Evolving Instance Specific Algorithm Configuration

Yuri Malitsky and **Deepak Mehta** and **Barry O'Sullivan**
Cork Constraint Computation Centre, University College Cork, Ireland
{y.malitsky | d.mehta | b.osullivan}@4c.ucc.ie

Abstract

Combinatorial problems are ubiquitous in artificial intelligence and related areas. While there has been a significant amount of research into the design and implementation of solvers for combinatorial problems, it is well-known that there is still no single solver that performs best across a broad set of problem types and domains. This has motivated the development of portfolios of solvers. A portfolio typically comprises either many different solvers, instances of the same solver tuned in different ways, or some combination of these. However, current approaches to portfolio design take a static view of the process. Specifically, the design of the portfolio is determined offline, and then deployed in some setting. In this paper we propose an approach to *evolving* the portfolio over time based on the problems instances that it encounters. We study several challenges raised by such a dynamic approach, such as how to re-tune the portfolio over time. Our empirical results demonstrate that our evolving portfolio approach significantly out-performed the standard static approach in the case when the type of instances observed change over time.

Introduction

Combinatorial problems arise in many real world domains like scheduling, planning, formal verification, etc. These problems can be formulated as Constraint Satisfaction Problems (Rossi, van Beek, and Walsh 2006) (CSP), Satisfiability problems (Biere et al. 2009) (SAT) or Mixed Integer Programming (MIP). Regardless of the formulation, many real-world problems are very large, often requiring thousands of variables with hundreds of thousands of constraints. The challenge is to find a satisfying and/or an optimal solution in this potentially large space of possibilities, a task that is known to be NP-Complete.

Due to the prevalence of combinatorial problems and their ability to represent numerous real world scenarios, solving these problems is an active field of research. Many efficient solvers have been developed like GECODE (Schulte, Lagerkvist, and Tack 2006) and CHOCO (CHOCO 2010) as CSP solvers, GLUCOSE (Audemard and Simon 2009) and MINISAT (Een and Sörensson 2005) as SAT solvers, and CPLEX (CPLEX 2003) and GUROBI (Optimization) as MIP solvers. Although these solvers are highly successful and competitive in their respective domains, it has been shown numerous times that even within a single domain, no single solver has the best performance over all instances. In other words, different solvers perform best on different problems.

This discovery has inspired a new line of research into portfolio algorithms which given a new problem instance try to predict which is the best solver to utilize on it. This is the driving idea behind approaches like SATZILLA (Xu et al. 2008), CP-HYDRA (O'Mahony et al. 2008), ISAC (Kadioglu et al. 2010) and 3S (Kadioglu et al. 2011). There are a number of ways that portfolio generation can be approached. One, is to run all the solvers in a portfolio on a benchmark set of training instances. Then using a collection of features that describe the structure of each instance, learn to predict the expected time of a solver on each instance. Therefore when a new instance needs to be solved, the solver with the lowest predicted time is chosen. This was the idea behind SATZILLA prior to 2009. Alternatively, instead of relying on a single decision, approaches like CP-HYDRA and 3S propose a schedule of solvers to run that maximize the probability that an instance would be solved. In 2012, SATZILLA changed its approach to using trees to compare every pair of solvers in its portfolio for a given instance. The solver that was usually predicted to be better was then run on the instance. Finally, approaches like ISAC use the instance feature vectors to cluster the training data into homogeneous groups and then select the best solver for each cluster.

All of the above mentioned portfolio approaches have dominated competitions in their respective categories at some point showing that it is not a good idea to rely on any single solver. The drawback to these approaches however is that once trained the learned portfolios remain static. We call this the one-shot learning approach, as the construction of a portfolio optimizing a given objective function (such as mean runtime, percent of instances solved) is done only once on a chosen set of training instances. The real world however is dynamic and constantly changing. It is possible that over time, the types of problems that a portfolio is expected to solve changes. This might not be a problem if the portfolio has been trained on a large dataset of problems that span the entire problem space. In this case, having seen representa-

tive instances of anything that might come up, the portfolio will be able to make the appropriate decisions. Unfortunately, such a large and all encompassing training set does not always exist. Furthermore, there is no reason why a portfolio approach should not iteratively improve its own performance as new instances become available. Consequently, it may be desirable to have a portfolio approach that evolves based on the set of instances that are solved in the online-phase.

The current state-of-the-art portfolio approaches do not provide any mechanism to evaluate the initial portfolio over time as more and more instances are solved in the online phase. Consequently, no steps are taken to further improve the performance of the initial portfolio based on the instances that are made available after the initial training phase. One possible direct solution to this is periodically relaunching the training approach. However, this can be very computationally expensive, raising the question of *when* is the best time to retrain. We therefore augment an existing portfolio approach, ISAC, to one that evaluates the current portfolio and provides ways to continuously improve it by taking advantage of growing set of available instances, enriched feature set and more up to-date set of available solvers. To the best of our knowledge this paper is the first work regarding the evolving aspect of the portfolio approaches for solving combinatorial optimization problems.

Instance Specific Algorithm Configuration

Instance-Specific Algorithm Configuration (ISAC), is a non-model-based portfolio approach that has been demonstrated to be highly effective in practice (Malitsky et al. 2011). Unlike similar approaches, such as Hydra (Xu, Hoos, and Leyton-Brown 2010) and ArgoSmart (Nikolić, Marić, and Janičić 2009), ISAC does not use regression-based analysis. Instead, it computes a representative feature vector in order to identify clusters of similar instances. The data is therefore clustered into non-overlapping groups and a single solver is selected for each group based on some performance characteristic. Given a new instance, its features are computed and it is assigned to the nearest cluster. The instance is then evaluated with the solver assigned to that cluster.

One of the advantages of ISAC over some of its competitor approaches is that it provides a way to both select a solver from a portfolio and in the absence of a portfolio to dynamically generate one from a single highly parameterized solver. To demonstrate how this is the case, for the remainder of this section, we will refer to a portfolio of solvers as a single solver with a single parameter. The setting of this single parameter determines which of the portfolio solvers to run.

ISAC works as follows (see Algorithm 1). In the learning phase, ISAC is provided with a parameterized solver A, a list of training instances I, their corresponding feature vectors F, and the minimum cluster size c. First, the gathered features are normalized so that every feature ranges from

Algorithm 1 Instance-Specific Algorithm Configuration

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

ISAC-Run(A, x, k, P, C, d, s, t)
$f \leftarrow$ Features(x)
$\bar{f}_i \leftarrow 2(f_i/s_i) - t_i \; \forall \; i$
$i \leftarrow \min_i(||\bar{f} - C_i||)$
return $A(x, P_i)$

$[-1, 1]$, 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 overtuning 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 (Hamerly and Elkan 2003) 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. In our case, we restrict clusters to have at least 50 instances. Beginning with the smallest cluster, the corresponding instances are redistributed to the nearest clusters, where proximity is measured by the Euclidean distance of each instance to the cluster's center. The final result of the clustering is a number of k clusters S_i, and a list of cluster centers C_i. Then, for each cluster of instances S_i, favorable parameters P_i are computed using the instance-oblivious tuning algorithm GGA (Ansótegui, Sellmann, and Tierney 2009).

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.

Evolving ISAC

Like other existing portfolio approaches, ISAC is based on one-shot learning. Once the training data has been clustered and a solver is selected for each cluster, no new learning is performed. This means that even as the learned portfolio interacts and is repeatedly used to solve new instances, that influx of new information is ultimately ignored. This also

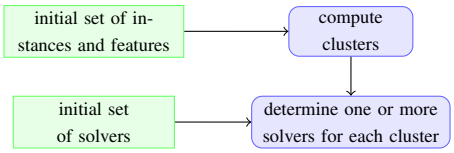

Figure 1: Initial phase of EISAC which is performed offline.

means that if the structure of the problems changes over time, the solvers that have been trained may no longer be suitable. Moreover, as time progresses, new features may be discovered which can have a dramatic impact on the quality of the clustering methods. Furthermore, the set of available solvers can change over time, which can also have a drastic impact on the current portfolio. To the best of our knowledge we are not aware of any portfolio approach for solving combinatorial optimization problems that takes these changes into account and provide solution to recompute the portfolio.

Of course, one solution is to re-launch ISAC from scratch after each such change. In practice, however, this can be very computationally expensive. For example, with 50 seconds timeout for 1000 instances, it can take 72 CPU days for training a portfolio (Malitsky and Sellmann 2012). In the case of a portfolio approach, every solver would need to be run on every instance in the training set. Alternatively, for the case with a parameterized single solver, new parameterizations would need to be tuned, a process that can take days or even weeks depending on the number of training instances, parameters, and the timeout. We therefore propose to extend ISAC to Evolving ISAC (EISAC). This new approach continuously evaluates the current portfolio and provides ways to continuously improve it by taking advantage of growing set of available instances, enriched feature set and more up to-date set of available solvers.

Notice that once the clustering of instances is known, ISAC determines the best solver for each cluster. As the current trend is to create computers with 2, 4 or even 8 cores, it is unusual to find single core machines still in use, and the number of cores will probably continue to double in the coming years. It is for this reason that for EISAC we also consider scenarios where κ CPU cores are available.

Let $\mathcal{I}_t$, $\mathcal{F}_t$, and S_t denote the set of instances, the feature set and the set of solvers known at time t and let $\mathcal{C}_t$ denote the clustering of instances at time t. We assume that each time period is associated with one or more changes in at least one of these sets. Following this notation $\mathcal{I}_0$, $\mathcal{F}_0$, and S_0 denote the initial sets of training instances, features, and solvers respectively.

EISAC is divided in to an initial phase and an evolving phase. Given an initial set of instances $\mathcal{I}_0$, an initial set of features $\mathcal{F}_0$, and κ number of cores, EISAC finds an initial set of clusters $\mathcal{C}_0$ and determines the κ best solvers from set $\mathcal{S}_0$ for each cluster. This process is summarized in Figure 1.

In the evolving phase, EISAC is tasked to solve an influx of

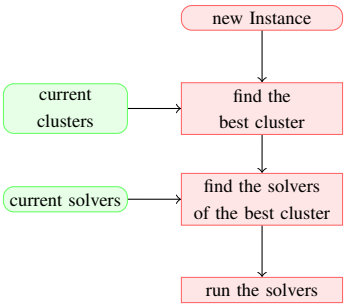

Figure 2: EISAC solving new instances.

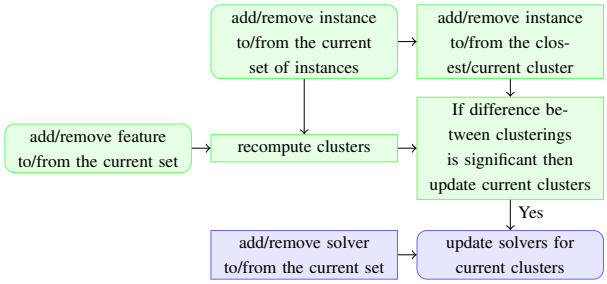

Figure 3: Evolving the ISAC portfolio.

new instances. As these instances come in, they are assigned to their corresponding clusters and are solved by the appropriately assigned solvers. This is illustrated in Figure 2. Once the instance is solved EISAC compares the original clustering with the one obtained by re-clustering all the instances. If the clusterings are sufficiently different, it accepts the new clustering and retrain the portfolio. If the two clusterings are relatively identical, then it sticks to the portfolio that has been used up to now. This process is summarized in Figure 3.

We therefore see that if the original training set is representative of all the new instances, then the new instances fall neatly into the initial clusters. In such a case, it never retrains the portfolio. On the other hand, if the new instances are very different from the initial set, this approach aims to automatically identify the best time to retrain the portfolio. Depending on the time we have to retrain a portfolio or how accurate we want to be, we can modify the threshold at which we consider two clusterings to be similar enough. In our experiments we use adjusted rand index to make this assessment.

As presented, there are a number of decisions that govern how EISAC behaves. These include answering questions like, how many new instances need to be solved before we consider re-clustering? How do we handle the introduction of new solvers in our portfolio, or solvers no longer being available? If we have an infinite influx of new instances, we cannot keep information about all of them in memory as our training set, so how many instances should we keep track of for re-clustering? We explain and discuss some of these issues below.

Updating Clusters

Updating the clusters of ISAC can potentially be very expensive, if we need to train or select a new solver for each cluster. It is therefore imperative to minimize the number of times this operation is performed while still maintaining good expected performance. There are two scenarios when we might wish to consider a new clustering:

- When a new instance is made available and it is added to the current set of instances, and when a number of instances are removed if the total number of instances exceeds the maximum number of instances that can be maintained at any time point.
- A new feature is added to the current set of features or if an existing feature is removed.

In each of these cases one would like to determine whether the existing clustering is still appropriate or if it should be modified.

Let δ be the time difference between the last time when EISAC was activated and the current time. Given a value δ EISAC recomputes the partition of the instances and compares it with the current partition. In the following we describe one way of comparing the similarity between two partitions.

The Rand index (Rand 1971; Hubert and Arabie 1985) or Rand measure (named after William M. Rand) is a measure of the similarity between two data clusterings. Given a set of instances $\mathcal{I}_t$ and two partitions of $\mathcal{I}_t$ to compare, $X = \{X_1, \ldots, X_k\}$ a partition of $\mathcal{I}_t$ into k subsets, and, $Y = \{Y_1, \ldots, Y_s\}$ a partition of $\mathcal{I}_t$ into s subsets, the Rand index is defined as follows:

- Let N_{11} denote the number of pairs of instances in $\mathcal{I}_t$ that are in the same set in X and in the same set in Y.
- Let N_{00} denote the number of pairs of instances in $\mathcal{I}_t$ that are in different sets in X and in different sets in Y.
- Let N_{10} denote the number of pairs of instances in $\mathcal{I}_t$ that are in the same set in X and in different sets in Y.
- Let N_{01} denote the number of pairs of instances in $\mathcal{I}_t$ that are in different sets in X and in the same set in Y.

The Rand index, denoted by R is defined as follows:

$$R = \frac{N_{11} + N_{00}}{N_{11} + N_{00} + N_{10} + N_{01}} = \frac{2(N_{11} + N_{00})}{n * (n-1)}$$

Intuitively, $N_{11} + N_{00}$ can be considered as the number of agreements between X and Y and $N_{10} + N_{01}$ as the number of disagreements between X and Y.

To correct for chance, the adjusted rand index (ARI) then normalizes the rand index to be between -1 and 1 by:

$$\texttt{ARI} = \frac{R - ExpectedIndex}{MaxIndex - ExpectedIndex}$$

If X is the current partition and Y is the new partition and if the Adjusted Rand Index is less than some chosen threshold, denoted by λ, then we replace the current partition of the instances with the new partition. If the current partition is replaced by the new partition then we may need to update the solvers for one or more clusters of the new partition. We experiment with a variety of thresholds and present our findings as part of the numerical results.

Updating Solver Selection for a Cluster

For EISAC we solve the optimization problem as described below to find κ best solvers for each cluster.

Let x_{ij} be a Boolean variable that determines if solver $j \in \mathcal{S}_t$ is chosen for instance i of some cluster C. Let T_{ij} denotes the time required for solving instance i using solver $j \in S_i$. Let y_j be a Boolean that denotes whether solver j is selected for the cluster C. For each instance $i \in C$ exactly one solver is selected:

$$\forall_{i \in C} : \quad \sum_{j \in \mathcal{S}_t} x_{ij} = 1 \tag{1}$$

The solver j is selected if it is chosen for any instance i:

$$\forall_{i \in C} \forall_{j \in \mathcal{S}_t} : x_{ij} \Rightarrow y_j \tag{2}$$

The number of selected solvers should be equal to κ:

$$\sum_{j \in S_t} y_j = \kappa \tag{3}$$

The objective is to minimize the time required to solve all the instances of the cluster, i.e.,

$$\min \sum_{i \in C} \sum_{j \in S_t} T_{ij} \cdot x_{ij} \tag{4}$$

Given a constant κ, the set of solvers $\mathcal{S}_t$ at time t, a cluster $C \in \mathcal{C}_t$, and the maxtrix T, $\texttt{computeBestSolvers}(C, \kappa, \mathcal{S}_t, T)$ denotes the κ best solvers obtained by solving the MIP problem composed of constraints (1)–(3) and the objective function (4). In the following we describe four cases when EISAC might update the set of κ best solvers for a cluster.

Removing Solvers. It may happen that an existing solver is no longer available now, which could happen when a solver is no longer supported by the developers, or when a new release is made then one would like to discard the previous version and update it with the new version. If the removed solver is used by any cluster then one can re-solve the above optimization problem for finding the current κ solvers.

Adding Solvers. When a new solver s is added to the set of solvers $\mathcal{S}_t$ at time t it can have an impact on the current portfolio. One way is to reconstruct the portfolio by running the solver s for all the instances in $\mathcal{I}_t$, which could be time consuming. Therefore, EISAC selects a sample from

each cluster and runs the solver s for only those samples. If adding the new solver to $\mathcal{S}_t$ improves the total execution time of solving the sample instances then EISAC computes the runtime of solver s for all the instances of the corresponding cluster. Otherwise, it avoids running the solver s for the remaining instances of the cluster. In EISAC the sample size for a cluster C is set to $(|\mathcal{I}_0|/|\mathcal{I}_t|) * |C|$. This allows EISAC to maintain the runtimes of all the solvers in $\mathcal{S}_t$ for at least $|\mathcal{I}_0|$ number of instances.

Removing Instances. If the clustering changes because of removing instances or because of change in the feature set, EISAC updates the current set of best solvers for each cluster by re-solving the above optimization problem.

Adding Instances. If the current clustering is replaced with the new clustering because of adding new instances then EISAC would recompute the κ best solvers for one or more clusters of the new clustering. In order to do so, one approach could be to compute the run-times of all the solvers for all the new instances and then recompute the κ best solver for each cluster. However, it would be more desirable if the κ best solvers for a cluster can be computed without computing the run-times of all the solvers for all the newly added instances. We, therefore, propose a lazy approach. The idea is to predict the expected best run-time of each solver for one or more instances whose run-times are unknown and use them to compute the κ best solvers for a cluster. If the newly computed best solvers are same as the previously known best solvers for a cluster then the task of solving new instances using many solvers is avoided. Otherwise, the current set of κ best solvers is updated and the actual run-times which are unknown for the instances are computed using the newly computed κ best solvers. This step is repeated until the newly computed best solvers are same as the previously known best solvers.

Algorithm 2 presents pseudo-code for computing κ best solvers for a cluster C lazily. Let $\mathcal{S}_{ti}$ be the set of solvers at time t for which the run-times are known for solving an instance i. We assume that $\mathcal{S}_{ti}$ is initialized to κ solvers which are used in the on-line phase to solve an instance i. Let $\mathcal{A}_t$ be the set of instances for which the run-times of all the solvers are known at time t (Line 2). Let r_{ip} denotes the p^{th} best solver for instance $i \in \mathcal{A}_t$ based on run-times (Line 3). Let e_p denotes the average ratio between the run-times of the best solver and the p^{th} best solver for each instance $i \in \mathcal{A}_t$ (Line 4). Let $C_u \subseteq C$ be the set of instances for which the run-times of one or more solvers in the set $\mathcal{S}_t$ are unknown (Line 5). The expected best run-time for a solver $j \in \mathcal{S}_t - \mathcal{S}_{ti}$ for an instance $i \in C_u$ is computed in as described below (Lines 6–9). If we assume that $\mathcal{S}_{ti}$ is the set of the $|\mathcal{S}_{ti}|$ worst solvers for instance $i \in C_u$, then the runtime of the best solver, b, of $\mathcal{S}_{ti}$ would have the p^{th} best runtime over all the solvers, where $p = |\mathcal{S}_t| - |\mathcal{S}_{ti}| + 1$, and the expected best run-time of a solver in $j \in \mathcal{S}_t - \mathcal{S}_{ti}$ would be $T_{ib} \cdot e_p$. Notice that different assumptions on the value of p would result in different performance of EISAC. Let N_C be the newly computed best solvers using expected best run-times (Line 10). Let B_C be the previously known κ best solvers for the cluster C. If N_C is same as B_C then the previously known κ solvers are still the best solvers for cluster C even when the known run-times are assumed to be worst, and the computation of the run-times of $\mathcal{S}_t - \mathcal{S}_{ti}$ solvers are avoided for each instance $i \in C_u$ (Lines 11–12). If N_C is different to B_C then the actual run-time of each solver $j \in N_C - \mathcal{S}_{ti}$ is computed for each instance $i \in C_u$, denoted by `computeRuntime`(i, j), and the best solvers for cluster C is updated to N_C (Lines 13–16).

Algorithm 2 updateBestSolvers(C,κ)

1: **loop**
2: $\mathcal{A}_t \leftarrow \{i | i \in \mathcal{I}_t \wedge |\mathcal{S}_{ti}| = |\mathcal{S}_t|\}$
3: $\forall i \in \mathcal{A}_t : T_{ir_{i1}} \leq \ldots \leq T_{ir_{i|\mathcal{S}_t|}}$
4: $\forall p < |\mathcal{S}_t| : e_p \leftarrow \frac{1}{|\mathcal{A}_t|} \sum_{i \in \mathcal{A}_t} \frac{T_{ir_{i1}}}{T_{ir_{ip}}}$
5: $C_u \leftarrow C - \mathcal{A}_t$
6: **for all** $i \in C_u$ **do**
7: $p \leftarrow |\mathcal{S}_t| - |\mathcal{S}_{ti}| + 1$
8: $b \leftarrow \arg\min_{j \in \mathcal{S}_{ti}}(T_{ij})$
9: $\forall j \in \mathcal{S}_t - \mathcal{S}_{ti} : \quad T_{ij} \leftarrow e_p \times T_{ib}$
10: $N_C \leftarrow$ `computeBestSolvers`$(C, \kappa, \mathcal{S}_t, T)$
11: **if** $N_C = B_C$ **then**
12: **return** B_C
13: **for all** $i \in C_u \wedge j \in N_C - S_{ti}$ **do**
14: $T_{ij} \leftarrow$ `computeRuntime`(i, j)
15: $S_{ti} \leftarrow S_{ti} \cup \{j\}$
16: $B_C \leftarrow N_C$

Empirical Results

In this section we demonstrate the effectiveness of using EISAC on SAT as well as MaxSAT instances.

SAT

For our experiments we use the SAT portfolio data made available by the SATzilla team after the 2011 SAT Competition (Data 2011). This dataset provides the runtimes of 31 top-tier SAT solvers with a 1,200 second timeout on over 3,000 instances spread across the Random, Crafted and Industrial categories. After filtering out the instances where every solver times-out, we are left with 2,524 instances. For each of these instances the dataset also provides all of the known SAT features, but we restrict our study to the 52 standard features (Nudelman et al. 2004) that do not rely on local search probing.

We use this dataset to simulate the scenario where instances are made available one at a time. Specifically, we start with a set of $\mathcal{I}_0$ instances for which we know the performance of every solver. Based on this initial set, we generate our initial clusters and select the solver that minimizes the PAR10 score of each cluster. PAR10 being the standard penalized average measurement in SAT where when a solver times out it is penalized as having taken 10 times the timeout to

Table 1: Comparison of performance of ISAC and EISAC on shuffled and ordered datasets using 200 or 500 training instances. We set the minimum cluster size to be either 50 or 100 and the adjusted rand index to either 0.5 or 0.95.

Shuffled		BS	ISAC c50	EISAC c50-λ0.5	EISAC c50-λ0.95	ISAC c100	EISAC c100-λ0.5	EISAC c100-λ0.95	VBS
200	Solved	1760	1776	1753	1759	1776	1752	1752	2324
	% Solved	75.7	76.0	75.4	75.7	76.0	75.4	75.4	100
	PAR10	3001	2923	3037	3006	2923	3038	3038	75.2
	# Train	1	1	275	329	1	166	166	-
500	Solved	1532	1548	1548	1539	1548	1548	1544	2024
	% Solved	75.7	76.4	76.4	76.0	76.4	76.4	76.3	100
	PAR10	3004	2912	2912	2962	2912	2912	2935	74.82
	# Train	1	1	1	674	1	1	104	-
Ordered		BS	ISAC c50	EISAC c50-λ0.5	EISAC c50-λ0.95	ISAC c100	EISAC c100-λ0.5	EISAC c100-λ0.95	VBS
200	Solved	1078	1078	1725	1793	1078	1741	1741	2324
	% Solved	46.3	46.3	74.2	77.2	46.3	74.9	74.9	100
	PAR10	6484	6484	3160	2821	6484	3084	3084	70.42
	# Train	1	1	49	160	1	9	9	-
500	Solved	791	795	1261	1606	817	817	1373	2024
	% Solved	39.1	39.3	62.3	79.3	40.4	40.4	67.8	100
	PAR10	7357	7334	4578	2556	7205	7205	3910	70.79
	# Train	1	1	4	611	1	1	97	-

complete. We then add δ instances to our dataset, evaluate them with the current portfolio, and then evaluate whether we should retrain. We use two thresholds for the adjusted rand index, 0.5 and 0.95. Simulating the scenario where we can only keep a certain number of instances for the retraining, once we add the δ new instances, we also remove the oldest δ instances.

We first consider the scenario where all the instances are shuffled and come randomly. We then also consider an ordering on the data, where first we iterate through the Industrial instances, followed by the crafted, and finally the instances that were randomly generated. This last experiment is meant to simulate the case where instances change over time. This is also the case where traditional portfolio approaches would fail because eventually they are tasked to solve instances they have never observed during training.

Table 1 presents our first test case where the instances come from a shuffled dataset. This is the typical case observed in competitions, where a representative set of the test data is available for training. The table presents the performance of a portfolio which have been given 200 or 500 training instances. The single best solver (BS) is choosing a single solver during training and then always using it during the test phase. Alternatively, the virtual best solver (VBS) is an oracle portfolio that for every instance always runs the best solver. The VBS represents the limit of what can be achieved by a portfolio. We also evaluate ISAC-c50 and ISAC-c100, trained with a minimum of 50 (respectively 100) instances in each cluster. Note that in this setting ISAC is performing better than BS. It is also important to note here that in the 2012 SAT Competition, the difference between the winning and second placed single engine solver was 3 instances and only 8 instances between the top 4 solvers. Therefore the improvement of 16 instances when training on 500 instances is significant. When compared to ISAC on this shuffled data, we see that EISAC is performing comparably to ISAC, although requiring significantly more training sessions. For each version of EISAC in the table we present the minimum cluster size and the adjusted rand index threshold. So EISAC-c100-λ0.95 has clusters with at least 100 instances and retrains as soon as the adjusted rand index drops below 0.95.

This comparable performance on shuffled data is to be expected. As the data is coming randomly, the initial training data was representative enough to capture the diversity. And even if the clusters change a little overtime, the basic assignment of solvers to instances doesn't really change. Note that the slight degradation between for the higher threshold in EISAC-c100 for 500 training instance, can likely be attributed to overtuning (or overfitting) in the numerous retraining steps. Also note that the lower performance for 500 training instances is misleading, since by adding 300 instances to our training set, we are removing 300 instances from the test set.

Table 2: Comparison of performance on ordered dataset using an approximation learning technique.

		BS	ISAC c50	EISAC c50-λ0.5	EISAC+ c50-λ0.5	VBS
200	Solved	1078	1078	1725	1671	2324
	PAR10	6484	6484	3160	3440	70.42
	# Train	1	1	49	44	-
	% Eval	100	100	100	59.5	-
500	Solved	791	795	1261	1264	2024
	PAR10	7357	7334	4578	4561	70.79
	# Train	1	1	4	3	-
	% Eval	100	100	100	83.8	-

The story becomes significantly different if the data is not shuffled as is the case at the bottom of Table 1. Here we see that the clusters and solvers chosen by ISAC initially are ill equipped to solve the future instances. EISAC on the other hand, is able to adapt to the changes and outperform ISAC by almost a factor of 2 in terms of the instances solved. What is also interesting is that for the case of 500 training

Table 3: SAT: Comparison of performance of ISAC, EISAC and EISAC+. on data where instances become progressively harder. Training set is composed of either 200 or 500 instances, the minimum cluster size to be 50 and the adjusted rand index is set to either 0.5 or 0.95.

Easy to Hard		BS	ISAC c50	EISAC c50-λ0.5	EISAC c50-λ0.95	EISAC+ c50-λ0.5	EISAC+ c50-λ0.95	VBS
200	Solved	1060	1035	1690	1823	1698	1741	2324
	% Solved	45.6	44.5	72.7	78.4	73.1	74.9	100
	PAR10	6621	6746	3383	2710	3343	3122	82.51
	# Train	1	1	58	155	57	185	-
	% Eval	100	100	100	100	83.8	83.3	-
500	Solved	1410	1057	1485	1526	1400	1532	2024
	% Solved	69.7	52.2	73.4	75.4	69.2	75.7	100
	PAR10	3753	5850	3322	3075	3811	3041	94.4
	# Train	1	1	4	611	3	663	-
	% Eval	100	100	100	100	87.8	83.1	-

instances and small clusters, this performance is achieved with only four re-training steps.

Table 2 examines the effectiveness of our proposed training technique. Instead of computing the time of every solver on every training instance during re-tuning, we lazily fill in this data until we converge on the expected best solver for a cluster. We call this approach EISAC+. Due to space limitations, we only present a comparison on the ordered dataset and for algorithms tuned with a minimum cluster size of 50. What we observe is that while performance is maintained with this lazy selection, we cut down the number of evaluations we need to 80% and occasionally to as low as 50%. This means that we can potentially speed up each training stage by a factor of 2 while still maintaining nearly identical performance.

Finally, Table 3 examines the effectiveness of EISAC and EISAC+ on a dataset where instances become progressively harder. Specifically we order the 2524 instances at our disposal according to the average runtime of all solvers. The first 200 (500) are used for training, and each instance is on average harder than the last. From the table we see that the regular version of ISAC struggles with this dataset, performing worse than the best single solver. This suggests that the structure of the easier instances is different from that of the harder instances. If we allow our portfolio to adapt to the changes, it is able to perform significantly better than the best single solver. Furthermore, EISAC+ is able to perform comparably to its regular counterpart but with fewer evaluations at each training stage.

We have also tried differing the frequency, δ, at which we consider retraining. The results presented in this paper are for cases where $\delta = 1$, but we also tried setting it to 10, 50, and 100. The results and trends presented, however, do not change significantly due to this change.

MaxSAT

Additionally, we explored how the Evolving ISAC approach behaves on the MaxSAT dataset, employing the instances from the 2012 MaxSAT Competition. This competition had 4 major categories: MaxSAT, Partial MaxSAT, Weighted MaxSAT, and Partial Weighted MaxSAT. Combining all these sets, results in 2681 instances. What differentiates this merged dataset from the one we had for SAT, is that the solvers used for each of the different types of instances are different. So while a solver can perform very well on one category, it might completely fail to read in instances of another type. In our case, whenever a solver is not able to solve an instance, it is marked as if it timed out.

For our solvers we employed 14 top-tier MaxSAT solvers with a 2,100 second timeout. Meanwhile the features we used are based on the original base SAT features plus a five features examining the weights of the soft clauses. Particularly what percentage of the clauses are soft, and what is the mean, standard deviation, minimum and maximum of the weights of these soft clauses.

Our experiments explore three scenarios. First, we take a look at the standard competition, train-once, situation where the training set is representative of the data to be seen in the test set. We can see under the rows labeled "Shuffled" in Table 5, that the adaptive approach while better than the best single solver, is not better than the plain vanilla version of ISAC. This however, quickly changes if the data is introduced first from the MaxSAT (MS) category, followed by Partial MaxSAT (PMS), then Weighted MaxSAT (WMS), and finally Partial Weighted MaxSAT (PWMS). These results are presented in the rows under "Ordered by Category" in Table 5. These experiments exemplify why it is necessary to continuously refine ones portfolio approach. The regular ISAC approach performs worse than the single best solver, since it is unable to generalize the portfolio it has learned to instances that it has never seen before. The EISAC approach however can cope with the changes. It is also important to note here, that EISAC+ is able to achieve nearly the same performance as EISAC, with half the number of evaluations during the tuning stage.

The third set of rows in Table 5, labeled "Easy to Hard" refers to a data set where the instances arrive in order of difficulty, from easy to hard. Here we define easy as an instance whose average runtime over all solvers in our portfolio is smallest. Here, we again see that the regular version of ISAC struggles to find a portfolio that generalizes to the harder instances. EISAC on the other hand, can double the number of instances solved.

Table 4: MaxSAT: Comparison of performance of ISAC, EISAC and EISAC+. on shuffled data (Shuffled), Ordered by Category (Ordered), and Ordered by difficulty (Easy to Hard) datasets using 200 or 500 training instances. We set the minimum cluster size to be 50 and the adjusted rand index to either 0.5 or 0.95.

Shuffled		BS	ISAC c50	EISAC c50-λ0.5	EISAC c50-λ0.95	EISAC+ c50-λ0.5	EISAC+ c50-λ0.95	VBS
	Solved	1472	1802	1540	1544	1502	1479	2260
	% Solved	59.3	72.6	62.1	62.2	60.5	59.6	91.1
200	PAR10	8187	5675	7640	7609	7949	8132	1831
	# Train	1	1	908	1266	872	1259	-
	% Eval	100	100	100	100	95.2	93.4	-
	Solved	1298	1724	1564	1555	1474	1475	1986
	% Solved	59.5	79.0	71.7	71.3	67.6	67.6	91.1
500	PAR10	8149	4252	5722	5800	6541	6532	1836
	# Train	1	1	495	1811	512	1843	-
	% Eval	100	100	100	100	93.2	83.8	-
Ordered by Category		BS	ISAC c50	EISAC c50-λ0.5	EISAC c50-λ0.95	EISAC+ c50-λ0.5	EISAC+ c50-λ0.95	VBS
	Solved	1466	1087	1830	1958	1741	1796	2268
	% Solved	59.1	43.8	73.8	78.9	70.2	72.4	91.4
200	PAR10	8236	11275	5325	4287	6033	5584	1764
	# Train	1	1	266	557	255	529	-
	% Eval	100	100	100	100	50.4	48.9	-
	Solved	1247	1131	1314	1555	1261	1422	2049
	% Solved	57.1	51.9	60.2	71.3	57.8	65.2	93.9
500	PAR10	8607	9656	7990	5789	8480	7006	1245
	# Train	1	1	145	871	155	866	-
	% Eval	100	100	100	100	58.3	56.9	-
Easy to Hard		BS	ISAC c50	EISAC c50-λ0.5	EISAC c50-λ0.95	EISAC+ c50-λ0.5	EISAC+ c50-λ0.95	VBS
	Solved	837	875	1520	1604	1509	1544	2241
	% Solved	33.7	35.2	61.2	64.6	60.8	62.2	90.3
200	PAR10	13297	12987	7812	7137	7902	7539	1988
	# Train	1	1	266	557	263	532	-
	% Eval	100	100	100	100	83.6	82.5	-
	Solved	539	687	1401	1403	1260	1352	1941
	% Solved	24.7	31.5	64.2	64.3	57.8	62.0	89.0
500	PAR10	15103	12742	7226	7207	8519	7674	2262
	# Train	1	1	145	871	123	854	-
	% Eval	100	100	100	100	88.0	88.4	-

Table 5: MaxSAT: Comparison of performance of ISAC, EISAC and EISAC+. on shuffled data (Shuffled), Ordered by Category, and Ordered by difficulty datasets using 200 or 500 training instances. We set the minimum cluster size to be 50 and the adjusted rand index to either 0.5 or 0.95.

Conclusions

We presented the importance of an evolving portfolio approach when handling datasets that can change over time. Specifically, we showed how or proposed changes can create EISAC, a portfolio approach that solves twice as many instances as its unmodified counterpart ISAC. We further showed how a lazy training method can help to significantly reduce the number of solver/instance combinations that need to be evaluated before the best solver for a cluster is selected.

As part of future research, EISAC can be further extended to handle all the scenarios mentioned in this paper. We would, however, like to argue that these extensions will not change the trends presented in this paper. Adding or removing features, if done properly, during the online stage of EISAC will only improve the quality of the clusters. And while a retuning might be required, this would be a one time modification since the creation of new features is not a frequent occurrence. The same logic follows for the addition and removal of solvers. If a solver is not used by any cluster, removing it has no effect on performance. Otherwise, a single retraining step might be necessary.

We would like to allow the possibility of maintaining more instances than those that are provided in the initial phase which could further improve the performance of EISAC. Currently the offline effort is measured coarsely which indicates the number of times EISAC is activated. Notice that many times EISAC just recomputes the clusters but no work is done on reconstructing the portfolio. Also in many occasions it only updates the best solver for only 1 or 2 clusters. Therefore, we would also like to devise more refined ways of computing the offline effort. Finally, it would be interesting to investigate the performance of EISAC by computing more than one solver for each cluster.

Acknowledgements

This work is supported by Science Foundation Ireland Grant No. 10/IN.1/I3032 and by the European Union FET grant (ICON) No. 284715.

References

Ansótegui, C.; Sellmann, M.; and Tierney, K. 2009. A gender-based genetic algorithm for the automatic configuration of algorithms. In Gent, I. P., ed., *CP*, volume 5732 of *Lecture Notes in Computer Science*, 142–157. Springer.

Audemard, G., and Simon, L. 2009. Glucose: a solver that predicts learnt clauses quality. *SAT Competition* 7–8.

Biere, A.; Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T. 2009. *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*. Amsterdam, The Netherlands, The Netherlands: IOS Press.

CHOCO, T. 2010. CHOCO: An open source java constraint programming library. *Ecole des Mines de Nantes. http://www.emn.fr/z-info/choco-solver/pdf/choco-presentation.pdf*.

CPLEX, S. 2003. Ilog. *Inc., Armonk, NY*.

Data, S. 2011. http://www.cs.ubc.ca/labs/beta/Projects/SATzilla/.

Een, N., and Sörensson, N. 2005. MINISAT: A SAT solver with conflict-clause minimization. *Sat* 5.

Hamerly, G., and Elkan, C. 2003. Learning the k in k-means. In *In Neural Information Processing Systems*, 2003. MIT Press.

Hubert, L., and Arabie, P. 1985. Comparing partitions. *Journal of Classification* 2(1):193–218.

Kadioglu, S.; Malitsky, Y.; Sellmann, M.; and Tierney, K. 2010. ISAC - instance-specific algorithm configuration. In Coelho, H.; Studer, R.; and Wooldridge, M., eds., *ECAI*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, 751–756. IOS Press.

Kadioglu, S.; Malitsky, Y.; Sabharwal, A.; Samulowitz, H.; and Sellmann, M. 2011. Algorithm selection and scheduling. In *Proceedings of the 17th international conference on Principles and practice of constraint programming*, CP'11, 454–469. Berlin, Heidelberg: Springer-Verlag.

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

Malitsky, Y.; Sabharwal, A.; Samulowitz, H.; and Sellmann, M. 2011. Non-model-based algorithm portfolios for SAT. In *SAT*, 369–370.

Nikolić, M.; Marić, F.; and Janičić, P. 2009. Instance-based selection of policies for SAT solvers. In *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing*, SAT '09, 326–340. Berlin, Heidelberg: Springer-Verlag.

Nudelman, E.; Devkar, A.; Shoham, Y.; and Leyton-Brown, K. 2004. Understanding random SAT: Beyond the clauses-to-variables ratio. In *In Proc. of CP-04*, 438–452.

O'Mahony, E.; Hebrard, E.; Holland, A.; Nugent, C.; and O'Sullivan, B. 2008. Using case-based reasoning in an algorithm portfolio for constraint solving. In *Irish Conference on Artificial Intelligence and Cognitive Science*.

Optimization, G. Inc.: Gurobi optimizer reference manual, 2012.

Rand, W. 1971. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association* 66(336):846–850.

Rossi, F.; van Beek, P.; and Walsh, T. 2006. *Handbook of Constraint Programming*. Foundations of Artificial Intelligence. New York, NY, USA: Elsevier.

Schulte, C.; Lagerkvist, M.; and Tack, G. 2006. Gecode. *Software download and online material at the website: http://www. gecode. org*.

Xu, L.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2008. SATzilla: portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research* 32(1):565–606.

Xu, L.; Hoos, H. H.; and Leyton-Brown, K. 2010. Hydra: Automatically configuring algorithms for portfolio-based selection. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 179–184.