Relating Training Instances to Automatic Design of Algorithms for Bin Packing via Features

Alexander E.I. Brownlee
University of Stirling
Stirling, UK
alexander.brownlee@stir.ac.uk

John R. Woodward
Queen Mary University of London
London, UK
j.woodward@qmul.ac.uk

Nadarajen Veerapen
University of Stirling
Stirling, UK
nadarajen.veerapen@cs.stir.ac.uk

ABSTRACT

Automatic Design of Algorithms (ADA) treats algorithm choice and design as a machine learning problem, with problem instances as training data. However, this paper reveals that, as with classification and regression, for ADA not all training sets are equally valuable.

We apply genetic programming ADA for bin packing to several new and existing benchmark sets. Using sets with narrowly-distributed features for training results in highly specialised algorithms, whereas those with well-spread features result in very general algorithms. Variance in certain features has a strong correlation with the generality of the trained policies.

KEYWORDS

Automatic design of algorithms; features; bin packing

1 INTRODUCTION

The Automatic Design of Algorithms (ADA) [2] seeks to build algorithms, which perform better than human designed algorithms. The algorithms are trained/designed using set of problem instances, then applied to a set of unseen problem instances. Recent research has shown how features of the problem instances can assist the process of automatically configuring existing algorithms [1], select heuristics [6] or predict runtime [5].

In practice, ADA is a machine learning procedure, and needs representative data to generalise well. 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 sets: these policies are applied to all the benchmark sets. The contributions 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) showing that variance in features 3, 4, 5, 6 aids fitting to training data; and in 2, 11–21 to more general policies;
(4) two new sets of benchmark bin packing instances.

Further detail and analysis of our experiments can be found in our technical report [3].

2 METHODOLOGY

We adopt a simple framework using genetic programming (GP) to evolve a *packing policy*: a function that gives a score s to each bin (including a to-be-used empty bin), given an item to be placed. The item is placed in the highest-scored bin score with enough remaining capacity. For equally scored bins, the first is chosen.

We also studied best-fit, which chooses the bin with the least spare capacity after adding the item; worst-fit, which chooses the bin with the most capacity; and *scaled-fit*, a generalisation of best-fit where the target remaining capacity is a fixed fraction τ of bin size.

2.1 GP applied to bin packing

Our approach used typical GP operators, with terminals: remaining bin capacity after item; bin capacity; random constant. The fitness to be minimised targeted policies that find the minimal number of bins with a correction to also reduce bloat.

Our experiments were in two stages. (1) For each training set, the GP was repeated 30 times, generating 30 packing policies tailored to that training set. Evolution was performed using EpochX with population size 1000, 100 generations, and other parameters the EpochX defaults. (2) Evolved policies from (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.

2.2 Benchmark Instances

Benchmark instances selected from the literature as training and test data were: 2cbp, Augmented irup & non-irup, Random, bw-2bp, falkenauer-t, falkenauer-u, hard28, mv-2bp, orlib, scholl 1/2/3, schwerin 1/2 and waescher (as per [3]). We also devised two new sets: "Stirling instances" [3], intended to provide varying levels of difficulty by allowing items to take sizes in bands defined as a fraction of the bin capacity. 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. We used two bin capacities: 100 and 150.

2.3 Instance Features

The considered features are *static features 1-10*: item sizes are integers (T/F); Number of items; Mean item size divided by bin capacity; Std dev in the item sizes divided by bin capacity; Information entropy in the item sizes, divided by bin capacity; max, min, median item size divided by bin capacity; max item size divided by min item size; compression ratio for the list of item sizes. *Performance features 11-21*: the number of bins needed 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 [7]. Our analysis focuses on statistical analysis of the results, but these illustrations help indicate the distribution of the instances in terms of their features. The full set of illustrations are available at http://hdl.handle.net/11667/108, and a subset in [3].

Published in Proceedings of the Genetic and Evolutionary Computation Conference Companion by ACM. The original publication is available at: https://doi.org/10.1145/3205651.3205748

3 RESULTS AND DISCUSSION

For each training set, GP was repeated 30 times, generating 30 packing policies. These policies were each applied to all 3581 instances.

Footprints. The *footprint* (instances on which the policy performs well) [4] for each policy was determined. We define the footprint as the set of instances for which a policy found a solution using the minimal known number of bins.

As our GP approach is relatively simplistic, it (unsurprisingly) struggles to beat best-fit on many instances. These are hard instances needing a more sophisticated approach, and easy instances where both approaches work well. The 1243 instances where GP outperformed best-fit fall between the extremes. We focus our analysis on these, and on the impact that their spread of features have on generating policies via GP.

3.1 Footprint Metrics

We now analyse the footprints of the policies generated by GP using each set of training instances.

Evolved policies. In each training set, although the specific policies from each GP run were different, they were qualitatively similar. Some sets had policies all consisting of a fixed constant (equivalent to first-fit) or just the remaining capacity (best-fit); for most others GP found complex trees with 20-30 nodes.

Generality. Most policies generalise well: most footprints are made up of unseen instances rather than the training set. Exceptions are 2cbp and orlib, for which large parts of the footprint comprises training instances.

Success on training set. There is considerable variation in the success of policies on the instances used to train them. In some cases it was rare for any training instances to appear in the footprint, yet, by chance, these policies were still able to find the optimal solutions for other, unseen, instances.

3.2 Variation in features

How do the features for the instances related to the success of trained policies? We computed the standard deviation for each feature within the set's instances [3].

To compare the variation in features for each instance set, and the footprint of the policies trained on that set, we computed 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) is strongly correlated with footprint size. 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; and the amount of variation in the performance features is also a strong indicator of general performance for the resulting ADA-generated policies.

Features 3, 4, 5, and 6 (related to the distribution of item sizes within each instance) are strongly correlated with how well the policies were fitted to the training instances. If the item size 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.

4 CONCLUSION

It is well known that, in machine learning, not all data sets are equal, and this carries over to automatically designing optimization algorithms for bin packing.

These results indicate that instance sets that are tightly packed in feature space lead to evolved policies that do not generalise; indeed, GP fails to find policies that even perform well on training data. The conclusion that algorithms trained on narrow training data do not generalise is unsurprising, but at least confirms intuition.

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.

5 ACKNOWLEDGEMENTS

Work funded by UK EPSRC [grants EP/N002849/1, EP/J017515/1], experiments run on EPSRC funded ARCHIE-WeSt HPC [EP/K000586/1].

6 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] 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.

[2] 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.

[3] A. E. I. Brownlee, J. R. Woodward, and N. Veerapen. Relating training instances to automatic design of algorithms for bin packing via features (detailed experiments and results). Technical report, University of Stirling, 2018. http://hdl.handle.net/1893/26957.

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

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

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

[7] 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.