CORE Metadata, citation and similar papers at core.ac.uk
Provided by Stirling Online Research Repository

Relating Training Instances to Automatic Design of Algorithms for Bin Packing via Features (Detailed Experiments and Results)

Alexander E.I. Brownlee
Division of Computing Science and Mathematics
University of Stirling
Stirling, UK
alexander.brownlee@stir.ac.uk

John R. Woodward
School of Electronic Engineering and Computer Science
Queen Mary University of London
London, UK
j.woodward@qmul.ac.uk

Nadarajen Veerapen
Division of Computing Science and Mathematics
University of Stirling
Stirling, UK
nadarajen.veerapen@cs.stir.ac.uk

ABSTRACT

Automatic Design of Algorithms (ADA) shifts the burden of algorithm choice and design from developer to machine. Constructing an appropriate solver from a set of problem instances becomes a machine learning problem, with instances as training data. An efficient solver is trained for unseen problem instances with similar characteristics to those in the training set. However, this paper reveals that, as with classification and regression, for ADA not all training sets are equally valuable.

We apply a typical genetic programming ADA approach for bin packing problems to several new and existing public benchmark sets. Algorithms trained on some sets are general and apply well to most others, whereas some training sets result in highly specialised algorithms that do not generalise.

We relate these findings to features (simple metrics) of instances. Using instance sets with narrowly-distributed features for training results in highly specialised algorithms, whereas those with well-spread features result in very general algorithms. We show that variance in certain features has a strong correlation with the generality of the trained policies.

Our results provide further grounding for recent work using features to predict algorithm performance, and show the suitability of particular instance sets for training in ADA for bin packing.

KEYWORDS

Automatic design of algorithms; features; bin packing

This technical report is provided in support of the paper "Relating Training Instances to Automatic Design of Algorithms for Bin Packing via Features" published at GECCO 2018, Kyoto, Japan.

1 INTRODUCTION

The Automatic Design of Algorithms (ADA) seeks to build algorithms, which perform better than human designed algorithms. The algorithms are trained on one set of problem instances, where the algorithm undergoes adaptation (i.e. learning), and are then applied to another set of unseen problem instances. ADA has been employed to design algorithms for domains such as decision tree induction, probability distributions, scheduling and others [6, 14, 17]. Recent research has shown how features of the problem instances (e.g. metrics computed over the instance, or results of probing by heuristics) can be exploited as part of the process to automatically configure existing algorithms [4], select heuristics [21, 24, 38] or predict runtime for algorithm variants [18, 19].

In practice, ADA is a machine learning procedure, with training and test instances. It is well known that representative data is needed if a machine learning algorithm is to generalise well from the training data. By *representative*, we mean that the training data contain enough information to induce an underlying rule that captures what it means for data to belong to the specific problem.

In this paper we investigate the relationship between training data and ADA, in terms of features of the training instances, for the well known combinatorial optimisation problem of bin-packing. We use a typical genetic programming approach to generate packing policies for several benchmark instance sets. These policies are then applied to all the instances from all the benchmark sets. We compare the performance of the generated policies across all instances with the features of the benchmark sets that were used to train the policies. While standard benchmark problem instances are often employed in the optimisation literature, and often for training and testing in ADA, this paper demonstrates that not all problem instance sets are equal, at least for bin packing. Therefore one should choose problem instances which are representative of the types of problems one is trying to solve.

The contributions of this paper are:

(1) an analysis of a large number of benchmark instances for bin packing, used as training data for ADA;
(2) the insight that training sets with more variation lead to better trained packing policies;
(3) the observation that variance in certain bin packing instance features (3, 4, 5, 6 from Table 2 for fitting to training data; 2 and 11–21 for generally performing policies) is most helpful for ADA;
(4) two new sets of benchmark bin packing instances.

We start in Section 2 by summarising related work in bin packing, ADA and features for optimisation. In Section 3 we describe our methodology, including our method for automatically designing packing policies using Genetic Programming (GP), a description of the benchmark instance sets that we used in our experiments, and a summary of numeric features calculated for the instances. In Section 4 we present the results of the study: how the ADA-generated algorithms performed on the instances; how much the features varied within each instance set; and connections between performance and features. Finally, we draw our conclusions in Section 5 and consider the implications of the work.

2 RELATED WORK

2.1 Bin Packing and Metaheuristics

Bin Packing is a well-known combinatorial NP-hard problem, with applications from dividing resources between physical containers, to distributing processing tasks in cloud computing. The goal of bin-packing is to pack items of various sizes into bins of a fixed size, using the fewest bins.

Metaheuristics have been used to successfully tackle bin-packing for some time [7, 12, 13, 26], and has continued to attract attention.

Sim et al [30] introduce a method inspired by Artificial Immune Systems which learn continuously over time in a lifelong learning paradigm. As the system develops, it maintains a small set of heuristics which can adapt to incoming problem instance. Gomez and Terashima-Marín [15] use a multi-objective approach based on evolutionary computation, minimizing the number of bins used to accommodate the items, and the total time required to make the allocations. They produce sets of variable length rules representing hyper-heuristics.

While some papers describe algorithms which operate directly on items to be packed, other papers employ hyper-heuristics which operate on the space of rules. These rules then operate on the items themselves. While these rules have been represented using Genetic Programming, they have also been represented as look up tables [2]. This has the advantage that a single look up table (or matrix as they call it) represents a single rule. However one drawback of this approach is that there are a large number of gaps in the matrix. A simpler approach which overcomes this is to use splines [11]. These papers are relevant to the automated design of algorithms, as many of them deliver a competitive algorithm for packing, based either on components of existing heuristics, or by altering part of an existing algorithm.

2.2 Features for Optimisation

The use of features to classify problems and estimate algorithm performance is currently an active area of research. Considerable success has been reported in using machine learning combined with algorithm portfolios to choose efficient solvers for challenging combinatorial problems such as SAT [21, 24, 38]. A similar approach was used to configure parameters for an existing algorithm for continuous black-box optimisation problems [4], and for algorithm runtime prediction [18] for SAT, MIP and TSP. One study has looked at the use of features for bin packing to analyse algorithm performance [19]. More general work in estimating problem hardness or difficulty includes [31, 33, 34]; this has also been extended to evolving new test instances with different properties [32].

For all of these methods to realise their full potential, it is critical that the relationship between instance features and performance is understood. There is an underlying assumption with these approaches that instances with similar features have similar properties: that algorithms good at one instance will be good at another with similar features. This is certainly the case for ADA: we make the assumption that a training set of instances with similar features will yield an automatically designed algorithm suitable for similar instances. Consequently our experiments focus on this: automatically designing packing policies using one set of instances, and testing them on other sets of instances. The hypothesis is that ADA trained algorithms will perform similarly well on benchmark sets with similar distributions of features, and so sets with a wide spread of features will result in more generally applicable algorithms.

3 METHODOLOGY

In applying ADA to any problem, the goal is to design an algorithm that performs well on a set of instances. For bin packing, the goal is to minimise the total number of bins used to pack all items across all the instances in the training set. In this work we are not seeking to outperform the state of the art – rather, we are seeking to gain understanding of the ADA process for a typical approach. We adopt a straightforward framework in which genetic programming (GP) is used to evolve a *packing policy*: a function that gives a score s to all current bins (including a to-be-used empty bin), given an item to be placed. The item is then placed in the bin with the highest score (after excluding bins for which there is not enough remaining capacity to hold the item). If there is a tie in the scores, the first bin with that score is chosen.

A simple policy might be what is referred to as *best-fit.* With this, the score is:

$$s = \begin{cases} 0 & \text{if} \quad i > c \\ \frac{1}{1+(c-i)} & \text{otherwise} \end{cases}$$

where c is the bin capacity and i the item's size. That is, choose the bin with the least capacity remaining after the item is placed in it. An alternative might be worst-fit, which always chooses the bin which would leave the largest space. This will always be the to-be-used empty bin, so will maximise the number of bins used and hence provide a useful baseline.

We will later refer to *scaled-fit*, a generalisation of best-fit. Each scaled-fit policy has a threshold parameter $0 \leq \tau \leq 1$. When finding a bin into which to place an item, the score for each bin is equal to $abs(\tau c - i)$ where c is bin capacity and i the item's size. τc is an ideal remaining capacity to be left in a bin after an item is placed. So when $\tau = 0$, this becomes best-fit, and when $\tau = 1$ it becomes worst-fit.

3.1 GP applied to bin packing

Our approach for automatically designing packing policies used the GP terminals and operators specified in Table 1. Within this framework, the three simplest programs that GP can find are:

constant (the constant might be either a Real value, or BinCap, with bin capacity for the instance) this means all bins score the same, which corresponds to another simple policy *first-fit*: always choose the first bin in the list of bins into which the item fits

RemCap bins are scored according to remaining capacity, which is equivalent to *worst-fit*

constant-RemCap bins are scored inversely to remaining capacity, which is equivalent to *best-fit*

The fitness score to be minimised targeted policies that find the minimal number of bins while avoiding bloat. The integer part of the fitness score was the total number of bins used by the policy to pack all items across all instances in the training set. The fractional part was the number of nodes in the GP tree, normalised to (0,1).

Table 1: Terminals and operators used in GP

Name	Description
Multiply	Product of 2 child nodes
Add	Sum of 2 child nodes
Subtract	Left child node minus right child node
Max2	Maximum of 2 child nodes
Min2	Minimum of 2 child nodes
Square	Single child node squared
DoubleERC	Randomly generated constant (0-1)
RemCap	Remaining bin capacity if item is added
BinCap	Bin capacity

Our experiments were in two stages. Stage (1). For each training set, the GP was repeated 30 times, generating 30 packing policies tailored to that training set. Stage (2). The evolved policies from stage (1) were each applied to all 3581 instances from all benchmark sets in the study. The number of bins required by each policy on each instance was recorded. Unlike stage (1), application of the packing policies to instances is deterministic, so stage (2) did not require multiple repeats.

Evolution in Stage (1) was performed using the EpochX [25] library. The population size was 1000, and evolution was terminated after 100 generations. Otherwise the GP parameters were as per the EpochX defaults. Some empirical exploration was used to determine these parameters (in particular, longer runs with smaller populations were tried), and these were found to yield reasonable results. Extensive parameter tuning was not deployed since the focus here is on the effect of the training sets, not the GP algorithm.

3.2 Benchmark Instances

A wide range of benchmark instances were selected from the literature to be used as training and test data for this study. These were: 2cbp [1]; Augmented irup, Augmented non-irup and Random [10]; bw-2bp [5]; falkenauer-t and falkenauer-u [12]; hard28 [27]; mv-2bp [20]; orlib [3][1]; scholl 1, 2 and 3 [28]; schwerin 1, 2 [29]; and waescher [37]. We also devised two new sets of benchmarks, described in Section 3.3. Where instance sets were for 2-dimensional bin packing, the second dimension was ignored .

3.3 Stirling Instances

We also introduced a new set of benchmark instances that we will collectively refer to as the Stirling Instances. These were intended to provide varying levels of difficulty by allowing items to take sizes in bands defined as a fraction of the bin capacity. Some of the instances allow a wide spread of item sizes, from zero up to the full bin capacity; others allow only very small items or very large items. The largest ones always require n bins, serving as a useful benchmark for which all algorithms will perform equally well (at least, in terms of the number of bins used: how efficiently they find this solution will of course vary).

These instances were generated in a similar fashion to [8]. Each instance is parametrised by a bin capacity c, number of items n, and lower/upper bounds on the item sizes l and u. Item sizes are sampled uniformly at random in this range. In generating our instances, we used two bin capacities: 100 and 150. The latter was following [8], the former so that the item size bands reached the bin capacity. For both bin sizes, (lower bound, upper bound) for item sizes were: 0,100; 0,50; 51,100; 0,25; 26,50; 51,75; 76,100. We refer to the sets as stirling_generated_binCapacity_lb_ub.

[1]It is worth noting that orlib instances are actually a combination of falkenauer-t and falkenauer-u

Table 2: Instance features

Feature	Description
1.	All item sizes in the instance are integers (rather than fractional): True/False
2.	Number of items in the instance
3.	Mean item size divided by the bin capacity
4.	Standard deviation in the item sizes divided by bin capacity
5.	Information entropy in the item sizes, divided by bin capacity
6.	Maximum item size divided by bin capacity
7.	Minimum item size divided by bin capacity
8.	Median item size divided by bin capacity
9.	Maximum item size divided by minimum item size
10.	Compression ratio for the list of item sizes (applying gzip to the string representation of the list)
11-21.	The scaled-fit "performance features", with τ increasing from 0 to 1 in 0.1 increments

3.4 Instance Features

Our experiments consider the distributions of values for features over the instances in benchmark sets, and how these distributions relate to the performance of algorithms built by ADA.

The features we used are listed in Table 2. We refer to features 1-10 as *static features*; features that are constant with respect to a given bin packing instance (an instance being a bin capacity, and a list of items to be packed). Most of these static features were taken from [19], where they were used for constructing a regression model to estimate algorithm performance.

In several other works using features to estimate algorithm performance for other combinatorial problems (e.g. [24]), the behaviour of some simple local search or other heuristic methods was used for some features. We refer to features of this kind as *performance features*. In our experiments, these were the number of bins found when applying scaled-fit policies with τ values from 0 to 1 inclusive, in 0.1 increments.

These features can be used to visualise the sets of instances, following the well-known work of Smith-Miles et al [32, 34]. We use the non-linear dimensionality reduction technique called t-Distributed Stochastic Neighbor Embedding (t-SNE) [35] to plot the instances in two dimensions. This technique has become fairly popular, especially for machine learning datasets, because, in general, it is able to preserve much of the local and global structure of the data, it is not limited to linear relationships like the more traditional Principal Components Analysis and it scales well on larger datasets. t-SNE minimises the divergence between the distribution that measures pairwise similarities of high-dimensional points and the one that measures pairwise similarities of the corresponding low-dimensional points. The latter distribution is computed as a normalised Student-t kernel with a single degree of freedom. Since the normalized Student-t kernel has heavy tails, this allows for dissimilar points to be modelled by low-dimensional counterparts that are also far apart. This creates more space to accurately model small pairwise distances, or local structure, in the low dimensional space. Since the features span a number of types and domains, dissimilarity between points is measured using the Gower distance [16] which is a linear combination of appropriate distance metrics for each data type and scaled to fall between 0 and 1.

Our later analysis will focus on statistical analysis of the results to draw our conclusions, but these illustrations serve as a helpful guide to the overall distribution of the instances in terms of their features (i.e. in *feature space*). Figure 1 shows all 3581 instances from the sets we studied, rendered using t-SNE over the features listed in Table 2. At a high level we can observe that some sets have a wide spread of features among their instances (e.g. Scholl 1 and Scholl 2); some have very little variation in their features (e.g. Schwerin 1 and Schwerin 2); and some have tight clusters of instances, with the clusters spread out over most of the feature space (e.g. Falkenauer-t, Falkenauer-u and the Stirling instances we generated). These figures aggregate the 21 features in to 2 dimensions, but as part of our study we computed the standard deviation for each feature over all the instances in each benchmark set and the variation seen in the figures does indeed carry over to the raw data.

It is worth noting that not every feature has the same importance [36]. Consequently, considering each feature as equally important can cause misleading similarity/diversity analysis. In [23] authors incorporate latent (hidden) features extracted from performance data to address this issue. In [22], a similar approach to our own was performed to deliver instance set analysis across different problems including bin packing. The results showed that even instances from totally different problems can be closer in terms of features than the ones from the exact same problem domain. Much broader work building on the present study could analyse the characteristics of the training sets beyond pre-determined features or fixed instance sets (e.g. training on Falkenauer-u assuming that they are similar). Here, we focus on fixed training sets as a logical first step in our strand of analysis.

4 RESULTS AND DISCUSSION

For each training set, the GP was repeated 30 times, generating 30 packing policies tailored to that training set. These evolved policies were each then applied to all 3581 instances and the number of bins required by each policy recorded. Unlike the GP stage, application of the packing policies to instances is deterministic, so this stage did not require multiple repeats.

4.1 Algorithm Footprints

After running every policy on every instance, the *footprint* for each policy was determined. The concept of an algorithm footprint was introduced by [9] and developed by [31], and is the set of instances on which a particular algorithm is known to perform well. We define the algorithm footprint for our bin-packing problems as the set of instances for which a policy (the 'algorithm') found a solution using the minimal known number of bins (so, the instances for which that policy found an optimal solution).

For comparison, we also computed the footprint for the scaled-fit policies spanning from best-fit to worst-fit. The footprint for best-fit 2338 instances (65% of all instances), and the footprint for worst-fit is 121 instances (3% of all). Analysing the features for the instances in these footprints, best-fit has a footprint covering much of the feature space; worst-fit only performs well on a narrow set of instances (the Stirling instances for which the optimal solutions had one item per bin). The scaled-fit policies between 0 (best-fit) and 1 (worst-fit) have footprints gradually increasing between these two extremes. As our GP approach to the problem is relatively simplistic, it is no surprise that it struggles to beat best-fit on many instances. These will fall into two categories: very hard instances that need a more sophisticated approach than either our GP or best-fit, and very easy instances for which both approaches work well. The 1243 instances on which the GP approach outperformed best-fit (i.e. using fewer bins than best-fit) fall in between these extremes, and so are of particular interest. In the remainder of the paper, we focus our analysis on these instances (referred to as the *filtered instances*), and on the impact that their spread of features have on generating policies via GP. These are illustrated in Figure 2.

4.2 Footprint Metrics

We now analyse the footprints of the policies generated by GP using each set of training instances. Table 3 shows the following metrics for the footprints among the filtered instances. Each row corresponds to one set of training instances, and the 30 policies generated using that set. These attempt to summarise the performance of the policies trained using each instance set. We will draw comparisons between these performances and the features in the instance sets in Section 4.3.

T The number of instances in the training set after filtering

F The footprint size: i.e. the number of instances in the footprint. The mean *m* and standard deviation *sd* over the 30 policies is given.

Fr The fraction of the 1143 filtered instances that contained in the footprint. This is intended to measure how general the trained policies are. If *Fr* is near one, the footprint includes many instances so can be said to be general. If *Fr* is near zero, the footprint is small and specialised to a narrow set of instances.

TF The number of instances in both the training set and the footprint. This serves as a sanity check in the training process. Given that the training focuses on generating a policy tailored to the training instances, we would expect to have generated a policy that finds the optima for some of the training instances, making *TF* non-zero.

T_1 The fraction of the footprint comprising instances from the training set. This is intended to measure how well the trained policies generalise beyond the training set. If T_1 is near zero, the footprint includes many instances that were not in the training data: the policy has generalised well to unseen instances. If T_1 is near one, the footprint is mostly the training instances, so the policy has not generalised beyond the training data (it has overfitted). Note this figure is affected by training set size: two

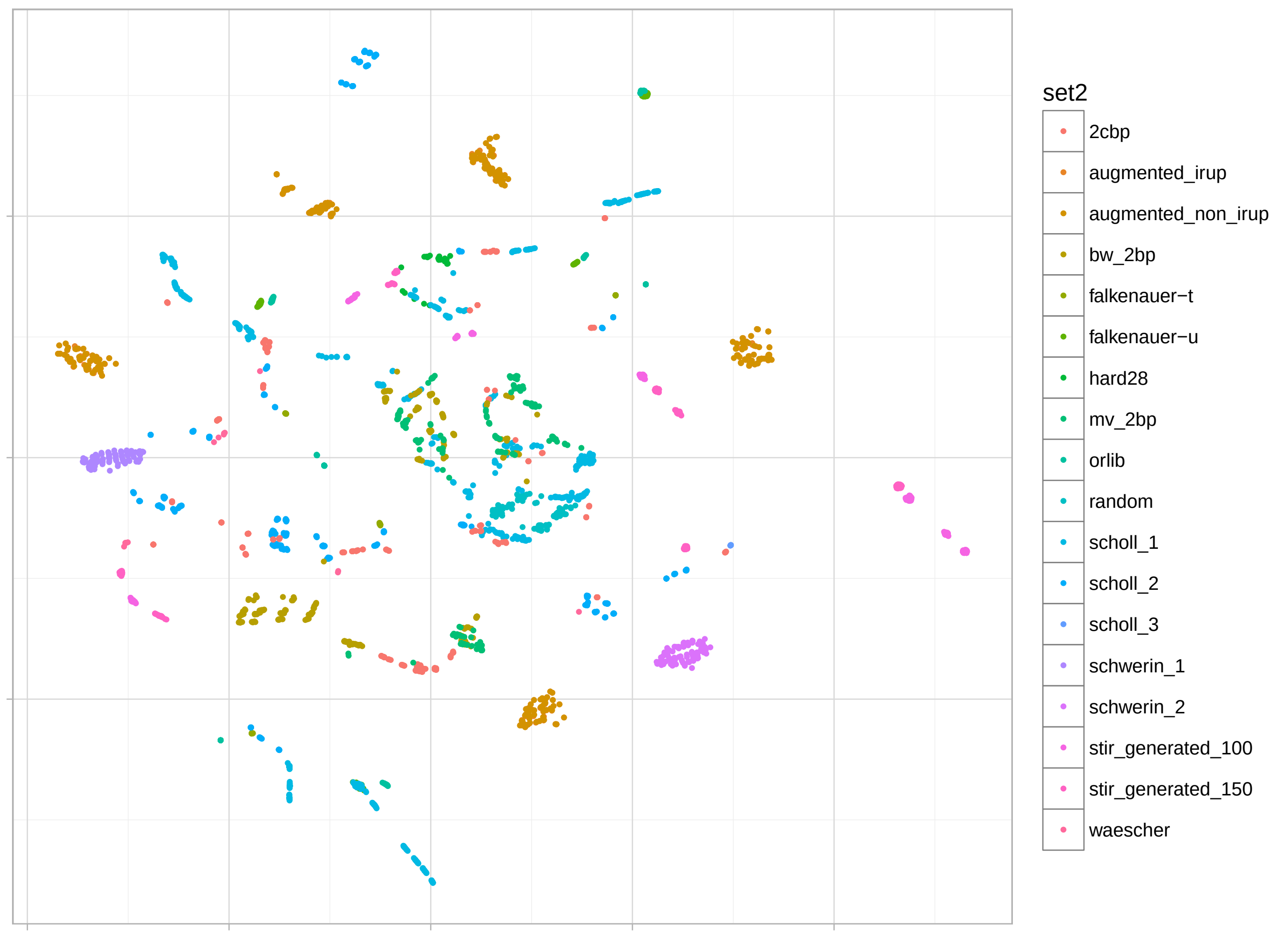


Figure 1: All instance sets rendered in feature space

policies with the same footprint will have different T_1 if one has a larger training set than the other.

T_2 The fraction of the training set instances in the footprint. This shows how well the policies have fitted to the training instances, similar in intent to R^2 for a regression model.

P Summarises the 30 evolved policies for each training set. c represents a program comprising of a single constant, which will score all bins equally and so revert to "first-fit". r represents a program comprising only of the remCap terminal, which corresponds to "best-fit", and $c - r$ corresponds to a constant minus remCap. * represents any program more complex than any of these. Where two symbols are given, there was a roughly equal balance between two types of policies.

There are several observations to make from these results before we move on to the relationship with instance features.

Evolved policies. Within each of the training sets, although the specific policies evolved in each GP run were different, they were similar in qualitative terms. Some sets settled exclusively on policies consisting of a fixed constant (equivalent to first-fit) or just the remCap variable (equivalent to best-fit); for most of the others GP found trees with 20-30 nodes, combining most of the operator and terminal types.

Generality. Overall, most of the policies generalise to unseen instances: T_1 is generally below 0.2 meaning that most of the footprints are made up of unseen instances rather than those from the training set. Exceptions are 2cbp and orlib, for which 0.95 and 0.38 of the footprint comprises training instances. However, T_1 is influenced by the size of the training set: for the very small sets, a policy will only need to do well on a small fraction of the unseen instances for those to outnumber the training instances. In contrast, Fr is always below 0.1: although the footprints include unseen instances, they only cover a small subset of all instances.

Success on training set. There is considerable variation in the success of policies on the instances used to train them. T_2 varies between 0.00 and 0.64: meaning that in some cases it was rare for any training instances to appear in the footprint. However, by chance, these policies were still able to find the optimal solutions for other, unseen, instances.

Performance for policies trained on Stirling instances. The Stirling instances were designed with tightly controlled structures

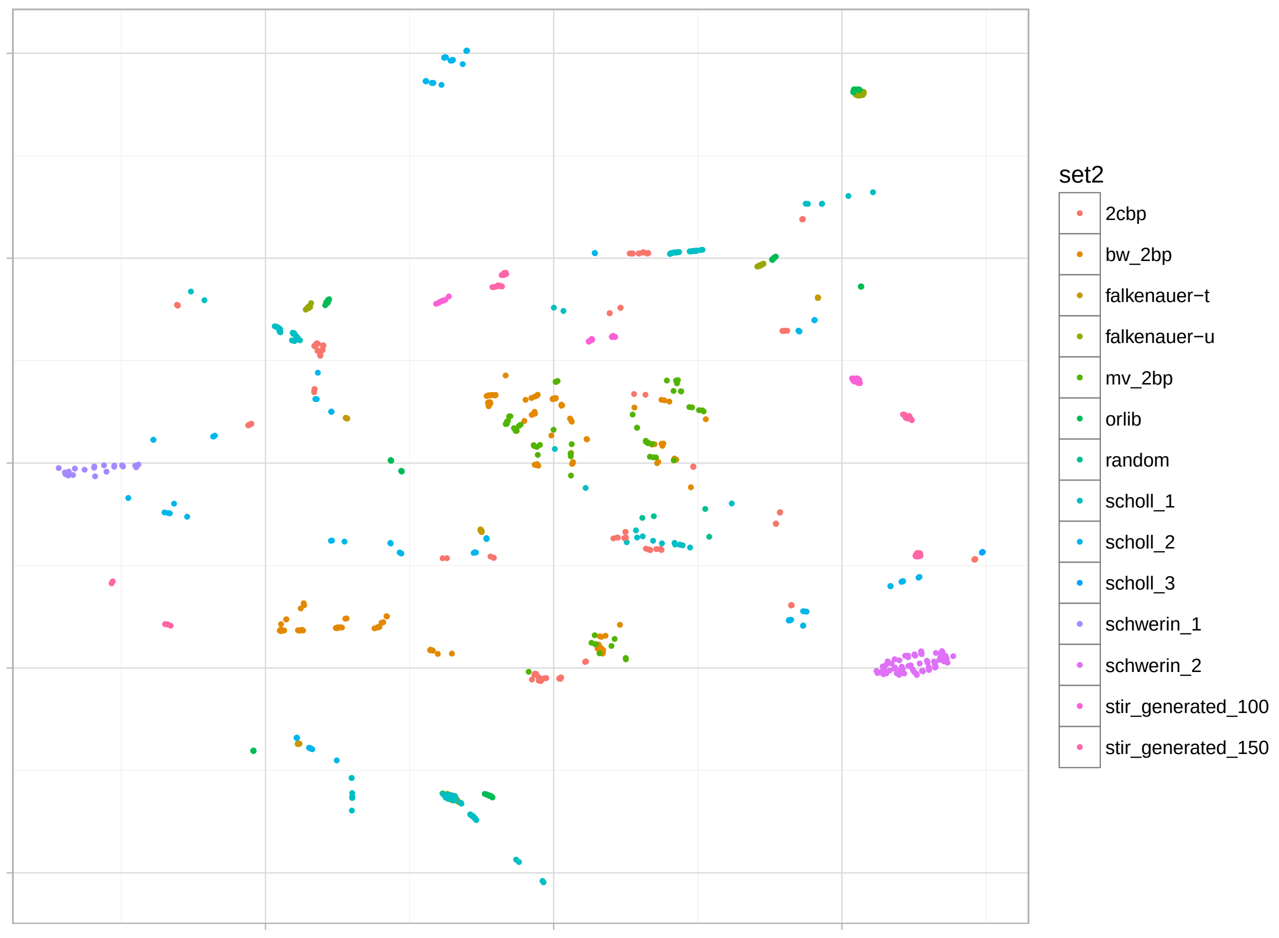


Figure 2: The subset of instances for which it was possible to improve on bestFit using ADA, rendered in feature space

(i.e. narrow bands of item sizes) to simplify identification of patterns in the trained policies. Set 100_26_50 yields policies that are highly specialised. These overfitted policies are influenced by random noise and so result in widely varied footprints across the 30 evolved policies and do not generalise well (with relatively small average footprint sizes).

Figures 3 and 4 have been added to illustrate the training sets and the footprints in qualitative terms, so that as well as the size of the sets/footprints, some insight can be made into the kinds of instances in them (at least in terms of their spread of features). Figures 3a and 4a show that the instances in the Falkenauer-u and Schwerin 1 sets have high and low variation in their features respectively. Figure 3b shows the footprint for the policies trained on Falkenauer-u. This footprint had a high Fr metric, meaning it cover a large number of instances: the illustration shows that these are also spread out over a wide range of features and instance sets. It also had a relatively high F_2, reflected by many of the training set instances featuring in the footprint. By way of contrast, Figure 4b shows the footprint for Schwerin 1. Its Fr metric is relatively low, reflected by a much smaller footprint (which is also less spread out, indicating a lower spread of features in the instances the policy is successful on). Its F_2 metric is zero, reflected in there being none of the training instances in the footprint.

4.3 Variation in features

We now consider how the features for our bin packing instances (listed in Section 3.4 related to the success of the trained policies. In order to capture the variation in features for each instance set, we computed the standard deviation for each feature within the set's instances. These are shown in Table 4. Only the orlib instances showed any variation in Feature 1 (integer/fractional object sizes) within the set. Features 2, 4 and 5 (metrics on the instance sizes) had negligible variation among the Stirling instances, as a result of their design. The variation in Features 11-21 (the performance features) was similar across the training sets. That is, among the instances of Falkenauer-u, orlib, augmented irup/non-irup and scholl_1/2 these features showed large variation; these features varied much less among the instances of the other sets. Given that these make up half the features, this is visible when the 21 features are projected into 2 dimensions: note the Falkenauer-u instances have a much

Table 3: Footprint metrics for each training set (see text for detailed explanation). Set: Training Set; T: number of instances in training set; P: summary of the 30 policies generated for the training set by the repeat GP runs; F: number of instances in the footprints of the 30 policies; F: fraction of all instances in the footprint; TF: number of instances in both the training set and the footprint; T_1: fraction of the footprint in the training set (overfitting to training set); T_2: fraction of the training set in the footprint (fit to training set). For all, *m* is mean, *sd* is standard deviation. Omitted points result from division by zero.

	Set	T	F		Fr		TF		T_1		T_2		P
			m	sd	m	sd	m	sd	m	sd	m	sd	
1	2cbp	182	87	27	0.07	0.02	19.90	13.54	0.21	0.13	0.11	0.07	*
2	augmented_irup	0	54	0	0.04	0.00	0.00	0.00	0.00	0.00			c
3	augmented_non_irup	0	54	0	0.04	0.00	0.00	0.00	0.00	0.00			c
4	bw_2bp	167	67	15	0.05	0.01	62.07	4.59	0.95	0.11	0.37	0.03	*
5	falkenauer-t	80	101	61	0.08	0.05	21.20	23.71	0.16	0.20	0.27	0.30	c*
6	falkenauer-u	72	120	63	0.10	0.05	27.90	25.06	0.17	0.15	0.39	0.35	c*
7	hard28	0	54	0	0.04	0.00	0.00	0.00	0.00	0.00			c
8	mv_2bp	63	63	12	0.05	0.01	2.03	3.37	0.03	0.04	0.03	0.05	c-r
9	orlib	149	113	25	0.09	0.02	41.63	11.57	0.38	0.10	0.28	0.08	*
10	random	4	54	0	0.04	0.00	0.00	0.00	0.00	0.00	0.00	0.00	c
11	scholl_1	138	64	15	0.05	0.01	0.90	3.06	0.01	0.03	0.01	0.02	c-r
12	scholl_2	127	57	11	0.05	0.01	0.47	0.82	0.01	0.02	0.00	0.01	c
13	scholl_3	10	60	23	0.05	0.02	0.60	2.28	0.01	0.02	0.06	0.23	c
14	schwerin_1	20	54	0	0.04	0.00	0.00	0.00	0.00	0.00	0.00	0.00	c
15	schwerin_2	81	58	23	0.05	0.02	2.70	14.79	0.02	0.08	0.03	0.18	c
16	waescher	0	54	0	0.04	0.00	0.00	0.00	0.00	0.00			c
17	Stirling_generated_100_1_100	25	58	15	0.05	0.01	3.90	2.87	0.07	0.05	0.16	0.11	*
18	Stirling_generated_100_1_25	0	58	0	0.05	0.00	0.00	0.00	0.00	0.00			c-r
19	Stirling_generated_100_1_50	8	65	13	0.05	0.01	5.13	0.51	0.08	0.02	0.64	0.06	*
20	Stirling_generated_100_26_50	30	61	36	0.05	0.03	9.07	5.54	0.19	0.15	0.30	0.18	*
21	Stirling_generated_100_51_100	0	38	25	0.03	0.02	0.00	0.00					c
22	Stirling_generated_100_51_75	0	34	26	0.03	0.02	0.00	0.00					rc
23	Stirling_generated_100_76_100	0	41	23	0.03	0.02	0.00	0.00					c
24	Stirling_generated_150_1_100	21	68	14	0.05	0.01	5.93	1.46	0.09	0.04	0.28	0.07	*
25	Stirling_generated_150_1_25	2	54	3	0.04	0.00	0.20	0.55	0.00	0.01	0.10	0.28	c
26	Stirling_generated_150_1_50	4	50	12	0.04	0.01	2.00	0.00	0.04	0.02	0.50	0.00	*
27	Stirling_generated_150_26_50	30	94	41	0.08	0.03	14.27	4.61	0.17	0.11	0.48	0.15	*
28	Stirling_generated_150_51_100	30	87	52	0.07	0.04	7.30	1.18	0.13	0.11	0.24	0.04	*
29	Stirling_generated_150_51_75	0	54	0	0.04	0.00	0.00	0.00	0.00	0.00			r
30	Stirling_generated_150_76_100	0	36	26	0.03	0.02	0.00	0.00					c

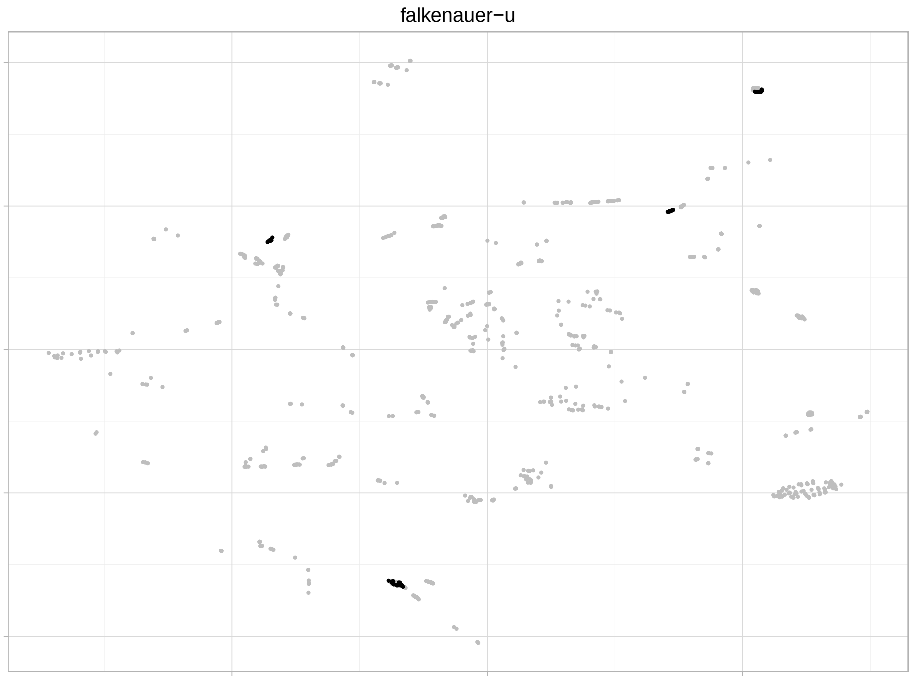
wider spread in Figure 3a than the Schwerin 1 instances in Figure 4a (in fact, Schwerin 1 showed little variation in any of the features).

We now draw a comparison between the variation in features for each instance set, and the performance of the policies that were generated when the sets were used as training data. We measure performance in terms of the footprint metrics given in the previous section. Table 5 gives the statistical correlations between the metrics for the footprints, with the standard deviation in the features across the instances of the corresponding training set. The variation in Features 2 (number of items to pack) and 11-21 (performance features) has a strong positive correlation with the *Fr* metric. This metric measures each policy's footprint size, which captures how generally applicable the policy is. It is logical that features 2 and 11-21 have somewhat similar influence: with feature 2 being the number of items to pack and features 11-21 being number of bins required using different heuristics we would expect them to be correlated. We conclude two things from this result:

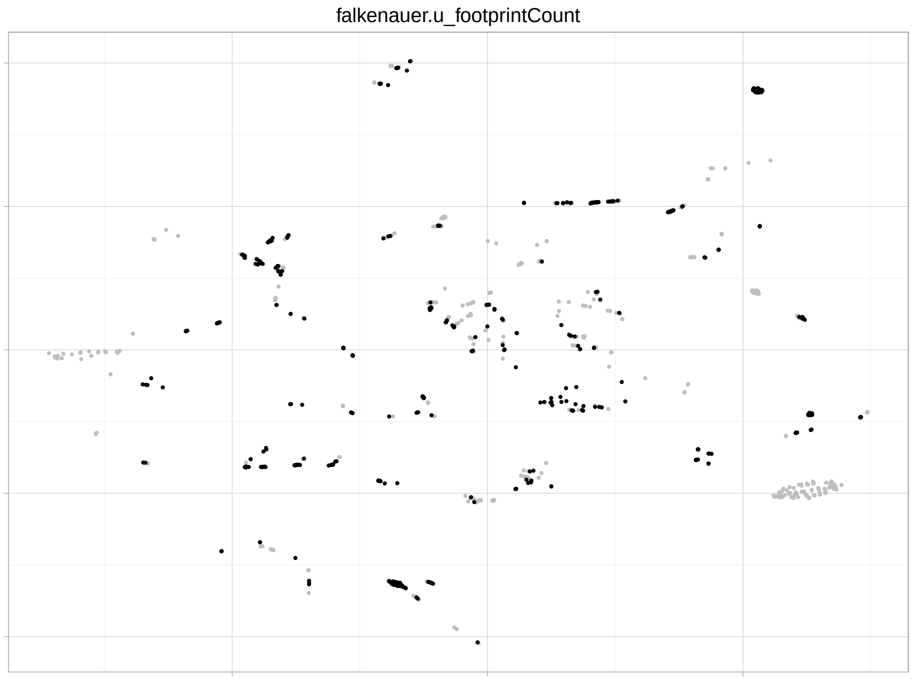
(1) that the amount of variation in the number of items among the instances (feature 2) is a strong indicator of how general the policies trained on it will be. If all the instances in a set have the same number of items, the policies generated for them will be much less likely to perform well in general.
(2) the amount of variation in the performance of the scaledFit policies (features 11-21) is also a strong indicator of general performance for the resulting ADA-generated policies.

In contrast, little can be said about the relationship between variation in features and the F_1 metric, as all these correlations are weak. This is specifically targeted to measure how well the policy performs on instances outside the training set.

On a qualitative level, Figure 3b shows how the well spread out training set (Falkenauer-u) leads to general policies with a widely spread footprint, whereas Figure 4b shows how a training set with a narrow range of features leads to policies with a small,

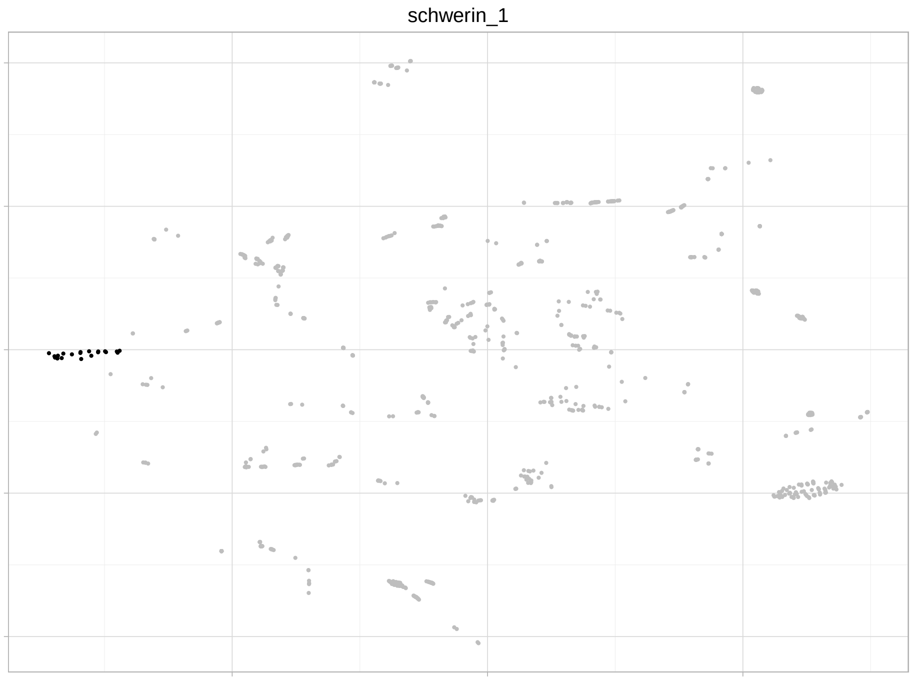


(a) Instances: Black points are instances in the set, grey are all others

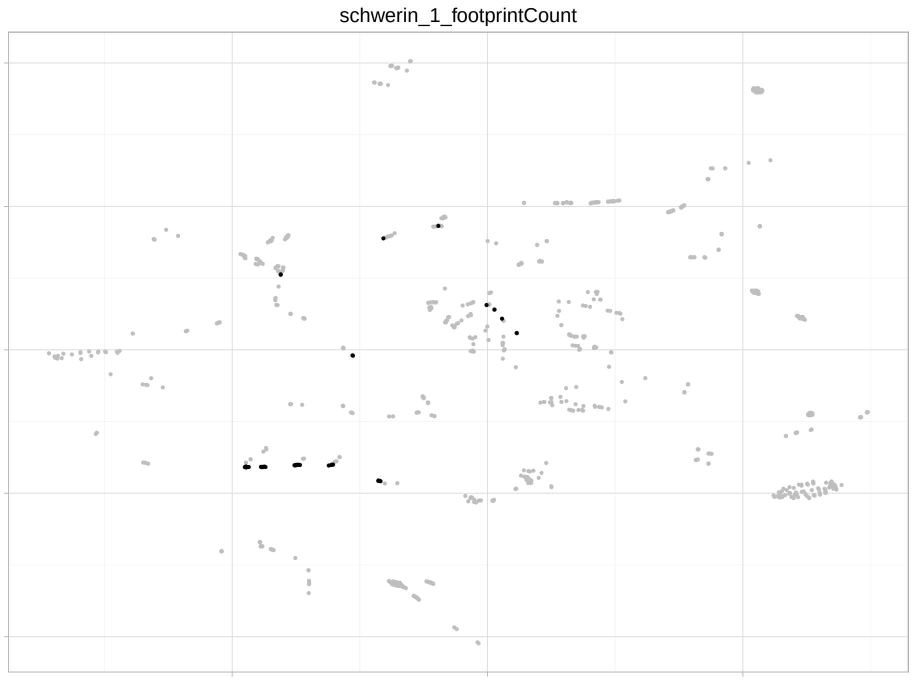


(b) Footprint: Black shows the instances which appear in more than half of the 30 footprints for the policies generated for Falkenauer-u

Figure 3: Feature-space locations of the instances for Falkenauer_u and the footprints for the policies trained on that set



(a) Instances: Black points are instances in the set, grey are all others



(b) Footprint: Black shows the instances which appear in more than half of the 30 footprints for the policies generated for Schwerin_1

Figure 4: Feature-space locations of the instances for Schwerin_1 and the footprints for the policies trained on that set

Table 4: Set & std dev for each feature. Features 12-20 omitted to save space but have broadly the same distribution as 11 & 21

	Set	1	2	3	4	5	6	7	8	9	10	11	21
1	2cbp	0.0	67	0.0050	0.0024	0.000	0.253	0.056	0.138	0.082	0.110	28.3	67.2
2	augmented_irup	0.0	284	0.0005	0.0004	0.000	0.002	0.001	0.015	0.094	0.007	94.5	283.6
3	augmented_non_irup	0.0	284	0.0005	0.0004	0.000	0.002	0.002	0.016	0.093	0.007	94.5	283.6
4	bw_2bp	0.0	28	0.0064	0.0037	0.003	0.311	0.034	0.171	0.334	0.118	16.3	28.3
5	falkenauer-t	0.0	170	0.0019	0.0004	0.000	0.004	0.000	0.007	0.000	0.079	64.5	170.5
6	falkenauer-u	0.0	339	0.0011	0.0004	0.000	0.002	0.002	0.013	0.001	0.071	136.7	338.5
7	hard28	0.0	16	0.0002	0.0001	0.000	0.050	0.010	0.024	0.268	0.007	7.5	16.3
8	mv_2bp	0.0	28	0.0089	0.0039	0.001	0.065	0.047	0.218	0.344	0.116	22.4	28.4
9	orlib	0.5	292	0.0016	0.0004	0.000	0.085	0.058	0.049	0.007	0.070	122.9	292.0
10	random	0.0	0	0.0009	0.0005	0.000	0.053	0.052	0.054	0.037	0.023	3.1	0.0
11	scholl_1	0.0	175	0.0035	0.0015	0.000	0.135	0.104	0.099	0.342	0.118	94.2	174.7
12	scholl_2	0.0	175	0.0017	0.0006	0.000	0.142	0.073	0.086	0.007	0.108	43.6	174.7
13	scholl_3	0.0	0	0.0000	0.0000	0.000	0.001	0.000	0.005	0.000	0.004	0.5	0.0
14	schwerin_1	0.0	0	0.0000	0.0000	0.000	0.000	0.000	0.002	0.000	0.005	0.0	0.0
15	schwerin_2	0.0	0	0.0000	0.0000	0.000	0.000	0.000	0.002	0.000	0.004	0.1	0.0
16	waescher	0.0	51	0.0013	0.0008	0.000	0.151	0.004	0.085	0.045	0.061	5.4	51.1
17	(anon)_generated_100_1_100	0.0	0	0.0002	0.0001	0.000	0.009	0.007	0.036	0.271	0.005	4.7	0.0
18	(anon)_generated_100_1_25	0.0	0	0.0000	0.0000	0.000	0.002	0.000	0.010	0.002	0.004	1.0	0.0
19	(anon)_generated_100_1_50	0.0	0	0.0001	0.0000	0.000	0.006	0.003	0.022	0.063	0.004	2.4	0.0
20	(anon)_generated_100_26_50	0.0	0	0.0000	0.0000	0.000	0.002	0.000	0.010	0.000	0.004	1.2	0.0
21	(anon)_generated_100_51_100	0.0	0	0.0001	0.0000	0.000	0.006	0.003	0.022	0.000	0.006	0.0	0.0
22	(anon)_generated_100_51_75	0.0	0	0.0000	0.0000	0.000	0.002	0.000	0.010	0.000	0.004	0.0	0.0
23	(anon)_generated_100_76_100	0.0	0	0.0000	0.0000	0.000	0.002	0.000	0.010	0.000	0.004	0.0	0.0
24	(anon)_generated_150_1_100	0.0	0	0.0001	0.0000	0.000	0.006	0.005	0.024	0.181	0.005	3.0	0.0
25	(anon)_generated_150_1_25	0.0	0	0.0000	0.0000	0.000	0.001	0.000	0.007	0.001	0.004	0.7	0.0
26	(anon)_generated_150_1_50	0.0	0	0.0001	0.0000	0.000	0.004	0.002	0.015	0.042	0.004	1.5	0.0
27	(anon)_generated_150_26_50	0.0	0	0.0000	0.0000	0.000	0.001	0.000	0.007	0.000	0.004	0.7	0.0
28	(anon)_generated_150_51_100	0.0	0	0.0001	0.0000	0.000	0.004	0.002	0.015	0.000	0.006	4.2	0.0
29	(anon)_generated_150_51_75	0.0	0	0.0000	0.0000	0.000	0.001	0.000	0.007	0.000	0.004	0.0	0.0
30	(anon)_generated_150_76_100	0.0	0	0.0000	0.0000	0.000	0.001	0.000	0.007	0.000	0.004	0.0	0.0

narrowly spread footprint (which does not even include the training instances).

T_2 is positively correlated with features 3, 4, 5, and 6, features 5 and 6 being strong correlations. T_2 measures how well the policies were fitted to the training instances. Features 3–6 relate to the distribution of item sizes within each instance. The strong positive correlation shows that if this distribution changes greatly between instances, it is (perhaps counter-intuitively) easier to find a policy which will perform well on them. We hypothesise that this is because there is enough variation between the instances to force the generated policy to be more general, so still performing well on all training instances rather than a small subset of them. Testing this will be considered in future work.

All figures from these experiments, including those omitted here to save space, are available from [url to be confirmed], along with the full table of standard deviations in the features.

5 CONCLUSION

It is well known that, in machine learning, not all data sets are equal. This paper shows that, in the context of automatically designing optimization algorithms for bin packing, this is still the case.

More interesting is that these results indicate that instance sets that are tightly packed in feature space (that is, with homogeneous instances) lead to evolved policies that fail to generalise well. In fact, these sets are so difficult that GP fails to find a policy that even performs well on the training data. The conclusion that algorithms trained on narrow training data do not generalise is unsurprising, but does at least confirm intuition. In contrast, policies with a broad spread of instances in feature space lead to better evolved policies. This is important to understand as we shift towards using machine effort rather than human effort to develop efficient solvers for new, unseen, problems. For ADA to work successfully, it is critical that the training instances are well-balanced and matched to the instances that the new solvers will ultimately be applied to. Again, this is analogous to the situation in machine learning.

Our experiments also revealed which particular features for bin packing should be widely varied to achieve good performance for ADA. High variation in Features 3, 4, 5, 6 (all connected with variation in item sizes) is a strong indicator for good fitting to the training instances. Variation in Feature 2 (number of items) and features 11-21 (performance features) are strong indicators for instance sets that will lead to generally performing policies.

Table 5: Correlations for stats vs feature SDs for each set

Feature	Correlation feature SD with (F)	(T_1)	(T_2)
1	0.46	0.08	0.26
2	0.70	0.01	0.14
3	0.09	−0.16	0.52
4	0.02	−0.12	0.56
5	−0.01	0.15	0.90
6	0.02	−0.11	0.72
7	0.04	−0.39	0.20
8	−0.09	−0.20	0.44
9	−0.22	−0.06	0.36
10	0.30	−0.21	0.39
11	0.72	0.03	0.18
12	0.73	0.03	0.19
13	0.73	0.04	0.20
14	0.73	0.04	0.20
15	0.73	0.05	0.20
16	0.72	0.05	0.18
17	0.71	0.02	0.15
18	0.71	0.03	0.15
19	0.71	0.03	0.14
20	0.70	0.02	0.14
21	0.70	0.01	0.14

It will be interesting to consider more general concepts of algorithm footprint in future work. Other footprint definitions that might be interesting to consider could be based on the performance relative to average or median performance. Furthermore, this study has only considered broad training groups of instances derived from existing benchmark sets. It will be interesting to consider the trade-off between instance spread and performance or algorithm generality, using much more tightly controlled groups of instances. It would also be useful to consider adding complementary results using an approach that uses a varied training set (rather than pre-existing training sets in the present study) by taking instances from different classes to improve generality.

6 ACKNOWLEDGEMENT

Work funded by UK EPSRC [grants EP/N002849/1, EP/J017515/1]. Results obtained using the EPSRC funded ARCHIE-WeSt HPC [EPSRC grant EP/K000586/1].

7 DATA ACCESS STATEMENT

The data sets, including all computed features, the evolved policies, and their performances, and the visualisations for all feature sets, are available from http://hdl.handle.net/11667/108.

REFERENCES

[1] OR Group, Bologna. http://or.dei.unibo.it/library.

[2] S. Asta, E. ..zcan, and A. J. Parkes. Champ: Creating heuristics via many parameters for online bin packing. *Expert Systems with Appl.*, 63:208 – 221, 2016.

[3] J. Beasley. OR Lib. http://people.brunel.ac.uk/ mastjjb/jeb/orlib/binpackinfo.html.

[4] N. Belkhir, J. Dréo, P. Savéant, and M. Schoenauer. Feature based algorithm configuration: A case study with differential evolution. In *PPSN XIV*, pages 156–166. Springer, 2016.

[5] J. O. Berkey and P. Y. Wang. Two-dimensional finite bin-packing algorithms. *Journal of the Operational Research Society*, 38(5):423–429, may 1987.

[6] J. Branke, S. Nguyen, C. W. Pickardt, and M. Zhang. Automated design of production scheduling heuristics: A review. *IEEE Transactions on Evolutionary Computation*, 20(1):110–124, 2016.

[7] E. K. Burke, M. R. Hyde, and G. Kendall. Evolving bin packing heuristics with genetic programming. In T. P. Runarsson, H.-G. Beyer, E. Burke, J. J. Merelo-Guervós, L. D. Whitley, and X. Yao, editors, *Parallel Problem Solving from Nature - PPSN IX*, pages 860–869, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[8] E. K. Burke, M. R. Hyde, G. Kendall, and J. R. Woodward. The scalability of evolved on line bin packing heuristics. In *Proc. IEEE CEC*, pages 2530–2537, 2007.

[9] D. Corne and A. Reynolds. *Optimisation and generalisation: Footprints in instance space*, volume 6238 of *LNCS*, pages 22–31. Part 1 edition, 2010.

[10] M. Delorme, M. Iori, and S. Martello. Bin packing and cutting stock problems: Mathematical models and exact algorithms. *Euro. J Oper. Res.*, 255(1):1–20, 2016.

[11] J. H. Drake, J. Swan, G. Neumann, and E. Özcan. Sparse, continuous policy representations for uniform online bin packing via regression of interpolants. In *EvoCOP*, volume 10197 of *LNCS*, pages 189–200, 2017.

[12] E. Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2(1):5–30, 1996.

[13] E. Falkenauer and A. Delchambre. A genetic algorithm for bin packing and line balancing. In *Proceedings 1992 IEEE International Conference on Robotics and Automation*, pages 1186–1192 vol.2, May 1992.

[14] A. A. Freitas and G. L. Pappa. Genetic programming for automatically constructing data mining algorithms. In J. Wang, editor, *Encyclopedia of Data Warehousing and Mining, Second Edition (4 Volumes)*, pages 932–936. IGI Global, 2009.

[15] J. C. Gomez and H. Terashima-Marín. Evolutionary hyper-heuristics for tackling bi-objective 2d bin packing problems. *GP & EM*, Mar 2017.

[16] J. C. Gower. A General Coefficient of Similarity and Some of Its Properties. *Biometrics*, 27(4):857–871, 1971.

[17] L. Hong, J. Woodward, J. Li, and E. Özcan. Automated design of probability distributions as mutation operators for evolutionary programming using genetic programming. In *EuroGP*, pages 85–96. Springer, 2013.

[18] F. Hutter, L. Xu, H. H. Hoos, and K. Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79–111, jan 2014.

[19] E. López-Camacho, H. Terashima-Marín, G. Ochoa, and S. E. Conant-Pablos. Understanding the structure of bin packing problems through principal component analysis. *International Journal of Production Economics*, 145(2):488–499, oct 2013.

[20] S. Martello and D. Vigo. Exact solution of the two-dimensional finite bin packing problem. *Management Science*, 44(3):388–399, mar 1998.

[21] P. Matos, J. Planes, F. Letombe, and J. a. Marques-Silva. A MAX-SAT Algorithm Portfolio. In *Proc. ECAI 2008*, pages 911–912, Amsterdam, 2008. IOS Press.

[22] M. Mısır. Matrix factorization based benchmark set analysis: A case study on hyflex. In *Asia-Pacific Conference on Simulated Evolution and Learning*, pages 184–195. Springer, 2017.

[23] M. Mısır and M. Sebag. Alors: An algorithm recommender system. *Artificial Intelligence*, 244:291–314, 2017.

[24] E. Nudelman, A. Devkar, Y. Shoham, and K. Leyton-Brown. Understanding Random SAT: Beyond the Clauses-to-Variables Ratio. In *CP-04*, pages 438–452.

[25] F. Otero, T. Castle, and C. Johnson. EpochX. In *Proc. GECCO Companion*, 2012.

[26] C. Reeves. Hybrid genetic algorithms for bin-packing and related problems. *Annals of Operations Research*, 63(3):371–396, Jun 1996.

[27] J. E. Schoenfield. Fast, exact solution of open bin packing problems without linear programming. *Draft, US Army Space & Missile Defense Command, Huntsville AL, USA*, 2002.

[28] A. Scholl, R. Klein, and C. Jrgens. Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *C&OR*, 24(7):627–645, 1997.

[29] P. Schwerin and G. Wäscher. The bin-packing problem: A problem generator and some numerical experiments with FFD packing and MTP. *International Transactions in Operational Research*, 4(5-6):377–389, nov 1997.

[30] K. Sim, E. Hart, and B. Paechter. A lifelong learning hyper-heuristic method for bin packing. *Evolutionary Computation*, 23(1):37–67, 2015. PMID: 24512321.

[31] K. Smith-Miles, D. Baatar, B. Wreford, and R. Lewis. Towards objective measures of algorithm performance across instance space. *Computers & Operations Research*, 45:12–24, May 2014.

[32] K. Smith-Miles and S. Bowly. Generating new test instances by evolving in instance space. *Computers & OR*, 63:102–113, Nov 2015.

[33] K. Smith-Miles and L. Lopes. Measuring instance difficulty for combinatorial optimization problems. *Computers & OR*, 39(5):875–889, May 2012.

[34] K. Smith-Miles, B. Wreford, L. Lopes, and N. Insani. Predicting metaheuristic performance on graph coloring problems using data mining. In *Hybrid Metaheuristics*, pages 417–432. Springer, 2013.

[35] L. Van Der Maaten and G. Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[36] M. Wagner, M. Lindauer, M. Mısır, S. Nallaperuma, and F. Hutter. A case study of algorithm selection for the traveling thief problem. *J Heur.*, pages 1–26, 2017.

[37] G. Wäscher and T. Gau. Heuristics for the integer one-dimensional cutting stock problem: A computational study. *OR Spektrum*, 18(3):131–144, sep 1996.

[38] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Satzilla: portfolio-based algorithm selection for sat. *J of Artificial Intell. Res.*, pages 565–606, 2008.