second timeout [4].

B. E_Dist Approach

Table I shows the results of the best performing solver over all the instances in each of the four benchmarks. The table then presents the performance of an algorithm portfolio of the 17 local search solvers tuned with ISAC using the complete set of 48 features. In this scenario, once the training instances are clustered, the best performing solver in the portfolio is assigned to each cluster. The decision of the solver to use is based on the best average runtime. In all cases the average runtime was improved. The change is especially significant for RAND. Furthermore, as is expected, in all cases ISAC usually doesn't use the solver that is found best over all instances, which suggests that certain solvers are better at solving certain instances while sacrificing performance on other types. For the HAND benchmark however, the time gain is minimal and one fewer instance is solved. Judging by the performance of ISAC with feature filtering, this poor performance is due to a poor clustering as a result of some confounding and noisy features.

Table I then shows the results from running ISAC after the features are filtered using the euclidean distance evaluation criteria, E_Dist. It is interesting to note that for both of the datasets, it is possible to maintain the same level of performance with significantly fewer features. This is especially true for feed-forward selection that uses less than a quarter of the features. However, we also observe that there is no significant improvement of the overall performance of the resulting portfolio.

C. E_Time Approach

Once we use a more accurate evaluation criteria, E_Time, we observe that ISAC's performance can be boosted, as seen in Table I. Here, feed-forward selection is again the best performing approach and we observe improvements in both of our datasets, although we also observe an increase in the number of used features. This seems to support the assumption that not all of the established 48 features are necessary, and are in fact damaging to the clustering in ISAC.

It is interesting to note that the features found by forward selection do not overlap much for four benchmarks with only two features appearing in both sets. The first is the maximum ratio of positive to negative literals per clause. The second is the number of unary literals in the problem.

Also of note is that feed-forward selection only chooses

CP	cpHydra (k = 40)	All Features	Forward	Backward
Parscore	2667	2421	2124	**1994**
Std. deviation	6695	6083	5756	**5589**
Avg. runtime	286.1	278.8	242.3	**234.5**
Std. deviation	617.0	613.5	579.5	**567.4**
# features	36	36	7	**29**
# clusters	-	18	20	**19**
# solved	807	807	822	**829**
% solved	86.77	86.77	88.39	**89.14**

Table II
RESULTS ON THE CP BENCHMARK, COMPARING THE BEST PERFORMING SOLVER "CPHYDRA", ISAC USING "ALL FEATURES", AND THE FORWARD AND BACKWARD FILTERING ALGORITHMS USING THE E_TIME EVALUATION FUNCTION.

the local search probing features twice for RAND. In backward selection however, almost all these probing features are used in both benchmarks. These features are stochastic with potentially a lot of variance between computations. They are also very computationally expensive, especially for larger instances. Fortunately, according to a comparison of the Forward and Backward selection algorithms these features are not needed and do not improve performance.

The number of clusters does not change drastically when switching from the euclidean distance, E_Dist, to the time performance evaluation functions, E_Time. This suggests that simply increasing or decreasing the number of clusters is not enough to improve performance. This also validates our clustering approach, showing that if a cluster contains similar instances, then it is possible to tune a very high quality solver for those instances.

In these experiments, there is little difference in the features found by backward selection and our hybrid approach. This is not very surprising since the clustering is based on a linear relation between the features, and it is unlikely that a removed feature would become beneficial once another feature is removed.

These results on SAT are encouraging, and as can be seen in Table II they also extend to the CP domain. When comparing all the single solvers, cpHydra significantly outperforms the other solvers. The closest competitor is Mistral-prime (1.313) which solves 780 instances. As shown in the table, applying ISAC to tune a portfolio algorithm leads to marginal improvements in the average runtime. However, once feature filtering is applied the performance of the tuned portfolio improves significantly. Using backward filtering creates a performance gap to cpHydra equal to the one separating cpHydra from its closest competitor.

D. E_Time (GGA) Approach

Running the feature filtering algorithms using the runtimes of the 100 GGA generated solvers yields the times presented under "E_Time (GGA)" in Table I. In this case, the forward selection algorithm worsens performance on RAND, but otherwise remains competitive with the original ISAC with less than a 11% of the features. Backward selection does not worsen performance on any datasets, and even gives the best performance of all the approaches on the RAND dataset, while removing a significant number of features in all cases.

In all cases, we see that using feedforward selection greatly reduces the number of needed features while usually improving the overall performance of the resulting tuned solver. Backward selection on the other hand seems like a more conservative approach, removing fewer features but it also offers consistent improvements over all datasets. This suggests that there are some features that should not be used for clustering, and all filtering algorithms remove these. But there are also some dependancies between the features, and including these features is important to improve the quality of the clusters used by ISAC.

While feed-forward selection generally outperformed backward selection on all datasets when using the portfolios of solvers, we see that when using the GGA generated solvers, backward selection clearly outperforms forward selection. When using a portfolio of solvers, we have access to more variety in solvers, as opposed to using a set of parameterized versions of the same solver. This seems to indicate that feedforward selection has a higher need for diversity in solvers, as it struggles with picking out the most important features starting from none, whereas backward selection is able to successfully remove a large set of noisy features and provide performance gains using the less diverse GGA solvers.

VII. CONCLUSIONS

Instance-Specific Algorithm Configuration (ISAC) is a newly proposed methodology that has been very successful in tuning a wide range of solvers for SAT, MIP, and SC. However, the approach was very dependent on the quality of features it was provided. It is well known in the machine learning community that a poorly selected set of features can have a strong detrimental effect on performance. Standard approaches for feature filtering were not applicable due the computational cost associated with the evaluation of a clustering using a subset of features. In this paper we showed three modifications to the evaluation function that remove the expensive portion of the approach. Applying feature filtering to ISAC, we observe performance gains in both SAT and CP domains while reducing the size of the feature sets.

These performance gains are important in the case of SAT since the 48 features are already a subset of a larger set of 89 features which has been carefully studied for the last ten years. Yet even in this case, we show that proper feature filtering does not worsen the performance but has a chance to greatly improve it. This observation is confirmed in the CP domain where the features have not been as carefully vetted.

Just applying ISAC on all 36 features didn't improve the number of solved instances. But once feature filtering was applied the performance also improved.

REFERENCES

[1] B. Adenso-Diaz and M. Laguna. Fine-tuning of Algorithms using Fractional Experimental Design and Local Search. *Operations Research*, 54(1):99–114, 2006.

[2] C. Ansotegui-Gil, M. Sellmann, K. Tierney. A Gender-Based Genetic Algorithm for the Automatic Configuration of Algorithms. *CP 2009*, 142–157, 2009.

[3] ArgoSAT. *http://argo.matf.bg.ac.rs/software/*.

[4] CPAI08 Competition. *http://www.cril.univ-artois.fr/CPAI08/*.

[5] C.P. Gomes and B. Selman. Algorithm Portfolios. *Artificial Intelligence*, 126(1–2):43–62, 2001.

[6] G. Hamerly and C. Elkan. Learning the K in K-Means. *NIPS*, 2003.

[7] H.H. Hoos. Adaptive Novelty+: Novelty+ with adaptive noise. *AAAI*, 2002.

[8] F. Hutter, D. Tompkins, H.H. Hoos. RSAPS: Reactive Scaling And Probabilistic Smoothing. *CP*, 2002.

[9] F. Hutter, H.H. Hoos, K. Leyton-Brown, T. Stuetzle. ParamILS: An Automatic Algorithm Configuration Framework. *JAIR*, 36:267–306, 2009.

[10] IBM. *IBM CPLEX* Reference manual and user manual. V12.1, IBM 2009.

[11] S. Kadioglu, Y. Malitsky, M. Sellmann, K. Tierney. ISAC – Instance-Specific Algorithm Configuration. *ECAI*, 751–756, 2010.

[12] A.R. KhudaBukhsh, L. Xu, H.H. Hoos, K. Leyton-Brown. SATenstein: Automatically Building Local Search SAT Solvers From Components. *IJCAI*, 2009.

[13] K. Leyton-Brown, E. Nudelman, G. Andrew, J. McFadden, Y. Shoham. A Portfolio Approach to Algorithm Selection. *IJCAI*, 1542–1543, 2003.

[14] C.M. Li and W.Q. Huang. G2WSAT: Gradient-based Greedy WalkSAT. *SAT*, 3569:158–172, 2005.

[15] M. Nikolic, F. Maric, P. Janici. Instance Based Selection of Policies for SAT Solvers. *Theory and Applications of Satisfiability Testing*, 326–340, 2009.

[16] E. O'Mahony, E. Hebrard, A. Holland, C. Nugent, B. O'Sullivan. Using Case-based Reasoning in an Algorithm Portfolio for Constraint Solving. *Irish Conference on Artificial Intelligence and Cognitive Science*, 2008.

[17] D.N. Pham and Anbulagan. ranov. Solver description. *SAT Competition*, 2007.

[18] D.N. Pham and C. Gretton. gnovelty+. Solver description. *SAT Competition*, 2007.

[19] S. Prestwich. VW: Variable Weighting Scheme. *SAT*, 2005.

[20] J.R. Rice. The algorithm selection problem. *Advances in Computers*, 65–118, 1976.

[21] SAT Competition. *http://www.satcomptition.org*.

[22] SAT Competition 2011. *http://www.cril.univ-artois.fr/SAT11/*.

[23] K.A. Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Comput. Surv.*, 41(1): 6:1–6:25, 2009.

[24] B. Silverthorn and R. Miikkulainen. Latent Class Models for Algorithm Portfolio Methods. *AAAI*, 2010.

[25] J. Thornton, D.N. Pham, S. Bain, V. Ferreira. Additive versus multiplicative clause weighting for SAT. *PRICAI*, 405–416, 2008.

[26] D.A.D Tompkins, F. Hutter, H.H. Hoos. saps. Solver description. *SAT Competition*, 2007.

[27] R. Vilata and Y. Drissi. A perspective view and survey of meta-learning. *Artificial Intelligence, Rev. 18*, 77–95, 2002.

[28] W. Wei, C.M. Li, H. Zhang. Combining adaptive noise and promising decreasing variables in local search for SAT. Solver description. *SAT Competition*, 2007.

[29] W. Wei, C.M. Li, H. Zhang. Deterministic and random selection of variables in local search for SAT. Solver description. *SAT Competition*, 2007.

[30] W. Wei, C.M. Li, H. Zhang. adaptg2wsatp. Solver description. *SAT Competition*, 2007.

[31] L. Xu, H. H. Hoos, K. Leyton-Brown. Hydra: Automatically Configuring Algorithms for Portfolio-Based Selection. *AAAI*, 2010.

[32] L. Xu, F. Hutter, H.H. Hoos, K. Leyton-Brown. SATzilla2009: an Automatic Algorithm Portfolio for SAT. Solver description. *SAT Competition*, 2009.

[33] L. Xu, F. Hutter, H.H. Hoos, K. Leyton-Brown. SATzilla: Portfolio-based Algorithm Selection for SAT. *JAIR*, 32(1):565–606, 2008.