SNNAP: Solver-based Nearest Neighbor for Algorithm Portfolios

Marco Collautti, Yuri Malitsky, Deepak Mehta, and Barry O'Sullivan

Cork Constraint Computation Centre
Department of Computer Science, University College Cork, Ireland
collautt@dei.unipd.it, (y.malitsky, d.mehta, b.osullivan)@4c.ucc.ie

Abstract. The success of portfolio algorithms in competitions in the area of combinatorial problem solving, as well as in practice, has motivated interest in the development of new approaches to determine the best solver for the problem at hand. Yet, although there are a number of ways in which this decision can be made, it always relies on a rich set of features to identify and distinguish the structure of the problem instances. In this paper, we show how one of the more successful portfolio approaches, ISAC, can be augmented by taking into account the past performance of solvers as part of the feature vector. Testing on a variety of SAT datasets, we show how our new formulation continuously outperforms an unmodified/standard version of ISAC.

1 Introduction

It is becoming increasingly recognized in the constraint programming (CP) and satisfiability (SAT) communities that there is no single best solver for all problems. Instead solvers tend to excel on a particular set of instances while offering subpar performance on everything else. This observation has led to the pursuit of approaches that, given a collection of existing solvers, attempt to select the most appropriate one for the problem instances at hand. The way in which these portfolio solvers make the selection, however, varies greatly. One approach can be to train a regression model to predict the performance of each solver, selecting the expected best one [16, 15]. Alternatively, a ranking approach can be used over all solvers [6]. It is also possible to train a forest of trees, where each tree makes a decision for every pair of solvers, deciding which of the two is likely to be better, selecting the one voted upon most frequently [17]. Research has also been conducted on creating a schedule of solvers to call in sequence rather than committing to a single one [7, 12]. An overview of many of these approaches is presented in [9]. Yet regardless of the implementation, portfolio-based approaches have been dominating the competitions in satisfiability (SAT) [7, 16], constraint programming (CP) [12], and quantified Boolean formulae (QBF) [14].

One of the most successful portfolio techniques is referred to as Instance-Specific Algorithm Configuration (ISAC) [8]. Originally the approach was demonstrated to outperform the then reigning regression-based approach, SATzilla, in the SAT domain [16], as well as a number of other existing portfolios [11]. The

approach was later embraced by the 3S solver which won 2 gold medals in the 2011 SAT competition [1]. ISAC's applicability has also been demonstrated in other domains such as set covering (SCP) and mixed integer (MIP) problems [8].

The guiding principle behind ISAC is to cluster the training instances based on a representative feature vector. It is assumed that if the feature vector is good, then the instances within the same cluster will all have similar underlying structure and will therefore yield to the same solver. Up to now, however, ISAC has relied on a pre-specified set of features and an objective-oblivious clustering. This means that any anomalous behavior of the features could result in an improper clustering. Similarly, it cannot be assumed the identified clusters group the instances in an optimal manner. In short, there has been a disconnect between the clustering objective and the performance objective.

For this reason we propose a new approach, SNNAP, in which we try to use the past performance of the solvers in the portfolio as part of the feature vector. This approach redefines the feature vector to automatically encode our desired objective of having similar instances in the same cluster, and is thus able to significantly improve the results of the original algorithm. However, unlike feature landmarking [13, 3] where the evaluation of multiple simple algorithms provide insight into the success of more complex algorithms, SNNAP tackles a domain where a problem needs to only be evaluated once, but highly efficiently.

In this paper, we will first give an overview of the ISAC methodology and demonstrate that straightforward feature filtering is not enough to significantly improve performance. We will then show a number of modifications to the employed distance metric that improve the quality of the clusterings, and thus the overall performance. We will conclude by presenting SNAPP, an approach that combines predictive modeling as a way of generating features to be used by dynamic clustering. To demonstrate our results, we use a collection of SAT datasets: one that contains only randomly generated instances, one with only handcrafted instances, one with industrial instances, and finally a mixture of all three.

2 ISAC

The fundamental principle behind ISAC is that instances with similar features are likely to have commonalities in their structure, and that there exists at least one solver that is best at solving that particular structure. Therefore the approach works as presented in Algorithm 1. In the training phase, ISAC is provided with a list of training instances T, their corresponding feature vectors F, and a collection of solvers A.

First, the computed features are normalized so that every feature ranges in [-1,1]. This normalization helps keep all the features at the same order of magnitude, and thereby avoids the larger values being given more weight than smaller values. Using these normalized values, the instances are clustered. Although any clustering approach can be used, in practice *g-means* [5] is employed in order to avoid specifying the desired number of clusters. This clustering ap-

Algorithm 1 ISAC

```
1: function ISAC-Train(T, F, A)
2:     (F̄) ← Normalize(F)
3:     (k, C, S) ← Cluster(T, F̄)
4:     for all i = 1, ..., k do
5:         BS_i ← FindBestSolver(T, S_i, A)
6:     end for
7:     return (k, C, BS)
8: end function
```

```
1: function ISAC-Run(x, k, C, BS)
2:     j ← FindClosestCluster(k, x, C)
3:     return BS_j(x)
4: end function
```

proach assumes that a good cluster is one in which the data follows a Gaussian distribution. Therefore, starting with all instances in the same cluster, *g-means* iteratively applies 2-means clustering, accepting the partition only if the two new clusters are more Gaussian than their predecessor. Once the instances are clustered, we add an additional step that merges all clusters smaller than a predefined threshold into their neighboring clusters. The final result of the clustering is a set of k clusters S, and a set of cluster centers C. For each cluster we then determine a single solver, which is usually the one that has best average performance on all instances in its cluster. When the procedure is presented with a previously unseen instance x, ISAC assigns it to the nearest cluster and runs the solver designated to that cluster on the instance x.

In practice, this standard version of ISAC has been continuously shown to perform well, commonly outperforming the choice of a single solver on all instances. In many situations ISAC has even outperformed existing state-of-the-art regression-based portfolios [11]. However, the current version of ISAC also accepts the computed clustering on faith, even though it might not be optimal or might not best capture the relationship between problem instances and solvers. It also does not take into account that some of the features might be noisy or misleading. Therefore, the following sections will show the advantages of applying feature filtering. We will then show how the feature vector can be extended to include the performance of solvers on the training instances. This has the effect of increasing the chances that instances for which the same solver performs well are placed into the same cluster. Finally, we extend the approach to combine predictive modeling and dynamic clustering to find the best cluster for each new instance.

3 Experimental Setup

The satisfiability (SAT) domain was selected to test our proposed methodologies due to the large number of diverse instances and solvers that are available. We

compiled four datasets using all instances from the 2006 SAT Competition to the present day 2012 SAT Challenge [2]. These four datasets contain 2140, 735, 1098 and 4243 instances, respectively, as follows:

RAND: instances have been generated at random;
HAND: instances are hand-crafted or are transformations from other NP-Complete problems;
INDU: instances come from industrial problems;
ALL: instances are the union of the previous datasets.

We rely on the standard set of 115 features that have been embraced by the SAT community [16]. Specifically, using the feature code made available by UBC,[1] we compute features with the following settings: *-base, -sp, -dia, -cl, -ls, and -lobjois*; but having observed previously that the features measuring computation time are unreliable, we discard those 9. These features cover information such as the number of variables, number of clauses, average number of literals per clause, proportion of positive to negative literals per clause, etc. Finally, we also run 29 of the most current SAT solvers, many of which have individually shown very good performance in past competitions. Specifically, we used:

- `clasp-2.1.1_jumpy`
- `clasp-2.1.1_trendy`
- `ebminisat`
- `glueminisat`
- `lingeling`
- `lrglshr`
- `picosat`
- `restartsat`
- `circminisat`
- `clasp1`
- `cryptominisat_2011`
- `eagleup`
- `gnoveltyp2`
- `march_rw`
- `mphaseSAT`
- `mphaseSATm`
- `precosat`
- `qutersat`
- `sapperlot`
- `sat4j-2.3.2`
- `sattimep`
- `sparrow`
- `tnm`
- `cryptominisat295`
- `minisatPSM`
- `sattime2011`
- `ccasat`
- `glucose_21`
- `glucose_21_modified`.

Each of the solvers was run on every instance with a 5,000 second timeout. We then removed instances that could not be solved by any of the solvers within the allotted time limit. We further removed instances that we deemed too easy, i.e. those where every solver could solve the instance within 15 seconds. This resulted in our final datasets comprising of 1949 Random, 363 Crafted, and 805 industrial instances, i.e. 3117 instances in total.

All the experiments presented in this paper were evaluated using 10-fold cross validation, where we averaged the results over all folds. We then repeat each experiment 10 times to decrease the bias of our estimates of the performance. In our experiments we commonly present both the average and the PAR-10 performance; PAR-10 is a weighted average where every timeout is treated as having taken 10 times the timeout. We also present the percentage of instances not solved.

We evaluate all our results comparing them to three benchmark values:

[1] `http://www.cs.ubc.ca/labs/beta/Projects/SATzilla/`

– *Virtual Best Solver (VBS)*: This is the lower bound of what is achievable with a perfect portfolio that, for every instance, always chooses the solver that results in the best performance.
– *Best Single Solver (BSS)*: This is the desired upper bound, obtained by solving each instance with the solver whose average running time is the lowest on the entire dataset.
– *Instance-Specific Algorithm Configuration (ISAC)*: This is the pure ISAC methodology obtained with the normal set of features and clustering.

Our results will always lie between the *VBS* and the *BSS* with the ultimate goal to improve over *ISAC*. Results will be divided according to their dataset.

4 Applying Feature Filtering

The current standard version of ISAC assumes that all features are equally important. But as was shown in [10], this is often not the case, and it is possible to achieve comparable performance with a fraction of the features, usually also resulting in slightly better overall performance. The original work presented in [10] only considered a rudimentary feed-forward selection. In this paper we utilize four common and more powerful filtering techniques: Chi-squared, information gain, gain ratio, and symmetrical uncertainty. Because all these approaches depend on a classification for each instance, we use the name of the best solver for that purpose.

Chi-squared. The Chi-squared test is a correlation-based filter and makes use of "contingency tables". One advantage of this function is that it does not need the discretization of continuous features. It is defined as:

$$\chi^2 = \sum_{ij} (M_{ij} - m_{ij})^2 / m_{ij} \qquad \text{where } m_{ij} = M_{i.} M_{.j} / m$$

M_{ij} is the number of times objects with feature values $Y = y_j, X = x_i$ appear in a dataset, y_i are classes and x_j are features.

Information gain. Information gain is based on information theory and is often used in decision trees and is based on the calculation of entropy of the data as a whole and for each class. For this ranking function continuous features must be discretized in advance.

Gain ratio. This function is a modified version of the *information gain* and it takes into account the *mutual information* for giving equal weight to features with many values and features with few values. It is considered to be a stable evaluation.

Symmetrical uncertainty. The symmetrical uncertainty is built on top of the mutual information and entropy measures. It is particularly noted for its low bias for multivalued features.

Table 1: Results on the SAT benchmark, comparing the Virtual Best Solver (VBS), the Best Single Solver (BSS), the original ISAC approach (ISAC) and ISAC with different feature filtering techniques: "chi.squared", "information.gain", "symmetrical.uncertainty" and "gain.ratio".

RAND	*Runtime - avg (std)*	*Par 10 - avg (std)*	*% not solved - avg (std)*
BSS	1551 (0)	13154 (0)	25.28 (0)
ISAC	826.1 (6.6)	4584 (40.9)	8.1 (0.2)
chi.squared	1081 (42.23)	7318 (492.7)	14 (1)
information.gain	851.5 (32.33)	5161 (390)	8.7 (0.8)
symmetrical.uncertainty	840.2 (13.15)	4908 (189.5)	8.76 (0.4)
gain.ratio	830.3 (21.3)	4780 (210)	9 (0.4)
VBS	358 (0)	358 (0)	0 (0)
HAND	*Runtime - avg (std)*	*Par 10 - avg (std)*	*% not solved - avg (std)*
BSS	2080 (0)	15987 (0)	30.3 (0)
ISAC	1743 (34.4)	13994 (290.6)	26.5 (0.9)
chi.squared	1544 (37.8)	11771 (435)	23.5 (0.9)
information.gain	1641 (38.9)	12991 (443)	24.3 (0.9)
symmetrical.uncertainty	1686 (27.3)	13041 (336)	25.7 (0.7)
gain.ratio	1588 (43.7)	12092 (545)	22.4 (1)
VBS	400 (0)	400 (0)	0 (0)
INDU	*Runtime - avg (std)*	*Par 10 - avg (std)*	*% not solved - avg (std)*
BSS	871 (0)	4727 (0)	8.4 (0)
ISAC	763.4 (4.7)	3166 (155.6)	5.2 (0.7)
chi.squared	708.1 (25.3)	3252 (218)	5.8 (0.4)
information.gain	712.6 (7.24)	2578 (120)	4.3 (0.3)
symmetrical.uncertainty	716.4 (16.76)	2737 (150)	4.4 (0.3)
gain.ratio	705.4 (19.9)	2697 (284)	4.1 (0.6)
VBS	319 (0)	319 (0)	0 (0)
ALL	*Runtime - avg (std)*	*Par 10 - avg (std)*	*% not solved*
BSS	2015 (0)	4726 (0)	30.9 (0)
ISAC	1015 (10.3)	6447 (92.4)	11.8 (0.2)
chi.squared	1078 (29.7)	7051 (414)	11.79 (0.8)
information.gain	1157 (18.9)	7950 (208)	15 (0.4)
symmetrical.uncertainty	1195 (28.7)	8067 (341)	15.6 (0.7)
gain.ratio	1111 (17.4)	6678 (225)	13.39 (0.5)
VBS	353 (0)	353 (0)	0 (0)

Restricting our filtering approaches to finding the best 15 features, we can see in Table 1 that the results are highly dependent on the dataset taken into consideration. For the random dataset there is no major improvement due to using just a subset of the features. Yet we can achieve almost the same result as the original ISAC by just using a subset of the features calculated using the *gain.ratio* function, a sign that not all the features are needed. We can further

observe the improvements are more pronounced in the hand-crafted and industrial datasets. For them, the functions that give the best results are, respectively, *chi.squared* and *gain.ratio*, but in the latter the result is almost identical to the one given by *chi.squared*.

These results show that not all features are necessary for the clustering, and that it is possible to improve performance through the careful selection of the features. However, we also observe that the improvements can sometimes be minor, and are highly dependent on the filtering approach used and the dataset it is evaluated on.

5 Extending the Feature Space

While the original version of ISAC employs Euclidean distance for clustering, there is no reason to believe that this is the best distance function. As an alternative one might learn a weighted distance metric, where the weights are tuned to match the desired similarity between two instances. For example, if two instances have the same best solver, then the distance between these two instances should be small. Alternatively, when a solver performs very well on one instance, but poorly on another, it might be desirable for these instances to be separated by a large distance.

In our initial experiments, we have trained a distance function that attempts to capture this desired behavior. Yet the resulting performance often times proved worse than the standard Euclidean distance. There are a number of reasons for this. First, while we know that some instances should be closer or farther from each other, the ideal magnitude of the distance cannot be readily determined. Second, the effectiveness of the distance function depends on near perfect accuracy since any mistake can distort the distance space. Third, the exact form of the distance function is not known. It is, for example, possible that even though two instances share the same best solver, they should nevertheless be allowed to be in opposite corners of the distance space. We do not necessarily want every instance preferring the same solver to be placed in the same cluster, but instead want to avoid contradictory preferences within the same cluster.

Due to these complications, we propose an alternate methodology for refining the feature vector. Specifically, we propose to add the normalized performance of all solvers as part of the features. In this setting, for each instance, the best performing solver is assigned a value of -1, while the worst performing is assigned to 1. Everything in between is scaled accordingly. The clustering is then done on both the set of the normal features and the new ones. During testing, however, we do not know the performance of any of the solvers beforehand, so we set all those features to 0.

We see in Table 2 that the performance of this approach (called NormTimes ISAC) was really poor: never comparable with the running times of the normal ISAC. The main reason was that we were taking into consideration too many solvers during the computation of the new features.

Table 2: Results on the SAT benchmark, comparing the Best Single Solver (BSS), the Virtual Best Solver (VBS), the original ISAC approach (ISAC), the ISAC approach with extra features coming from the running times: "NormTimes ISAC" has the normalized running times while "BestTwoSolv ISAC" takes into consideration just the best two solvers per each instance.

RAND	*Runtime - avg (std)*	*Par 10 - avg (std)*	*% not solved - avg (std)*
BSS	1551 (0)	13154 (0)	25.28 (0)
ISAC	826.1 (6.6)	4584 (40.9)	8.1 (0.2)
NormTimes ISAC	1940 (-)	15710 (-)	30 (-)
BestTwoSolv ISAC	825.6 (5.7)	4561 (87.8)	8.1 (0.2)
VBS	358 (0)	358 (0)	0 (0)
HAND	*Runtime - avg (std)*	*Par 10 - avg (std)*	*% not solved - avg (std)*
BSS	2080 (0)	15987 (0)	30.3 (0)
ISAC	1743 (34.4)	13994 (290.6)	26.5 (0.9)
NormTimes ISAC	1853 (-)	14842 (-)	28.3 (-)
BestTwoSolv ISAC	1725 (29.2)	13884 (124.4)	26.5 (0.8)
VBS	400 (0)	400 (0)	0 (0)
INDU	*Runtime - avg (std)*	*Par 10 - avg (std)*	*% not solved - avg (std)*
BSS	871 (0)	4727 (0)	8.4 (0)
ISAC	763.4 (4.7)	3166 (155.6)	5.2 (0.7)
NormTimes ISAC	934.3 (-)	5891 (-)	10.8 (-)
BestTwoSolv ISAC	750.5 (2.4)	2917 (157.3)	4.7 (0.4)
VBS	319 (0)	319 (0)	0 (0)
ALL	*Runtime - avg (std)*	*Par 10 - avg (std)*	*% not solved - avg (std)*
BSS	2015 (0)	4727 (0)	30.9 (0)
ISAC	1015 (10.3)	6447 (92.4)	11.8 (0.2)
NormTimes ISAC	1151 (-)	6923 (-)	12.5 (-)
BestTwoSolv ISAC	1019 (11.5)	6484 (172.3)	11.9 (0.3)
VBS	353 (0)	353(0)	0 (0)

As an alternative, we consider that matching the performance of all solvers is too constraining. Implicitly ISAC assumes that a good cluster is one where the instances all prefer the same solver. For this reason we decided to take into account only the performance of the best two solvers per each instance. This was accomplished by extending the normal set of features with a vector of new features (one per each solver), and assigning a value of 1 to the components corresponding to the best two solvers and 0 to all the others. In the testing set, since we again do not know which are the best two solvers before hand, all the new features are set to the constant value of 0. As can be seen in Table 2, depending on which of the four datasets was used we got different results (this approach is called bestTwoSolv ISAC): we observed a small improvement in the

hand-crafted and industrial datasets, while for the other two datasets the results were almost the same as the pure ISAC methodology.

The drawbacks of directly extending the feature vector with the performance of solvers are two-fold. First, the performance of the solvers is not available prior to solving a previously unseen test instance. Secondly, even if the new features are helpful in determining a better clustering, there is usually a large number of original features that might be resisting the desired clustering. Yet even though these extensions to the feature vector did not provide a compelling case to be used instead of the vanilla ISAC approach, they nonetheless supported the assumption that by considering solver performances on the training data it is possible to improve the quality of the overall clustering. This has inspired the Solver-based Nearest Neighbor Approach that we describe in the next section.

6 SNNAP

There are two main takeaway messages from extending the feature vector with solver performances. First, the addition of solver performances can be helpful, but the inclusion of the original features can be disruptive for finding the desired cluster. Second, it is not necessary for us to find instances where the relation of every solver is similar to the current instance. It is enough to just know the best two or three solvers for an instance. Using these two ideas we propose SNNAP which is presented as Algorithm 2.

During the training phase the algorithm is provided with a list of training instances T, their corresponding features vectors F and the running times R of every solver in our portfolio. We then train a single model PM for every solver to predict the expected runtime on a given instance. We have claimed previously that such models are difficult to train properly since any misclassification can result in the selection of the wrong solver. In fact, this was partly why the original version of ISAC outperformed these types of regression-based portfolios. Clusters provide better stability of the resulting prediction of which solver to choose. We are, however, not interested in using the trained model to predict the single best solver to be used on the instance. Instead, we just want to know which solvers are going to behave well on a particular instance.

For training the model, we scale the running times of the solvers on one instance so that the scaled vector will have a mean of 0 and unitary standard deviation. We saw that this kind of scaling is crucial in helping the following phase of prediction. Thus we are not training to predict runtime. We are learning to predict when a solver will perform much better than usual. Doing so, for every instance, every solver that behaves better than one standard deviation from the mean will receive a score less than -1, the solvers which behaves worse than one standard deviation from the mean a score greater of 1, and the others will lie in between. Random forests [4] were used as the prediction model.

In the prediction phase the procedure is presented with a previously unseen instance x, the prediction models PM (one per each solver), the training instances T, their running times R (and the scaled version $\bar{R}$), the portfolio of

Algorithm 2 Solver-based Nearest Neighbor for Algorithm Portfolios

```
1: function SNNAP-TRAIN(T, F, R)
2:     for all instances i in T do
3:         R̄_i ← Scaled(R_i)
4:     end for
5:     for all solver j in the portfolio do
6:         PM_j ← PredictionModel(T, F, R̄)
7:     end for
8:     return PM
9: end function
```

```
1: function SNNAP-RUN(x, PM, T, R, R̄, A, k)
2:     PR ← Predict(PM, x)
3:     dist ← CalculateDistance(PR, T, R̄)
4:     neighbors ← FindClosestInstances(dist, k)
5:     j ← FindBestSolver(neighbors, R)
6:     return A_j(x)
7: end function
```

solvers A and the size of the desired neighborhood k. The procedure first uses the prediction models to infer the performances PR of the solvers on the instance x, using its originally known features. SNNAP then continues to use these performances to compute a distance between the new instance and every training instance, selecting the k nearest among them. The distance calculation takes into account only the scaled running time of the instances of the training set and the predicted performances PR of the different solvers on the instance x. At the end the instance x will be solved using the solver that behaves best (measured as the average running time) on the k neighbors previously chosen.

It is worth highlighting again that we are not trying to predict the running times of the solvers on the instances but, after scaling, we predict a ranking amongst the solvers on a particular instance: which will be the best, which the second best, etc. Moreover, as shown in the next section, we are not interested in learning a ranking among all the solvers, but just among a small subset of them, specifically for each instance which will be the best n solvers.

6.1 Choosing the Distance Metric

The k-nearest neighbors approach is usually used in conjunction with the weighted Euclidean distance; unfortunately the Euclidean distance does not take into account the performances of the solvers in a way that is helpful to us. What is needed is a distance metric that takes into account the performances of the solvers and that would allow the possibility of making some mistakes in the prediction phase without too much prejudice on the performances. Thus the metric should be trained with the goal that the k-nearest neighbors always prefer to

be solved by the same solver while instances that prefer different solvers are separated by a large margin.

Given two instances a, b and the running times of the m algorithms in the portfolio A on both of them $R_{a_1}, \dots R_{a_m}$ and $R_{b_1}, \dots R_{b_m}$, we identify which are the best n solvers on each $(A_{a_1}, \dots A_{a_n})$ and $(A_{b_1}, \dots A_{b_n})$ and define their distance as a Jaccard distance:

$$1 - \frac{|intersection((A_{a_1}, \dots A_{a_n}), (A_{b_1}, \dots A_{b_n}))|}{|union((A_{a_1}, \dots A_{a_n}), (A_{b_1}, \dots A_{b_n}))|}$$

Using this definition two instances that will prefer the exact same n solvers will have a distance of 0, while instances which prefer completely different solvers will have a distance of 1. Moreover, using this kind of distance metric we are no longer concerned with making small mistakes in the prediction phase: even if we switch the ranking between the best n solvers the distance between two instances will remain the same. In our experiments, we focus on setting $n = 3$, as with higher values the performances degrades.

6.2 Numerical Results

In our approach, with the Jaccard distance metric, for each instance we are interested in knowing which are the n best solvers. In the prediction phase we used random forests which achieved high levels of accuracy: as stated in Table 3 we correctly made 91, 89, 91 and 91% (respectively RAND, HAND, INDU and ALL datasets) of the predictions. We compute these percentages in the following manner. There are 29 predictions made (one per each solver) per each instance, giving us a total of 5626, 1044, and 2320 predictions per category. We define accuracy as the percentage of matches between the predicted best n solvers and the true best n.

Having tried different parameters we use the performance of just the $n = 3$ best solvers in the calculation of the distance metric and a neighborhood size of 60. Choosing a larger number of solvers degrades the results. This is most likely due to scenarios where one instance is solved well by a limited number of solvers, while all the others time out.

As we can see in Table 4 the best improvement, as compared to the standard ISAC, is achieved in the hand-crafted dataset. Not only are the performances

Table 3: Statistics of the four datasets used: instances generated at random "RAND", hand-crafted instances "HAND", industrial instances "INDU" and the union of them "ALL".

	RAND	*HAND*	*INDU*	*ALL*
Number of instances considered	1949	363	805	3117
Number of predictions	5626	1044	2320	9019
Accuracy in the prediction phase	91%	89%	91%	91%

Table 4: Results on the SAT benchmark, comparing the Best Single Solver (BSS), the original ISAC approach (ISAC), our SNNAP approach (SNNAP) (also with feature filtering) and the Virtual Best Solver (VBS)

RAND	*Runtime - avg (std)*	*Par 10 - avg (std)*	*% not solved - avg (std)*
BSS	1551 (0)	13154 (0)	25.28 (0)
ISAC	826.1 (6.6)	4584 (40.9)	8.1 (0.2)
SNNAP	791.4 (15.7)	4119 (207)	7.3 (0.2)
SNNAP + Filtering	723 (9.27)	3138 (76.9)	5.28 (0.1)
VBS	358 (0)	358 (0)	0 (0)
HAND	*Runtime - avg (std)*	*Par 10 - avg (std)*	*% not solved - avg (std)*
BSS	2080 (0)	15987 (0)	30.3 (0)
ISAC	1743 (34.4)	13994 (290.6)	26.5 (0.9)
SNNAP	1063 (33.86)	6741 (405.5)	12.4 (0.4)
SNNAP + Filtering	995.5 (18.23)	6036 (449)	10.5 (0.4)
VBS	400 (0)	400 (0)	0 (0)
INDU	*Runtime - avg (std)*	*Par 10 - avg (std)*	*% not solved - avg (std)*
BSS	871 (0)	4727 (0)	8.4 (0)
ISAC	763.4 (4.7)	3166 (155.6)	5.2 (0.7)
SNNAP	577.6 (21.5)	1776 (220.8)	2.6 (0.4)
SNNAP + Filtering	540 (15.52)	1630 (149)	2.4 (0.4)
VBS	319 (0)	319 (0)	0 (0)
ALL	*Runtime - avg (std)*	*Par 10 - avg (std)*	*% not solved - avg (std)*
BSS	2015 (0)	4727 (0)	30.9 (0)
ISAC	1015 (10.3)	6447 (92.4)	11.8 (0.2)
SNNAP	744.2 (14)	3428 (141.2)	5.8 (0.2)
SNNAP + Filtering	692.9 (7.2)	2741 (211.9)	4.5 (0.1)
VBS	353 (0)	353 (0)	0 (0)

improved by 60%, but also the number of unsolved instances is halved; this also has a great impact on the PAR10 evaluation.[2] It is interesting to note that the hand-crafted dataset is the one that proves to be most difficult, in terms of solving time, while being the setting in which we achieved the most improvement.

We also achieved a significant improvement, although lower than that with the Hand-crafted dataset, on the Industrial and ALL ($\sim$ 25%) datasets. Here the number of unsolved instances was also halved. In the random dataset we achieved the lowest improvement but, yet, we were able to overtake significantly the standard ISAC approach.

[2] PAR10 score is a penalized average of the runtimes: for each instance that is solved within 5000 seconds (the timeout threshold), the actual runtime in seconds denotes the penalty for that instance. For each instance that is not solved within the time limit, the penalty is set to 50000, which is 10 times the original timeout.

Table 5: Matrix for comparing instances solved and not solved using SNNAP and ISAC for the four datasets: RAND, HAND, INDU and ALL. Values are in percentages.

RAND		
SNNAP \ISAC	*Solved*	*Not Solved*
Solved	89.4	3.3
Not Solved	2.5	4.8

HAND		
SNNAP \ISAC	*Solved*	*Not Solved*
Solved	70.2	17.4
Not Solved	3.3	9.1

INDU		
SNNAP \ISAC	*Solved*	*Not Solved*
Solved	93.9	3.5
Not Solved	0.9	1.7

ALL		
SNNAP \ISAC	*Solved*	*Not Solved*
Solved	85.8	8.4
Not Solved	2.4	3.4

We have also applied feature filtering to SNNAP and the results are shown in Table 4. Feature filtering is again proving beneficial, significantly improving the results for all our datasets and giving us a clue that not all 115 features are essential. Results in the table have been reported only for the more successful ranking function (gain.ratio for the Random dataset, chi.squared for Hand-crafted and Industrial and the overAll dataset).

These consistent results for SNNAP are encouraging. In particular, it is clear that the dramatic decrease in the number of unsolved instances is highly important, as they are key to lowering the average and the PAR10 scores. This result can also be observed in Table 5, where we can see the percentage of instances solved/not solved by each approach. In particular the most significant result is achieved, again, in the HAND dataset where the number of instances not solved by ISAC, but solved by our approach is 17.4% of the overall instances, while the number of instances not solved by our approach but solved by ISAC is only 3.3%. As we can see this difference is also considerable in the other three datasets. Deliberately, we chose to show this matrix only for the version of ISAC and SNNAP without feature filtering as it offers an unbiased comparison between the two approaches, as we have shown that ISAC does not improve after feature filtering.

Another useful statistic is represented by the number of times that an approach is able to select the best solver for a given instance. In the random dataset SNNAP is able to select the best solver for 39% of the instances, as compared with 35% for ISAC. For the Hand-crafted dataset those values are 25% and 17%, respectively, for the Industrial 29% and 21%, respectively, and for the ALL 32% and 26%, respectively. These values suggest that ISAC is already behaving well on the RAND dataset and, for this reason, the improvement achieved with our new approach is smaller in that case, while the improvement is more significant

Table 6: Frequencies of solver selections for VBS, ISAC and SNNAP. Results are expressed as percentages and entries with value < 0.5 have been reported as '-'. In bold the top three solvers for each approach are identified. Reported are also the Best Single Solver (BSS) for each dataset. Solvers are: 1: clasp-2.1.1_jumpy, 2: clasp-2.1.1_trendy, 3: ebminisat, 4: glueminisat, 5: lingeling, 6: lrglshr, 7: picosat, 8: restartsat, 9: circminisat, 10: clasp1, 11: cryptominisat_2011, 12: eagleup, 13: gnoveltyp2, 14: march_rw, 15: mphaseSAT, 16: mphaseSATm, 17: precosat, 18: qutersat, 19: sapperlot, 20: sat4j-2.3.2, 21: sattimep, 22: sparrow, 23: tnm, 24: cryptominisat295, 25: minisatPSM, 26: sattime2011, 27: ccasat, 28: glucose_21, 29: glucose_21_modified.

RAND	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
BSS	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	100	-	-
VBS	-	-	-	-	-	-	-	-	-	-	-	17	6	**19**	2	-	-	-	-	-	3	**18**	5	-	-	4	**24**	-	-
ISAC	-	6	1	-	-	-	-	-	-	-	-	8	-	**20**	-	**12**	-	-	-	-	-	-	-	-	-	-	**52**	-	-
SNNAP	-	5	-	-	-	-	-	-	-	-	-	3	1	**27**	2	2	-	-	-	-	0	**12**	4	-	-	2	**41**	-	-
HAND	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
BSS	-	-	-	-	-	-	-	-	-	-	-	-	-	-	100	-	-	-	-	-	-	-	-	-	-	-	-	-	-
VBS	-	3	1	2	2	1	**7**	-	1	**13**	4	-	5	7	8	1	3	2	-	1	**7**	**10**	5	3	1	6	3	2	2
ISAC	-	**22**	2	1	1	1	2	-	-	1	-	2	-	1	**40**	-	-	-	-	-	-	**26**	-	-	-	-	-	-	-
SNNAP	-	10	1	1	1	-	-	-	-	**32**	-	-	-	1	**25**	-	-	-	-	-	2	4	-	4	-	**19**	-	-	-
INDU	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
BSS	-	-	-	-	100	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
VBS	-	5	1	3	**15**	**8**	3	2	5	4	4	-	-	-	3	1	1	9	-	-	-	-	-	4	5	-	-	**20**	3
ISAC	-	1	-	1	**31**	5	-	-	6	-	**17**	-	-	-	-	-	-	5	-	-	-	-	-	-	2	-	-	**32**	-
SNNAP	-	1	-	1	**38**	-	-	-	-	-	2	-	-	-	4	1	-	**19**	-	-	-	-	-	4	-	-	-	**29**	-
ALL	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
BSS	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	100	-	-	-	-	-	-	-	-	-	-	-	-	-
VBS	-	2	-	1	4	2	2	1	1	3	2	11	4	**13**	3	-	1	3	-	-	3	**13**	4	2	1	3	**15**	5	1
ISAC	-	6	1	1	**14**	-	-	-	2	-	-	2	-	12	5	**12**	-	-	-	-	-	4	-	4	-	2	**31**	2	-
SNNAP	-	5	-	1	10	-	-	-	-	3	-	2	1	**18**	2	5	-	3	-	-	-	**11**	2	3	-	2	**24**	4	3

in the other three datasets. These results also show that there is room for further improvement.

The final thing we analyze are the frequencies with which the solvers are chosen by the three strategies: the Virtual Best Solver (VBS), ISAC and SNNAP. Table 6 presents the frequency with which each of the 29 solvers in our portfolio were selected by each strategy, highlighting the best single solver (BSS) for each category. In this table we can see that ISAC tends to favor selecting the Best Single Solver. This is particularly clear in the Hand-crafted dataset: the BSS (15) is chosen only in 8% of the instances by the VBS, while it is the more frequently chosen solver by ISAC. Note also that in the ALL dataset the VBS approach never chooses the BSS, while this solver is one of the top three chosen by ISAC. On the other hand, the more often a solver is chosen by the VBS, the more often it is chosen by SNNAP. This big discrepancy between the VBS and BSS in this dataset is one of the reason for the poorer performance of ISAC and one of the reasons for the improvement observed when using SNNAP.

7 Conclusions

Instance-Specific Algorithm Configuration (ISAC) is a successful approach to tuning a wide range of solvers for SAT, MIP, set covering, and others. This approach assumes that the features describing an instance are enough to group instances so that all instances in the cluster prefer the same solver. Yet there is no fundamental reason why this hypothesis should hold. In this paper we show that the assumptions that ISAC makes can be strengthened. We show that not all employed features are useful and that it is possible to achieve similar performance with only a fraction of the features that are available. We then show that it is possible to extend the feature vector to include the past performances of solvers to help guide the clustering process. In the end, however, we introduce an alternative view of ISAC which uses the existing features to predict the best three solvers for a particular instance. Using k-nearest neighbors, the approach then scans the training data to find other instances that preferred the same solvers, and uses them as a dynamically formed training set to select the best solver to use. We call this methodology Solver-based Nearest Neighbors for Algorithm Portfolios (SNNAP).

The benefit of the SNNAP approach over ISAC is that the cluster formulated by the k-NN comprises of instances that are most similar to the new instance, something that ISAC assumes but has no way of enforcing. Additionally, the approach is not as sensitive to incorrect decisions by the predictive model. For example, it does not matter if the ranking of the top three solvers is incorrect, any permutation is acceptable. Furthermore, even if one of the solvers is incorrectly included in the top n, the k-NN generates a large enough training set to find a reasonable solver that is likely to work well in general.

This synergy between prediction and clustering that enforces the desired qualities of our clusters is the reason that SNNAP consistently and significantly outperforms the traditional ISAC methodology. Consequently, this paper

presents a solver portfolio for combinatorial problem solving that out-performs the state-of-the-art in the area.

Acknowledgements

This work was supported by the European Commission through the ICON FET-Open project (Grant Agreement 284715) and by Science Foundation Ireland under Grant 10/IN.1/I3032.

References

1. SAT Competition 2011. http://www.cril.univ-artois.fr/SAT11/
2. SAT Competitions. http://www.satcompetition.org/
3. Bensusan, H., Giraud-Carrier, C.: Casa Batlo is in Passeig de Gracia or landmarking the expertise space. ECML Workshop on Meta-Learning: Building Automatic Advice Strategies for Model Selection and Method Combination (2000)
4. Breiman, L.: Random forests. Machine Learning 45(1), 5–32 (2001)
5. Hamerly, G., Elkan, C.: Learning the k in k-means. NIPS (2003)
6. Hurley, B., O'Sullivan, B.: Adaptation in a CBR-based solver portfolio for the satisfiability problem. ICCBR (2012)
7. Kadioglu, S., Malitsky, Y., Sabharwal, A., Samulowitz, H., Sellmann, M.: Algorithm selection and scheduling. CP pp. 454–469 (2011)
8. Kadioglu, S., Malitsky, Y., Sellmann, M., Tierney, K.: ISAC – Instance-Specific Algorithm Configuration. ECAI pp. 751–756 (2010)
9. Kotthoff, L., Gent, I., Miguel, I.P.: An evaluation of machine learning in algorithm selection for search problems. AI Communications (2012)
10. Kroer, C., Malitsky, Y.: Feature filtering for instance-specific algorithm configuration. ICTAI pp. 849–855 (2011)
11. Malitsky, Y., Sellmann, M.: Instance-specific algorithm configuration as a method for non-model-based portfolio generation. CPAIOR pp. 244–259 (2012)
12. O'Mahony, E., Hebrard, E., Holland, A., Nugent, C., O'Sullivan, B.: Using case-based reasoning in an algorithm portfolio for constraint solving. AICS (2008)
13. Pfahringer, B., Bensusan, H., Giraud-Carrier, C.: Meta-learning by landmarking various learning algorithms. ICML (2000)
14. Pulina, L., Tacchella, A.: A self-adaptive multi-engine solver for quantified boolean formulas. Constraints 14(1), 80–116 (2009)
15. Silverthorn, B., Miikkulainen, R.: Latent class models for algorithm portfolio methods. AAAI (2010)
16. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Satzilla: Portfolio-based algorithm selection for SAT. CoRR (2011)
17. Xu, L., Hutter, F., Shen, J., Hoos, H.H., Leyton-Brown, K.: Satzilla2012: Improved algorithm selection based on cost-sensitive classification models (2012), SAT Competition