New Representations in Genetic Programming for Feature Construction in *k*-means Clustering

Andrew Lensen, Bing Xue, and Mengjie Zhang

School of Engineering and Computer Science,
Victoria University of Wellington, PO Box 600, Wellington 6140, New Zealand
`{Andrew.Lensen,Bing.Xue,Mengjie.Zhang}@ecs.vuw.ac.nz`

Abstract. *k*-means is one of the fundamental and most well-known algorithms in data mining. It has been widely used in clustering tasks, but suffers from a number of limitations on large or complex datasets. Genetic Programming (GP) has been used to improve performance of data mining algorithms by performing feature construction — the process of combining multiple attributes (features) of a dataset together to produce more powerful constructed features. In this paper, we propose novel representations for using GP to perform feature construction to improve the clustering performance of the *k*-means algorithm. Our experiments show significant performance improvement compared to *k*-means across a variety of difficult datasets. Several GP programs are also analysed to provide insight into how feature construction is able to improve clustering performance.

Keywords: Cluster Analysis; Feature Construction; Genetic Programming; k-means; Evolutionary Computation

1 Introduction

Clustering is a common data mining task which groups similar data items (instances) of a dataset into homogeneous groups (clusters) [1,2]. *k*-means [1,3] is one of the most widely used clustering algorithms due to its simple design and low computational cost. However, it suffers from several limitations: the quality of the clustering solution (partition) is heavily dependent on the initial cluster centroids, and cluster quality quickly decreases as the number of clusters (K) increases.

Feature construction (FC) is a common technique used to improve the performance of learning algorithms in data mining [4]. FC algorithms produce powerful high-level CFs (CFs) by combining features from the original feature set. By only using a few CFs instead of the full feature set, data mining algorithms can train more efficiently (due to a smaller search space) and more effectively, while generally producing more concise and understandable solutions [5].

Genetic Programming (GP) [6] is an Evolutionary Computation (EC) [7] technique, which has been shown to be effective at performing FC, especially on classification problems [8, 9]. GP, like other EC algorithms, produces solutions to a problem by performing a population-based heuristic search, using Darwinian inspired principles to encourage co-operation between solutions. Tree-based GP has been extensively used for FC as its representation can easily combine features in different ways [9]. One of the most successful approaches in classification tasks has been to use a *wrapper* approach, where GP is used to construct features which are then fed into an existing classifier.

This allows the performance of an existing, well-founded classifier to be improved by using a smaller number of more powerful features.

Unlike in classification tasks, there has been very little work conducted using GP for FC for clustering [10]. Most existing work does not examine the clustering performance on a large number of clusters, and no work has been proposed using a wrapper approach where GP produces CFs that are fed to an existing clustering algorithm. The performance of k-means could be improved by using such an approach, where a GP individual produces several CFs which are then fed into k-means to perform clustering. In this way, the performance of k-means can be improved beyond what is possible with the original features alone. Traditional GP program designs output only a single value from a single individual, meaning only a single CF is created.While a single CF may be adequate on easy datasets with a small K, when there are many clusters it would be very difficult to accurately partition the dataset using a single value. Hence, new GP representations would need to be developed to produce multiple CFs. The evolved system can also be taught to produce good clusters according to any measure of cluster quality, as GP individuals will learn to produce CFs to maximise the fitness of the wrapped k-means algorithm. In contrast, standard k-means simply minimises intra-cluster variance without considering any other indicators of cluster quality. As clustering will be performed on a constructed feature space, using a clustering algorithm more advanced than k-means may not be necessary, as GP should learn to produce features tailored to the clustering algorithm being wrapped.

This paper aims to explore the potential of using GP for FC with a wrapper approach to improve the clustering performance of k-means. We will:

- Propose new GP representations for producing multiple CFs from a single GP individual,
- Investigate suitable fitness functions for improving cluster quality,
- Evaluate our proposed representations and fitness functions across a variety of datasets, and
- Analyse evolved GP trees to understand the usefulness of the CFs they produce.

2 Background

2.1 Clustering

A variety of clustering algorithms have been proposed which are effective on different datasets and problems. These can be generally grouped into a number of categories such as hierarchical, density, partitional, or graph-based algorithms [11]. Popular algorithms in each of these categories includes single- or complete-linkage clustering [11]; DBSCAN [12]; k-means [3]; and the Highly Connected Subgraphs (HCS) algorithm [13]. This paper focuses on applying GP to FC with k-means, a centroid-based, partitional algorithm which is described below.

The k-means algorithm begins by randomly selecting K instances from the dataset to act as the K initial cluster centres. Each instance in the dataset is then assigned to the nearest cluster centre using a distance measure such as Euclidean distance. The centres of each cluster are then recomputed by finding the mean of all instances in the cluster. Each instance is then again assigned to its nearest cluster and cluster centres are recomputed. This process continues for a number of iterations or until the clusters stabilise. While k-means is efficient and has generally good performance, it has a number of limitations: the clusters produced are very sensitive to the initial centres chosen, when K is high performance tends to decrease, and the clusters produced will tend to be compact but may not be well-separated or well-connected. Furthermore, as k-means uses a distance measure to assign instances to clusters, it cannot produce non-hyper-spherical clusters [1]. Hyper-spherical clusters are those where instances lie in a hyper-spherical region around the cluster mean; in a two-dimensional feature space, this produces circular clusters. Clusters need not be hyper-spherically shaped; a dataset may contain clusters of varying shape (e.g. elliptical, spiral, ring, etc. [14]).

2.2 Related Work

When GP is used for FC, terminal nodes generally draw from the feature set, and function nodes are operators which operate on real values, such as the arithmetic functions. The small amount of work using GP for clustering uses a variety of approaches [10], several of which perform FC within the GP tree. However, no existing work uses a wrapper approach where an existing clustering algorithm is used for clustering. We discuss existing work which has used GP for FC in clustering in this subsection.

Boric and Estévez [15] proposed a multi-tree approach where each GP individual contains multiple trees, each of which corresponds to a single cluster. An instance is fed to to each tree, and the tree with the highest output is chosen as the instance's assigned cluster. This approach implicitly performs FC within the trees, as the terminal set contains features (only some of which are used), and the function set contains arithmetic operators to combine features. However, the authors only tested their approach on datasets with a relatively low K (a maximum of $K = 7$). If K was higher, e.g. $K = 40$, it is likely that this approach may perform poorly as training 40 trees simultaneously is very difficult due to a very large search space.

Ahn et al. [16] proposed an approach using a simple GP program design where the output of the tree directly maps to a cluster. The terminal set consists of the feature set and a random constant, and the function set contains arithmetic operators. As in Boric's work, FC is implicitly performed within a tree. The output of the tree is rounded using integer rounding to assign an instance to a cluster. This rounding technique can cause issues as it introduces an uneven search space — an output of 0.99 maps to the "0" cluster, despite being much closer in distance to the "1" cluster. Using such a multi-threshold system is a technique which is known to give poor performance in multi-class classification [5]. Using GP to perform only FC in combination with an existing clustering algorithm will avoid this issue.

3 Proposed Method

In this section, we propose two new representations for performing FC using GP for clustering. We also introduce two fitness functions which can be used to train GP with k-means to improve clustering performance.

3.1 Multi-Tree Representation

To allow multiple CFs to be produced by a single GP individual, we propose an extension to the standard single-tree GP representation, whereby an individual contains multiple trees, producing multiple CFs. The number of trees, t, required is dependent on the dataset being clustered — generally, a higher K necessitates a higher t.

The function set used contains a number of standard arithmetic operators ($+, -, \times, \div, |+|, |-|$), as well as the *max* and *min* operators. Each of these operators take two children and produce a single output. The $\div$ operator is protected division; if the second child (the divisor) is 0, the operator returns 1. The final operator in the function set is if, which takes three children and returns the value of $child_2$ if $child_1$ is positive; otherwise it returns the value of $child_3$. The if operator is used to allow conditional behaviour in the GP program. The terminal nodes consist of the features of the dataset (f_1 through to f_m for m features) as well as a random double in the range $[0, 1]$.

When a multi-tree approach is used, the crossover and mutation operators in the evolutionary process must be adapted. In this work, we use a common approach whereby crossover is performed by selecting two random individuals, selecting a random tree from each of the individuals, and then selecting a random sub-tree from each tree to use for crossover. Mutation is performed by choosing a random tree from a random individual to be mutated.

While the multi-tree approach is reasonably straightforward to design, it has a number of limitations; most notably, t must be set in advance. The crossover operator used may also be problematic; as any two random trees can be chosen for crossover, trees being crossed over may not correspond to similar CFs, and so the CFs produced are unlikely to be fully distinct from each other. Redundancy across a constructed feature set may affect the efficiency and interpretability of a given solution.

3.2 Vector Representation

To address these issues, we also propose a single-tree approach which utilises a vector representation to produce multiple CFs from a single tree. The vector representation has no t parameter and so no parameter tuning is required. We use a similar function set as in the previous approach, but adapt each function to take two vectors as input and produce a vector as output. Each function operates on the input vectors in a pairwise manner, and the output vector has length equal to that of the smaller vector. We also introduce a *concat* function which takes two vectors as input and outputs a vector which is the result of appending the second vector to the end of the first. This *concat* function allows vectors of variable length to be generated, allowing GP to automatically generate a dynamic number of CFs. By using several *concat* functions in a single tree, the constructed feature vector will grow as the tree is evaluated from bottom to top. The terminal set remains the same as in the previous approach, however each terminal node now outputs a **vector** containing the terminal value.

3.3 Fitness Function

When K is fixed, the most common fitness function is the $\sum$ Intra fitness [11]:

$$\sum \text{Intra} = \sum_{i=1}^{K} \sum_{I_a \in C_i} d(I_a, Z_i) \qquad (1)$$

where C_i represents the i^{th} cluster, $I_a \in C_i$ represents an instance in the i^{th} cluster and Z_i represents the mean of the i^{th} cluster. This fitness function is what is minimised by k-means — when K is known, we should encourage all clusters to be as compact as possible, by minimising $\sum Intra$. One limitation of this measure is that clusters are encouraged towards hyper-sphericality; clusters will be unlikely to form non-spherical shapes that can occur on certain datasets.

One way of avoiding this problem is to use a fitness function based on connectedness. Connectedness measures the extent to which instances are in the same cluster as their immediate neighbours; close instances are similar and should fall in the same cluster. We propose a new fitness function, based on that proposed by Handl et al. [17], that computes the mean connectedness of all clusters in a partition:

$$\text{Connectedness} = \frac{1}{K}\sum_{i=1}^{K}\frac{1}{|C_i|}\sum_{I_a,I_b\in C_i} d_{inverse}(I_a,I_b) \tag{2}$$

$$d_{inverse}(I_a,I_b) = min\Big[\frac{1}{d(I_a,I_b)},10\Big] \tag{3}$$

The above fitness function, which should be maximised, encourages clusters to contain instances which are close together. Equations (2) and (3) contain a number of extensions to the one proposed by Handl et al. [17]:

1. Closer neighbours are weighted more strongly, by directly using the distance between neighbours in the fitness calculation. The inverse distance is capped to a maximum of 10 (i.e. when dist ≤ 0.1) to prevent very similar/identical instances overly affecting fitness. The value of 10 was chosen empirically.
2. The average connectedness is calculated across the set of clusters, instead of summing over all instance pairs. This discourages solutions with one very large cluster (with very good connectedness) and several very small clusters.
3. The average connectedness within a cluster is used (instead of summing), to prevent very close instances from being over-emphasised in the fitness measure.

It is hoped that by using connectedness, GP will produce features that allow for non-hyper-spherical clustering — although k-means itself will create hyper-spherical clusters in terms of the CFs, the CFs created by GP need not be linear transformations of the original features. The ability of our wrapper approach to train k-means based on different fitness measures allows k-means to be adapted to perform well on datasets that it would otherwise struggle on, especially when it is used with only a few CFs.

4 Experiment Design

Each combination of the two representations and two fitness functions were evaluated on a range of datasets using a variety of metrics. In addition, k-means was run across all datasets using the full feature set. As all of the methods are non-deterministic, each method is run 30 times using different seeds; the mean for each metric is computed.

Table 1 shows the evolutionary parameters used for all the GP methods across all the datasets. For the multi-tree approach, t is set to 7 — this was found empirically to be the required number of trees in order to give good performance across all datasets. k-means is also run for 100 iterations, or until convergence is reached (i.e. when cluster centres do not move across iterations). The initial cluster centres for k-means are randomly selected from the dataset. The seed of k-means is determined using a hashing function applied to a GP tree so that each tree produces consistent partitions.

Table 1: GP parameter settings

Parameter	Value	Parameter	Value
Generations	100	Crossover Rate	80%
Population Size	1024	Mutation Rate	20%
Minimum Depth	2	Elitism	top-10
Maximum Depth	8	Selection Type	Tournament
Initial Population	Half-and-half	Tournament Size	7

Table 2: Datasets used in the experiments.

Real-World UCI datasets from [18].				Synthetic datasets from [17].			
Name	No. of Features	No. of Instances	No. of Classes	Name	No. of Features	No. of Instances	No. of Classes
Iris	4	150	3	10d10c	10	2730	10
Wine	13	178	3	10d20c	10	1014	20
Movement Libras	90	360	15	10d40c	10	1938	40
				50d10c	50	2699	10
Breast Cancer	9	683	2	50d20c	50	1255	20
				50d40c	50	2335	40
Image Segmentation	18	683	7	100d10c	100	2893	10
				100d20c	100	1339	20
Dermatology	34	359	6	100d40c	100	2212	40

4.1 Datasets

A range of synthetic and real-world datasets were used to comprehensively evaluate the proposed methods, as shown in Table 2 Datasets were scaled so that each feature had values in $[0, 1]$, to prevent bias towards features with large ranges.

The synthetic datasets are chosen from a widely-used study by Handl et al. [17]. These datasets contain 10, 50, or 100 features and 10, 20, or 40 clusters. The synthetic datasets are used to test the performance of the proposed methods when K is high; k-means and other existing methods perform poorly on high K values.

The real-world classification datasets were chosen from the UCI machine learning repository [18], which has been commonly used in clustering studies. We use these real-world datasets to evaluate how well our proposed methods can re-create the known classifications (as is common in the literature); the class labels are **not** used during training, and are only used to evaluate how well the partitions produced match the known classes. As clustering a classification dataset is harder than clustering a specifically designed clustering dataset, we generally use real-world datasets with small K, but also include the Movement Libras dataset (with $K = 15$) to give an indication of performance on hard classification problems.

4.2 Evaluation Metrics

Clustering performance is measured using the two *internal* metrics defined previously, which directly measure the quality of a cluster partition. Connectedness (see Equation (2)) evaluates how well neighbouring instances are allocated to the same cluster, and $\sum$ Intra Distance (see Equation (1)) indicates how compact the clusters are.

In addition, we use two *external* metrics to measure how well the cluster partitions produced correspond to the dataset's class labels. These are class purity, which measures the homogeneity of each cluster with respect to the class labels, and the F-measure, which measures how well pairs of instances agree in terms of the clusters they are allocated to and their class labels. These measures are defined as follows:

1. Class purity: computed according to the following steps:
   (a) For each cluster, find the majority class label of that cluster's instances.
   (b) Count the number of correctly classified instances in the cluster, where an instance is correctly classified if it belongs to the majority class.
   (c) Calculate class purity as the fraction of correctly classified instances across the dataset.
2. F-measure: We use an adaptation of the F-measure used in classification. We consider each pair of instances in turn (as it is not possible to directly decide if a given instance is in the "right" cluster) and select from the following cases:
   (a) Same class label and assigned the same cluster: true positive (TP).
   (b) Same class label and assigned **different** clusters: false negative (FN).
   (c) **Different** class labels and assigned **different** clusters: true negative (TN).
   (d) **Different** class labels and assigned the same cluster: false positive (FP).

   The F-measure is then calculated in the normal way using the total number of TPs, FPs, and FNs:

$$\text{F-measure} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{4}$$

$$\text{precision} = \frac{TPs}{TPs + FPs} \tag{5}$$

$$\text{recall} = \frac{TPs}{TPs + FNs} \tag{6}$$

5 Results and Analysis

Tables 3 and 4 show the performance of the four GP methods and k-means (using all features (AF)) across the six real-world and nine synthetic datasets respectively. MTConn and MTIntra are the multi-tree approaches using the connectedness and $\sum$ Intra fitness function respectively, with $t = 7$. VectorConn and VectorIntra are the two vector approaches, each using one of the fitness functions proposed. Each metric is labelled with an "↑" or "↓" if it should be maximised or minimised respectively. The four metrics are: Conn (connectedness), $\sum$ Intra ($\sum$ intra distance), Purity (class purity), and FM (the F-measure). For the GP methods, each result is labelled with a "+" or a "−" if it is significantly better or worse than the k-means baseline according to a Student's t-test performed with a 95% confidence interval. A lack of a "+" or "−" indicates no significant difference.

5.1 Results on Real-World Datasets

The GP methods generally perform well compared to k-means across the real-world datasets. All four of the GP methods are significantly better in terms of the F-measure on the Iris, Wine, and Image Segmentation datasets. At least one of the GP methods is significantly better than k-means on all remaining real-world datasets; GP is only significantly worse than k-means when using connectedness on the Breast Cancer dataset, where the $\sum$ Intra fitness measure gives much better performance. The connectedness

Table 3: Performance on Real-World Datasets.

Method	Conn ↑	Intra ↓	Purity ↑	FM ↑	Conn ↑	Intra ↓	Purity ↑	FM ↑
	Iris				Wine			
MTConn7	223.4^{+}	29.59^{+}	0.8989^{+}	0.8308^{+}	90.13^{+}	88.99^{-}	0.9723^{+}	0.9444^{+}
MTIntra	26.49^{-}	29.28^{+}	0.8867^{+}	0.8111^{+}	7.621^{+}	88.7^{+}	0.9663^{+}	0.933^{+}
VectorConn	223.1^{+}	29.59^{+}	0.9502^{+}	0.9086^{+}	90.12^{+}	89.01^{-}	0.9697^{+}	0.9392^{+}
VectorIntra	26.49^{-}	29.28^{+}	0.8867^{+}	0.8111^{+}	7.618^{+}	88.7^{+}	0.9661^{+}	0.9325^{+}
k-means AF	26.77	31.39	0.8116	0.7544	7.561	88.74	0.9491	0.8998
	Movement Libras				Breast Cancer			
MTConn7	19.28^{+}	424.9^{-}	0.4424^{-}	0.3417	895.8^{+}	369.7^{-}	0.9101^{-}	0.8445^{-}
MTIntra	5.473^{+}	400.2^{+}	0.472	0.3527	15.63^{+}	331.6^{+}	0.9675^{+}	0.9423^{+}
VectorConn	19.1^{+}	414.3	0.4583	0.3434	898.1^{+}	376.7^{-}	0.8972^{-}	0.824^{-}
VectorIntra	5.486^{+}	399.0^{+}	0.4749^{+}	0.3542^{+}	15.64^{+}	331.6^{+}	0.9669^{+}	0.9413^{+}
k-means AF	5.134	414.5	0.4619	0.3439	15.52	332.0	0.9609	0.9313
	Image Segmentation				Dermatology			
MTConn7	798.4^{+}	877.1^{+}	0.6832^{+}	0.5886^{+}	42.28^{+}	377.1^{+}	0.946^{+}	0.9324^{+}
MTIntra	25.39^{+}	869.5^{+}	0.6654^{+}	0.5717^{+}	3.176^{+}	376.2^{+}	0.8655^{+}	0.7915
VectorConn	797.1^{+}	873.8^{+}	0.6859^{+}	0.5894^{+}	41.7^{+}	382.8	0.9063^{+}	0.8764^{+}
VectorIntra	25.38^{+}	872.2^{+}	0.6655^{+}	0.5726^{+}	3.063	380.9^{+}	0.8538	0.7839
k-means AF	24.78	908.6	0.6383	0.5582	3.022	386.5	0.8349	0.7569

fitness measure, however, gives very good results on the Dermatology dataset, improving performance over *k*-means significantly. The fact that different fitness functions perform better on different datasets shows the usefulness of our proposed methods to train using a range of criteria, unlike the original *k*-means algorithm. Both the multi-tree and the vector approaches appear to perform similarly on the real-world datasets, with an exception on the Iris dataset, where the vector approach is superior when connectedness is used in terms of the external metrics.

5.2 Results on Synthetic Datasets

The GP methods continue to perform well compared to *k*-means on the synthetic datasets. All four methods have a significantly higher F-measure value than *k*-means on the datasets with 20 or 40 clusters. These datasets are the most difficult as they require separating the dataset into the greatest number of distinct groups. *k*-means performs very poorly when there is a large number of clusters (e.g. $K = 40$); GP is able to effectively perform FC to significantly improve the performance of *k*-means on the hardest datasets, while only using a small subset of the feature set. Some GP methods perform significantly worse on the simple 10d10c and 50d10c datasets, but at least one GP method is still significantly better than *k*-means in these cases.

The connectedness and $\sum$ Intra fitness measures are again superior on different datasets. The method using connectedness are significantly better than *k*-means on the 50d10c dataset, whereas those using $\sum$ Intra fitness are significantly worse. The inverse is true on the 10d10c dataset, however. In general, the multi-tree approach seems slightly better than the vector approach, especially on the datasets with highest dimensionality such as 100d20c and 100d40c. Future testing is required to evaluate which method is superior, and more work could be done to improve each method by further exploring alternative representations or fitness functions.

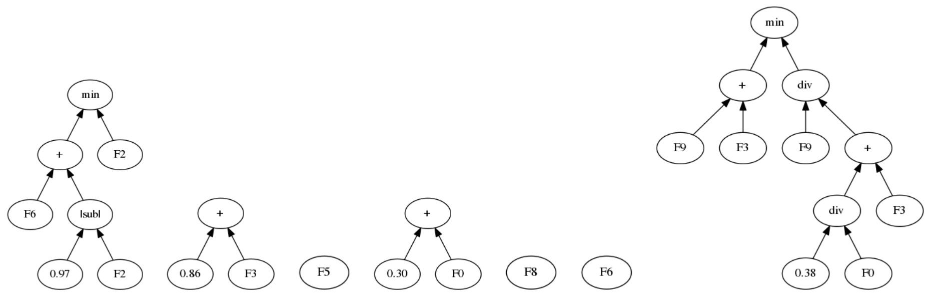
Table 4: Performance on Synthetic Datasets.

Method	Conn $\uparrow$	Intra $\downarrow$	Purity $\uparrow$	FM $\uparrow$	Conn $\uparrow$	Intra $\downarrow$	Purity $\uparrow$	FM $\uparrow$
	10d10c				10d20c			
MTConn	823.3^{+}	719.1^{-}	0.9019^{-}	0.7836^{-}	177.0^{+}	213.2^{+}	0.9948^{+}	0.9919^{+}
MTIntra	17.67^{-}	710.1^{+}	0.9294	0.878	16.5^{+}	213.0^{+}	0.9948^{+}	0.9919^{+}
VectorConn	827.3^{+}	713.9	0.9153^{-}	0.8025^{-}	177.0^{+}	213.5^{+}	0.9941^{+}	0.9887^{+}
VectorIntra	18.05^{+}	706.3^{+}	0.9404^{+}	0.8926^{+}	16.48^{+}	213.5^{+}	0.9938^{+}	0.9906^{+}
k-means AF	17.88	712.4	0.9291	0.8571	15.29	254.8	0.8732	0.7969
	10d40c				50d10c			
MTConn	173.2^{+}	406.5^{+}	0.9747^{+}	0.9311^{+}	589.4^{+}	1480.0^{-}	0.7325^{-}	0.5167^{+}
MTIntra	16.34^{+}	405.0^{+}	0.977^{+}	0.9456^{+}	17.14^{-}	1220.0^{+}	0.7392	0.4785^{-}
VectorConn	173.6^{+}	403.3^{+}	0.9789^{+}	0.9409^{+}	587.3^{+}	1437.0^{-}	0.7278^{-}	0.5005
VectorIntra	16.32^{+}	404.1^{+}	0.9771^{+}	0.9494^{+}	17.22^{-}	1216.0^{+}	0.7397	0.4795^{-}
k-means AF	15.75	436.8	0.9182	0.8628	17.49	1317.0	0.744	0.4939
	50d20c				50d40c			
MTConn	163.3^{+}	583.2^{-}	0.7138^{+}	0.4996^{+}	163.3^{+}	894.7^{-}	0.685	0.4397^{+}
MTIntra	17.43	493.8^{+}	0.7456^{+}	0.4776^{+}	18.95^{-}	833.8^{+}	0.6952^{+}	0.4269^{+}
VectorConn	162.5^{+}	555.7	0.7212^{+}	0.4832^{+}	165.4^{+}	850.4^{+}	0.7082^{+}	0.4106^{+}
VectorIntra	17.52	487.2^{+}	0.7412^{+}	0.4351^{+}	19.3^{+}	797.1^{+}	0.7165^{+}	0.3759^{+}
k-means AF	17.33	546.5	0.6868	0.3823	19.16	865.2	0.6791	0.2618
	100d10c				100d20c			
MTConn	521.8^{+}	2123.0^{-}	0.7598	0.5311	126.0^{+}	885.4^{-}	0.7084	0.4657^{+}
MTIntra	15.81^{+}	1776.0^{+}	0.7835^{+}	0.5825^{+}	13.66^{+}	764.9^{+}	0.7481^{+}	0.4598^{+}
VectorConn	519.5^{+}	2077.0^{-}	0.7595	0.5446	125.5^{+}	850.5	0.7122	0.4451^{+}
VectorIntra	15.89^{+}	1771.0^{+}	0.7839^{+}	0.5854^{+}	13.74^{+}	749.6^{+}	0.7466^{+}	0.4331^{+}
k-means AF	15.14	1968.0	0.748	0.5255	13.31	844.2	0.7033	0.38
	100d40c							
MTConn	114.8^{+}	1234.0^{-}	0.6963	0.4629^{+}				
MTIntra	14.18^{-}	1118.0^{+}	0.7181^{+}	0.462^{+}				
VectorConn	116.0^{+}	1159.0	0.7142^{+}	0.4418^{+}				
VectorIntra	14.45	1061.0^{+}	0.7344^{+}	0.4028^{+}				
k-means AF	14.55	1184.0	0.6904	0.2675				

6 Evolved Program Analysis

It is often useful when using GP to evaluate and analyse some of the high-performing individuals produced during the evolutionary process. Doing so allows us to understand what properties of a given tree allow it to perform well, which leads to a better understanding of the problem as well allowing the GP method to be improved further. In addition, analysing evolved programs increases the confidence in our proposed method by demonstrating how it is able to achieve the good results we claim. In this section, we analyse a number of GP trees with high F-measure across a range of datasets.

An example of the multi-tree approach can be seen in Figure 1. The seven trees produced by an individual with a very high F-measure value of 0.9947 is shown, along with the constructed feature set generated which consists of seven features, one from each tree. Of these trees, three are simply performing feature selection of a single feature, two add a constant value to a single feature, and the remaining two are performing more advanced FC. In total, seven of the original 10 features are used. Although the dimensionality has not been greatly reduced, performance is still much higher than that of the original *k*-means algorithm (which achieves an F-measure



(a) The evolved multi-tree GP individual with 7 trees.

$$\left[min(F_6 + |0.97 - F_2|, F_2),\ (0.86 + F_3),\ F_5,\ (0.3 + F_0),\ F_8,\ F_6,\ min\Big((F_9 + F_3),\ \frac{F9}{(\frac{0.38}{F_0} + F_3)}\Big)\right]$$

(b) Constructed feature set generated by the programs.

Fig. 1: An evolved individual on the 10d20c dataset using the *multi-tree* approach (F-measure = 0.9947).

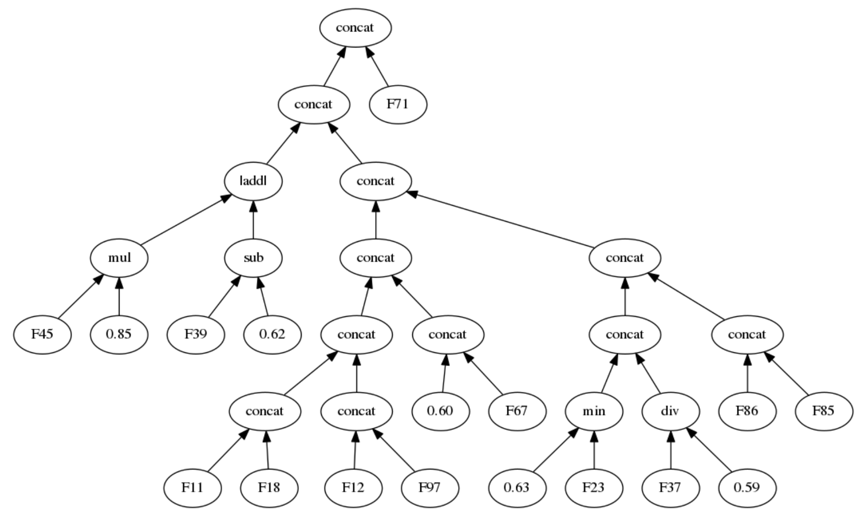
value of 0.7969). This further highlights the ability of GP to improve performance by selecting the most important features, and creating more powerful high-level features.

Figure 2 shows a GP individual using the vector approach with high performance on the hardest synthetic dataset (100d40c). The individual is a reasonably concise tree, with a maximum width of eight nodes and a depth of seven. The tree itself is shown in Figure 2a, and the output of the tree as shown in Figure 2b. The tree selects feature values as terminal nodes, and outputs a constructed feature vector of length 12, containing 11 "constructed" features and one constant value. Of these CFs, one is an arithmetic combination of two selected features and two constants, two are operations applied to a selected feature and a constant value, and the remaining nine are unchanged selected features. k-means achieved an F-measure value of 0.2618 on average; this GP individual produced nearly double the F-measure value while only using 12 features compared to the 100 original features that k-means used. This large increase in performance shows the power of GP in improving performance by creating more powerful high-level features while also reducing dimensionality.

A useful property of the vector approach is its ability to dynamically produce a variable number of CFs. For example, on the Iris dataset which has only three classes, it is unnecessary to have seven CFs (as occurs for $t = 7$ in the multi-tree approach) and having so many features may reduce the interpretability of the solution. Figure 3 shows a high performing, very simple GP individual produced on the Iris dataset. This tree is very easy to analyse: it simply selects two of the four features in the dataset (F_3 and F_2). By not selecting the other misleading or redundant features, this GP tree significantly improves the ability of k-means to produce a good cluster partition.

7 Conclusion

This work showcased the ability of GP to be used for feature construction for clustering; the performance of k-means was significantly improved by using GP to automatically construct a few high-level features. We proposed two representations, using a multi-

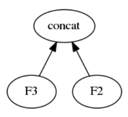


(a) The GP tree.

$\left[|(F_{45} \times 0.85) + (F_{39} - 0.62)|,\, F_{11},\, F_{18},\, F_{12},\, F_{97},\, 0.6,\, F_{67},\, min(0.63, F_{23}),\, \frac{F_{37}}{0.59},\, F_{86},\, F_{85},\, F_{71}\right]$

(b) Constructed feature set generated by the program.

Fig. 2: An evolved individual on the 100d40c dataset using the *vector* approach (F-measure of 0.499).



(a) The GP tree.

$[F3,\, F2]$

(b) Feature set.

Fig. 3: An evolved individual on the Iris dataset with using the vector approach (F-measure = 0.9233).

tree and a vector approach, and explored two potential fitness functions (connectedness and $\sum$ intra fitness) that could be used for training high performing GP trees. Both representations and fitness functions were shown to give significantly improved performance compared to the base k-means algorithm across a range of real-world and difficult synthetic datasets. A number of evolved GP trees were analysed and shown to perform effective and efficient FC even in a very small tree.

As GP has seen little use in FC for clustering, there are many promising future research areas that could be explored. The representations and fitness functions used in this work could be further improved, and many other representations and fitness functions are possible. For example, the vector approach could be adapted to directly encourage shorter constructed feature vectors to be produced (thereby producing more powerful CFs). The multi-tree approach would be improved if the number of trees could be determined automatically — for example, using a heuristic based on K (a higher number of clusters should genuinely mean more CFs are required). It may also be worthwhile to investigate using other clustering algorithms besides k-means; while in theory it is possible for GP to produce optimal CFs that k-means can use to produce perfect partitions, other algorithms may be more powerful and perform well with a wider range of CFs. The methods we proposed were all designed to work when K was pre-defined, as k-means requires K to be known. This may be inflexible in practice — extending these methods to automatically determine K would be beneficial.

References

1. Jain, A.K.: Data clustering: 50 years beyond k-means. Pattern Recognition Letters **31**(8) (2010) 651–666
2. García, A.J., Gómez-Flores, W.: Automatic clustering using nature-inspired metaheuristics: A survey. Appl. Soft Comput. **41** (2016) 192–213
3. J. A. Hartigan, M.A.W.: Algorithm AS 136: A k-means clustering algorithm. Journal of the Royal Statistical Society. Series C (Applied Statistics) **28**(1) (1979) 100–108
4. Liu, H., Motoda, H.: Feature extraction, construction and selection: A data mining perspective. Springer Science & Business Media (1998)
5. Espejo, P.G., Ventura, S., Herrera, F.: A survey on the application of genetic programming to classification. IEEE Trans. Systems, Man, and Cybernetics, Part C **40**(2) (2010) 121–144
6. Koza, J.R.: Genetic programming: on the programming of computers by means of natural selection. Volume 1. MIT press (1992)
7. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. Natural Computing Series. Springer (2015)
8. Neshatian, K., Zhang, M., Andreae, P.: A filter approach to multiple feature construction for symbolic learning classifiers using genetic programming. IEEE Trans. Evolutionary Computation **16**(5) (2012) 645–661
9. Tran, B., Xue, B., Zhang, M.: Genetic programming for feature construction and selection in classification on high-dimensional data. Memetic Computing **8**(1) (2016) 3–15
10. Nanda, S.J., Panda, G.: A survey on nature inspired metaheuristic algorithms for partitional clustering. Swarm and Evolutionary Computation **16** (2014) 1–18
11. Aggarwal, C.C., Reddy, C.K., eds.: Data Clustering: Algorithms and Applications. CRC Press (2014)
12. Ester, M., Kriegel, H., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA. (1996) 226–231
13. Hartuv, E., Shamir, R.: A clustering algorithm based on graph connectivity. Inf. Process. Lett. **76**(4-6) (2000) 175–181
14. Tseng, L.Y., Yang, S.B.: A genetic clustering algorithm for data with non-spherical-shape clusters. Pattern Recognition **33**(7) (2000) 1251–1259
15. Boric, N., Estévez, P.A.: Genetic programming-based clustering using an information theoretic fitness measure. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC). (2007) 31–38
16. Ahn, C.W., Oh, S., Oh, M.: A genetic programming approach to data clustering. In: Proceedings of the International Conference on Multimedia, Computer Graphics and Broadcasting (MulGraB), Part II. (2011) 123–132
17. Handl, J., Knowles, J.D.: An evolutionary approach to multiobjective clustering. IEEE Trans. Evolutionary Computation **11**(1) (2007) 56–76
18. Lichman, M.: UCI machine learning repository (2013)