Evolutionary Computation

Genetic Programming for Evolving Similarity Functions for Clustering: Representations and Analysis

--Manuscript Draft--

Manuscript Number:	
Full Title:	Genetic Programming for Evolving Similarity Functions for Clustering: Representations and Analysis
Article Type:	Original Article
Corresponding Author:	Andrew Lensen Victoria University of Wellington Wellington, NEW ZEALAND
Other Authors:	Bing Xue, PhD
	Mengjie Zhang, PhD
Abstract:	Clustering is a difficult and widely-studied data mining task, with many varieties of clustering algorithms proposed in the literature. Nearly all algorithms use a similarity measure such as a distance metric (e.g.\ Euclidean distance) to decide which instances to assign to the same cluster. These similarity measures are generally pre-defined and cannot be easily tailored to the properties of a particular dataset, which leads to limitations in the quality and the interpretability of the clusters produced. In this paper, we propose a new approach to automatically evolving similarity functions for clustering using genetic programming. We introduce a new genetic programming-based method which automatically selects a small subset of features (feature selection) and then combines them using a variety of functions (feature construction) to produce dynamic and flexible similarity functions that are specifically designed for a given dataset. We demonstrate how the evolved similarity functions can be used to perform clustering using a graph-based representation. The results of a variety of experiments across a range of large, high-dimensional datasets show that the proposed approach can achieve higher and more consistent performance than the benchmark methods. We further extend the proposed approach to automatically produce multiple complementary similarity functions by using a multi-tree approach, which gives further performance improvements. We also analyse the interpretability and structure of the automatically evolved similarity functions to provide insight into how and why they are superior to standard distance metrics.
Keywords:	Cluster analysis; genetic programming; feature construction
Manuscript Classifications:	Artificial intelligence; Data mining; Genetic programming

Genetic Programming for Evolving Similarity Functions for Clustering: Representations and Analysis

Andrew Lensen Andrew.Lensen@ecs.vuw.ac.nz
Evolutionary Computation Research Group, Victoria University of Wellington, Wellington 6140, New Zealand

Bing Xue Bing.Xue@ecs.vuw.ac.nz
Evolutionary Computation Research Group, Victoria University of Wellington, Wellington 6140, New Zealand

Mengjie Zhang Mengjie.Zhang@ecs.vuw.ac.nz
Evolutionary Computation Research Group, Victoria University of Wellington, Wellington 6140, New Zealand

Abstract

Clustering is a difficult and widely-studied data mining task, with many varieties of clustering algorithms proposed in the literature. Nearly all algorithms use a similarity measure such as a distance metric (e.g. Euclidean distance) to decide which instances to assign to the same cluster. These similarity measures are generally pre-defined and cannot be easily tailored to the properties of a particular dataset, which leads to limitations in the quality and the interpretability of the clusters produced. In this paper, we propose a new approach to automatically evolving similarity functions for clustering using genetic programming. We introduce a new genetic programming-based method which automatically selects a small subset of features (feature selection) and then combines them using a variety of functions (feature construction) to produce dynamic and flexible similarity functions that are specifically designed for a given dataset. We demonstrate how the evolved similarity functions can be used to perform clustering using a graph-based representation. The results of a variety of experiments across a range of large, high-dimensional datasets show that the proposed approach can achieve higher and more consistent performance than the benchmark methods. We further extend the proposed approach to automatically produce multiple complementary similarity functions by using a multi-tree approach, which gives further performance improvements. We also analyse the interpretability and structure of the automatically evolved similarity functions to provide insight into how and why they are superior to standard distance metrics.

Keywords

Cluster analysis, automatic clustering, genetic programming, similarity function, feature selection, feature construction.

1 Introduction

Clustering is a fundamental data mining task (Fayyad et al., 1996), which aims to group related/similar instances into a number of clusters where the data is unlabelled. It is one of the key tasks in exploratory data analysis, as it enables data scientists to reveal the underlying structure of unfamiliar data, which can then be used for further analysis (Jain, 2010).

Nearly all clustering algorithms utilise a similarity measure, usually a distance function, to perform clustering as close instances are similar to each other, and expected to be in the same cluster. The most common distance functions, such as Manhattan or Euclidean distance, are quite inflexible: they consider all features equally despite features often varying significantly in their usefulness. Consider a weather dataset of daily records which contains the two following features: *day of week* and *rainfall (mm)*. Clusters generated using the rainfall feature will give insight into what days are likely to be rainy, which may allow better prediction of whether we should take an umbrella in the future. Clusters generated with the day of week feature however are likely to give little insight or be misleading — intuitively, we know that the day of the week has no effect on long-term weather patterns and so any clusters produced could mislead us. Ideally, we would like to perform *feature selection* to select only the most useful features in a dataset. These distance functions also have uniform behaviour across a whole dataset, which makes them struggle with common problems such as clusters of varying density or separation, and noisy data. Indeed, trialling a range of similarity measures is commonly a tedious but necessary parameter tuning step when performing cluster analysis. *Feature construction* is the technique of automatically combining existing low-level features into more powerful high-level features. Feature construction could produce similarity measures which are better fitted to a given dataset, by combining features in a non-uniform and flexible manner. These *feature reduction* techniques have been shown to be widely effective in both supervised and unsupervised domains (Xue et al., 2016; Aggarwal and Reddy, 2014).

A variety of representations have been proposed for modelling clustering solutions. The graph representation models the data in an intuitive way, where instances (represented as nodes) are connected by an edge if they are *similar enough* (von Luxburg, 2007). This is a powerful representation that allows modelling a variety of cluster shapes, sizes, and densities, unlike the more common prototype-based representations such as k-means. However, algorithms using graph representations are very dependent on the criterion used to select edges (von Luxburg, 2007). One of the most common criteria is to simply use a fixed threshold (von Luxburg, 2007), which indicates the distance at which two instances are considered too dissimilar to share an edge. Such a threshold must be determined independently for every dataset, and this approach typically does not allow varying thresholds to be used in different clusters. Another popular criterion is to put an edge between each instance and its N-nearest neighbours (von Luxburg, 2007), where N is a small integer value such as 2, 3, or 4. N must also be determined before running the algorithm, with results being very sensitive to the N value chosen. Again, this method does not allow for varied cluster structure.

Many of the above issues can be tackled by using a similarity function which is able to be *automatically* tailored to a specific dataset, and which can treat different clusters within a dataset differently. Genetic programming (GP) (Koza, 1992) is an evolutionary computation (EC) (Eiben and Smith, 2015) method that automatically evolves *programs*. The most common form of GP is *tree-based* GP, which models solutions in the form of a tree which takes input (such as a feature set) and produces an output, based on the functions performed within the tree. We hypothesise that this approach can be used to automatically evolve similarity functions that are represented as GP trees, where a tree takes two instances as input and produces an output that corresponds to how similar those two instances are. By automatically combining only the most relevant features (i.e. performing feature selection and construction), more powerful and specific similarity functions can be generated to improve clustering performance on a range of

datasets. GP can also use multiple trees to represent each individual/solution. Multi-tree GP has the potential to automatically generate multiple complementary similarity functions, which are able to *specialise* on different clusters in the dataset. To our best knowledge, such an approach has not been investigated to date.

1.1 Goals

This work aims to propose the first approach to using GP for automatically evolving similarity functions with a graph representation for clustering (GPGC). This work is expected to improve clustering performance while also producing more interpretable clusters which use only a small subset of the full feature set. We will investigate:

- how the output of an evolved similarity function can be used to create edges in clustering with a graph representation;
- what fitness function should be used to give high-quality clustering results;
- whether using multiple similarity functions to make a consensus decision can further improve clustering results; and
- whether the evolved similarity functions are more interpretable and produce simpler clusters than standard distance functions.

A piece of preliminary work was presented in our previous research (Lensen et al., 2017a), which proposed evolving a single similarity function with a single-tree approach for clustering. This work extends the preliminary work significantly by providing more detailed and systematic description and justification and introducing a multi-tree approach, as well as much more rigorous comparisons to existing techniques and more detailed analysis of the proposed method.

2 Background

2.1 Clustering

A huge variety of approaches have been proposed for performing clustering (Xu and II, 2005; Aggarwal and Reddy, 2014), which can be generally categorised into hard, soft (fuzzy), or hierarchical clustering methods. In hard and soft clustering, each instance belongs to exactly one or to at least one cluster respectively. In contrast, hierarchical clustering methods build a hierarchy of clusters, where a parent cluster contains the union of its child clusters. The majority of work has focused on hard clustering, as partitions where each instance is in exactly one cluster tend to be easier to interpret and analyse. A number of distinct models have been proposed for performing hard clustering: prototype-based models (including the most famous clustering method k-means (J. A. Hartigan, 1979), and its successor k-means++ (Arthur and Vassilvitskii, 2007)), density-based models (e.g. DBSCAN (Ester et al., 1996) and OPTICS (Ankerst et al., 1999)), graph-based models (e.g. the Highly Connected Subgraph (HCS) algorithm (Hartuv and Shamir, 2000)), and statistical approaches such as distribution-based (e.g. EM clustering) and kernel-based models. Prototype-based models produce a number of prototypes, each of which corresponds to a unique cluster, and then assigns each instance to its nearest prototype using a distance function, such as Euclidean distance. While these models are the most popular, they are inherently limited by their use of prototypes to define clusters: when there are naturally clusters that are non-hyper-spherically shaped, prototype-based models will tend to perform poorly as minimising the distance of instances to the prototype encourages spherical clusters. This problem is further exemplified when a cluster is non-convex.

Graph-based clustering algorithms (von Luxburg, 2007) represent clusters as distinct graphs, where there is a path between every pair of instances in a cluster graph. This representation means that graph-based measures are not restricted to clusters with hyper-spherical or convex shapes. The HCS algorithm (Hartuv and Shamir, 2000) uses a similarity graph which connects instances sharing a similarity value (e.g. distance) above a certain threshold, and then iteratively splits graphs which are not *highly connected* by finding the minimum cut, until all graphs are highly connected. Choosing a good threshold value in HCS can be difficult when there is no prior knowledge of the data.

EC techniques have also been applied to clustering successfully (Lorena and Furtado, 2001; Picarougne et al., 2007; Nanda and Panda, 2014; García and Gómez-Flores, 2016; Sheng et al., 2016) with many genetic algorithms (GA) and particle swarm optimisation (PSO) techniques used to automatically evolve clusters. Again, the majority of the literature tends to use prototype-based models, and little work uses feature reduction techniques to improve the performance of clustering methods and to produce more interpretable clusters. There is notably a deficit of methods using GP for clustering, and no current methods, asides from our preliminary work (Lensen et al., 2017a), that use GP to automatically evolve similarity functions. Relevant EC clustering methods will be discussed further in the related work section.

2.2 Feature Reduction

Feature reduction is a common strategy used to improve the performance of data mining algorithms and interpretability of the models or solutions produced (Liu and Motoda, 2012). The most common feature reduction strategy is *feature selection*, where a subset of the original feature set is selected for use in the data mining algorithm. Using fewer features can decrease training time, produce more concise and understandable results, and even improve performance by removing irrelevant/misleading features or reducing over-fitting (in supervised learning). Feature selection has been extensively studied, on a range of problems, such as classification (Tang et al., 2014) and clustering (Alelyani et al., 2013). *Feature construction*, another feature reduction strategy, focuses on automatically producing new high-level features, which combine multiple features from the original feature set in order to produce more powerful constructed features (CFs). As with feature selection, the use of feature construction can improve performance and interpretability by automatically combining useful features.

Research into the use of EC techniques for performing feature reduction has become much more popular during the last decade, due to the ability of EC techniques to efficiently search a large feature set space. Feature selection has been widely studied using PSO and GAs (García-Pedrajas et al., 2014; Xue et al., 2016), and GP has emerged as a powerful feature construction technique due to its tree representation allowing features to be combined in a hierarchical manner using a variety of functions (Espejo et al., 2010; Neshatian et al., 2012). Despite this, the use of EC for feature reduction in clustering tasks has thus far been relatively unexplored. Given that clustering is an unsupervised learning task with a huge search space, especially when there are many instances, features, or clusters, good feature reduction methods for clustering are needed.

2.3 Subspace Clustering

Another approach for performing feature reduction in clustering tasks is *subspace clustering* (Liu and Yu, 2005; Müller et al., 2009), where each cluster is located in a subspace of the data, i.e. it uses only a subset of the features. In this regard, each cluster is able to

correspond to a specific set of features that are used to characterise that cluster, which has the potential to produce better-fitted and more interpretable clusters. Several EC methods have been proposed for performing subspace clustering (Vahdat and Heywood, 2014; Peignier et al., 2015). However, subspace clustering intrinsically has an even larger search space than normal clustering, as the quantity and choice of features must be made for every cluster, rather than only once for the dataset (Parsons et al., 2004). In this paper, we do not strictly perform subspace clustering, but rather we allow the proposed approach to use different features in different combinations across the cluster space.

2.4 Related Work

There has been a handful of work proposed that uses a graph-based representation in conjunction with EC for performing clustering. One notable example is the MOCK algorithm (Handl and Knowles, 2007), which uses a GA with a locus-based graph approach to perform multi-objective clustering. Another GA method has also been proposed, which take inspiration from spectral clustering and uses either a label-based or medoid-based encoding to cluster the similarity graph (Menéndez et al., 2014).

The use of GP for performing clustering is very sparse in the literature, with only about half a dozen pieces of work proposed. One early work uses a grammar-based approach (Falco et al., 2005) where a grammar is evolved to assign instances to clusters based on how well they matched the evolved *formulae*. Instances that do not match any formulae are assigned to the closest centroid. This assignment technique, and the fitness function used, means that the proposed method is biased towards hyper-spherical clustering. Boric et al. (Boric and Estévez, 2007) proposed a multi-tree representation, where each tree in an individual corresponds to a single cluster. This method required the number of trees (t) to be set in advance, i.e. the number of clusters must be known *a priori*, which may not be available in many cases. A single-tree approach has also been proposed (Ahn et al., 2011), which uses integer rounding to assign each instance to a cluster based on the output of the evolved tree. Such an approach is unlikely to work well on datasets with a relatively high K, and this method produces a difficult search space due to clusters having an implicit order. Another proposed approach (Coelho et al., 2011) uses GP to automatically build consensus functions that combine the output of a range of clustering algorithms to produce a fusion partition. This is suggested to combine the benefits of each of the clustering algorithms, while avoiding their limitations. Each of the clustering algorithms use a fixed distance function to measure similarity between instances, and several of the algorithms require that K is known in advance. More recently, a GP approach has been proposed based on the idea of novelty search (Naredo and Trujillo, 2013), where in lieu of an explicit fitness function, the *uniqueness* (novelty) of a solution in the behavioural landscape is used to determine whether it is used in subsequent generations. This approach was only tested on problems with two clusters, and it is unclear how it would scale as K increases, given that the behavioural landscape would become exponentially larger.

While there has been very little work utilising feature construction techniques for improving the performance of clustering, there has been a significant amount of study into using feature selection for clustering problems (Dy and Brodley, 2004; Alelyani et al., 2013), with dimensionality reduction approaches such as Principal Component Analysis (PCA) (Jolliffe, 2011) being used with both EC (Kuo et al., 2012) and non-EC clustering methods. In addition, there has been some work using EC to perform simultaneous clustering and feature selection, with the aim of concurrently tailoring

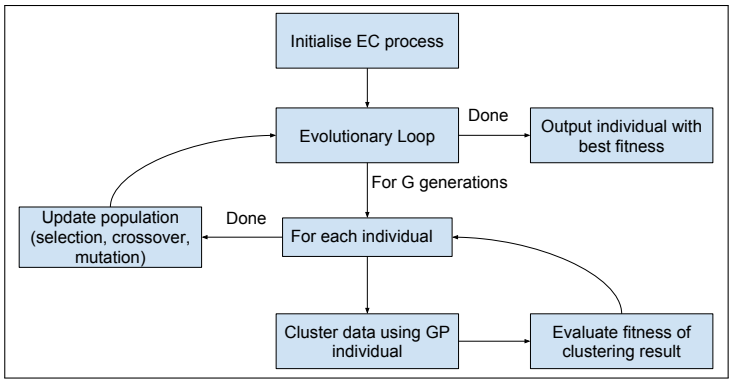


Figure 1: The overall flow of the proposed GPGC algorithm. The clustering process is discussed in detail in Section 3.2, and is shown in Algorithm 1.

the features selected to the clusters produced. PSO, in particular, has been shown to be effective on this task (Sheng et al., 2008; Lensen et al., 2017b).

The clustering literature has an overwhelming focus on producing novel clustering algorithms which employ a wide range of techniques for modelling and searching the clustering problem space. However, there has been very little focus on new techniques for automatically creating more appropriate and more powerful similarity measures to accurately model the relationships between instances on a specific dataset. GP, with its intrinsic function-like solution structure, is a natural candidate for automatically evolving similarity functions tailored to the data it is trained on. GP, and EC methods in general, have been shown to be effective on large dataset sizes and dimensionality; GP has the potential to evolve smaller, more refined, and more interpretable similarity functions on very big datasets. This paper investigates the capability of GP for automatically constructing power similarity functions.

3 Proposed Approaches

An overview of the proposed GPGC algorithm is shown in Fig. 1. We discuss the different parts of this overall algorithm in the following subsections.

3.1 GP Program Design

To represent a similarity function, a GP tree must take two instances as input and produce a single floating-point output corresponding to the similarity of the two instances. Therefore, we define the terminal set as all feature values of both instances, such that there are $2m$ possible terminals for m features (I_0F_0 and I_1F_0 through to I_0F_{m-1} and I_1F_{m-1}), as well as a random floating-point value (for scaling purposes). The function set comprises of the normal arithmetic functions ($+$, $-$,$\times$,$\div$), two absolute arithmetic functions ($|+|$ and $|-|$), and the max, min and if operators. All of these functions asides from if take two inputs and output a single value which is the result of applying that function. The if function takes three inputs and outputs the second input if the first is positive, or the third input if it is not. We include the if, max, and min functions to allow conditional behaviour within a program, in order to allow the creation of similarity functions which operate differently across the feature space. The $\div$ operator is protected division: if the divisor (the second input) is zero, the operator will return a value of one. An example of a similarity function using this GP program design is

shown in Fig. 2.

3.2 Clustering Process

As we are using a graph representation, every pair of instances which are deemed close enough by an evolved GP tree should be connected by an edge. As discussed before, we would like to refrain from using a fixed similarity threshold as varying thresholds may be required across a dataset due to varying cluster density. We therefore use the approach where each instance is connected to a number of its most similar neighbours (according to the evolved similarity function).

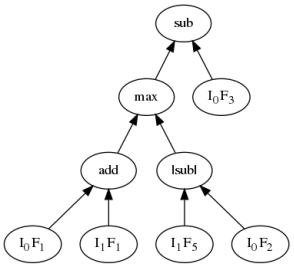


Figure 2: An example of of a similarity function with the expression $sub\big(max(add(I_0F_1, I_1F_1), |sub|(I_1F_5, I_0F_2)), I_0F_3\big)$.

To find the most similar neighbour of a given instance for an evolved similarity function requires comparing the instance to every other instance in the dataset. Normally, when using a distance metric, these pairwise similarities could be precomputed; however, in the proposed algorithm, these must be computed separately for every evolved similarity function, giving $O(n^2)$ comparisons for every GP individual on every generation of the training process, given n instances. In order to reduce the computational cost, we use a heuristic whereby each instance is only compared to its l nearest neighbours based on Euclidean distance. The set of nearest neighbours can be computed at the start of the EC training process, meaning only $O(nl)$ comparisons are required per GP individual. By using this approach, we balance the flexibility of allowing an instance to be connected to many different neighbours with the efficiency of using a subset of neighbours to compare to. As we use Euclidean distance only to give us the *order* of neighbours, the problems associated with Euclidean distance at high dimensions should not occur. We found in practice that setting l as $l = \lceil \sqrt[3]{n} \rceil$ gave a good neighbourhood size that is proportional to n, while ensuring l is at least 2 (Lensen et al., 2017b).

Algorithm 1 shows the steps used to produce a cluster for a given GP individual, X. For each instance I in the dataset, the nearest l neighbours are found using the precomputed Euclidean distance mappings. Each of these l neighbours is then fed into the bottom of the tree (X) along with I. The tree is then evaluated, and produces an output corresponding to the similarity between I and that neighbour. The neighbour with the highest similarity is chosen, and an edge is added between it and I. As in (Lensen et al., 2017a), we tested adding edges to more than one nearest neighbour, but found that performance tended to drop. Once this process has been completed for each $I \in Dataset$, the set of edges formed will give a set of graphs, where each graph represents a single cluster. These graphs can then be converted to a set of clusters by assigning all instances in each graph to the same cluster.

3.3 Fitness Function

The most common measures of cluster quality are cluster compactness and separability (Aggarwal and Reddy, 2014). A good cluster partition should have distinct clusters which are very dense in terms of the instances they contain, and which are far away from other clusters. A third, somewhat less common measure, is the instance connectedness, which measures how well a given instance lies in the same cluster as its nearby

Algorithm 1: Process to produce a cluster using a given GP individual (X) and the number of neighbours (l).

```
Edges = {};
for I ∈ Dataset do Choose edge
    Neighbours = nearestNeighbours(I, l);
    Neighbour_Best = ∅;
    Similarity_Best = −∞;
    for Y ∈ Neighbours do Test neighbour
        similarity = evaluate(X, I, Y);
        if similarity > Similarity_Best then
            Neighbour_Best = Y;
        end
    end
    add edge from I to Neighbour_Best to Edges;
end
Cluster = graphToCluster(Edges);
```

neighbours. The majority of the clustering literature measures performance in a way that implicitly encourages hyper-spherical clusters to be produced, by minimising each instance's distance to its cluster mean, and maximising the distance between different cluster means. Such an approach is problematic, as it introduces bias in the shape of clusters produced, meaning elliptical or other non-spherical clusters are unlikely to be found correctly.

As a graph representation is capable of modelling a variety of cluster shapes, we instead propose using a fitness function which balances these three measures of cluster quality in a way that gives minimal bias to the shape of clusters produced. We discuss each of these in turn below:

Compactness To measure the *compactness* of a cluster, we choose the instance in the cluster which is the furthest away from its nearest neighbour in the same cluster; that is, the instance which is the most isolated within the cluster. The distance between that instance and its nearest neighbour, called the *sparsity* of the cluster, should be minimised. We define sparsity in Equation (1), where C_i represents the i^{th} cluster of K clusters, $I_a \in C_i$ represents an instance in the i^{th} cluster, and $d(I_a, I_b)$ is the Euclidean distance between two instances.

$$\text{Sparsity} = \max_{I_a \in C_i} \left\{ \min_{I_b \in C_i} d(I_a, I_b) \middle| I_a \neq I_b \right\} \tag{1}$$

Separability To measure the separation of a cluster, we find the minimum distance from that cluster to any other cluster. This is equivalent to finding the minimum distance between the instances in the cluster and all other instances in the dataset which are not in the same cluster, as shown in Equation (2). The separation of a cluster should be maximised to ensure that it is distinct from other clusters.

$$\text{Separation} = \min_{I_a \in C_i} \left\{ \min_{I_b \notin C_i} d(I_a, I_b) \right\} \tag{2}$$

Connectedness An instance's *connectedness* is measured by finding how many of its c nearest neighbours are assigned to the same cluster as it, with higher weighting given to neighbours which are closer to the given instance, as shown in Equation (3). To prevent connectedness from encouraging spherical clusters, c must be chosen to be adequately small — otherwise, large cluster blobs will form. We found that setting

$c = 10$ provided a good balance between producing connected instances and allowing varying cluster shapes. The mean connectedness of a dataset should be maximised.

$$\text{Connectedness} = \frac{1}{K}\sum_{i=1}^{K}\frac{1}{|C_i|}\sum_{I_a \in C_i,\ I_b \in N_{I_a}} d_{inv}(I_a, I_b)\big|I_b \cap C_i \tag{3}$$

where N_{I_a} gives the c nearest neighbours of I_a, and

$$d_{inv}(I_a, I_b) = min\Big[\frac{1}{d(I_a, I_b)}, 10\Big] \tag{4}$$

The inverse distance between two instances is capped at 10, to prevent very close instances from overly affecting the fitness measure.

Our proposed fitness function is a combination of these three measures (Equations (1)–(3)): we find each cluster's ratio of sparsity: separation (as they are competing objectives) as shown in Equation (5), and then measure the partition's fitness by also considering the connectedness, as shown in Equation (6). This fitness function should be maximised.

$$\text{Mean SpaSep} = \frac{1}{K}\sum_{i=1}^{K}\frac{\text{Sparsity}}{\text{Separation}} \tag{5}$$

$$\text{Fitness} = \frac{\text{Connectedness}}{\text{Mean SpaSep}} \tag{6}$$

3.4 Using a Multi-Tree Approach

As previously discussed, using a single fixed similarity function means that every pair of instances across a dataset must be compared identically, i.e. with all features weighted equally regardless of the characteristics of the given instances. By using GP to automatically evolve similarity functions containing conditional nodes (if, max, and min), we are able to produce trees which will measure similarity dynamically. However, a tree is still limited in its flexibility, as there is an inherent trade-off between the number of conditional nodes used and the complexness of the constructed features in a tree — more conditionals will tend to mean simpler constructed features with fewer operators (and vice versa), due to the limitations on tree depth and training time.

To tackle these issues, while still maintaining reasonable tree depth and training time, we propose evolving a number of similarity functions concurrently. Using this approach, a pair of instances will be assigned a similarity score by each similarity function, which are then summed together to give a total measure of how similar the instances are. In this regard, each similarity function provides a measure of its *confidence* that two instances should lie in the same cluster, allowing different similarity functions to specialise on different parts of the dataset. This is implemented using GP with a multi-tree approach, where each GP individual contains not only one but multiple trees. An example of this structure is shown in Fig. 3 with the number of trees, $t = 3$. As all t similarity functions are evolved concurrently in a single individual, a set of cohesive functions will be evolved that work well together, but that are not expected to be good similarity functions independently. In this way, a GP individual can be thought of as a *meta-function*. The core of the clustering process remains the same with this approach, with the only change being that the most similar neighbour for a given instance is based on the sum of similarities given by all trees in an individual. This change to the algorithm is shown in Algorithm 2.

There are several factors that must be considered when extending the proposed algorithm to use a multi-tree approach: how to perform crossover when there are multiple trees to crossover between, and how many trees to use. These two factors will

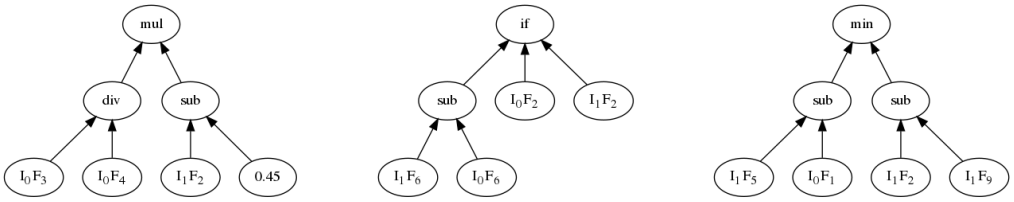


Figure 3: An example of a multi-tree similarity function.

Algorithm 2: Choosing the most similar neighbour to an instance (I) in the multi-tree approach for individual X.

```
for Y ∈ Neighbours do Test neighbour
    similarity_sum = 0;
    for T ∈ X do Each tree
        similarity_sum += evaluate(T, I, Y);
    end
    if similarity_sum > Similarity_Best then
        Neighbour_Best = Y;
    end
end
```

be discussed in the following paragraphs. A third consideration is the maximum tree depth — we use a smaller tree depth when multiple trees are used, as each tree is able to be more specialised and so does not require as many nodes to produce a good similarity function. Mutation is performed as normal, by randomly choosing a tree to mutate.

3.4.1 Crossover Strategy

In standard GP, crossover is performed by selecting two individuals, randomly selecting a subtree from each of these two individuals, and swapping the selected subtrees to produce new offspring. In multi-tree GP, a tree within each individual must also be selected. There are a number of possible methods for doing so (Haynes and Sen, 1997; Thomason and Soule, 2007), as discussed below:

Random-index crossover The most obvious method is to randomly select a tree from each individual, which we term *random-index crossover* (RIC). This method may be problematic when applied to our proposed approach, as it reduces the ability of each tree to specialise, by exchanging information between trees which may have different "niches".

Same-index crossover An alternative method to avoid the limitations of RIC is to always pick two trees at the same index in each individual. For example, selecting the third tree in both individuals. This method, which we call *same-index crossover* (SIC), allows an individual to better develop a number of distinct trees while still encouraging co-operation between individuals through the crossover of related trees.

All-index crossover The SIC method can be further extended by performing crossover between every pair of trees simultaneously, i.e. crossover between every i^{th} tree in both individuals, where $i \in [1, t]$ for t trees. This approach, called *all-index crossover* (AIC) allows information exchange to occur more aggressively between individuals, which should increase training efficiency. However, it introduces the require-

ment that the effect of performing all pairs of crossovers gives a net fitness increase which may limit the exploitation of individual solutions during the EC process.

We will compare each of these crossover approaches to investigate which type of crossover is most appropriate.

3.4.2 Number of Trees

The number of trees used in a multi-tree approach must strike a balance between the performance benefit gained by using a large number of specialised trees and the difficulty in training many trees successfully. When using either the SIC or RIC crossover methods, increasing the number of trees used will reduce the chance proportionally that a given tree is chosen for crossover/mutation, thereby decreasing the rate at which each tree is refined. When the AIC method is used, a larger number of trees increases the probability that a crossover will not improve fitness, as the majority of the trees are unlikely to gain a performance boost when crossed over in the later stages of the training process when small tweaks to trees are required to optimise performance. We will investigate the effect of the number of trees used on the fitness obtained later in this paper.

4 Experiment Design

4.1 Benchmark Techniques

We compare our proposed single-tree approach (GPGC) to a variety of baseline clustering methods, which are listed below. We also compare the single- and multi-tree approaches, to investigate the effectiveness of using additional trees.

- k-means++ (Arthur and Vassilvitskii, 2007), a commonly used partitional algorithm. Standard k-means++ cannot automatically discover the number of clusters, and so K is pre-fixed for this method. We use this as an example of a relatively simple and widely used method in the clustering literature.

- OPTICS (Ankerst et al., 1999), a well-known density-based algorithm. OPTICS requires a contrast parameter, ξ, to be set in order to determine where in the dendrogram the cluster partition is extracted from; we test OPTICS with a range of ξ values in $[0.001, 0.005, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5]$ and report the best result in terms of the Adjusted Rand Index (defined in Section 4.4).

- Two naïve graph-based approaches which connect every instance with an edge to its n-nearest neighbours (von Luxburg, 2007). We test with both $n = 2$ (called NG-2NN) and $n = 3$ (called NG-3NN) in this work. Note that the case where $n = 1$ (NG-1NN) is similar to the clustering process used in Algorithm 3.2; we exclude NG-1NN as it produces naive solutions with a fixed distance function.

- The Markov Clustering (MCL) algorithm (Van Dongen, 2000), another clustering algorithm using a graph-based representation, which simulates random walks through the graph and keeps instances in the same cluster when they have a high number of paths between them.

- The multi-objective clustering with automatic k-determination (MOCK) algorithm (Handl and Knowles, 2007) introduced earlier, as an example of a well-known high-quality EC clustering method.

Table 1: Datasets generated using a Gaussian distribution (Handl and Knowles, 2007).

Name	m	n	K
10d10cGaussian	10	2730	10
10d20cGaussian	10	1014	20
10d40cGaussian	10	1938	40

Table 2: Datasets generated using an Elliptical distribution (Handl and Knowles, 2007).

Name	m	n	K	Name	m	n	K
10d10c	10	2903	10	100d10c	100	2893	10
10d20c	10	1030	20	100d20c	100	1339	20
10d40c	10	2023	40	100d40c	100	2212	40
10d100c	10	5541	100	1000d10c	1000	2753	10
50d10c	50	2699	10	1000d20c	1000	1088	20
50d20c	50	1255	20	1000d40c	1000	2349	40
50d40c	50	2335	40	1000d100c	1000	6165	100

4.2 Datasets

We use a range of synthetic clustering datasets to evaluate the performance of our proposed approach, with varying cluster shapes, numbers of features (m), instances (n) and clusters (K). We avoid using real-world datasets with class labels as done in previous clustering studies, as there is no requirement that classes should correspond well to homogeneous clusters (von Luxburg et al., 2012) — for example, clustering the well-known Iris dataset will often produce two clusters, as the versicolor and virginica classes overlap significantly in the feature space. The datasets were generated with the popular generators introduced by Handl et al. (Handl and Knowles, 2007). The first generator uses a Gaussian distribution, which produces a range of clusters of varying shapes at low dimensions, but produces increasingly hyper-spherical clusters as m increases. As such, we use this generator only at a small m, to produce the datasets shown in Table 1. The second generator produces clusters using an elliptical distribution, which produces non-hyper-spherical clusters even at large dimensionality. A wide variety of datasets were generated with this distribution, with m varying from 10 to 1000, and K varying from 10 to 100, as shown in Table 2. Datasets with $K = 10$ clusters have between 50 and 500 instances per cluster, whereas datasets with a higher K have between 10 and 100 to limit the memory required. These datasets allow our proposed approach to be tested on high-dimensional problems. All datasets are scaled so that each feature is in $[0, 1]$ to prevent feature range overly affecting the distance calculations used in the clustering process. As a generator is used, the cluster that each instance is assigned to is known — i.e. the datasets provide a *gold standard* in the form of a "cluster label" for each instance. While this label is not used during training, it is useful for evaluating the clusters produced by the clustering methods.

4.3 Parameter Settings

The non-deterministic methods (k-means++, GPGC, MOCK, MCL) were run 30 times, and the mean results were computed. k-means++, GPGC and MOCK were run for 100 iterations, by which time k-means++ had achieved convergence. All benchmarks use Euclidean distance. The GP parameter settings for the single- and multi-tree GPGC

Table 3: Common GP Parameter Settings

Parameter	Setting	Parameter	Setting
Generations	100	Population Size	1024
Mutation	20%	Crossover	80%
Elitism	top-10	Selection Type	Tournament
Min. Tree Depth	2	Max. Tree Depth	5 (MT), 7 (ST)
Tournament Size	7	Pop. Initialisation	Half-and-half

methods, are based on standard parameters (Poli et al., 2008), and are shown in Table 3; the multi-tree (MT) approach uses a smaller maximum tree depth than the single-tree (ST) approach due to having multiple more-specific trees. The MOCK experiments used the attainment score method to select the best solution from the multi-objective pareto front.

4.4 Evaluation Metrics

To evaluate the performance of each of the clustering algorithms, we use the three measures defined previously (connectedness, sparsity, and separation), as well as the Adjusted Rand Index (ARI), which compares the cluster partition produced by an algorithm to the gold standard provided by the cluster generators in an adjusted-for-chance manner (Vinh et al., 2010).

Given a cluster partition C produced by an algorithm and a gold standard cluster partition G, the ARI is calculated by first generating a contingency table where each entry n_{ij} denotes the number of instances in common between C_i and G_j, where C_i is the i-th cluster in C, and G_j is the j-th cluster in G. In addition, the sum of each row and column is computed, denoted as a_i and b_j respectively. As before, n is the total number of instances. The ARI is then calculated according to Equation (7), which finds the frequency of occurrence of agreements between the two clusterings, while adjusting for the chance grouping of instances.

$$\text{ARI} = \frac{\sum_{ij}\binom{n_{ij}}{2} - [\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}]/\binom{n}{2}}{\frac{1}{2}[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}] - [\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}]/\binom{n}{2}} \tag{7}$$

5 Results and Discussion

We provide and analyse the results of our experiments in this section. We begin by comparing each of the proposed multi-tree approaches to the single-tree GPGC approach in order to decide which version of GPGC is the more effective (Section 5.1). We then compare the best of these approaches, GPGC-AIC, to the benchmark methods to examine how well our proposed method performs relative to existing clustering methods (Section 5.2). The effect of the number of trees on the performance of the multi-tree approach is analysed in Section 5.3.

5.1 GPGC using Multiple Trees

To further improve the performance of the proposed GPGC approach, we proposed an extension to use a multi-tree GP design in Section 3.4. To analyse the effectiveness of this extension, and determine which type of multi-tree crossover is most effective, we evaluated the three crossover methods (RIC, SIC, AIC) against the single-tree GPGC

Table 4: Crossover: Datasets using a Gaussian Distribution

(a) 10d10cGaussian

Method	Fitness	K	Conn.	Spar.	Sep.	ARI
GPGC	19.23	21.5	41.9	0.293	0.140	0.750
AIC	23.75^{+}	8.8	51.4^{+}	0.324^{-}	0.154^{+}	0.880^{+}
RIC	24.47^{+}	8.1	52.4^{+}	0.324^{-}	0.156^{+}	0.859^{+}
SIC	24.66^{+}	7.6	52.9^{+}	0.326^{-}	0.157^{+}	0.833^{+}

(b) 10d20cGaussian

Method	Fitness	K	Conn.	Spar.	Sep.	ARI
GPGC	63.00	19.7	47.3	0.268	0.375	0.991
AIC	63.79^{+}	19.5	47.3	0.268	0.377^{+}	0.980
RIC	63.26	19.7	47.3^{+}	0.269	0.376	0.988
SIC	63.33	19.7	47.3	0.268	0.376	0.991

(c) 10d40cGaussian

Method	Fitness	K	Conn.	Spar.	Sep.	ARI
GPGC	57.81	34.8	48.6	0.267	0.331	0.958
AIC	60.37^{+}	33.7	48.9^{+}	0.265^{+}	0.337^{+}	0.943
RIC	60.05^{+}	34.0	48.9^{+}	0.266^{+}	0.336^{+}	0.955
SIC	58.89	34.6	48.8	0.267	0.334	0.958

approach. We used $t = 7$ trees based on initial tests — the effect of varying t is discussed further in Section 5.3. Tables 4 and 5 show the results of these experiments on the datasets generated using a Gaussian and elliptical distribution respectively. For each of the four methods, we provide the (mean) number of clusters (K), as well as four metrics of cluster quality: fitness achieved, connectedness (Conn), sparsity (Spar), separation (Sep), and the ARI. Connectedness, sparsity, and separation are defined in the same way as in the fitness function. We performed a student's t-test at a 95% confidence interval comparing each of the multi-tree approaches to the single-tree approach on each of the metrics. A "$+$" indicates a method was significantly better than the single-tree GPGC method, a "$-$" indicates it was significantly worse, and no symbol indicates no significant difference was found.

The most noticeable result of using a multi-tree approach is that the fitness achieved by the GP process is significantly improved across all datasets with the exception of the 10d40c and 10d100c datasets, where the multi-tree approaches were significantly worse or had similar fitness to GPGC. On the datasets generated using a Gaussian distribution, the multi-tree approaches are able to find the number of clusters much accurately on 10d10cGaussian, and achieve a significantly higher ARI result. On the 10d40cGaussian dataset, both AIC and RIC achieved significantly better fitness, connectedness, sparsity, and separation than GPGC.

The multi-tree approaches also tend to produce clusters that are both better connected and better separated than GPGC on the datasets generated with an elliptical distribution. It seems that using multiple trees allows the GP evolutionary process to better separate clusters, while still ensuring that similar instances are placed in the same cluster. Sparsity is either increased (i.e. made worse) or is similar compared to GPGC when a multi-tree approach is used — this suggests that the single tree approach was overly favouring reducing sparsity at the expense of the overall fitness. Another interesting pattern is that the number of clusters (K) found by the multi-tree approaches was always lower than that found by GPGC; given that GPGC tended to over-estimate K, this can be seen as further evidence that using multiple trees improves clustering performance. Furthermore, a smaller K is likely to directly improve connectedness as more instances will have neighbours in the same cluster, and separation since having fewer clusters increases the average distance between neighbouring clusters.

In terms of the ARI, the multi-tree approaches were significantly better than GPGC on a number of elliptically-generated datasets, with the RIC, SIC, and AIC methods being significantly better on 1, 3, and 5 datasets respectively. Both the AIC and RIC

Table 5: Crossover: Datasets using an Elliptical Distribution.

(a) 10d10c

Method	Fitness	K	Conn.	Spar.	Sep.	ARI
GPGC	19.17	36.4	61.2	0.157	0.061	0.737
AIC	21.35^{+}	24.0	71.7^{+}	0.166^{-}	0.059	0.814^{+}
RIC	20.68^{+}	23.1	73.7^{+}	0.168^{-}	0.058	0.799^{+}
SIC	21.13^{+}	24.7	72.9^{+}	0.164^{-}	0.057	0.806^{+}

(b) 10d20c

Method	Fitness	K	Conn.	Spar.	Sep.	ARI
GPGC	43.69	27.8	73.3	0.150	0.098	0.663
AIC	49.11^{+}	22.5	77.4^{+}	0.154^{-}	0.106^{+}	0.666
RIC	49.51^{+}	21.9	77.7^{+}	0.154	0.106^{+}	0.656
SIC	50.28^{+}	21.8	78.3^{+}	0.155^{-}	0.106^{+}	0.677

(c) 10d40c

Method	Fitness	K	Conn.	Spar.	Sep.	ARI
GPGC	36.57	55.6	70.9	0.136	0.079	0.579
AIC	33.42^{-}	49.2	76.6^{+}	0.139	0.072^{-}	0.522^{-}
RIC	31.92^{-}	53.2	75.1^{+}	0.138	0.070^{-}	0.486^{-}
SIC	31.95^{-}	51.1	75.9^{+}	0.140^{-}	0.071^{-}	0.539

(d) 10d100c

Method	Fitness	K	Conn.	Spar.	Sep.	ARI
GPGC	31.80	109.5	74.0	0.131	0.066	0.424
AIC	32.40	106.4	75.6	0.131	0.066	0.421
RIC	30.84	134.9	72.6	0.127^{+}	0.064	0.442
SIC	31.78	113.4	74.1	0.130	0.066	0.444

(e) 50d10c

Method	Fitness	K	Conn.	Spar.	Sep.	ARI
GPGC	31.66	12.6	57.8	0.445	0.276	0.962
AIC	42.49^{+}	10.0	59.5^{+}	0.472^{-}	0.341^{+}	0.987^{+}
RIC	41.21^{+}	10.0	60.1^{+}	0.465^{-}	0.328^{+}	0.977
SIC	40.21^{+}	10.4	59.9^{+}	0.459	0.320^{+}	0.969

(f) 50d20c

Method	Fitness	K	Conn.	Spar.	Sep.	ARI
GPGC	30.36	25.2	50.7	0.350	0.234	0.807
AIC	36.78^{+}	21.1	51.8^{+}	0.359	0.273^{+}	0.837
RIC	34.82^{+}	21.5	51.3	0.362^{-}	0.268^{+}	0.841
SIC	34.06^{+}	22.1	51.3	0.358	0.261^{+}	0.848

(g) 50d40c

Method	Fitness	K	Conn.	Spar.	Sep.	ARI
GPGC	29.58	49.8	54.6	0.304	0.196	0.726
AIC	34.36^{+}	45.2	56.1^{+}	0.308	0.220^{+}	0.810^{+}
RIC	32.13^{+}	46.2	55.8^{+}	0.306	0.209^{+}	0.721
SIC	32.27^{+}	46.3	55.8^{+}	0.307	0.208^{+}	0.776

(h) 100d10c

Method	Fitness	K	Conn.	Spar.	Sep.	ARI
GPGC	39.40	10.4	47.9	0.611	0.545	0.993
AIC	44.41^{+}	9.8	48.1	0.621^{-}	0.580^{+}	0.998
RIC	44.88^{+}	9.7	48.2	0.622^{-}	0.584^{+}	0.997
SIC	42.87^{+}	10.0	48.1	0.617	0.569	0.996

(i) 100d20c

Method	Fitness	K	Conn.	Spar.	Sep.	ARI
GPGC	28.18	22.2	38.4	0.527	0.449	0.883
AIC	31.99^{+}	20.6	38.2	0.535	0.494^{+}	0.917^{+}
RIC	31.31^{+}	20.7	38.6	0.530	0.481^{+}	0.905
SIC	31.59^{+}	20.5	38.3	0.538	0.492^{+}	0.921^{+}

(j) 100d40c

Method	Fitness	K	Conn.	Spar.	Sep.	ARI
GPGC	21.60	50.2	39.7	0.436	0.282	0.724
AIC	25.02^{+}	45.9	40.6^{+}	0.440	0.316^{+}	0.771
RIC	24.50^{+}	47.6	40.7^{+}	0.438	0.311^{+}	0.777
SIC	23.64^{+}	49.2	39.9	0.438	0.304^{+}	0.792^{+}

(k) 1000d10c

Method	Fitness	K	Conn.	Spar.	Sep.	ARI
GPGC	11.60	10.1	15.0	2.132	1.704	0.980
AIC	12.60^{+}	9.7	14.9	2.126	1.784^{+}	0.987
RIC	12.55^{+}	9.7	15.0	2.122	1.776^{+}	0.984
SIC	12.48^{+}	9.6	15.2	2.115	1.754	0.978

(l) 1000d20c

Method	Fitness	K	Conn.	Spar.	Sep.	ARI
GPGC	9.22	23.1	12.0	1.539	1.325	0.834
AIC	11.48^{+}	19.6	12.4^{+}	1.575^{-}	1.511^{+}	0.810
RIC	11.37^{+}	19.5	12.2	1.580^{-}	1.533^{+}	0.790
SIC	10.96^{+}	19.4	12.3	1.589^{-}	1.498^{+}	0.804

(m) 1000d40c

Method	Fitness	K	Conn.	Spar.	Sep.	ARI
GPGC	8.48	47.5	14.0	1.376	1.006	0.797
AIC	10.14^{+}	42.5	14.2	1.387	1.130^{+}	0.803
RIC	10.01^{+}	42.6	14.2	1.384	1.126^{+}	0.832
SIC	9.66^{+}	44.1	14.2	1.382	1.086^{+}	0.828

(n) 1000d100c

Method	Fitness	K	Conn.	Spar.	Sep.	ARI
GPGC	7.91	132.5	15.8	1.172	0.761	0.839
AIC	9.90^{+}	117.2	16.0	1.189^{-}	0.901^{+}	0.916^{+}
RIC	9.06^{+}	119.9	16.0^{+}	1.186^{-}	0.839^{+}	0.863
SIC	8.56	124.7	16.0	1.172	0.797	0.853

methods have significantly better fitness than GPGC on these datasets, while SIC is not significantly better on 1000d100c. Given that the AIC method generally has the highest fitness across the datasets, and often the best ARI, we believe that this method is the most effective version of our proposed algorithm. Based on this, we use GPGC-AIC in the next section to compare to other clustering methods.

Table 6: Baselines: Datasets using a Gaussian distribution

(a) 10d10cGaussian

Method	K	Conn	Spar	Sep	ARI
AIC	8.8	51.4	0.324	0.154	0.880
k-means++	10.0	50.4	0.341^{-}	0.133^{-}	0.848
MCL	8.0	52.8^{+}	0.323	0.137^{-}	0.910
MOCK	13.6	41.4^{-}	0.291^{+}	0.167^{+}	0.963^{+}
NG-2NN	4.0	45.3^{-}	0.317^{+}	0.193^{+}	0.368^{-}
NG-3NN	1.0	57.6^{+}	0.428^{-}	0.000^{-}	0.248^{-}
OPT-0.01	22.0	35.5^{-}	0.211^{+}	0.109^{-}	0.571^{-}

(b) 10d20cGaussian

Method	K	Conn	Spar	Sep	ARI
AIC	19.5	47.3	0.268	0.377	0.980
k-means++	20.0	43.8^{-}	0.273^{-}	0.293^{-}	0.872^{-}
MCL	20.0	47.2^{-}	0.269^{-}	0.373^{-}	0.998^{+}
MOCK	20.7	45.7^{-}	0.265^{+}	0.359^{-}	0.990
NG-2NN	19.0	47.3^{+}	0.268	0.381^{+}	0.965^{-}
NG-3NN	19.0	47.3^{+}	0.268	0.381^{+}	0.965^{-}
OPT-0.05	15.0	46.9^{-}	0.305^{-}	0.297^{-}	0.521^{-}

(c) 10d40cGaussian

Method	K	Conn	Spar	Sep	ARI
AIC	33.7	48.9	0.265	0.337	0.943
k-means++	40.0	44.5^{-}	0.270^{-}	0.260^{-}	0.895^{-}
MCL	40.0	47.2^{-}	0.264^{+}	0.335	0.999^{+}
MOCK	38.0	46.4^{-}	0.261^{+}	0.321^{-}	0.960
NG-2NN	40.0	46.3^{-}	0.261^{+}	0.332^{-}	0.984^{+}
NG-3NN	37.0	47.6^{-}	0.263^{+}	0.342^{+}	0.951
OPT-0.05	27.0	47.6^{-}	0.298^{-}	0.242^{-}	0.338^{-}

5.2 GPGC-AIC compared to the Benchmarks

Tables 6 and 7 show how the proposed GPGC-AIC method compares to the six benchmarks across the datasets tested. For each of the seven methods, we provide the (mean) number of clusters (K), as well as the same four metrics of cluster quality as before. Note that k-means++ requires K to be pre-defined, and so always obtains the correct K value. We use the same significance test as in Section 5.1.

Table 6 shows the results on the datasets that were generated using a Gaussian distribution. In terms of the ARI, the AIC method is significantly worse than either the MCL or MOCK method across the three datasets, but generally outperforms all the other baselines. As these datasets were generated with a Gaussian distribution, they tend to contain very well-formed hyper-spherical clusters, and so methods such as MCL are very effective at clustering these correctly.

On the results for the datasets generated using an elliptical distribution, shown in Table 7, GPGC is significantly better than all baselines excluding MOCK across all datasets containing 10 features (10d*c). While the MOCK method is competitive (or better) on the datasets with 10 and 20 clusters, it achieves a very poor ARI on the more difficult datasets with 40 and 100 clusters, where the AIC method is clearly superior. The remaining baseline methods are nearly always significantly worse than AIC on these datasets. The NG baselines are particularly inconsistent, with the number of clusters and ARI values varying by up to three times depending on the number of nearest neighbours chosen. AIC can also automatically find the number of clusters much more accurately than OPTICS, and produces less sparse and more separated clusters than k-means++ across these datasets. In contrast to the previous datasets, the MCL method struggles significantly with these non-hyper-spherical datasets — a pattern that is also true for the remaining datasets and which highlights a key weakness with the MCL method. Similar patterns are seen across the 50d*c datasets, with the exception being on 50d10c where NG-3NN achieves a near perfect result. On the high-dimensionality datasets (100d*c, 1000d*c), the MOCK method has a high variance in accuracy, with ARI values ranging from 0.434 to 0.897. In contrast, the AIC method achieves consistently good performance, with the lowest ARI achieved being 0.771 and the highest being 0.998. While the MOCK method is superior in two cases (1000d20c, 1000d40c),

Table 7: Baselines: Datasets using an Elliptical Distribution (Part 1).

(a) 10d10c

Method	K	Conn	Spar	Sep	ARI
AIC	24.0	71.7	0.166	0.059	0.814
k-means++	10.0	85.5^{+}	0.212^{-}	0.038^{-}	0.552^{-}
MCL	15.0	83.3^{+}	0.205^{-}	0.050^{-}	0.703^{-}
MOCK	17.8	86.1^{+}	0.180^{-}	0.065^{+}	0.793
NG-2NN	9.0	75.0^{+}	0.176^{-}	0.071^{+}	0.510^{-}
NG-3NN	5.0	82.3^{+}	0.187^{-}	0.085^{+}	0.323^{-}
OPT-0.1	6.0	90.7^{+}	0.091^{+}	0.036^{-}	0.232^{-}

(b) 10d20c

Method	K	Conn	Spar	Sep	ARI
AIC	22.5	77.4	0.154	0.106	0.666
k-means++	20.0	73.3^{-}	0.190^{-}	0.087^{-}	0.459^{-}
MCL	28.0	69.4^{-}	0.173^{-}	0.095^{-}	0.451^{-}
MOCK	27.5	79.3^{+}	0.152	0.113^{+}	0.752^{+}
NG-2NN	36.0	69.5^{-}	0.131^{+}	0.094^{-}	0.584^{-}
NG-3NN	16.0	83.1^{+}	0.145^{+}	0.110^{+}	0.355^{-}
OPT-0.1	8.0	87.8^{+}	0.119^{+}	0.056^{-}	0.138^{-}

(c) 10d40c

Method	K	Conn	Spar	Sep	ARI
AIC	49.2	76.6	0.139	0.072	0.522
k-means++	40.0	77.2	0.169^{-}	0.069^{-}	0.417^{-}
MCL	54.0	69.2^{-}	0.149^{-}	0.094^{+}	0.288^{-}
MOCK	28.8	84.2^{+}	0.146^{-}	0.087^{+}	0.232^{-}
NG-2NN	43.0	68.4^{-}	0.110^{+}	0.085^{+}	0.256^{-}
NG-3NN	13.0	73.6^{-}	0.131^{+}	0.107^{+}	0.082^{-}
OPT-0.1	10.0	89.3^{+}	0.106^{+}	0.057^{-}	0.065^{-}

(d) 10d100c

Method	K	Conn	Spar	Sep	ARI
AIC	106.4	75.6	0.131	0.066	0.421
k-means++	100.0	80.4^{+}	0.153^{-}	0.056^{-}	0.398
MCL	98.0	77.3^{+}	0.136^{-}	0.070^{+}	0.125^{-}
MOCK	62.0	86.2^{+}	0.137^{-}	0.074^{+}	0.087^{-}
NG-2NN	91.0	70.6^{-}	0.108^{+}	0.080^{+}	0.049^{-}
NG-3NN	26.0	73.4^{-}	0.096^{+}	0.084^{+}	0.030^{-}
OPT-0.2	2.0	78.6^{+}	0.184^{-}	0.138^{+}	0.024^{-}

(e) 50d10c

Method	K	Conn	Spar	Sep	ARI
AIC	10.0	59.5	0.472	0.341	0.987
k-means++	10.0	52.2^{-}	0.555^{-}	0.102^{-}	0.485^{-}
MCL	12.0	54.1^{-}	0.519^{-}	0.102^{-}	0.604^{-}
MOCK	14.4	56.7^{-}	0.494^{-}	0.217^{-}	0.811^{-}
NG-2NN	18.0	53.6^{-}	0.372^{+}	0.168^{-}	0.967^{-}
NG-3NN	11.0	58.8^{-}	0.456^{+}	0.302^{-}	0.999^{+}
OPT-0.2	2.0	73.5^{+}	0.550^{-}	0.111^{-}	0.236^{-}

(f) 50d20c

Method	K	Conn	Spar	Sep	ARI
AIC	21.1	51.8	0.359	0.273	0.837
k-means++	20.0	44.3^{-}	0.429^{-}	0.168^{-}	0.353^{-}
MCL	26.0	42.0^{-}	0.415^{-}	0.175^{-}	0.482^{-}
MOCK	24.3	50.3^{-}	0.375^{-}	0.273	0.884
NG-2NN	44.0	41.0^{-}	0.304^{+}	0.213^{-}	0.831
NG-3NN	23.0	49.0^{-}	0.369^{-}	0.286^{+}	0.808
OPT-0.1	8.0	65.3^{+}	0.279^{+}	0.152^{-}	0.125^{-}

(g) 50d40c

Method	K	Conn	Spar	Sep	ARI
AIC	45.2	56.1	0.308	0.220	0.810
k-means++	40.0	50.7^{-}	0.352^{-}	0.140^{-}	0.254^{-}
MCL	48.0	48.6^{-}	0.329^{-}	0.161^{-}	0.351^{-}
MOCK	42.7	56.6^{+}	0.315^{-}	0.253^{+}	0.867^{+}
NG-2NN	86.0	50.1^{-}	0.249^{+}	0.170^{-}	0.762^{-}
NG-3NN	46.0	56.3	0.279^{+}	0.217	0.738^{-}
OPT-0.1	26.0	65.1^{+}	0.231^{+}	0.129^{-}	0.081^{-}

(h) 100d10c

Method	K	Conn	Spar	Sep	ARI
AIC	9.8	48.1	0.621	0.580	0.998
k-means++	10.0	45.6^{-}	0.695^{-}	0.131^{-}	0.562^{-}
MCL	16.0	40.5^{-}	0.677^{-}	0.207^{-}	0.877^{-}
MOCK	28.8	45.7^{-}	0.590^{+}	0.170^{-}	0.548^{-}
NG-2NN	16.0	44.6^{-}	0.552^{+}	0.287^{-}	0.934^{-}
NG-3NN	11.0	45.8^{-}	0.647^{-}	0.513^{-}	0.989^{-}
OPT-0.005	98.0	35.7^{-}	0.383^{+}	0.127^{-}	0.319^{-}

(i) 100d20c

Method	K	Conn	Spar	Sep	ARI
AIC	20.6	38.2	0.535	0.494	0.917
k-means++	20.0	34.1^{-}	0.595^{-}	0.232^{-}	0.374^{-}
MCL	27.0	29.5^{-}	0.594^{-}	0.283^{-}	0.587^{-}
MOCK	24.6	35.7^{-}	0.573^{-}	0.487	0.897
NG-2NN	41.0	32.1^{-}	0.450^{+}	0.291^{-}	0.819^{-}
NG-3NN	25.0	34.7^{-}	0.529	0.520^{+}	0.965^{+}
OPT-0.1	13.0	43.6^{+}	0.405^{+}	0.189^{-}	0.148^{-}

(j) 100d40c

Method	K	Conn	Spar	Sep	ARI
AIC	45.9	40.6	0.440	0.316	0.771
k-means++	40.0	35.3^{-}	0.492^{-}	0.226^{-}	0.268^{-}
MCL	57.0	32.9^{-}	0.473^{-}	0.265^{-}	0.467^{-}
MOCK	41.8	39.4^{-}	0.476^{-}	0.371^{+}	0.784
NG-2NN	91.0	34.6^{-}	0.375^{+}	0.299^{-}	0.791
NG-3NN	49.0	36.1^{-}	0.431^{+}	0.393^{+}	0.711^{-}
OPT-0.001	140.0	25.6^{-}	0.593^{-}	0.150^{-}	0.430^{-}

its inconsistency makes it harder to use confidently in practice. On this set of datasets, the NG baselines achieve better results than previously in terms of the ARI, but GPGC is only ever significantly worse than one of NG-2NN and NG-3NN at most. GPGC also often has significantly better connectedness and separation than one or both of the NG baselines, suggesting it is a more consistent choice given that it is difficult to determine the number of nearest neighbours in advance (as the NG methods assume). All of the graph-based approaches are superior to k-means++ (due to the non-hyper-spherical

Table 7: Baselines: Datasets using an Elliptical Distribution (Part 2).

(k) 1000d10c

Method	K	Conn	Spar	Sep	ARI
AIC	9.7	14.9	2.126	1.784	0.987
k-means++	10.0	13.8^{-}	2.347^{-}	0.385^{-}	0.488^{-}
MCL	10.0	12.2^{-}	2.407^{-}	0.445^{-}	0.474^{-}
MOCK	16.0	14.5^{-}	2.079^{+}	0.995^{-}	0.800^{-}
NG-2NN	21.0	14.9	1.681^{+}	0.610^{-}	0.932^{-}
NG-3NN	10.0	15.5^{+}	2.078^{+}	1.541^{-}	0.947^{-}
OPT-0.01	93.0	12.2^{-}	1.074^{+}	0.477^{-}	0.257^{-}

(l) 1000d20c

Method	K	Conn	Spar	Sep	ARI
AIC	19.6	12.4	1.575	1.511	0.810
k-means++	20.0	9.7^{-}	1.885^{-}	0.832^{-}	0.376^{-}
MCL	24.0	9.3^{-}	1.828^{-}	0.748^{-}	0.339^{-}
MOCK	25.5	10.8^{-}	1.706^{-}	1.423^{-}	0.896^{+}
NG-2NN	47.0	9.8^{-}	1.409^{+}	1.014^{-}	0.736^{-}
NG-3NN	26.0	10.5^{-}	1.531^{+}	1.379^{-}	0.945^{+}
OPT-0.05	34.0	11.5^{-}	1.374^{+}	0.647^{-}	0.386^{-}

(m) 1000d40c

Method	K	Conn	Spar	Sep	ARI
AIC	42.5	14.2	1.387	1.130	0.803
k-means++	40.0	11.7^{-}	1.556^{-}	0.632^{-}	0.219^{-}
MCL	47.0	10.8^{-}	1.500^{-}	0.676^{-}	0.157^{-}
MOCK	41.7	13.6^{-}	1.488^{-}	1.231^{+}	0.887^{+}
NG-2NN	94.0	12.1^{-}	1.209^{+}	0.846^{-}	0.740^{-}
NG-3NN	52.0	13.7^{-}	1.346^{+}	1.130	0.898^{+}
OPT-0.1	13.0	17.6^{+}	1.094^{+}	0.584^{-}	0.068^{-}

(n) 1000d100c

Method	K	Conn	Spar	Sep	ARI
AIC	117.2	16.0	1.189	0.901	0.916
k-means++	100.0	14.5^{-}	1.270^{-}	0.523^{-}	0.103^{-}
MCL	204.0	11.5^{-}	1.242^{-}	0.482^{-}	0.189^{-}
MOCK	64.3	16.7^{+}	1.265^{-}	1.146^{+}	0.434^{-}
NG-2NN	229.0	14.7^{-}	1.014^{+}	0.693^{-}	0.761^{-}
NG-3NN	132.0	16.0	1.107^{+}	0.933	0.863^{-}
OPT-0.1	32.0	19.7^{+}	0.874^{+}	0.512^{-}	0.027^{-}

Table 8: Number of wins of each algorithm across the 17 datasets.

AIC	k-means++	MCL	MOCK	NG-2NN	NG-3NN	OPTICS
6	0	2	4	1	4	0

cluster shape), and OPTICS across these datasets. The AIC method also appears to be the method which predicts K most accurately overall across these datasets, especially where $K = 100$.

5.2.1 Summary

Table 8 shows the number of datasets for which each clustering method was the winner (i.e. highest mean ARI). The AIC method was the most successful, with six wins compared to four for the closest methods (NG-3NN and MOCK). This is consistent with our previous analysis, which showed AIC was the most consistent and best-performing method across datasets with high dimensionality, and was competitive with MOCK on the remaining datasets. The remaining baselines, with the exception of MCL, were almost always outperformed by at least one of AIC and MOCK. Given that MOCK is a multi-objective approach, we are hopeful that a future multi-objective variation of AIC would be able to improve the GPGC method even further and allow it to achieve a higher number of wins.

5.3 Number of Trees: Effect on Fitness

The number of trees to use in a multi-tree approach can be considered to be a form of parameter tuning. To investigate the effect of different numbers of trees on the training performance of the proposed multi-tree approach, we tested a range of trees for $t \in [1, 10]$, using the AIC crossover method. We limit t to a maximum of 10, as we found that multi-tree GP did not have improved performance, and trained more slowly at higher t values. Note that the $t = 1$ case is not equivalent to the single-tree GPGC method, due to the smaller maximum program depth of 5. For each dataset, we calculated the mean fitness of 30 runs for each value of t. The results are plotted in Fig. 4. Each plot corresponds to a single dataset, with a red dotted line indicating the baseline performance where $t = 1$, and each point corresponds to the *relative* fitness for a given

value of t. The error bars show the standard deviation of each point, for the 30 runs performed for that t value. The blue solid line is a trend-line fitted to the 10 points.

On the majority of the plots (14 out of 17), increasing the number of trees causes an increase in the fitness obtained; the 10d20cGaussian plot has no noticeable improvement as t is increased, while on the 10d40c and 10d100c plots, the fitness is actually reduced by using more trees. This is consistent with the results presented in Table 5, where the AIC method had significantly worse fitness on the 10d40c dataset, and was not significantly different on the 10d100c dataset — on the 10d100c plot, the fitness value for $t = 7$ is very close to the baseline (i.e. a relative fitness of 1). For the 10d20cGaussian plot, we hypothesise that there is little room for fitness improvement as t is increased, as shown by the small changes in fitness and ARI performance compared to GPGC in Table 4. On the 14 plots where there was a positive association, the fitness improvement is between 10% (on 1000d10c), and slightly over 50% (on 1000d100c), with improvements of around 25% on the majority of the datasets.

The optimal value of t varies depending on the dataset that is used — however, fitness tends to peak at a certain value of t for each dataset, before remaining relatively constant or dipping slightly (with the exception of those where t does not improve performance). The best value of t for each dataset is hence the lowest value of t for which performance is significantly better than all lower values of t, as this provides the best balance of maximising fitness while maintaining the interpretability and computational benefits of a lower t value. For most datasets, this value is between $t = 5$ and $t = 8$, with the exception of 50d20c and 1000d10c, where a t value of 9 and 4 seem to be best, respectively. Hence, we suggest a value of $t = 7$ or $t = 8$ may be best for an unknown dataset in order to ensure good fitness is obtained.

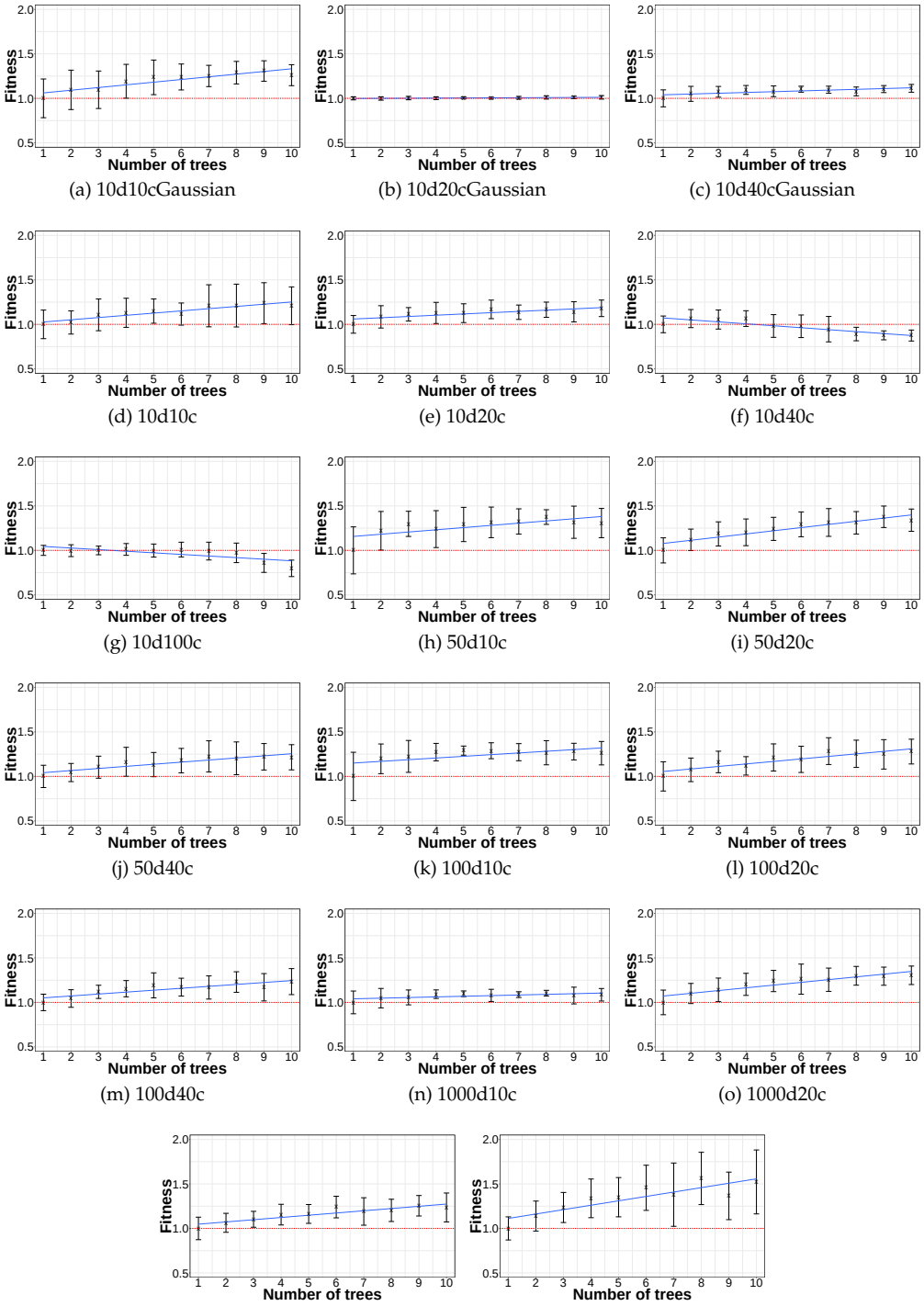
6 Further Analysis

6.1 Evolved GP Trees

In addition to achieving good clustering performance, our proposed methods are also expected to automatically select a subset of features and construct new, more powerful high-level features due to the tree-based GP structure used. To evaluate the feature manipulation performance of our proposed methods, we analyse an example evolved individual for both the single- and multi-tree approaches in this subsection.

Fig. 5 shows a single-tree individual evolved by GPGC on the 10d10c dataset, which has a very good ARI result of 0.9144. We can see that the tree produced is able to combine a number of different sub-trees to effectively construct a custom similarity function which can vary its behaviour across the dataset through the use of conditional *max* and *min* operators. A range of *building blocks* are used to find the similarity of two instances, from simple feature weighting operations (e.g. $0.572 \div I_1F_5$, $I_1F_9 + 0.659$) to more advanced feature comparisons (e.g. $min(I_0F_2, I_1F_8)$), with high-level features formed by combining these building blocks in a variety of ways. By evolving a similarity function tailored to this dataset, GPGC is able to out-perform the benchmark methods, which use the inflexible Euclidean distance function. This evolved function gives a different nearest neighbour to that of Euclidean distance for 97.04% of the instances, and on average chooses the 5.4th nearest neighbour according to Euclidean distance ordering. Clearly, GPGC has produced a significantly different ordering which is more appropriate for this dataset than normal Euclidean distance ordering.

Fig. 6 shows an example of a multi-tree individual evolved on the 1000d20c dataset, which has a very good ARI result of 0.9616. We have simplified the trees where appropriate to aid interpretability by computing constants and removing dead



(a) 10d10cGaussian (b) 10d20cGaussian (c) 10d40cGaussian

(d) 10d10c (e) 10d20c (f) 10d40c

(g) 10d100c (h) 50d10c (i) 50d20c

(j) 50d40c (k) 100d10c (l) 100d20c

(m) 100d40c (n) 1000d10c (o) 1000d20c

(p) 1000d40c (q) 1000d100c

Figure 4: Effect on training performance as the number of trees is increased.

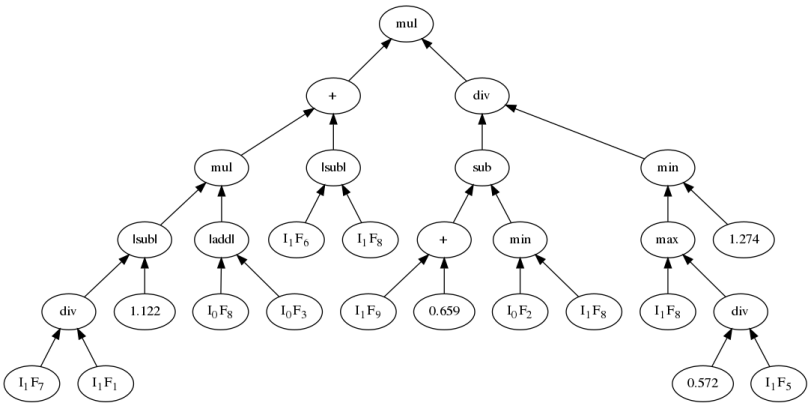


Figure 5: An evolved individual on the 10d10c dataset using the single-tree approach. Individual has a fitness of 21.37 and ARI of 0.9144 and produces $K = 22$ clusters.

branches. The seven evolved trees are generally quite simple, with only one tree (c) being the maximum depth of five, and one, two, two, and one trees having depths of four, three, two, and one, respectively. While it is difficult to understand why these trees perform well across each instance in the dataset, it is possible to gain insight by examining the general behaviour of each tree ((a)–(g)), as shown below:

(a) simple feature selection of I_1F_{312}.

(b) computing a weighted sum of two selected features.

(c) constructing a more powerful high-level feature by weighting and constructing non-linear combinations of five original features.

(d) thresholding I_1F_{458} so that it has a minor impact on the total similarity.

(e) finding the maximum of: a feature, a constant value, and the difference between two features. This gives varied behaviour based on the instances being considered.

(f) finding the absolute difference between two features, with one feature scaled.

(g) finds the maximum of two features, and then takes the result as a negative. In this way, the bigger the result, the less similar the two instances are said to be.

Each of the seven trees evaluated above had distinctive and interesting behaviour, which gives insight into which features are useful in the dataset, and into what relationships between features can be used to gauge instances' similarities accurately. In contrast, a standard distance function cannot provide such insight, as it uses the *full* feature set and performs only linear comparisons between instances' features. Of the 1000 features in the 1000d20c dataset, the example individual in Fig. 6 uses only 15 features to build its seven similarity functions. This means the clustering partition produced is much more interpretable than one produced by a standard nearest-neighbour graph-based clustering algorithm. This evolved meta-similarity function chooses the same nearest neighbour as that of Euclidean distance on only 2.67% of the instances in the dataset. On average, the 6.2th nearest neighbour is chosen as the first nearest neighbour: this is a similar trend to that of the previous example in that the neighbours have been significantly re-ordered to be tailored for this dataset.

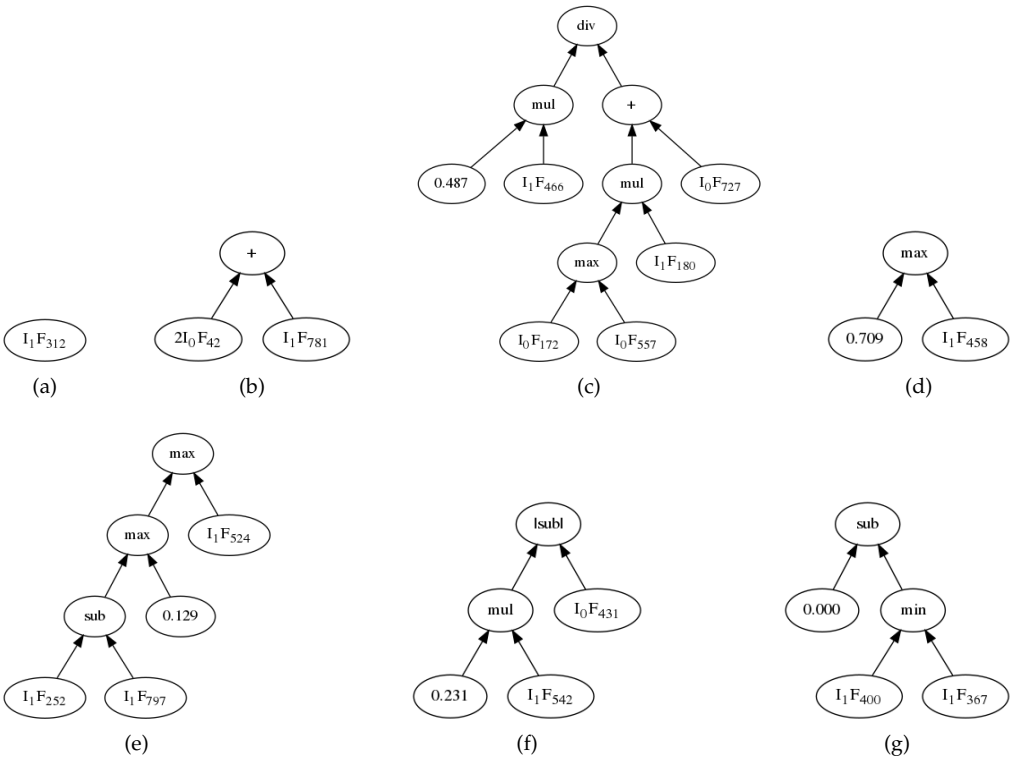


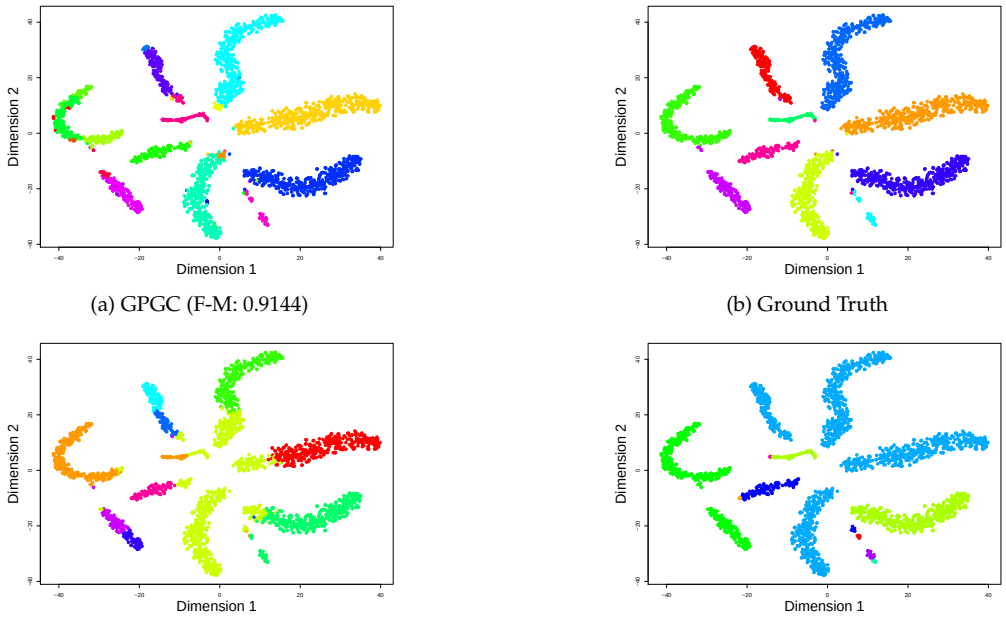
Figure 6: An evolved individual on the 1000d20c dataset using AIC crossover and $t = 7$. Individual has a fitness of 10.43 and ARI of 0.9616 and produces $K = 22$ clusters.

6.2 Visualising the Clusters Found

To further analyse the examples discussed in Section 6.1, we visualise the clusters produced compared to the ground truth clustering in this subsection using the commonly used t-SNE visualisation method (van der Maaten and Hinton, 2008), which minimises the probability distribution divergence between the two-dimensional visualisation and the original feature space.

Fig. 7 shows the clusters produced by GPGC, k-means++ and the NG-2NN methods on the 10d10c dataset. For GPGC, we use the same evolved individual as in Section 6.1. For k-means++, we chose the result with the highest ARI of the 30 runs. NG-2NN is deterministic, and so the single result is shown. In addition, the ground truth is shown in Fig. 7 for reference. It is clear that GPGC is able to most accurately reproduce the ground truth, with the majority of the clusters mapping to the ground truth well, asides from a few instances in each cluster. The exception is on the horseshoe-shaped cluster on the left of the visualisation, where GPGC has over-clustered the data by splitting this cluster in two. The k-means++ method clearly performs very poorly, with only the horseshoe-shaped cluster being clustered nearly correctly; all other clusters have significant overlap. The NG-2NN method also produces clearly incorrect clusters, with many clusters being combined, including four distinct clusters combined into one single blue cluster. GPGC is clearly able to better find the natural clusters compared to these baseline methods.

Fig. 8 (a) and (b) show the clusters produced by the GPGC-MT method (using the



(a) GPGC (F-M: 0.9144)

(b) Ground Truth

(c) k-means++ (F-M: 0.6134)

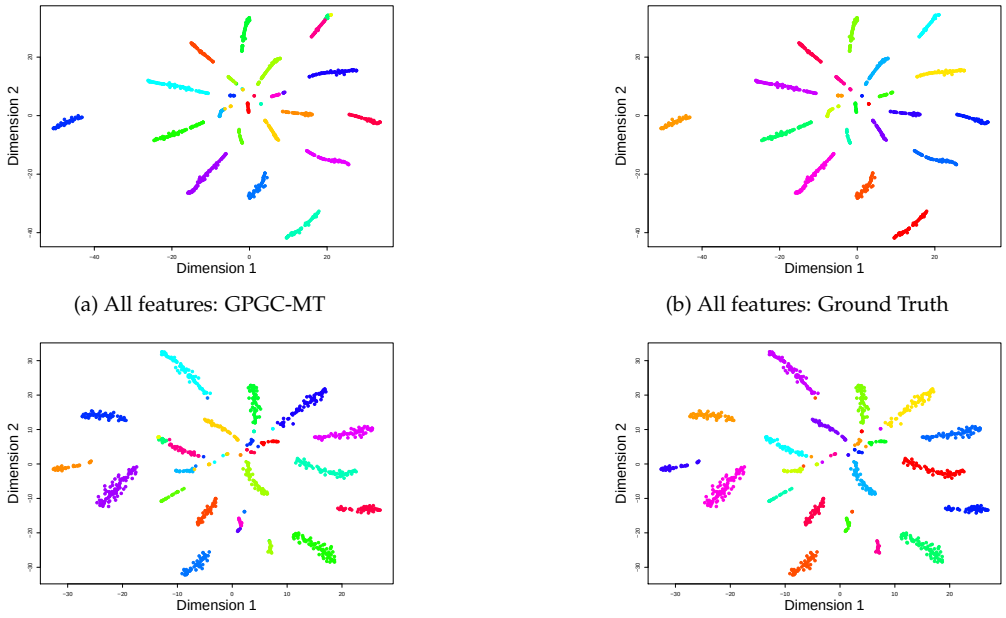
(d) NG-2NN (F-M: 0.5103)

Figure 7: Visualising the partitions chosen by a GP individual compared to the baseline methods on the 10d10c dataset. t-SNE (van der Maaten and Hinton, 2008) is used to reduce dimensionality to two dimensions. Each colour corresponds to a single cluster.

evolved tree discussed in Section 6.1) and the ground truth respectively on the 1000d20c dataset. GPGC-MT reproduces the ground truth accurately, with only a small amount of over-clustering. The GPGC-MT method uses only a subset of features in the evolved trees; in this case, only 15 of the 1000 features are used. To analyse whether using so few features would reduce the interpretability of the clusters produced, we performed another set of visualisations which used only the 15 selected features as input, as shown in Fig. 8 (c) and (d). The clusters shown in these visualisations are still very distinct and well-separated, which suggests that the GPGC-MT method was able to successfully perform feature selection implicitly in the evolved similarity functions. While t-SNE is able to reliably project the feature space into two dimensions, it does so at the cost of interpretability – the two dimensions produced cannot be easily mapped back to the original feature set, and so it is very difficult to analyse why a cluster contains certain instances. In contrast, GPGC-MT uses only a small subset of the feature set, and explicitly combines features in an interpretable manner in the evolved trees.

6.3 Evolutionary Process

To further analyse the learning effectiveness of the proposed methods (GPGC and the three multi-tree crossover approaches), we plot the fitness over the evolutionary process for the 10d10cGaussian and 1000d100c datasets, as shown in Fig 9 (a) and (b) respectively. For each dataset, we plot the mean fitness of the best individual at each generation, taken across the 30 independent runs. These datasets were selected as they represent the datasets with the lowest and highest m and K values, and are from each of the two different generators used. Both datasets show the same pattern for the single-tree compared with the multi-tree approaches: while all methods begin at similar fit-



(a) All features: GPGC-MT

(b) All features: Ground Truth

(c) Selected features: GPGC-MT

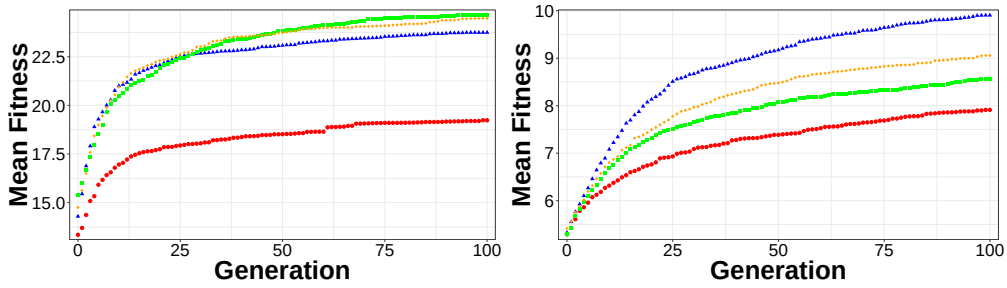
(d) Selected features: Ground Truth

Figure 8: Visualising the partitions chosen by a GP individual on the 1000d20c dataset. t-SNE (van der Maaten and Hinton, 2008) is used to reduce dimensionality to two dimensions. Figures (a) and (b) show the visualisations formed when all features are used by t-SNE, whereas (c) and (d) show the visualisations using only the features used in the GP tree. Each colour corresponds to a single cluster.

nesses, the multi-tree methods increase in mean fitness at a significantly faster rate, and reaches a much higher mean fitness overall. Indeed, the final fitness of GPGC at the 100th generation is achieved by each of the multi-tree approaches by generation 25 in both datasets. It is clear that the multi-tree approaches can train more efficiently (i.e. with a steeper initial slope), and effectively, by reaching a higher final fitness over the same number of generations. While the GPGC-AIC method is slightly outperformed by the other two crossover approaches on the 10d10cGaussian dataset, it is clearly the best method on the more difficult 1000d100c dataset, which reinforces our view that this is the best of the proposed approaches. While fitness appears to have levelled off by generation 100 on the 10d10cGaussian dataset, it appears that additional generations could improve the performance on the 1000d100c dataset even further.

7 Conclusions and Future Work

In this work, we proposed a novel approach to performing clustering, whereby GP was used to automatically evolve similarity functions in place of the commonly used inflexible distance metrics. The results of our experiments showed that the automatically generated similarity functions could improve the performance and consistency of clustering algorithms using a graph representation, while producing more interpretable similarity metrics, which consider only the most important features in a dataset. We also showed that a multi-tree GP approach could be utilised to further improve the performance by automatically evolving several highly-specific similarity functions, which



(a) 10d10cGaussian (b) 1000d100c

Figure 9: Fitness of the two proposed methods over the evolutionary process. The red dots correspond to GPGC, the blue triangles to GPGC-AIC, the green squares to GPGC-SIC, and the orange diamonds to GPGC-RIC respectively.

are able to specialise on different components of the overall clustering problem.

While the investigation in this paper was focused on graph-based clustering, due to its ability to model a range of cluster shapes, we also hope our proposed approaches can be applied to nearly any clustering method which uses a similarity function to perform clustering. By replacing the graph-based clustering approach with another given clustering algorithm, the evolved similarity functions will need to be optimised to work with that algorithm instead. We would also like to test our proposed approaches on real-world data which has "gold-standard" labels, but all real-world datasets we have found provide **class** labels only, which are not suitable for measuring cluster quality. This paper focused on using a scalar fitness function so as to constrain the scope of this work, allowing us to directly evaluate the quality of the proposed GP representation. In future work, we would like to extend our proposed fitness function by using an evolutionary multi-objective optimisation (EMO) approach — the three key measures of cluster quality (compactness, separability, connectedness) partially conflict with each other, and so using an EMO approach may allow better and more varied solutions to be generated. Initial experiments (see Appendix A) showed that GPGC had promise for subspace clustering, but that better performance could likely be achieved in the future by developing a new fitness function and designing new genetic operators to be specific to subspace clustering tasks. There is also scope for refining the GP program design used: the terminals and functions could be further tailored to the clustering domain through the use of other feature comparison operators.

References

Aggarwal, C. C. and Reddy, C. K., editors (2014). *Data Clustering: Algorithms and Applications*. CRC Press.

Ahn, C. W., Oh, S., and Oh, M. (2011). A genetic programming approach to data clustering. In *Proceedings of the International Conference on Multimedia, Computer Graphics and Broadcasting (MulGraB), Part II*, pages 123–132.

Alelyani, S., Tang, J., and Liu, H. (2013). Feature selection for clustering: A review. In *Data Clustering: Algorithms and Applications*, pages 29–60. CRC Press.

Ankerst, M., Breunig, M. M., Kriegel, H., and Sander, J. (1999). OPTICS: ordering points to identify the clustering structure. In *Proceedings of the International Conference on Management of Data*, pages 49–60.

Arthur, D. and Vassilvitskii, S. (2007). k-means++: the advantages of careful seeding. In *Pro-*

ceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 1027–1035.

Boric, N. and Estévez, P. A. (2007). Genetic programming-based clustering using an information theoretic fitness measure. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 31–38.

Coelho, A. L. V., Fernandes, E., and Faceli, K. (2011). Multi-objective design of hierarchical consensus functions for clustering ensembles via genetic programming. *Decision Support Systems*, 51(4):794–809.

Dy, J. G. and Brodley, C. E. (2004). Feature selection for unsupervised learning. *Journal of Machine Learning Research*, 5:845–889.

Eiben, A. E. and Smith, J. E. (2015). *Introduction to Evolutionary Computing*. Natural Computing Series. Springer.

Espejo, P. G., Ventura, S., and Herrera, F. (2010). A survey on the application of genetic programming to classification. *IEEE Trans. Systems, Man, and Cybernetics, Part C*, 40(2):121–144.

Ester, M., Kriegel, H., Sander, J., and Xu, X. (1996). 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*, pages 226–231.

Falco, I. D., Tarantino, E., Cioppa, A. D., and Gagliardi, F. (2005). A novel grammar-based genetic programming approach to clustering. In *Proceedings of the ACM Symposium on Applied Computing (SAC)*, pages 928–932.

Fayyad, U., Piatetsky-Shapiro, G., and Smyth, P. (1996). From data mining to knowledge discovery in databases. *AI Magazine*, 17(3):37–54.

García, A. J. and Gómez-Flores, W. (2016). Automatic clustering using nature-inspired metaheuristics: A survey. *Appl. Soft Comput.*, 41:192–213.

García-Pedrajas, N., de Haro-García, A., and Pérez-Rodríguez, J. (2014). A scalable memetic algorithm for simultaneous instance and feature selection. *Evolutionary Computation*, 22(1):1–45.

Handl, J. and Knowles, J. D. (2007). An evolutionary approach to multiobjective clustering. *IEEE Trans. Evolutionary Computation*, 11(1):56–76.

Hartuv, E. and Shamir, R. (2000). A clustering algorithm based on graph connectivity. *Inf. Process. Lett.*, 76(4-6):175–181.

Haynes, T. and Sen, S. (1997). Crossover operators for evolving a team. In Koza, J. R., Deb, K., Dorigo, M., Fogel, D. B., Garzon, M., Iba, H., and Riolo, R. L., editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 162–167, CA, USA. Morgan Kaufmann.

J. A. Hartigan, M. A. W. (1979). Algorithm AS 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108.

Jain, A. K. (2010). Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31(8):651–666.

Jolliffe, I. T. (2011). Principal component analysis. In *International Encyclopedia of Statistical Science*, pages 1094–1096. Springer.

Koza, J. R. (1992). *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press.

Kuo, R. J., Syu, Y. J., Chen, Z., and Tien, F. (2012). Integration of particle swarm optimization and genetic algorithm for dynamic clustering. *Inf. Sci.*, 195:124–140.

Lensen, A., Xue, B., and Zhang, M. (2017a). GPGC: genetic programming for automatic clustering using a flexible non-hyper-spherical graph-based approach. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO.*, pages 449–456. ACM.

Lensen, A., Xue, B., and Zhang, M. (2017b). Using particle swarm optimisation and the silhouette metric to estimate the number of clusters, select features, and perform clustering. In Squillero, G. and Sim, K., editors, *Proceedings of the 20th European Conference on the Applications of Evolutionary Computation (EvoApplications), Part I*, volume 10199 of *Lecture Notes in Computer Science*, pages 538–554. Springer.

Liu, H. and Motoda, H. (2012). *Feature selection for knowledge discovery and data mining*, volume 454. Springer Science & Business Media.

Liu, H. and Yu, L. (2005). Toward integrating feature selection algorithms for classification and clustering. *IEEE Trans. Knowl. Data Eng.*, 17(4):491–502.

Lorena, L. A. N. and Furtado, J. C. (2001). Constructive genetic algorithm for clustering problems. *Evolutionary Computation*, 9(3):309–328.

Menéndez, H. D., Barrero, D. F., and Camacho, D. (2014). A genetic graph-based approach for partitional clustering. *International Journal of Neural Systems*, 24(3).

Müller, E., Günnemann, S., Assent, I., and Seidl, T. (2009). Evaluating clustering in subspace projections of high dimensional data. In *Proceedings of the 35th International Conference on Very Large Data Bases (VLDB)*, pages 1270–1281.

Nanda, S. J. and Panda, G. (2014). A survey on nature inspired metaheuristic algorithms for partitional clustering. *Swarm and Evolutionary Computation*, 16:1–18.

Naredo, E. and Trujillo, L. (2013). Searching for novel clustering programs. In *Genetic and Evolutionary Computation Conference, GECCO '13, Amsterdam, The Netherlands, July 6-10, 2013*, pages 1093–1100.

Neshatian, K., Zhang, M., and Andreae, P. (2012). A filter approach to multiple feature construction for symbolic learning classifiers using genetic programming. *IEEE Trans. Evolutionary Computation*, 16(5):645–661.

Parsons, L., Haque, E., and Liu, H. (2004). Subspace clustering for high dimensional data: a review. *SIGKDD Explorations*, 6(1):90–105.

Peignier, S., Rigotti, C., and Beslon, G. (2015). Subspace clustering using evolvable genome structure. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO*, pages 575–582.

Picarougne, F., Azzag, H., Venturini, G., and Guinot, C. (2007). A new approach of data clustering using a flock of agents. *Evolutionary Computation*, 15(3):345–367.

Poli, R., Langdon, W. B., and McPhee, N. F. (2008). *A Field Guide to Genetic Programming*. lulu.com.

Sheng, W., Chen, S., Sheng, M., Xiao, G., Mao, J., and Zheng, Y. (2016). Adaptive multisubpopulation competition and multiniche crowding-based memetic algorithm for automatic data clustering. *IEEE Trans. Evolutionary Computation*, 20(6):838–858.

Sheng, W., Liu, X., and Fairhurst, M. C. (2008). A niching memetic algorithm for simultaneous clustering and feature selection. *IEEE Trans. Knowl. Data Eng.*, 20(7):868–879.

Tang, J., Alelyani, S., and Liu, H. (2014). Feature selection for classification: A review. In *Data Classification: Algorithms and Applications*, pages 37–64. CRC Press.

Thomason, R. and Soule, T. (2007). Novel ways of improving cooperation and performance in ensemble classifiers. In *Proceedings of the Genetic and Evolutionary Computation Conference, (GECCO)*, pages 1708–1715.

Vahdat, A. and Heywood, M. I. (2014). On evolutionary subspace clustering with symbiosis. *Evolutionary Intelligence*, 6(4):229–256.

van der Maaten, L. and Hinton, G. E. (2008). Visualizing high-dimensional data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605.

Van Dongen, S. M. (2000). *Graph clustering by flow simulation*. PhD thesis, University of Utrecht.

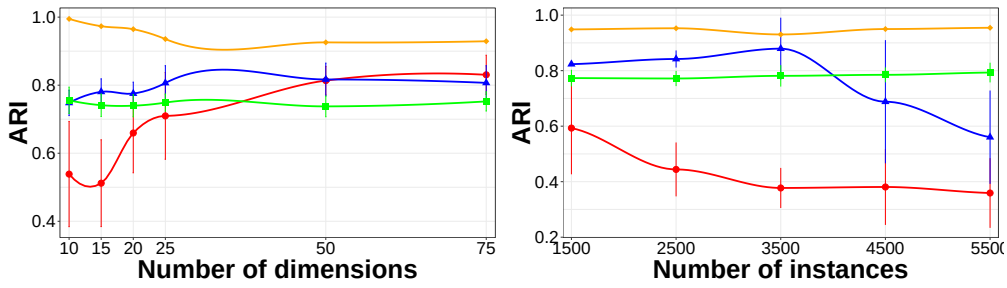
Vinh, N. X., Epps, J., and Bailey, J. (2010). Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *Journal of Machine Learning Research*, 11:2837–2854.

von Luxburg, U. (2007). A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416.

von Luxburg, U., Williamson, R. C., and Guyon, I. (2012). Clustering: Science or art? In *Proceedings of the Unsupervised and Transfer Learning Workshop held at ICML 2011*, pages 65–80.

Xu, R. and II, D. C. W. (2005). Survey of clustering algorithms. *IEEE Trans. Neural Networks*, 16(3):645–678.

Xue, B., Zhang, M., Browne, W. N., and Yao, X. (2016). A survey on evolutionary computation approaches to feature selection. *IEEE Trans. Evolutionary Computation*, 20(4):606–626.



(a) Varying the number of dimensions

(b) Varying the number of instances

Figure 10: ARI achieved by each clustering method on the OpenSubspace clustering datasets, where either the number of dimensions, or number of instances are varied.The red dots correspond to GPGC, the blue triangles to GPGC-AIC, the green squares to PROCLUS, and the orange diamonds to DOC respectively.

Appendix

A Subspace Clustering Results

The fitness function proposed in this paper is designed to use all features in the feature space when calculating distances, as our primary goal is to reduce dimensionality and produce interpretable similarity functions, while maintaining good cluster quality. However, a natural extension of our proposed approach is to apply it to subspace clustering problems, as our GP representation has the potential to use different feature subsets in different clusters. To investigate the plausibility of this extension, we applied GPGC and GPGC-AIC to datasets from OpenSubspace (Müller et al., 2009), a collection of popular subspace benchmarking datasets. We chose to use the PROCLUS and DOC algorithms for comparison as examples of commonly used cell-based and clustering-oriented subspace clustering algorithms respectively. We chose PROCLUS and DOC as they have been shown to have superior (or similar) performance to other subspace algorithms in their paradigm (Müller et al., 2009). We do not compare to a clustering algorithm from the third paradigm – density-based subspace clustering – as these methods all produced overlapping (i.e. non-crisp) clusters, which is not the focus of this study.

Fig. 10 shows the performance of each of the four methods (GPGC, GPGC-AIC, PROCLUS, and DOC) as the number of dimensions is varied (Fig. 10a) and the number of instances (Fig. 10b) is varied respectively. For each dataset, we plot the ARI achieved by each method, as well as the standard deviation across 30 runs (the vertical bars). The two proposed methods are competitive with PROCLUS as the dimensionality is increased, but PROCLUS clearly outperforms GPGC and is slightly better than GPGC-AIC as the number of instances is increased. The DOC method is clearly the best of all the methods on both categories of datasets. However, GPGC-AIC in particular shows some promise given it has not been optimised for this task, but can still achieve competitive results often with one common subspace clustering method. Furthermore, GPGC-AIC is clearly superior to vanilla GPGC, which further reinforces the findings throughout this paper. We hope to further improve GPGC-AIC and make it a competitive subspace clustering algorithm in the future by developing a new fitness function and designing new genetic operators.

Response Letter