Evolutionary Approaches to Improving the Layouts of Instance-Spaces

Kevin Sim[0000−0001−6555−7721] and Emma Hart[0000−0002−5405−4413]

Edinburgh Napier University
{k.sim, e.hart}@napier.ac.uk

Abstract. We propose two new methods for evolving the layout of an instance-space. Specifically we design three different fitness metrics that seek to: (i) reward layouts which place instances won by the same solver close in the space; (ii) reward layouts that place instances won by the same solver *and* where the solver has similar performance close together; (iii) simultaneously reward proximity in both class and distance by combining these into a single metric. Two optimisation algorithms that utilise these metrics to evolve a model which outputs the coordinates of instances in a 2d space are proposed: (1) a multi-tree version of GP (2) a neural network with the weights evolved using an evolution strategy. Experiments in the TSP domain show that both new methods are capable of generating layouts in which subsequent application of a classifier provides considerably improved accuracy when compared to existing projection techniques from the literature, with improvements of over 10% in some cases. Visualisation of the the evolved layouts demonstrates that they can capture some aspects of the performance gradients across the space and highlight regions of strong performance.

Keywords: Instance-space · Dimensionality-Reduction · Algorithm-Selection.

1 Introduction

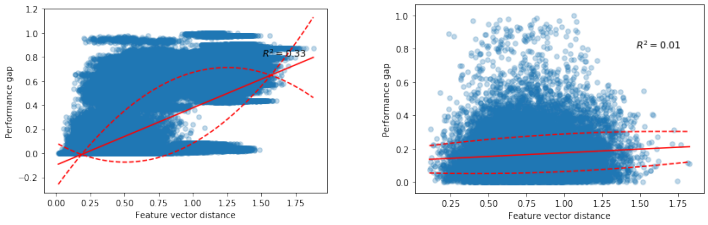
Instance Space Analysis is a methodology first proposed by Smith-Miles *et al.* in a series of papers [18, 14, 15] with the purpose of (1) providing a means of visualising the location of benchmark instances in a 2d space; (2) illustrating the 'footprint' of an algorithm (i.e. the regions of the space in which it performs well) and (3) calculating objective (unbiased) metrics of algorithmic power via analysis of the aforementioned footprints. Once an instance-space has been created, it can be used in various ways: for example, to identify regions in the space that are lacking representative data (followed by generation of instances targeted at filling these gaps) or developing automated algorithm selection tools to determine the best algorithm for solving a new instance.

A critical step of the methodology is clearly the projection of an instance described by a high-dimensional feature-vector into a 2d instance-space, with the quality of this projection having significant influence of the utility of the resulting space [14]. Three factors are important. Firstly, the projection method should

be model-based, i.e. it should learn a model that can be used to project future unseen instances into the space once it has been created — this rules out *embedding* approaches such as the popular t-sne [10] technique. Secondly, if instances are labelled according to the solver which produces the 'best' performance, instances with the same label should be co-located in the space (potentially in multiple distinct regions). Finally, within a subset of instances with the same label, we propose that the projection should locate instances where the solver provides similar performance closer together than those where the performance differs widely, i.e. performance should vary smoothly across a cluster of instances with the same label, thereby indicating regions in which the winning solver is particularly strong and vice-versa.

The vast majority of work in the area of instance-space creation has used Principal Component Analysis (PCA)[12] as the means of projecting to a 2d-space — despite the fact that this method is unsupervised and therefore does not take into account either instance-labels or relative performance of solvers. Alternative methods for dimensionality-reduction such as manifold-learning methods (e.g. UMAP [3]) which can be used in a supervised manner seek to place instances that are close in the high-dimensional space close in low-dimensional instance space: that is, an instance should retain the same nearest neighbours in the embedded space as in the input space. However, mapping from an high-dimensional *feature-space* to a low-dimensional feature-space that also smoothly captures variation in the *performance-space* for the purpose of instance-space creation poses a problem for most manifold-learning methods: neighbours in the performance-space are not necessarily neighbours in the feature-space. This is clearly illustrated in figure 1 which plots the distance in a feature-space against distance in the performance-space for all pairs of instances taken from a large set of 950 TSP instances, and shows there is very little correlation.

Therefore, in order to produce instance-spaces which attempt to satisfy all three criteria outlined above, we propose two evolutionary approaches to learn a model which maps from a high-dimensional feature-space to a low-dimensional instance-space. The first uses a multi-tree genetic programming (GP) algorithm to output the coordinates in a 2d space, while the second evolves the weights of a neural-network which outputs the 2d coordinates, using an evolution strategy (ES) to train the network. We propose three novel fitness functions that can be combined with either optimiser to address this. To evaluate the quality of the evolved layouts, *post-evolution* we apply multiple classifiers to the space to determine whether it facilitates algorithm-selection, and visualise the space to gain a qualitative understanding of whether the space smoothly reflect the performance gradient of the solvers. Both approaches are demonstrated to evolve layouts that improve the accuracy of classifiers trained on the 2d space compared to using layouts created using PCA and UMAP, with some progress towards improving layouts with respect to the performance gradients within a class.



(a) Instances won by LKCC (b) Instances won by CLK

Fig. 1: Scatter plots showing distance in feature-space (x-axis) vs performance-gap for pairwise comparisons of TSP instances, illustrating lack of correlation between these quantities

2 Related Work

A long line of work by Smith-Miles and her collaborators [18, 14, 15] has gradually refined the approach to defining an instance-space into a rigorous methodology, culminating in the freely available MATILDA toolbox [2]. The method has been applied in multiple domains within combinatorial optimisation (e.g. TSP [16], timetabling [17], knapsack [2], graph-colouring [18]) and more recently to machine-learning datasets [11]. In the vast majority of the work described in combinatorial optimisation, instance-spaces are created using PCA to project into low-dimensions. As this is an unsupervised method, in order to find projections that place instances won by the same solver in similar regions, an evolutionary algorithm is used to select a subset of features that - when projected via PCA - maximise classification accuracy using an SVM classifier. However, this potentially biases the projection towards SVM. A more recent approach in which a single instance-space is developed to represent multiple *machine-learning* datasets uses a new projection method that tries to minimise the approximation error $|F - f| + |P - p|$ where F, P correspond to the feature/performance vector in a high-d space and f, p to the same vectors in the low-d space, but cannot be directly translated to laying out an instance-space to reflect the performance gradients of *multiple* solvers.

More recently, Lensen *et al.* [7] proposed a GP approach to manifold learning of machine-learning datasets [19]. A multi-tree GP method is used to learn a 2d projection using a fitness function that attempts to maintain the same ordering between neighbours of an instance in both spaces. The quality of a learned embedding is estimated via a proxy measure calculated post-evolution —- applying a classifier to the newly projected data and measuring classification accuracy. In more recent work, the same authors propose further extensions that (1) optimise the embedding learned by GP to match a pre-computed UMAP embedding, and (2) optimise UMAP's own cost-function directly [13]. Most recently they adapt their approach to consider how local structure within an embedding can be better reflected, proposing a modified fitness functions that seeks to measure how well local topology is preserved by the evolved mapping [8]. However, a com-

mon trait in this body of work is that the various fitness functions proposed optimise embeddings such that the input and embedded neighbourhood of a given instance contain the same instances with the same ordering. As previously discussed in section 1, this is not necessarily desirable if the instance-space is to capture performance gradients as well as maintain neighbourhoods in the feature-space. These issues motivate our new approach which is detailed below.

3 Methods

Our goal is to evolve an instance-space layout that places instances won by the same solver in the same regions of a low-d feature-space while also attempting to maintain an ordering in the performance-space, i.e. *instances won by the same algorithm and eliciting similar performance from that algorithm should placed close together*. We investigate two approaches for creating the layout: the first uses an evolution-strategy to evolve the weights of a neural network that outputs the 2d coordinates of each instances. The second, inspired by the work of Lensen *et al*, uses a multi-tree genetic programming approach to achieve the same goal but with new fitness function(s). Assuming a set of instances, each of which is described by a high-dimensional feature-vector and is labelled with the solver that 'wins' the instance, we propose the following novel fitness functions:

1. Label-based ($\mathcal{L}$): maximise the proportion of the k nearest neighbours of an instance i in the embedded space that have the same label as i (averaged over all instances)
2. Distance-based ($\mathcal{D}$) :minimise the normalised average distance between the performance of an instance and that of its k nearest neighbours *that have the same label*. A penalty of 1 is added to this quantity for every neighbour that is wrongly labelled, in order to prevent the fitness function favouring a small number of correctly labelled neighbours with very small distance.
3. Combined ($\mathcal{L} + \mathcal{D}$): maximise $\mathcal{L} + (1 - \mathcal{D})$, i.e a linear combination of the previous fitness functions

We evolve mappings using each combination of fitness-function and optimisation algorithm (ES/GP), i.e. 6 combinations in total. Post-optimisation, we follow the approach of Lensen *et al.* and estimate the quality of an evolved instance-space using a proxy measure: we apply three off-the-shelf classifiers to the evolved layout to predict the best solver on an *unseen* set of instances, hence determining whether the layout facilitates algorithm-selection. Secondly, we provide a qualitative view of the extent to which instances that have similar performance are mapped to similar regions of the space using visualisation. Results are compared to PCA and UMAP. PCA is chosen as it is the method of choice to produce an instance-space in MATILDA. UMAP is selected as it can be used in a supervised fashion and therefore offers a comparison to our proposed supervised techniques.

3.1 Mapping using an Evolution Strategy

An ES is used to learn the hyper-parameters of a neural-network that given a set of features describing an instance, outputs the coordinates of the instance in a 2-d space. A population encodes a set of individuals that specify the real-valued weights of a fixed-size neural network. The neural network is a feed-forward neural network with f inputs corresponding to the number of features describing an instance, and $o = 2$ outputs specifying the new coordinates in a 2d space. The network has one hidden layer with $(f + o)/2$ neurons. The hidden neurons use a *relu* activation function, while the two output neurons have *sigmoid* activation.

To evolve the population to optimise the chosen fitness function(s) (as defined above), we apply the standard evolution strategy CMA-ES [4] due its prevalence in the literature in the context of neuro-evolution [9, 6, 5]. We use the default implementation of CMA-ES provided in the DEAP library [1]. This requires three parameters to be set: the centroid (set to 0.0), the value of *sigma* (set to 1.0) and the number of offspring *lambda* which is set to 50 in all experiments. The algorithm was run for 50 generations.

3.2 Mapping using a Multi-Tree Genetic Programming

We use a multi-tree GP representation in which each individual contains 2 GP trees, each representing a single dimension in the embedding, following the general approach of Lensen *et al.* [7]. The terminal set contains the n features describing an instance, with a mix of linear/non-linear functions (table 1). The algorithm is implemented using DEAP. All parameter settings are given in table 1. To enable direct comparison with the ES, the number of generations was fixed at 25, resulting in the same number of individual evaluations for both methods.

Table 1: GP parameters and settings

GP parameters	
Population size	100
Initialisation	GenHalfandHalf (5,10)
Selection	Tournament (size=7)
Crossover	1 pt
Mutation	GenFull (3, 5)
Max tree depth	17

GP Function Set	
ADD	COSINE
SUBTRACT	SIN
MULTIPLY	TANH
ProtectedDIVIDE	RELU
NEG	Eph. Const.(0,1)

3.3 Instance Data

We use the instances from the TSP domain provided via MATILDA [2]. This includes the meta-data associated with an instance that defines the values of features identified as relevant for the domain and the performance data from two

solvers (Chained Lin-Kernighan (CLK) an Lin-Kernighan with Cluster Compensation (LKCC) [18]). 950 instances are provided which are synthetically generated to be either easy or hard for each solver (see [18] for a detailed description of the generation method and solver), labelled by the best-solver determined according to the time taken to solve.

We conduct experiments using the 17 features given in the MATILDA metadata (denoted 'full features'). In addition we repeat experiments using the subset of 6 features that were selected by MATILDA to produce the instance-space projection using PCA, as described in the section 2, referred from here on as the 'reduced set'. Recall that this reduced set was specifically chosen to optimise a PCA projection but is used in all experiments to enable direct comparison. The reader is referred to the MATILDA toolbox [2] for a detailed description of each feature and the features designated as the 'reduced' feature-set.

4 Experiments

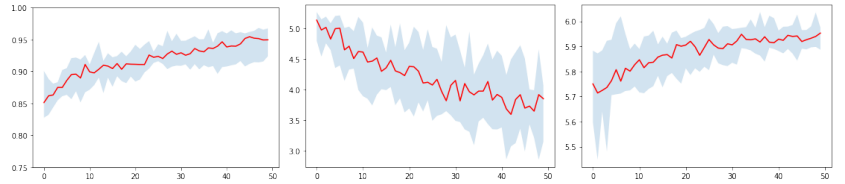
Experiments are conducted for each optimiser (ES/GP) combined with each of the three proposed fitness functions. For each combination we evolve layouts using both the full and reduced feature set. All experiments are repeated 10 times. The feature-data is normalised (specifically as this provides input in a suitable range to the neural network). For PCA and UMAP, standardised scaling is applied to the data to remove the mean and scale to unit variance as this is widely accepted as best-practice for these methods. The 950 instances are split into a 'training' set using 60% of the data (selected using stratified sampling) to preserve class distribution in each split. The same training dataset is used in all experiments to evolve the instance-space layout. In all experiments reported, the k nearest neighbour parameter required to calculate fitness was set to 15. (A series of preliminary experiments that varied k between 15 and 45 did not provide any statistical evidence that the setting influenced results).

Following evolution, we calculate a proxy measure of quality as described in section 2 to quantify the effectiveness of the evolved layout: three off-the-shelf classifiers are trained to predict the best solver (a binary classification problem) using the evolved 2d projection as input to the classifier. The classifier is trained using the same training data used to evolve the layout, then results are reported on the *held-out test set*. Three classifiers are chosen: Random Forest (generally cited as providing strong performance), support vector machines (as used in the MATILDA methodology), and finally a k-nearest neighbour classifier (which would be expected to perform well in the spaces evolved by the label-based method which also relies on neighbourhoods). For each evolved layout, we record the accuracy and F1 score (which combines the precision and recall of a classifier into a single metric by taking their harmonic mean) of each classifier. This is repeated using layouts created using PCA and UMAP on the same data as comparison. The standard scikit-learn implementation of PCA is used which does not require any parameter setting. UMAP requires a parameter *nearest_neighbor* which controls the balance between local versus global struc-

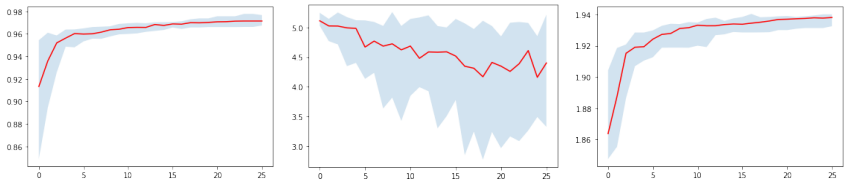
ture in the data (where low values emphasise local structure) which was set to 5 again following a brief empirical investigation. All other UMAP settings were left in their default setting as provided in the Python implementation [3]. As previously noted, UMAP is used in its supervised form.

5 Results

This section reports quantitative and qualitative results that (1) compare the quality of embedding (using classification accuracy/F1-score as a proxy) using each combination of optimiser/fitness-function to off-the-shelf methods (PCA, UMAP) and (2) provide a qualitative evaluation of the evolved instance-spaces with respect to the extent to which they appear to separate the two classes, and illustrate gradients in the performance data.



(a) NN (fitness= $\mathcal{L}$) (b) NN (fitness= $\mathcal{D}$) (c) fitness: $\mathcal{L} + (1 - \mathcal{D})$)



(d) GP (fitness= $\mathcal{L}$) (e) GP (fitness= $\mathcal{D}$) (f) fitness: $\mathcal{L} + (1 - \mathcal{D})$)

Fig. 2: Convergence curves for combinations of (optimiser, fitness function), obtained using the reduced feature-set as input. Top row shows results for the NN method, the bottom row for GP. Red line shows median value over 10 runs

5.1 Insights into Evolutionary Progress

Figure 2 plots convergence curves for each combination of optimiser/fitness function applied to evolving a layout in the reduced feature-space[1]. The GP method combined with label-based fitness measure $\mathcal{L}$ or the combined fitness measure exhibits less variance than the neural network approach, although both methods

[1] similar trends are observed in the plots obtained in the full feature space but not shown due to space limitations

show wide variance using the distance based function which only implicitly accounts for class-labels. Furthermore, the GP approaches converge more quickly than the neural-network equivalents.

Table 2: Median accuracy and F1 score per classifier using projection obtained by each combination of optimiser/fitness function, with comparison to PCA and UMAP. Top row - full feature set; bottom row - reduced feature set. **Bold** indicates that *accuracy* is better than both PCA/UMAP, *italics* that F1 is better than both. All results reported on held-out test set

Full Features	A(SVM)	F1(SVM)	A(KNN)	F1(KNN)	A(RndF)	F1(RndF)
GP-KNN ($\mathcal{L}$)	**0.920**	*0.911*	**0.972**	*0.972*	**0.959**	*0.958*
GP-DIST ($\mathcal{D}$)	**0.953**	*0.951*	**0.966**	*0.966*	**0.962**	*0.962*
GP-DIST-ADD ($\mathcal{L}+\mathcal{D}$)	**0.942**	*0.939*	**0.966**	*0.966*	**0.961**	*0.960*
NN-KNN ($\mathcal{L}$)	**0.912**	*0.912*	**0.920**	*0.919*	**0.915**	*0.913*
NN-DIST ($\mathcal{D}$)	0.833	0.782	**0.933**	*0.933*	**0.917**	*0.917*
NN-DIST-ADD ($\mathcal{L}+\mathcal{D}$)	0.800	0.711	0.778	0.710	0.800	0.711
PCA	0.850	0.843	0.847	0.845	0.858	0.844
UMAP	0.855	0.845	0.861	0.852	0.855	0.845

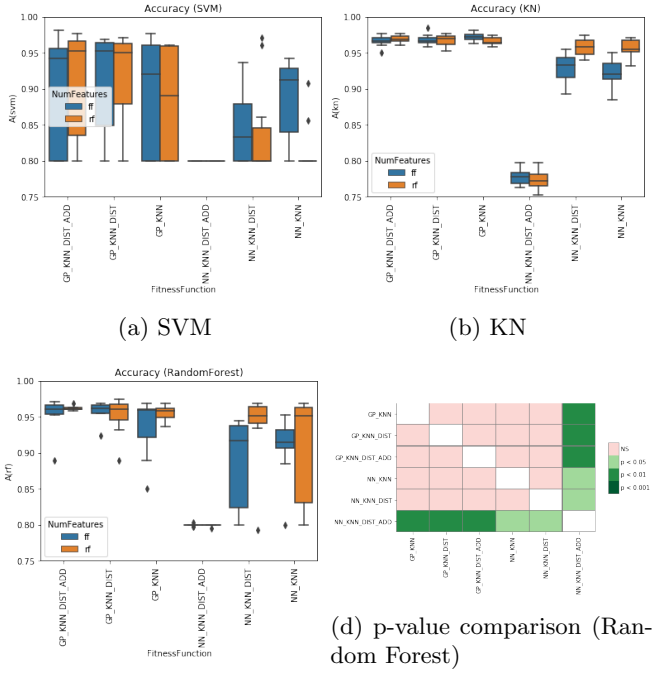
Reduced Features	A(SVM)	F1(SVM)	A(KNN)	F1(KNN)	A(RndF)	F1(RndF)
GP-KNN ($\mathcal{L}$)	0.891	0.897	**0.965**	*0.964*	**0.958**	*0.957*
GP-DIST ($\mathcal{D}$)	**0.950**	*0.949*	**0.970**	*0.969*	**0.961**	*0.960*
GP-DIST-ADD ($\mathcal{L}+\mathcal{D}$)	**0.952**	*0.952*	**0.968**	*0.968*	**0.961**	*0.960*
NN-KNN ($\mathcal{L}$)	0.800	0.711	**0.955**	*0.955*	**0.951**	*0.951*
NN-DIST ($\mathcal{D}$)	0.800	0.711	**0.958**	*0.957*	**0.951**	*0.951*
NN-DIST-ADD ($\mathcal{L}+\mathcal{D}$)	0.800	0.711	0.772	0.708	0.800	0.711
PCA	0.874	0.875	0.889	0.888	0.868	0.846
UMAP	0.909	0.907	0.903	0.902	0.913	0.911

5.2 Quantitative evaluation via proxy classification metrics

Table 2 shows the classification accuracy and F1-score obtained from each of three classifiers on the *unseen dataset* for each experiment using the projections evolved in the prior step, and compared to projections obtained from PCA and UMAP. With the exception of two combinations that use the neural network method ($\mathcal{L}$, $\mathcal{L}+\mathcal{D}$), it is clear that the evolved layouts enable all three classifiers tested to produce significantly better results that PCA and UMAP, with performance gains of over 10% in several cases. This demonstrates that the evolved layouts provided a good general basis for classification, in eliciting good performance from mutiple different types of classifier. As expected, the neighbour-based $\mathcal{L}$ fitness function creates layouts that favour the KNN classifier, which also relies on a neighbourhood method, but it is clear that the other classifiers (particularly Random Forest) are also competitive in a space evolved to favour similar neighbours. The GP approach generally outperforms the neural-network

approaches. The SVM classifier generally provides weaker results than the other two classifiers, although still markedly better than PCA/UMAP in 4 out of 6 experiments.

Figure 3 shows boxplots of results obtained on the test set from the 10 runs for each combination, plotted per classifier. Pairwise significance tests were conduct on each set of results per classifiers (Mann-Whitney test using Bonferroni correction). Similar plots of p-values were obtained for all three classifiers, however only one is shown due to space restrictions. The boxplots demonstrate that the SVM classifier has more variable performance in the evolved spaces while the other two classifiers appear robust to the projection; similarly, the neural approach tends to result in layouts in which classification performance is more variable than the spaces evolved using GP; this result is consistent across all three classifiers.



(a) SVM

(b) KN

(c) Random Forest

(d) p-value comparison (Random Forest)

Fig. 3: Boxplots showing distribution of results per proxy classifier for each layout combination. (d) shows a typical plot of p-values obtained from comparing pairs of methods: similar plots were obtained for all 3 classifiers

.

5.3 Qualitative Evaluation: Visualisation of layouts (by label)

Figure 4 shows examples of layouts obtained by the single run from the 10 runs of each combination that resulted in the best fitness for each combination of optimiser/fitness function. The plots are shaded by class label. A wide variety of layouts are observed. Note that UMAP (as expected) produces plots that favour local structure within the data. The fitness function $\mathcal{L}+\mathcal{D}$ that favours both positioning instance with the same label *and* similar performance tends to spread the instances more widely across the space. While all but one of the evolutionary methods (the exception being NN($\mathcal{L}+\mathcal{D}$) produce layouts that result in considerably higher classification accuracy than PCA/UMAP, the layouts are perhaps less easy to interpret to the human eye. In several cases, multiple instances are mapped to the same coordinates which improves classification but does not easily enable 'similar' instances (from an algorithm-selection perspective) to be easily identified. A trade-off thus exists: ultimately the choice of method depends on the priorities of the user, i.e. whether the goal is simply to select the best algorithm or from a more scientific perspective, to gain insights into algorithm performance relative to algorithm features.

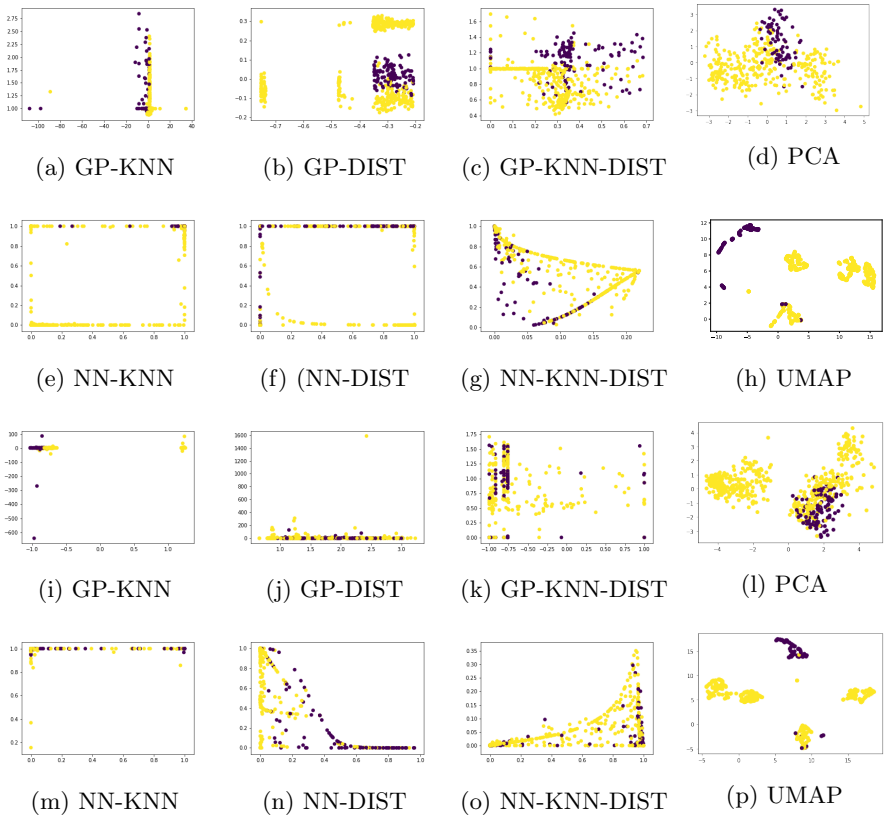
Furthermore, while the boxplots show low variance in classification accuracy across multiple runs, there is considerably variation in the layouts themselves (see figure 5 as an example). The different biases of the two fitness functions is clearly observed: note the tighter clustering of instances per class in the top row from rewarding near neighbours of the same class, while the distance based fitness function results in a wider spread of instances. This variability further emphasises the trade-offs to be considered as mentioned above.

5.4 Visualisation of layouts (by performance)

Finally we evaluate the extent to which the distance based fitness function results in layouts that place instances that elicit similar performance from the same algorithm close together. Figure 6 shows three examples obtained from separate runs of the GP optimiser with the distance-based fitness function. Instances 'won' by each class are shown on separate plots for clarity with the shading representing the relative performance of the algorithm (normalised between 0 and 1). Some clear clusters of similar performance are visible (e.g. particularly regarding the dark colours representing very strong performance), while there is some evidence of instances with weak performance (lightest colours) appearing towards the edge of clusters. However, there is clearly further work required to adapt this method to create smoother gradients across the space.

6 Conclusion

Instance-space analysis methods have been attracting increasing attention as a way of understanding the 'footprints' of an algorithm within a feature-space, enabling new insights into relative performance and algorithmic power [14], while additionally facilitating algorithm-selection. We first outlined the properties that



(a) GP-KNN (b) GP-DIST (c) GP-KNN-DIST (d) PCA

(e) NN-KNN (f) (NN-DIST (g) NN-KNN-DIST (h) UMAP

(i) GP-KNN (j) GP-DIST (k) GP-KNN-DIST (l) PCA

(m) NN-KNN (n) NN-DIST (o) NN-KNN-DIST (p) UMAP

Fig. 4: Embeddings from run with best accuracy. First row: reduced feature-set, GP plus PCA reduced features. Second row: reduced feature-set, NN, plus UMAP reduced features. Third/fourth row: as above using full feature set.

we believe an instance-space should embody, specifically that it should co-locate instances with the same class label, while also reflecting the performance-gradient across a cluster, i.e placing instances that elicit the similar performance from an algorithm close together. These factors are peculiar to the goal of instance-space analysis in optimisation. Specifically, they differ from the goals of standard dimensionality reduction methods which usually try to creating a mapping in which the ordering of neighbours of a point in a high-dimensional space is reflected in the low-dimensional space. As described in section 1 however, an ordering in the feature-space of a set of instances can differ extensively from the ordering within the performance-space, hence manifold-learning techniques might not be appropriate for instance-space analysis.

We demonstrate that both proposed optimisation methods are capable of generating layouts that provide considerable improvement in classification accuracy to UMAP and PCA, of over 10% in some cases. We also provide a more

qualitative analysis of the visualisations in terms of their ability to reflect performance gradients. Results suggest that the approach shows promise with the respect to the latter goal, although there is scope for refinement. The calculation of the k nearest-neighbours is time-consuming for a large dataset but this can be significantly improved using an efficient implementation (e.g. k-d trees) and is therefore not limiting. We deliberately chose not to conduct a wide exploration of possible neural architectures or to spend a large amount of effort in tuning the parameters of the proposed algorithms. This will be further investigated in future work. Finally, we intend to repeat the experiments in other combinatorial domains, using the MATILDA generated instance-spaces as a baseline.

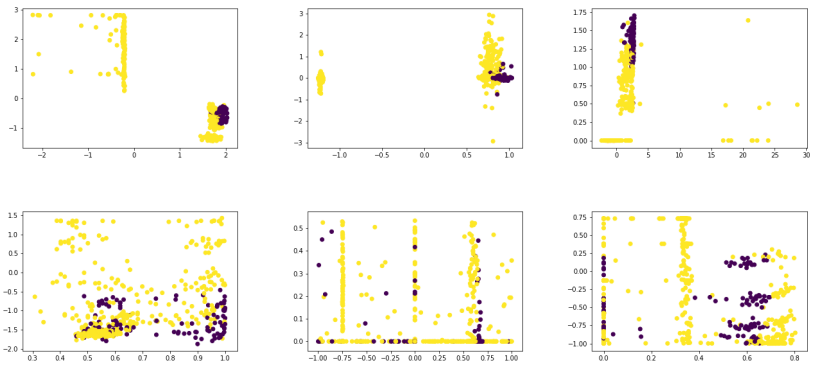


Fig. 5: Layouts from repeated runs of the same optimiser/fitness function. Top row - GP with nearest-neighbour fitness; bottom - GP with distance based fitness

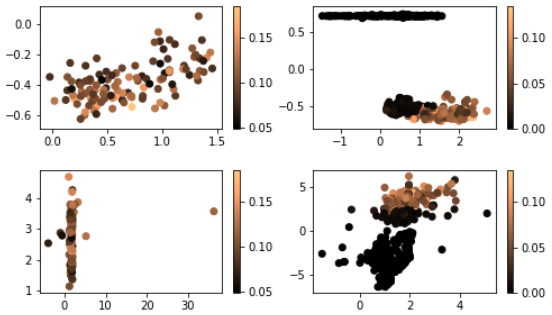


Fig. 6: Layouts from 2 runs of GP optimiser (fitness=Distance), one run per row. Instances shaded by relative performance and separated into two plots according to the class label of the instance

Acknowledgments

Hart gratefully acknowledges the support EPSRC EP/V026534/1

References

1. Deap: Distributed evolutionary algorithms in python. `https://deap.readthedocs.io/en/master/`
2. Matilda: Melbourne algorithm test instance library with data analytics. `https://matilda.unimelb.edu.au/matilda/`
3. Umap: Uniform manifold approximation and projection for dimension reduction. `https://umap-learn.readthedocs.io/en/latest/index.html`
4. Hansen, N., Müller, S.D., Koumoutsakos, P.: Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). Evolutionary computation **11**(1), 1–18 (2003)
5. Hasselmann, K., Ligot, A., Ruddick, J., Birattari, M.: Empirical assessment and comparison of neuro-evolutionary methods for the automatic off-line design of robot swarms. Nature communications **12**(1), 1–11 (2021)
6. Le Goff, L.K., Buchanan, E., Hart, E., Eiben, A.E., Li, W., de Carlo, M., Hale, M.F., Angus, M., Woolley, R., Timmis, J., et al.: Sample and time efficient policy learning with cma-es and bayesian optimisation. In: Artificial Life Conference Proceedings. pp. 432–440. MIT Press (2020)
7. Lensen, A., Xue, B., Zhang, M.: Can genetic programming do manifold learning too? In: European Conference on Genetic Programming. pp. 114–130. Springer (2019)
8. Lensen, A., Xue, B., Zhang, M.: Genetic programming for manifold learning: Preserving local topology. IEEE Transactions on Evolutionary Computation (2021)
9. Loshchilov, I., Hutter, F.: Cma-es for hyperparameter optimization of deep neural networks. arXiv preprint arXiv:1604.07269 (2016)
10. Van der Maaten, L., Hinton, G.: Visualizing data using t-sne. Journal of machine learning research **9**(11) (2008)
11. Muñoz, M.A., Villanova, L., Baatar, D., Smith-Miles, K.: Instance spaces for machine learning classification. Machine Learning **107**(1), 109–147 (2018)
12. Partridge, M., Calvo, R.A.: Fast dimensionality reduction and simple pca. Intelligent data analysis **2**(3), 203–214 (1998)
13. Schofield, F., Lensen, A.: Using genetic programming to find functional mappings for umap embeddings. In: 2021 IEEE Congress on Evolutionary Computation (CEC). pp. 704–711. IEEE (2021)
14. Smith-Miles, K., Baatar, D., Wreford, B., Lewis, R.: Towards objective measures of algorithm performance across instance space. Computers & Operations Research **45**, 12–24 (2014)
15. Smith-Miles, K., Bowly, S.: Generating new test instances by evolving in instance space. Computers & Operations Research **63**, 102–113 (2015)
16. Smith-Miles, K., Hemert, J.v., Lim, X.Y.: Understanding tsp difficulty by learning from evolved instances. In: International conference on learning and intelligent optimization. pp. 266–280. Springer (2010)
17. Smith-Miles, K., Lopes, L.: Generalising algorithm performance in instance space: A timetabling case study. In: International conference on learning and intelligent optimization. pp. 524–538. Springer (2011)
18. Smith-Miles, K., Lopes, L.: Measuring instance difficulty for combinatorial optimization problems. Computers & Operations Research **39**(5), 875–889 (2012)
19. Wang, Y., Huang, H., Rudin, C., Shaposhnik, Y.: Understanding how dimension reduction tools work: an empirical approach to deciphering t-sne, umap, trimap, and pacmap for data visualization. Journal of Machine Learning Research **22**(201), 1–73 (2021)