GPGC: Genetic Programming for Automatic Clustering using a Flexible Non-Hyper-Spherical Graph-Based Approach

Andrew Lensen
School of Engineering and Computer Science
Victoria University of Wellington
PO Box 600
Wellington 6140, New Zealand
andrew.lensen@ecs.vuw.ac.nz

Bing Xue
School of Engineering and Computer Science
Victoria University of Wellington
PO Box 600
Wellington 6140, New Zealand
bing.xue@ecs.vuw.ac.nz

Mengjie Zhang
School of Engineering and Computer Science
Victoria University of Wellington
PO Box 600
Wellington 6140, New Zealand
mengjie.zhang@ecs.vuw.ac.nz

ABSTRACT

Genetic programming (GP) has been shown to be very effective for performing data mining tasks. Despite this, it has seen relatively little use in clustering. In this work, we introduce a new GP approach for performing graph-based (GPGC) non-hyper-spherical clustering where the number of clusters is not required to be set in advance. The proposed GPGC approach is compared with a number of well known methods on a large number of data sets with a wide variety of shapes and sizes. Our results show that GPGC is the most generalisable of the tested methods, achieving good performance across all datasets. GPGC significantly outperforms all existing methods on the hardest ellipsoidal datasets, without needing the user to pre-define the number of clusters. To our knowledge, this is the first work which proposes using GP for graph-based clustering.

CCS CONCEPTS

•**Computing methodologies** → **Genetic programming;** *Cluster analysis; Feature selection;*

KEYWORDS

Cluster Analysis; Automatic Clustering; Graph-Based Clustering; Feature Construction; Genetic Programming; Evolutionary Computation

ACM Reference format:
Andrew Lensen, Bing Xue, and Mengjie Zhang. 2017. GPGC: Genetic Programming for Automatic Clustering
using a Flexible Non-Hyper-Spherical Graph-Based Approach. In *Proceedings of the Genetic and Evolutionary Computation Conference 2017, Berlin, Germany, July 15–19, 2017 (GECCO '17),* 8 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Evolutionary Computation (EC) [6] algorithms, a category of meta-heuristics algorithms founded on biological evolutionary principles, have been widely applied to a range of data mining tasks successfully. Clustering, the task of grouping similar instances in a dataset into a number of clusters (K), is one such data mining task that EC has performed very well on [10, 23]. Genetic programming (GP) [16] is a flexible EC method which represent solutions as simple computer programs. GP has seen relatively little use in clustering tasks, despite its flexible solution structure having the potential to allow for a range of different cluster shapes and sizes to be produced. In particular, GP has seen little use in clustering for producing non-hyper-spherical (NHS) clusters.

Hyper-spherical (HS) clusters are shaped in a way such that the instances in the cluster lie in a hyper-sphere around the cluster mean. While HS clusters are quite common, algorithms such as k-means which produce HS clusters will perform very poorly on datasets where clusters are ellipsoidal, curved, or otherwise non-uniform [15]. A common technique in the clustering domain for producing NHS clusters is to use an intuitive graph-based approach, where a cluster is defined by a graph containing instances which are connected by edges [21]. The full cluster partition then consists of a number of distinct graphs.

The techniques used for deciding which instance pairs should share an edge in graph-based clustering are generally quite naïve — for example, simple distance thresholds are often used [21]. Such techniques often limit the performance of graph-based clustering, and limit the extent to which algorithms can be tailored specifically to a given dataset. GP is well-known for its ability to generate dynamic decision tree-esque solutions which are automatically optimised for the dataset being used by the evolutionary process. We hypothesise that using GP to evolve dynamic individuals which decide which pairs of instances (nodes) are connected by an edge will allow the power and intuitive representation provided by a graph-based approach, while overcoming the limitations of a static design for forming edges.

In this work, we will propose a novel GP graph-based clustering (GPGC) algorithm designed to dynamically produce a range of cluster shapes including both HS and NHS clusters. To our knowledge, this is the first piece of work using GP to build cluster graphs, and one of the very few pieces of work using GP to do NHS clustering. More explicitly, we will:

- Introduce a new GP program representation to allow GP to dynamically define edges between instances (Section 3.1),
- Propose a new algorithm that uses the output of a GP tree to directly perform graph-based clustering (Section 3.2),

GECCO '17, Berlin, Germany

978-x-xxxx-xxxx-x/YY/MM…$15.00
DOI: 10.1145/nnnnnnn.nnnnnnn

- Design a new fitness function for evaluating NHS clusters while ensuring an appropriate number of good-quality clusters are generated (Section 3.3), and
- Comprehensively compare our proposed algorithm to a number of existing methods across a wide range of HS and NHS datasets (Section 5).

2 BACKGROUND

Many different clustering approaches have been proposed using a variety of techniques: hard and soft clustering allocate instances to one or more clusters respectively; partitional, density, hierarchical, and graph-based techniques are based on different fundamental ideas and perform differently on different types of datasets [1].

Hard partitional clustering algorithms, which are by far the most common clustering algorithms, assign every instance to exactly one cluster. The most famous algorithm in this area is k-means [14], also known as Lloyd's algorithm. k-means generates random initial clusters and then iteratively refines them by recomputing the cluster centres at each stage to minimise intra-cluster variance. k-means++ [3] is an improved version of k-means, which performs more intelligent selection of the initial (seed) values for clusters by maximising the distance between the initial cluster centres. It has been widely used with good results for clustering problems where HS clusters are appropriate, but is known to perform badly on clusters of other shapes.

The most famous of the density-based clustering algorithms is DBSCAN [7], which has since been largely succeeded by OPTICS [2]. OPTICS improves upon DBSCAN by detecting clusters in data of varying density more accurately, by ordering instances by their similarity. Unlike DBSCAN, OPTICS does not explicitly produce a partition; instead a parameter, ξ, is used — the value of ξ represents the relative decrease in density which represents a cluster boundary, e.g. $\xi = 0.1$ corresponds to a 10% drop in density.

Other clustering algorithms include agglomerative clustering, a hierarchical clustering algorithm which initially assigns all instances to their own clusters and then iteratively merges clusters greedily [1]. The most popular graph-based clustering algorithm is the Highly Connected Subgraph (HCS) [12] method, which uses a similarity graph of instances and then finds the most highly connected sub-graphs to be clusters.

2.1 Related Work

While a small range of methods have been proposed that use GP for clustering, most inherently encourage the production of HS clusters by using a fitness function which considers the intra-cluster distance to the cluster mean, and/or the inter-cluster distance to the dataset mean. One exception to this is a Novelty Search (NS)-GP clustering method [19], which evolves individuals based on how novel (i.e. sparse in the search space) they are relative to other individuals, and then chooses a final best individual based on a cluster distance ratio (CDR), which does not encourage HS clusters. Unfortunately, the method was designed only for $K = 2$ clustering problems, and was only tested on datasets with three features, making it unlikely to be useful on general clustering tasks.

Coelho et al. [5] propose an ensemble-based GP clustering method which combines the output of multiple different commonly used clustering algorithms using consensus functions as function nodes in GP tree in order to produce better combine clustering partitions. While this method achieved good results, it uses a multi-objective fitness function which considers compactness, which encourages HS clusters to be formed. Furthermore, two of the four base clustering algorithms used (k-means and hierarchical average linkage) are biased towards hyper-spherical clusters. While it is possible this method may still be able to produce NHS clusters, it is hoped that using a pure GP approach which is designed have no bias in cluster shape will be able to produce better results.

Boric et al. [4] and Falco et al. [8] also proposed two GP clustering methods which use a multi-tree and a grammar-based approach respectively. Both these approaches again use fitness functions which encourage HS clusters to be generated. Boricfis approach evolves one tree for each cluster, and then assigns individuals to clusters based on which tree outputs the maximum value. While this is an interesting approach, it does require that K is preset.

Other EC methods such as Genetic Algorithms (GAs) have been used with a graph-based clustering approach [17], but no existing work has proposed dynamically evolving similarity functions using feature construction as we do here.

In summary, existing GP clustering work has several limitations which we seek to overcome: they are commonly restricted to, or encourage, producing HS clusters (due to their representation/fitness function design); they often require K to be preset; and many proposed methods do not scale with a large number of features (m) or clusters — clustering is a very hard problem in huge search spaces.

3 METHOD

The core novelty of this work is the use of GP trees to compare two instances, and produce a single output value which gives a measure of how *clusterable* they are — that is, how strongly the GP tree believes two instances should lie in the same cluster. Clusters can then be formed by putting an edge between all *clusterable* pairs; a set of graphs will be produced, where each graph represents a cluster. GP is effectively being used to replace the distance functions (e.g. Euclidean distance) commonly used in clustering algorithms. By performing feature construction on the original feature set with a variety of arithmetic operators, it is thought that the evolutionary process will be able to tailor the "distance functions" produced to the dataset being trained on, without being restricted to producing HS clusters, thereby increasing clustering performance.

Several questions arise when designing a method using this idea:

- How can a GP tree take two instances as input?
- Which instance pairs should be compared, and what defines a *clusterable* pair?
- Which fitness function should be used, given we are attempting to perform NHS clustering and so cannot use the commonly used fitness functions proposed by other EC clustering work?

We explore each of these issues in the following subsections.

3.1 GP Program Design

Most work using GP for feature manipulation defines the terminal set as the features of the **single** instance being evaluated and, optionally, a random real value. We extend this approach to allow a

tree to take two instances by instead defining the terminal set as all the features in each of the **two** instances being compared, and a random real value (to allow the GP tree to scale features). Thus, with m features, there will be $2m + 1$ terminals. The function set contains the following functions: $\{+, -, \times, \div, |+|, |-|, \max, \min, \text{if}\}$. All functions except for *if* take two real inputs and output a single real value. *if* takes three real inputs and outputs the second input if the first input is positive; otherwise it outputs the third input. $|+|$ and $|-|$ represent absolute addition and subtraction; $\div$ is protected division — it returns 1 if the divisor (the second input) is 0.

3.2 Clustering Process

A given GP tree will output a single value which measures how *clusterable* the instances being evaluated are. Each instance, i, is said to be connected to the top c most clusterable instances with respect to it; i.e., the c instances which produce the largest tree output when fed into the tree along with instance i. Finding the c most clusterable instances across the dataset for each instance would require $n \times (n-1)$ evaluations of the GP tree for n instances, giving complexity of $\Theta(n^2)$. This may make the evolutionary process very slow, as each individual must be evaluated $n \times (n-1)$ times every generation. The largest datasets have $n \approx 6000$, which would mean 3.6×10^7 evaluations for each tree is required. However, it is usually not necessary to compare pairs of instances which are far apart — indeed, it is logical to think that an instance should only be allowed to be connected to a number of its nearest neighbours. As a graph-based structure is used, we can instead only compare each instance to its l nearest neighbours, which gives $\Theta(nl)$ complexity which reduces to $\Theta(n)$ when l is constant and $l \ll n$. In the $n \approx 6000$ case, only $6000 \times l$ evaluations per tree will be performed.

A number of different values of c and l were considered. It was determined that $c = 1$ performed well for all datasets, and that increasing c further quickly reduced the number of clusters produced, harming performance. It was found that l should be expressed as a function of n — the more instances present in a dataset, the more neighbours a given instance should be compared to, as cluster size typically increases relative to n. A suitable heuristic was found empirically where $l = \lceil \sqrt[3]{n} \rceil$, such that l is always at least 2. Using this heuristic, evaluating a GP tree will have complexity of $\Theta(n \times \lceil \sqrt[3]{n} \rceil)$.

Figure 1 shows the key steps required to produce a set of graphs for a given GP tree, and the full process for producing the cluster partition is shown below:

(1) Assign each instance to its own cluster.
(2) For each instance, feed itself (as the 1st instance) and each of its l nearest neighbours (in turn, as the 2nd instance) into the tree. Add a *directed* edge between the instance and the c neighbours which produce the highest tree output.
(3) Pick an instance. Find all instances it has a path to by performing a depth-first search for each of its l edges. Merge all these instances' clusters and the chosen instance's cluster into a single cluster.
(4) Repeat Step 3 for all instances.
(5) A partition with a number of clusters equal to the number of graphs has been generated.

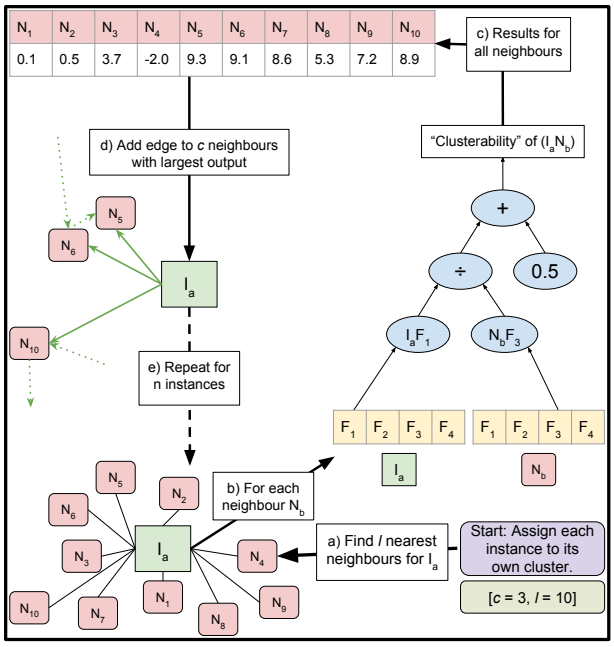


Figure 1: Building a set of graphs for a given GP tree.

3.3 Fitness Function

As previously discussed, the commonly used HS-based fitness functions (e.g. the trace scatter metric and the sum of squared error (SSE) [18]) cannot be directly used to evaluate the proposed approach as we wish to encourage NHS clusters to be produced. To address this, we propose a novel fitness function which measures three important indicators of cluster quality in a NHS manner. These indicators are as follows:

Connectedness, which measures the extent to which each instance is in the same cluster as its j nearest neighbours; instances which are similar should lie in the same cluster. When j is small ($j \ll n$), NHS clusters with good connectedness can be produced; when j becomes too large relative to n, the connectedness metric begins to essentially measure cluster compactness in a spherical manner, as each instance will be compared to all other instances in its cluster. Furthermore, as j increases, a smaller number of bigger clusters are produced, as instances are encouraged to be in the same cluster as an increasing number of neighbours. We trialled several small values of j and found $j = 10$ provided the best balance of encouraging connectedness while allowing flexible cluster shapes.

Compactness, which measures how tightly-packed a cluster is. Normally, compactness is calculated based on the distance from each instance to the cluster centre, or to each other instance in the cluster. However, these two approaches will encourage hyper-sphericality. Instead, we consider how far each instance is away from its nearest neighbour in the same cluster. To best punish clusters which contain instances distant from their neighbours, we find the **maximum** of the **minimum** distances between each instance in a given cluster and its nearest neighbour; we call this the **sparsity** of the cluster.

Separability, which measures how well neighbouring clusters are separated in the feature space. Again, normally separability is based on the distance between pairs of cluster centres, or the

distance from cluster centres to the overall dataset centre. As metrics based on cluster centres will encourage HS clusters, we instead propose finding the minimum distance between each instance in a given cluster and the instances in all other clusters. That is, we find the **minimum** of the **minimum** distances between each instance in a cluster and all other instances not in that cluster.

A good partition will be maximally connected, minimally sparse, and maximally separated. Each cluster in a partition should have a high ratio of sparsity to separability — we call this ratio the Cluster Sparsity:Separability index (CS:S). In addition, each cluster should have a high mean connectedness amongst its instances. To balance these three indicators, we propose the following fitness function:

$$\text{Fitness} = \frac{\text{Mean Connectedness}}{\text{Mean CS:S}} \tag{1}$$

where

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

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

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 N_{I_a} gives the j nearest neighbours to I_a. We use $j = 10$ in this work. $d(I_a, I_b)$ is the Euclidean distance between two instances, defined in the standard way.

$$\text{Mean CS:S} = \frac{1}{K}\sum_{i=1}^{K}\frac{\max_{I_a \in C_i}\left(\min_{I_b \in C_i, I_b \neq I_a} d(I_a, I_b)\right)}{\min_{I_a \in C_i}\left(\min_{I_b \notin C_I} d(I_a, I_b)\right)} \tag{4}$$

The above fitness function is also novel in our work in that it does not contain a component based on the number of clusters, K. It was found that by balancing these three metrics, K was able to be found with relative accuracy on many datasets based on the inherent cluster quality — this is an improvement over previous work which included a component in the fitness function which explicitly punished large K.

4 EXPERIMENT DESIGN

This section details the baseline methods used to compare to the proposed GPGC algorithm, describes the datasets used, the metrics used to evaluate cluster quality, and provides the parameter settings used in the experiments performed.

4.1 Baseline Methods

We use k-means++ [3] and OPTICS [2] as our baseline partitional and density-based clustering methods. We use the OPTICSXi algorithm provided by the ELKI [22] framework as our OPTICS baseline. In addition, we use a PSO algorithm using a medoid representation [13] as our baseline EC method — we struggled to obtain the source code of any recent GP-based clustering algorithms for comparison. By using a prototype-based method (k-means++), a density-based method (OPTICS) and a baseline EC method, we will compare our proposed method to a range of existing methods.

4.2 Datasets

We used a range of clustering datasets with different cluster shapes and different numbers of clusters (K), features (m), and instances (n) in order to comprehensively evaluate our proposed method. The first two groups of datasets were generated using two widely used synthetic clustering data generators designed by Handl et al. [11]. The first generator uses a Gaussian distribution to generate clusters. Table 1 shows the characteristics of the Gaussian datasets. The second generator uses an ellipsoidal distribution. A large variety of datasets were produced using this dataset, with between 10 and 1000 features, and 10 and 100 clusters. These datasets are inherently more difficult than the Gaussian datasets, due to their larger number of features and clusters, and their less uniform shapes. In particular, existing methods struggle significantly on the 100 feature and 1000 feature datasets; GPGC will be tested to see if it can perform well at these high dimensions. The final group of datasets are sourced from a collection of hand-crafted datasets curated by Fränti et al. [9]. This group has datasets with a range of different shapes (as shown in Figure 2) which are used to evaluate if clustering methods can adapt to varying cluster densities and geometries. These datasets only have two features each, but vary in the number of instances and clusters they contain, as detailed in Table 3. All datasets are pre-processed by scaling each feature to fall between 0 and 1 to reduce the effect of feature range on the clustering process.

Table 1: Generated gaussian datasets [11].

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

Table 2: Generated ellipsoid datasets [11].

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		1000	2349	40
50d40c	50	2335	40	1000d100c	1000	6165	100

Table 3: Hand-crafted datasets [9].

Name	m	n	K	Name	m	n	K
Jain	2	373	2	Flame	2	240	2
R15	2	600	15	Compound	2	399	6
D31	2	3100	31	Pathbased	2	300	3
Aggregation	2	788	7	Spiral	2	312	3

4.3 Evaluation Metrics

The clustering performance of a given algorithm can be measured using a variety of metrics. In this work, we evaluate the GPGC algorithm compared to the baseline methods based on four criteria:

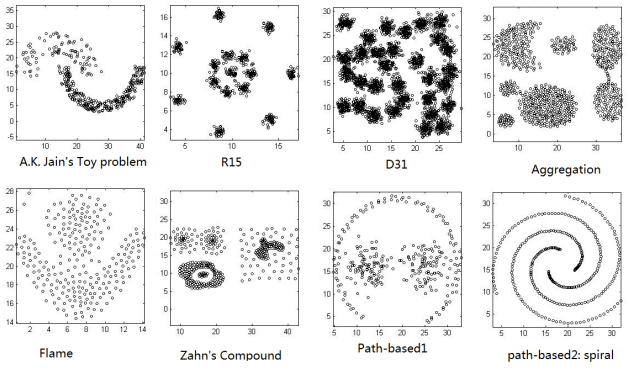


Figure 2: Visualisation of hand-crafted datasets [9].

the number of clusters (K) produced (which should be as close to the actual number of clusters as possible), the sum of the distance between each instance and its cluster centre (called $\sum$ Intra), and two external metrics, which compare the clusters produced to the actual known clusters provided by the dataset. As we are using a number of NHS datasets, we focus less on the $\sum$ Intra metric, as it is naturally optimised by forming HS clusters. The two external metrics used are described below.

F-Measure: The F-Measure is a commonly used metric for measuring accuracy in classification tasks. Unfortunately, the F-measure cannot be directly applied to evaluating clustering algorithms, as there is no ideal method for mapping the clusters produced to the actual known clusters. Instead, we use an analogue to the F-measure, where instance pairs are evaluated to determine whether or not they should appear in the same cluster or not, according to the known clusters. More explicitly, each possible pair of instances is considered and one of the following cases is selected:

(1) Both instances in the same known cluster and the algorithm assigns them to the same cluster: true positive (TP).
(2) Both instances in the same known cluster but the algorithm assigns them to **different** clusters: false negative (FN).
(3) Both instances in **different** known clusters and the algorithm assigns them to **different** clusters: true negative (TN).
(4) Both instances in **different** known clusters but the algorithm assigns them to the same cluster: false positive (FP).

The F-measure is then calculated as it is in the classification domain based on the sum of each of the TPs, FPs, and FNs:

$$\text{F-measure} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{5}$$

$$\text{recall} = \frac{TPs}{TPs + FNs} \tag{6} \qquad \text{precision} = \frac{TPs}{TPs + FPs} \tag{7}$$

Class purity: A metric which represents how uniform each cluster is in terms of the known cluster labels of the instances it contains. Generally, it is expected that good clusters will have instances which belong to the same known cluster. The class purity is computed as follows. Firstly, find the majority known cluster label for each cluster, and then count the number of instances in each cluster which have the majority known cluster label. Then compute the class purity as the total count divided by the number of total instances.

Table 4: Gaussian datasets

(a) 10d10cGaussian

Method	#Clusters	$\sum$ Intra	Class Purity	F-Measure
GPGC	20.27	735.9	0.8959	0.7554
k-means++	10.00	713.2^{+}	0.9299^{+}	0.8647^{+}
OPTICS–0.005	39.00	783.1^{-}	0.8721	0.5721^{-}
PSO Medoid	10.00	720.5	0.9116	0.8526^{+}

(b) 10d20cGaussian

Method	#Clusters	$\sum$ Intra	Class Purity	F-Measure
GPGC	19.87	213.3	0.9965	0.9963
k-means++	20.00	235.6^{-}	0.9287^{-}	0.886^{-}
OPTICS–0.001	26.00	212.7	0.9891^{-}	0.9015^{-}
PSO Medoid	20.00	219.6^{-}	0.9749^{-}	0.9589^{-}

(c) 10d40cGaussian

Method	#Clusters	$\sum$ Intra	Class Purity	F-Measure
GPGC	34.03	426.0	0.9518	0.9528
k-means++	40.00	423.0	0.9417^{-}	0.8936^{-}
OPTICS–0.001	55.00	397.0^{+}	0.9861^{+}	0.8483^{-}
PSO Medoid	40.00	424.2	0.9438	0.8991^{-}

4.4 Parameter Settings

Each of the three baseline methods and the proposed GPGC algorithm require parameters to be set before they are used. For both the k-means++ and PSO Medoid algorithms, K is set to the known number of clusters. k-means++, the PSO method, and GPGC are all run for 100 iterations/generations, by which point convergence has nearly always occurred. The PSO method uses standard settings for its parameters [24]: a swarm size of 30, $v_{max} = 6$, $w = 0.729844$, and $c_1 = c_2 = 1.49618$. The proposed GPGC algorithm uses a population size of 1,024, 20% mutation, 80% crossover, top-10 elitism, and minimum and maximum tree depths of 2 and 7 respectively [20]. The initial population is generated using the half-and-half method, and tournament selection is used with a tournament size of 7 [20]. The GPGC and PSO parameters are consistent across all the datasets tested. For the OPTICS algorithm, ξ must be set per-dataset in order to achieve reasonable results. In order to increase the confidence in making a fair comparison to OPTICS, we ran OPTICS several times on each dataset with different ξ values from the range $[0.001, 0.005, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5]$. All of the methods excluding OPTICS are non-deterministic in nature and so were run 30 times each, and the mean results computed.

5 RESULTS AND ANALYSIS

The results on each of the three types of datasets (Gaussian, ellipsoid, and handcrafted) are shown in Tables 4, 5, and 6 respectively. For each dataset, we report the number of clusters as well as the performance on the clustering metrics for each of the four methods. For the OPTICS method, only the ξ value with the highest F-Measure is included. A Student's t-test with a 95% confidence interval was performed on each baseline result — when the baseline was significantly better or worse than GPGC, the result is marked with a "+" or "−" respectively. No mark indicates no significant difference was found. More "−" markings will indicate GPGC was able to outperform existing methods more often. The results on each type of dataset will be analysed in turn in the following paragraphs.

Gaussian datasets: GPGC is significantly better than all baseline methods on two of the three Gaussian datasets in terms of the F-measure results, despite it not being specifically designed for HS clustering and not having K pre-defined (in contrast to k-means++ and the PSO Medoid method). The two HS methods significantly outperform GPGC on the $K = 10$ dataset; at low dimensions, k-means++ can perform very well on HS datasets as it is specifically designed to minimise intra-cluster variance in regard to the cluster centre. GPGC is also able to find K much more accurately than OPTICS, while achieving much higher F-measure values.

Ellipsoid datasets: GPGC is consistently the best method across the ellipsoid datasets, significantly outperforming all other methods in terms of the F-measure results on all datasets except for k-means++ on 10d100c (where there is no significant difference). GPGC achieves particularly impressive results on the datasets with the highest numbers of features and clusters (the 100d and 1000d datasets), where the baseline methods struggle considerably. For example, on the hardest datasets with 1000 features and 100 clusters, GPGC achieves an F-measure more than twice that of the next closest result, and over 8 times better than the commonly used k-means++ algorithm. GPGC is also able to accurately find the number of clusters on a majority of datasets, finding K within 25% of the true value on all datasets except 10d10c. This is a promising result considering that the fitness function used does not consider the number of clusters at all — GPGC is able to naturally find the correct K based on the data structure alone.

Handcrafted datasets: On the handcrafted datasets, GPGC has the best F-measure performance on two datasets (aggregation and spiral), and is 2nd to OPTICS on two others (compound and jain). These four datasets are clearly have extremely non-uniform cluster shapes, and so GPGC is able to significantly outperform the k-means++ and PSO distance-based algorithms. The clustering results on the aggregation dataset are visualised in Figure 3. These visualisations clearly show how GPGC is able to find well-formed clusters of varying shape. Interestingly, GPGC differs from the ground truth only in that it merges clusters which are connected by a small number of instances — we argue that in doing so, GPGC has not produced a strictly incorrect partition. k-means++ clearly performs poorly due to the circular clusters produced, and OPTICS seems to struggle with this dataset — it mistakenly combines several distinct clusters into the same cluster, and also breasks two bigger clusters up by adding an additional obviously incorrect cluster.

Both the path-based and flame datasets, for which GPGC achieves the 3rd and 4th best F-measure result respectively, do not have a clear separation between their two clusters; instances within each cluster are similar distances apart to instances **between** clusters. On the pathbased dataset, the boundary between the two "blob" clusters are quite fuzzy relative to the large arc cluster. This is an inherent limitation of the GPGC method, as the proposed fitness function encourages solutions to be produced which have clear separation between clusters and minimal gaps within clusters. We hypothesise that an alternate fitness function may be able to give much better results on these datasets — unlike OPTICS or k-means++, GPGC can be tailored to be more suited to a particular clustering problem by changing the measure of cluster quality used. The final two datasets, r15 and d31, contain clusters which are roughly circular, and so it is unsurprising that GPGC is outperformed by the PSO and k-means++ methods. It would be interesting to examine how GPGC performs relative to the baselines if these handcrafted datasets were extended to have a higher m — as shown on the elliptical dataset results, GPGC performs very well at high dimensionality where the other baselines fail.

Summary: The results showed that the proposed GPGC algorithm significantly outperforms all baseline methods across the ellipsoid datasets, with particularly impressive performance on the

Table 5: Ellipsoid datasets (Part 1)

(a) 10d10c

Method	#Clusters	$\sum$ Intra	Class Purity	F-Measure
GPGC	30.90	578.2	0.9588	0.7995
k-means++	10.00	602.1	0.7446^{-}	0.5471^{-}
OPTICS-0.05	32.00	1124.0^{-}	0.3541^{-}	0.2387^{-}
PSO Medoid	10.00	582.5	0.7781^{-}	0.5881^{-}

(b) 10d20c

Method	#Clusters	$\sum$ Intra	Class Purity	F-Measure
GPGC	25.90	199.4	0.8389	0.6824
k-means++	20.00	179.2^{+}	0.715^{-}	0.5082^{-}
OPTICS-0.001	69.00	160.3^{+}	0.8602	0.3619^{-}
PSO Medoid	20.00	179.4^{+}	0.7433^{-}	0.5451^{-}

(c) 10d40c

Method	#Clusters	$\sum$ Intra	Class Purity	F-Measure
GPGC	53.93	339.2	0.7918	0.5728
k-means++	40.00	299.1^{+}	0.6881^{-}	0.4424^{-}
OPTICS-0.001	118.00	322.4	0.7616^{-}	0.3418^{-}
PSO Medoid	40.00	311.6^{+}	0.6929^{-}	0.4781^{-}

(d) 10d100c

Method	#Clusters	$\sum$ Intra	Class Purity	F-Measure
GPGC	118.00	964.2	0.6941	0.4291
k-means++	100.00	719.8^{+}	0.6724	0.4051
OPTICS-0.001	290.00	856.4^{+}	0.7084	0.2829^{-}
PSO Medoid	100.00	930.8	0.574^{-}	0.3819^{-}

(e) 50d10c

Method	#Clusters	$\sum$ Intra	Class Purity	F-Measure
GPGC	10.63	1558.0	0.9646	0.9592
k-means++	10.00	1307.0^{+}	0.7412^{-}	0.4833^{-}
OPTICS-0.05	28.00	2099.0^{-}	0.5752^{-}	0.3684^{-}
PSO Medoid	10.00	1242.0^{+}	0.7605^{-}	0.5073^{-}

(f) 50d20c

Method	#Clusters	$\sum$ Intra	Class Purity	F-Measure
GPGC	22.73	597.1	0.8909	0.7967
k-means++	20.00	522.9^{+}	0.7047^{-}	0.3698^{-}
OPTICS-0.005	73.00	421.3^{+}	0.8995	0.4017^{-}
PSO Medoid	20.00	508.3^{+}	0.7561^{-}	0.4822^{-}

(g) 50d40c

Method	#Clusters	$\sum$ Intra	Class Purity	F-Measure
GPGC	48.63	926.9	0.9036	0.797
k-means++	40.00	827.3^{+}	0.6922^{-}	0.2549^{-}
OPTICS-0.001	150.00	652.9^{+}	0.9122	0.4042^{-}
PSO Medoid	40.00	844.3^{+}	0.7438^{-}	0.4463^{-}

Table 5: Ellipsoid datasets (Part 2)

(h) 100d10c

Method	#Clusters	∑ Intra	Class Purity	F-Measure
GPGC	10.17	2092.0	0.9894	0.9916
k-means++	10.00	1907.0^{+}	0.7713^{-}	0.5463^{-}
OPTICS-0.001	92.00	1571.0^{+}	0.9779^{-}	0.4547^{-}
PSO Medoid	10.00	1814.0^{+}	0.7862^{-}	0.5983^{-}

(i) 100d20c

Method	#Clusters	∑ Intra	Class Purity	F-Measure
GPGC	20.43	961.0	0.917	0.8759
k-means++	20.00	821.3^{+}	0.698^{-}	0.3638^{-}
OPTICS-0.01	76.00	610.4^{+}	0.9395^{+}	0.3859^{-}
PSO Medoid	20.00	767.8^{+}	0.7647^{-}	0.4678^{-}

(j) 100d40c

Method	#Clusters	∑ Intra	Class Purity	F-Measure
GPGC	48.60	1261.0	0.8755	0.7415
k-means++	40.00	1111.0^{+}	0.7041^{-}	0.2725^{-}
OPTICS-0.001	140.00	919.1^{+}	0.9037^{+}	0.4387^{-}
PSO Medoid	40.00	1115.0^{+}	0.7551^{-}	0.455^{-}

(k) 1000d10c

Method	#Clusters	∑ Intra	Class Purity	F-Measure
GPGC	9.73	6656.0	0.9844	0.9818
k-means++	10.00	5862.0^{+}	0.7408^{-}	0.4929^{-}
OPTICS-0.001	86.00	5049.0^{+}	0.9586^{-}	0.4507^{-}
PSO Medoid	10.00	5511.0^{+}	0.7617^{-}	0.549^{-}

(l) 1000d20c

Method	#Clusters	∑ Intra	Class Purity	F-Measure
GPGC	21.13	2524.0	0.8918	0.8249
k-means++	20.00	2092.0^{+}	0.7105^{-}	0.3578^{-}
OPTICS-0.001	67.00	1574.0^{+}	0.9384^{+}	0.4687^{-}
PSO Medoid	20.00	1915.0^{+}	0.7718^{-}	0.4777^{-}

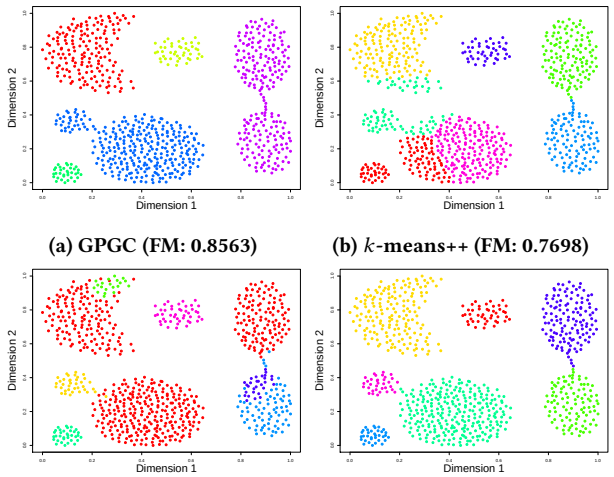
(m) 1000d40c

Method	#Clusters	∑ Intra	Class Purity	F-Measure
GPGC	43.63	4368.0	0.8968	0.7923
k-means++	40.00	3697.0^{+}	0.6837^{-}	0.2197^{-}
OPTICS-0.001	132.00	3037.0^{+}	0.9302^{+}	0.4302^{-}
PSO Medoid	40.00	3604.0^{+}	0.7586^{-}	0.3914^{-}

(n) 1000d100c

Method	#Clusters	∑ Intra	Class Purity	F-Measure
GPGC	127.30	8314.0	0.9665	0.8725
k-means++	100.00	8388.0	0.6597^{-}	0.1021^{-}
OPTICS-0.001	356.00	6471.0^{+}	0.9361^{-}	0.4313^{-}
PSO Medoid	100.00	10840.0^{-}	0.6118^{-}	0.2019^{-}

high-dimensional datasets; achieve the best performance on the majority of the Gaussian datasets; and outperform a majority of the baselines on the handcrafted datasets which had most varied shapes. While the GPGC algorithm struggled on a handful of handcrafted datasets, it was shown to be the best performing method in general across all datasets evaluated, without any parameter tuning across datasets being required. Both the k-means++ and PSO methods required K to be pre-defined, and the OPTICS algorithm had to be tested with a range of ξ values in order to find one which could



(a) GPGC (FM: 0.8563) **(b) k-means++ (FM: 0.7698)**

(c) OPTICS–0.05 (FM: 0.5478) **(d) Ground Truth**

Figure 3: Cluster partitions produced by GPGC compared to k-means++, OPTICS, and the ground truth. Partitions for GPGC and k-means++ are the median result of the 30 runs.

give acceptable results. However, the GPGC algorithm does have the ability to be tailored to a given clustering problem by adjusting the fitness function proposed to match the characteristics of the problem (if known). The fitness function proposed allowed both generalisability (it works well across the majority of datasets), and allowed GPGC to achieve the best performance on many datasets. We believe this is due to the careful design of the fitness function to ensure that it can measure clustering performance accurately independent of cluster shape — using three very important measures of cluster quality (connectedness, compactness, separability) allows a holistic evaluation of cluster quality, and using these measures in an NHS manner prevents bias towards clusters of particular shape.

6 CONCLUSION

In this study we introduced the GPGC algorithm, a novel graph-based GP algorithm for performing clustering while also automatically determining K. As far as we are aware, this is the first work using GP with a graph-based approach for clustering. Our results showed that GPGC was able to achieve good performance on a variety of datasets with a range of characteristics, without requiring any per-dataset parameter tuning. GPGC achieved results that were far superior to existing methods on the ellipsoid datasets, significantly better on most of the Gaussian datasets, and performed well on the handcrafted datasets with the most non-uniformly shaped clusters.

As little work has been done in this area previously, there are a number of future directions which could be taken to extend the GPGC algorithm. Firstly, as the fitness function essentially balances three different competing criteria, it is likely using a multi-objective approach would have the potential to further increase performance. In addition, the function and terminal sets used could be further refined. Currently, a GP tree is able to compare any feature from instance a to any feature from instance b. However, clustering is usually performed by comparing the values of the *same* feature in each of two instances; allowing comparison of different features may introduce large areas of "bad" search space. Furthermore, the

Table 6: Handcrafted datasets

(a) jain

Method	#Clusters	∑ Intra	Class Purity	F-Measure
GPGC	2.93	77.65	1.0	0.8775
k-means++	2.00	82.01	0.882^{-}	0.8178^{-}
OPTICS-0.1	3.00	87.98^{-}	1.0	0.9879^{+}
PSO Medoid	2.00	82.01	0.8819^{-}	0.8176

(b) r15

Method	#Clusters	∑ Intra	Class Purity	F-Measure
GPGC	11.07	35.03	0.7304	0.651
k-means++	15.00	19.37^{+}	0.9252^{+}	0.8989^{+}
OPTICS-0.005	20.00	16.5^{+}	0.9833^{+}	0.9255^{+}
PSO Medoid	15.00	16.91^{+}	0.9771^{+}	0.9597^{+}

(c) d31

Method	#Clusters	∑ Intra	Class Purity	F-Measure
GPGC	76.73	339.8	0.4965	0.3876
k-means++	31.00	121.8^{+}	0.9039^{+}	0.8605^{+}
OPTICS-0.05	56.00	192.6^{+}	0.7852^{+}	0.518^{+}
PSO Medoid	31.00	127.8^{+}	0.8686^{+}	0.8024^{+}

(d) aggregation

Method	#Clusters	∑ Intra	Class Purity	F-Measure
GPGC	5.00	136.9	0.8274	0.8563
k-means++	7.00	94.79^{+}	0.8973^{+}	0.7691^{-}
OPTICS-0.05	7.00	225.9^{-}	0.6409^{-}	0.5478^{-}
PSO Medoid	7.00	92.99^{+}	0.9028^{+}	0.7857^{-}

(e) flame

Method	#Clusters	∑ Intra	Class Purity	F-Measure
GPGC	2.00	76.71	0.6458	0.6976
k-means++	2.00	57.68^{+}	0.8494^{+}	0.7521^{+}
OPTICS-0.05	2.00	60.41^{+}	0.9875^{+}	0.9767^{+}
PSO Medoid	2.00	58.02^{+}	0.84^{+}	0.7393^{+}

(f) compound

Method	#Clusters	∑ Intra	Class Purity	F-Measure
GPGC	4.00	62.62	0.7794	0.8615
k-means++	6.00	46.06^{+}	0.818^{+}	0.6455^{-}
OPTICS-0.1	6.00	65.31	0.9098^{-}	0.9186^{+}
PSO Medoid	6.00	45.21^{+}	0.8512^{+}	0.6589^{-}

(g) pathbased

Method	#Clusters	∑ Intra	Class Purity	F-Measure
GPGC	4.67	57.45	0.7136	0.6182
k-means++	3.00	50.82^{+}	0.7433	0.6586^{+}
OPTICS-0.05	1.00	$+\infty^{-}$	0.3667^{-}	0.4995^{-}
PSO Medoid	3.00	50.83^{+}	0.7439	0.6588^{+}

(h) spiral

Method	#Clusters	∑ Intra	Class Purity	F-Measure
GPGC	4.10	91.66	0.9306	0.8318
k-means++	3.00	62.92^{+}	0.3478^{-}	0.3277^{-}
OPTICS-0.005	6.00	97.87^{-}	0.8718^{-}	0.7354^{-}
PSO Medoid	3.00	62.99^{+}	0.3484^{-}	0.3286^{-}

result of comparing two different features may vary depending on which instance is "first". Consider, for example, a sub-tree of the form $a_5 - b_2$. The result of this expression will vary depending on the order of our two instances. While these limitations are not inherently problematic (GP can likely learn to avoid these problems), we hypothesise that an alternative approach may allow better searching, ergo better performance.

REFERENCES

[1] Charu C. Aggarwal and Chandan K. Reddy (Eds.). 2014. *Data Clustering: Algorithms and Applications.* CRC Press.

[2] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. 1999. OPTICS: Ordering Points To Identify the Clustering Structure. In *Proceedings of the International Conference on Management of Data.* 49–60.

[3] David Arthur and Sergei Vassilvitskii. 2007. k-means++: the advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA).* 1027–1035.

[4] Neven Boric and Pablo A. Estévez. 2007. Genetic programming-based clustering using an information theoretic fitness measure. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC).* 31–38.

[5] André L. V. Coelho, Everlândio Fernandes, and Katti Faceli. 2011. Multi-objective design of hierarchical consensus functions for clustering ensembles via genetic programming. *Decision Support Systems* 51, 4 (2011), 794–809.

[6] A. E. Eiben and James E. Smith. 2015. *Introduction to Evolutionary Computing.* Springer.

[7] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 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.* 226–231.

[8] Ivan De Falco, Ernesto Tarantino, Antonio Della Cioppa, and Francesco Gagliardi. 2005. A novel grammar-based genetic programming approach to clustering. In *Proceedings of the ACM Symposium on Applied Computing (SAC).* 928–932.

[9] Pasi Fränti and others. 2015. Clustering datasets. http://cs.uef.fi/sipu/datasets/. (2015).

[10] Adán José García and Wilfrido Gómez-Flores. 2016. Automatic clustering using nature-inspired metaheuristics: A survey. *Appl. Soft Comput.* 41 (2016), 192–213.

[11] Julia Handl and Joshua D. Knowles. 2007. An Evolutionary Approach to Multiobjective Clustering. *IEEE Trans. Evolutionary Computation* 11, 1 (2007), 56–76.

[12] Erez Hartuv and Ron Shamir. 2000. A clustering algorithm based on graph connectivity. *Inf. Process. Lett.* 76, 4-6 (2000), 175–181.

[13] Eduardo R. Hruschka, Ricardo José Gabrielli Barreto Campello, Alex Alves Freitas, and André Carlos Ponce Leon Ferreira de Carvalho. 2009. A Survey of Evolutionary Algorithms for Clustering. *IEEE Trans. Systems, Man, and Cybernetics, Part C* 39, 2 (2009), 133–155.

[14] M. A. Wong J. A. Hartigan. 1979. Algorithm AS 136: A K-Means Clustering Algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 28, 1 (1979), 100–108.

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

[16] John R Koza. 1992. *Genetic programming: on the programming of computers by means of natural selection.* Vol. 1. MIT press.

[17] Héctor D. Menéndez, David F. Barrero, and David Camacho. 2014. A Genetic Graph-Based Approach for Partitional Clustering. *Int. J. Neural Syst.* 24, 3 (2014). DOI: http://dx.doi.org/10.1142/S0129065714300083

[18] Satyasai Jagannath Nanda and Ganapati Panda. 2014. A survey on nature inspired metaheuristic algorithms for partitional clustering. *Swarm and Evolutionary Computation* 16 (2014), 1–18.

[19] Enrique Naredo and Leonardo Trujillo. 2013. Searching for novel clustering programs. In *Genetic and Evolutionary Computation Conference, GECCO '13, Amsterdam, The Netherlands, July 6-10, 2013.* 1093–1100.

[20] Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. 2008. *A Field Guide to Genetic Programming.* lulu.com.

[21] Satu Elisa Schaeffer. 2007. Graph clustering. *Computer Science Review* 1, 1 (2007), 27–64.

[22] Erich Schubert, Alexander Koos, Tobias Emrich, Andreas Züfle, Klaus Arthur Schmid, and Arthur Zimek. 2015. A Framework for Clustering Uncertain Data. *PVLDB* 8, 12 (2015), 1976–1979.

[23] Weiguo Sheng, Shengyong Chen, Mengmeng Sheng, Gang Xiao, Jiafa Mao, and Yujun Zheng. 2016. Adaptive Multisubpopulation Competition and Multiniche Crowding-Based Memetic Algorithm for Automatic Data Clustering. *IEEE Trans. Evolutionary Computation* 20, 6 (2016), 838–858.

[24] Frans Van Den Bergh. 2006. *An analysis of particle swarm optimizers.* Ph.D. Dissertation. University of Pretoria.