Genetic Programming for Evolving Similarity Functions Tailored to Clustering Algorithms

Hayden Andersen*, Andrew Lensen†, Bing Xue‡
School of Engineering and Computer Science, Victoria University of Wellington
PO Box 600, Wellington 6140, New Zealand
Email: *andershayd@ecs.vuw.ac.nz, †andrew.lensen@ecs.vuw.ac.nz, ‡bing.xue@ecs.vuw.ac.nz

***Abstract*—Clustering is the process of grouping related instances of unlabelled data into distinct subsets called clusters. While there are many different clustering methods available, almost all of them use simple distance-based (dis)similarity functions such as Euclidean Distance. However, these and most other predefined dissimilarity functions can be rather inflexible by considering each feature equally and not properly capturing feature interactions in the data. Genetic Programming is an evolutionary computation approach that evolves programs in an iterative process that naturally lends itself to the evolution of functions. This paper introduces a novel framework to automatically evolve dissimilarity measures for a provided clustering dataset and algorithm. The results show that the evolved functions create clusters exhibiting high measures of cluster quality.**

***Index Terms*—Clustering, Genetic Programming, Similarity Function, Feature Selection**

I. Introduction

Clustering is an unsupervised learning task [1] that has the goal of grouping similar instances of unlabelled data into distinct subsets known as clusters [2]. Nearly all clustering algorithms employ some form of predefined dissimilarity function to evaluate the difference between instances when deciding if they should be grouped. One common category of dissimilarity functions is distance functions, which calculate dissimilarity based on how close two instances are in the feature space. The most commonly used distance function is Euclidean Distance [3], [4], which simply measures the straight-line distance between two instances. However, this comes with its own issues.

First, distance measures will treat all features as equally important, when in reality some features can be much more useful than others, and some can be virtually useless [5]. A good example of this is a situation where you are clustering different weather patterns — rainfall is essential as a feature, while the day of the week is likely to not influence the patterns themselves. These distance measures can also be inflexible, as they use a predefined function to evaluate the distance between instances. For more complex data, this can fail to properly represent the relationships present, and a function to better represent the complexity of the data is required.

Genetic Programming (GP) is an Evolutionary Computation (EC) method that evolves solutions for the target problem, usually in the form of a program tree [6]. These trees can represent a function by taking the form of an expression tree, with internal nodes representing numeric operations and leaf nodes representing values in the data and constant values. GP has been shown to be very successful in handling various complex problems [7]. In particular, the flexible tree-based representation provides GP with the advantage of evolving various forms of functions without any predefined structure of the function. GP has also been used for implicit or explicit feature selection which can make the evolved function powerful by removing unnecessary components through using only the selected features [8], but existing work focuses mainly on supervised learning, with little work on unsupervised learning, e.g. clustering.

This paper explores using GP to evolve new dissimilarity functions for clustering problems, using the features of the two instances being compared as inputs and producing an output that represents how dissimilar the two instances are. This has the potential to solve many of the issues encountered when using a simple distance function. In the process of evolving the trees, GP can automatically select relevant features and utilise them in the trees, i.e. the dissimilarity functions in this work. This provides a form of feature selection, as not all features will be selected for use in the dissimilarity functions (i.e. not all features will be used in the leaf nodes of the evolved GP tree). This will have the effect of automatically preventing useless features from being selected, and will allow more relevant features to have a higher impact on the overall dissimilarity through the use of constant values and positive interactions between features to be utilised for improving the overall performance. These tailored functions are expected to allow clustering algorithms to create better clusters than simple predefined functions.

Previous research [5] applied the concept of using GP to evolve similarity functions for clustering algorithms utilising graph theory, achieving competitive results. However, the method introduced heavily ties the GP process to the graph representation of the clusters, which prevents it from being able to train dissimilarity functions for other clustering methods. In this work, we aim to significantly extend this concept to provide a framework that can evolve high-quality dissimilarity functions for any provided clustering algorithm, with the following specific goals:

- propose a framework for evolving dissimilarity functions for any clustering algorithm, with an improved terminal set and fitness function;

- evaluate the proposed approach against a selection of commonly used predefined functions on several datasets of varying complexity; and
- visualise and analyse the evolved dissimilarity functions to better understand why they can work well.

II. Background

A. Clustering

Clustering is the most widely known problem in the unsupervised learning domain [1]. At the highest level, it is the process of segmenting a collection of instances into multiple subsets known as clusters [1]. Versions of the clustering problem exist where instances can belong to multiple clusters at once [9], or hierarchies of clusters (clusters of clusters) can be created [1]. This paper focuses on hard clustering, where each individual belongs to at most one cluster.

There is a vast range of clustering methods, which can broadly be split into several different categories [10]. In this paper, we use a canonical method from each of the four main categories:

1) k-means++ (partitional clustering): a cluster is defined by its centre, according to the members of that cluster. The clusters are randomly initialised, and then the cluster centres are iteratively updated until the algorithm converges [10]. An example of this is the k-means algorithm [10]. In this work, we use the refined k-means++ algorithm, which selects cluster centres such that the initial clusters are well-spread [11].
2) Agglomerative clustering (hierarchical clustering): a hierarchy of clusters is created by initially starting with each instance in its own cluster, and then repetitively merging these clusters until the required number of clusters are formed [12].
3) HDBSCAN (density-based clustering): based on the premise that a continual region of high density (many instances) in the feature space represents a single cluster. Hierarchical DBSCAN (HDBSCAN) is a state-of-the-art extension of the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm [13] that combines it with hierarchical clustering to extract clusters based on their stability.
4) Graph-based clustering uses a graph theoretic approach to represent clusters as a set of disjoint graphs, where the nodes are instances, joined by edges. In this paper, each instance (node) is connected to its k-nearest neighbours based on a dissimilarity function.

B. Genetic Programming

Genetic programming (GP) is a form of evolutionary algorithm (EA). GP starts from a population of randomly generated programs made up of functions and terminals. Functions take one or more inputs and return an output, and can be arithmetic operations, programmatic operations, domain-specific functions, etc. [14]. The terminal set can be any input values, and usually contains information acquired from task-specific inputs. A fitness measure is used to evaluate the fitness of individual members of the population, and the fittest members are selected to be passed on to the next generation. These members then undergo reproduction and mutation to produce the next generation of offspring. This continues until either a certain number of generations have passed or some other stopping criterion is reached.

A common use of GP is to represent numerical functions in a tree-based representation, taking feature values as input and producing a single numerical output calculated by the rules defined in the tree [6].

C. Related Work

The Genetic Programming Graph-based Clustering (GPGC) algorithm [15], [5] was the first method proposed to automatically evolve similarity functions. GPGC is a graph-based evolutionary clustering algorithm that evaluates a similarity function using GP and then connects each instance to the most similar other instance in the dataset. The fitness function used in GPGC combines sparsity (intra-cluster distance), separation (inter-cluster distance), and connectedness (a measure of how well points are clustered with other nearby points) metrics [15]. One important improvement made to GPGC was the introduction of a multitree approach, where each GP individual consisted of multiple smaller trees as opposed to a single larger tree. This was found to significantly improve the performance on a variety of datasets [5].

A similar piece of work to the idea presented in this paper was recently published [16]. This paper evolved constructed features for use in the clustering domain, working as a wrapper method around the k-means++ algorithm. While the authors achieved high-quality results, this method was never tested on any algorithm other than k-means++, and so there is no indication that it is a valid method to be applied to any clustering algorithm.

There is very little other work in the literature on using GP for clustering. One proposed method used multitree GP to evolve a set of ”membership functions”, where each tree corresponds to a specific cluster. Each instance was then placed into the cluster for which there is the highest output from the function tree that corresponds to that cluster [17]. A potential issue with this method is that it is unlikely to scale well as the number of clusters increases, as it necessitates evolving a tree for every cluster.

A seminal work applying evolutionary computation to the clustering domain is the Multiobjective Clustering with Automatic K-determination (MOCK) algorithm. The MOCK algorithm is split into two parts. First, a Pareto front of clustering solutions is created, using measures of compactness and connectedness as the two objectives. In the second part, a single model is selected from the Pareto front based on the shape of the front [18]. While MOCK bears similarity to this paper in that EAs are applied to multiobjective clustering, MOCK directly treats the clusters as the representation in the algorithm. In contrast, this paper proposes the evolution of a dissimilarity function, which is then used to create clusters using a wrapper-style algorithm. The work by Handl et al. [18]

also introduced a simple algorithm for creating artificial clustering datasets with ellipsoidal clusters and arbitrary cluster orientations.

There has been some work done in evaluating the quality of different dissimilarity functions in the clustering domain. Balcan et al. [19] created a theoretical framework that can be used to evaluate what properties of a similarity function are required or important for clustering purposes. While this paper provides useful theoretical underpinnings, there was no practical evaluation to evidence that the theoretical concepts are useful in practice. This paper aims to evolve functions that have a strong performance in clustering — in theory, this means that it should be able to learn the properties suggested in this paper through the GP process.

III. Proposed Method

The main idea of the proposed method is to evolve a population of dissimilarity functions that can be used in place of general distance metrics in pre-existing clustering algorithms. The algorithms chosen were k-means++, HDBSCAN, graph clustering, and agglomerative (single-link) clustering. This range of algorithms gives at least one algorithm for each of the main categories of clustering methods and allows for the flexibility of the system to be rigorously tested.

A. GP Representation

In this work, we represent similarity functions as GP trees, where the leaves are taken from the terminal set and internal nodes are taken from the function set. A real example of a high-quality evolved tree is shown in Figure 1. The trees are evaluated recursively in a bottom-up manner, such that the input of each function is the output of evaluating its children.

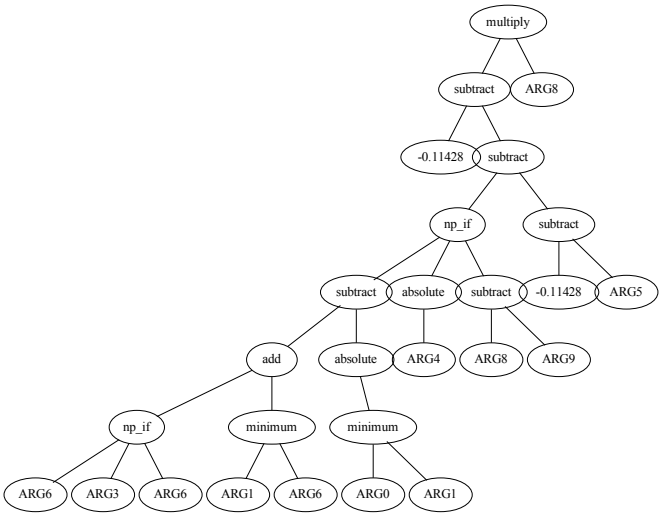

Fig. 1. Sample GP representation of a similarity function

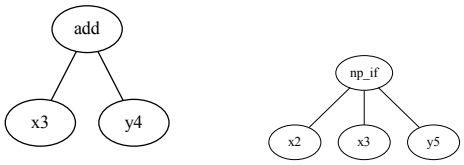

Fig. 2. Example trees using naive terminal set

B. Terminal Set

Previous work [15], [5] used a terminal set that comprised the features of each of the two instances. Mathematically, the terminal set T was defined by Equation 1, where $dim(I)$ is the dimensionality (number of features) of the data and x and y are the two instances being compared.

$$T = \{x_i | i \in \{1, 2, .., dim(I)\}\} \cup \{y_i | i \in \{1, 2, .., dim(I)\}\} \tag{1}$$

There are several limitations to such an approach. Firstly, with this terminal set, it is possible (and likely) for GP to learn dissimilarity functions containing features of only one of the points. This is nonsensical: if a tree, for example, contains only the features of instance a, it would give the same output regardless of the second instance, i.e. $d(a, b) = d(a, c)d(a, d), ...$ Another issue with this terminal set is that it does not ensure that the evolved dissimilarities are symmetrical. For example, the simple similarity functions shown in Figure 2 would result in a different similarity between A and B than they would get from B and A.

Given these trees, imagine two instances with five features, $a = [-0.4, -0.1, 0.3, 0.8, 0.6]$ and $b = [-0.1, 0.8, 0.9, 0.7, -0.5]$. Equations 2 and 3 show the resulting dissimilarities calculated for these instances, taken in both directions. As can be seen, the result is completely different between the two directions.

$$\begin{aligned} d(a, b) &= 0.3 + 0.7 = 1.1 \\ d(b, a) &= 0.9 + 0.8 = 1.7 \end{aligned} \tag{2}$$

$$\begin{aligned} d(a, b) &= \begin{cases} 0.3, & \text{if} -0.1 > 0 \\ -0.5, & \text{otherwise} \end{cases} = -0.5 \\ d(b, a) &= \begin{cases} 0.9, & \text{if } \ 0.8 > 0 \\ 0.6, & \text{otherwise} \end{cases} = \ 0.9 \end{aligned} \tag{3}$$

To produce a better terminal set, we take into account a property that is already known about the features — the corresponding features between two data points are related, as they represent the same aspect of the data. For example, in the well-known Iris dataset, the petal length of one instance is most easily compared to the petal length of another instance. In fact, this is a property that is always assumed in existing

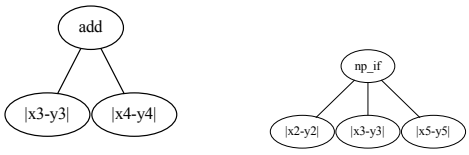

Fig. 3. Example trees using improved terminal set

similarity functions (e.g. Euclidean distance), as they will compare the same feature from each point. To encode this information, the terminal set is taken as the absolute value of the difference between each feature index on the two points. Mathematically, the terminal set T is defined by Equation 4.

$$T = \{|x_i - y_i| | i \in \{1, 2, .., dim(I)\}\} \tag{4}$$

The use of absolute values in the definition of the terminal set ensures that the symmetry is preserved in the resulting dissimilarity functions. Figure 3 shows a similar tree to figure 2, however, now the similarity will be symmetric between A and B.

Using the same two example instances as earlier, $a = [-0.4, -0.1, 0.3, 0.8, 0.6]$ and $b = [-0.1, 0.8, 0.9, 0.7, -0.5]$, Equations 5 and 6 show the resulting dissimilarities calculated for these instances. Now the symmetry property is preserved.

$$\begin{aligned} d(a,b) &= |0.3 - 0.9| + |0.8 - 0.7| = 0.6 + 0.1 = 0.7 \\ d(b,a) &= |0.9 - 0.3| + |0.7 - 0.8| = 0.6 + 0.1 = 0.7 \end{aligned} \tag{5}$$

$$\begin{aligned} d(a,b) &= \begin{cases} |0.3 - 0.9|, & \text{if } |-0.1 - 0.8| > 0 \\ |0.6 - -0.5|, & \text{otherwise} \end{cases} \\ &= \begin{cases} 0.6, & \text{if } 0.9 > 0 \\ 1.1, & \text{otherwise} \end{cases} \\ &= 0.6 \\ d(b,a) &= \begin{cases} |0.9 - 0.3|, & \text{if } |0.8 - -0.1| > 0 \\ |-0.5 - 0.6|, & \text{otherwise} \end{cases} \\ &= \begin{cases} 0.6, & \text{if } 0.9 > 0 \\ 1.1, & \text{otherwise} \end{cases} \\ &= 0.6 \end{aligned} \tag{6}$$

In addition to the derived terminals, a random floating-point constant in the range $[-1, 1]$ is added to the terminal set for scaling purposes and to allow weighting of subtrees.

C. Function Set

The function set contains the standard arithmetic functions: $\{+, -, \times, \div\}$, where $\div$ refers to protected division ($\frac{x}{0} := 1$), as well as the *max*, *min*, *if*, and *abs* operators. All of these functions except *if* and *abs* take two inputs and output a single value based on the function, while *abs* only takes one input and *if* takes three inputs. The *if* operator will output the second input if the first is non-negative, or the third input if the first input is negative. These operators are based on those used in previous research [5].

D. Fitness Evaluation

The silhouette criterion [20] is used to measure the fitness of GP individuals. The silhouette of a clustering solution is a measure of how similar an instance is to instances of its own cluster, compared to instances of other clusters [20]. While originally created as an aid for graphical representation of clusters, it is a good general measure of the quality of a clustering solution. The silhouette of an instance in the solution is defined by Equation 7, where a_i is the average distance from instance i to all other instances in the same cluster and b_i is the minimum of the average distances from instance i to all instances in any other cluster. The definitions of a_i and b_i are given by Equations 8 and 9 respectively where C_i is the cluster containing instance i and $d(i, j)$ is the Euclidean distance from instance i to instance j [20].

Equation 8 gives a measure of how similar an instance is to instances of its own cluster and Equation 9 gives a measure of how similar an instance is to instances of other clusters. Equation 7 then takes the difference between these values.

$$s_i = \frac{b_i - a_i}{max(a_i, b_i)} \tag{7}$$

$$a_i = \frac{1}{|C_i| - 1} \sum_{j \in C_i, i \neq j} d(i, j) \tag{8}$$

$$b_i = \min_{k \neq i} \frac{1}{C_k} \sum_{j \in C_k} d(i, j) \tag{9}$$

The fitness of the individuals is calculated as a wrapper around the supplied clustering algorithm. The evolved GP tree is supplied as a dissimilarity function to the clustering algorithm to create clusters. Once the clusters are generated, the silhouette measure of each instance in the dataset is calculated, and from these, the mean silhouette is calculated. This is shown in Equation 10, where s_i refers to the silhouette of instance i, calculated according to Equation 7.

$$f = \frac{1}{|I|} \sum_{i=1}^{|I|} s_i \tag{10}$$

This is then returned as the overall fitness of the GP individual. With this fitness function, the larger the fitness value, the better an individual, i.e. the fitness function (Equation 10) is a **maximisation** function.

E. Overall Algorithm

A diagram of the overall algorithm is shown in Figure 4. First, a random population of trees is generated using the Ramped-Half-and-Half approach. Then, every individual in the population is evaluated according to the fitness function shown by Equation 10. A chosen number of the best solutions are put

aside for elitism. Then, the selection operator is performed to select good individuals, which are then used by the genetic operators, i.e. mutation and crossover, to generate offspring to form the child population. Finally, the child population is appended to the best solutions that were put aside to form the new population in the next generation. This process continues until a preset number of generations have passed, at which point the individual with the best fitness is returned.

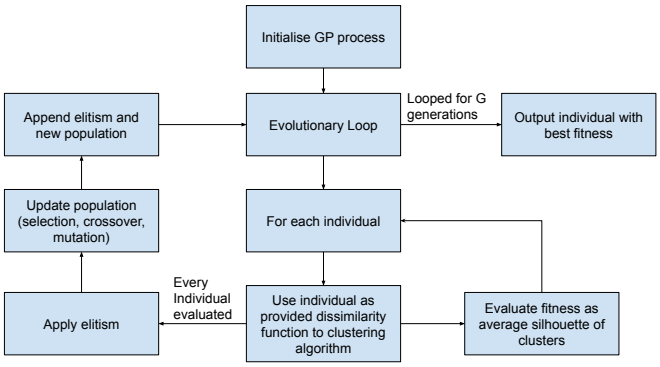

Fig. 4. The proposed method

IV. Experiment Design

The proposed method was implemented using DEAP [21], an evolutionary computation framework that is lightweight and easily extensible. The library was modified to add elitism [22], so that a given number of high performing individuals in the population kept to the next generation.

All algorithms used except for the graph clustering algorithm were implemented using the PyClustering library [23]. The naïve graph clustering was implemented by hand as no existing libraries handled the process of building cluster graphs.

A. Parameter Settings

Table I shows the parameter settings chosen for the GP process. These are standard GP parameters [22] except for the population size, which is kept slightly lower to reduce the computation time. As part of preliminary testing a number of different maximum tree depths were explored, and a maximum depth of 7 was found to give the best trade off between performance and computation time.

B. Datasets

The datasets used in this paper are sourced from two different pieces of work. These are all artificial datasets, as these allow the performance of different algorithms to be compared to known *ground truth*s. This approach is common in the clustering literature. Some of the clustering literature has used classification datasets, with the class labels removed from the clustering process. However, research has shown that this can result in misleading results, as there is no guarantee in classification tasks that classes correspond one-to-one to well-formed clusters [24].

TABLE I
Parameters settings used for GP

Parameter	Value
Population	256
Tournament Size	7
Elitism	10
Generations	100
Crossover Probability	80%
Mutation Probability	20%
Population Initialisation	Half and Half
Crossover	One Point
Mutation	Random Subtree Replacement
Max Tree Depth	7

TABLE II
Datasets used for evaluation

	HAWKS		MOCK	
Dimensions	10	20	50	100
Clusters	10	20	10	10
Instances	1000	1000	2698	2892

The first set of datasets are those used in [16]. These are generated using HAWKS [25], which uses genetic algorithms to generate datasets with clusters of dynamic shapes and sizes, targeting a given silhouette score. Each one of these datasets consists of 1000 points, with varying numbers of dimensions and clusters. Each cluster is hyper-spherical in shape, and as the dimensionality of these datasets increases, they get easier to correctly cluster.

The second set of datasets were generated using the well-known generators used in the MOCK paper [18]. These datasets are used as they create clusters that are not completely hyper-spherical, and instead are ellipsoidal in shape. This represents a difficult clustering challenge [26].

Table II shows the four datasets used in this paper.

C. Adjusted Rand Index

The Adjusted Rand Index (ARI) is an evaluation metric that compares a given set of clusters produced by a clustering method to provided 'gold standard' clusters [27]. This provides an overall measure of similarity between the clusters produced and the gold standard clusters.

The ARI is based on the Rand Index, which is an evaluation metric that compares the similarity between the two cluster groups without correcting for chance. If a true positive (TP) is taken to mean the case where two instances are in the same cluster in both groups and a true negative (TN) is taken to mean the case where two instances are in different clusters in both groups, then Equation 11 shows the calculation of the Rand Index [28].

$$RI = \frac{TP + TN}{TP + TN + FP + FN} \tag{11}$$

The ARI is a modification on the Rand index to account for random chance in pairings.

Given a clustering X and gold standard Y, a contingency table $[n_{ij}]$ is first built where each n_{ij} is equal to the number of instances in common between cluster X_i and cluster Y_j. Once this has been built, the sum of each row and column is computed as a_i and b_j. With n as the total number of instances, the ARI is calculated according to Equation 12.

$$ARI = \frac{\sum_{ij}\binom{n_{ij}}{2} - [\sum_i \binom{a_i}{2}\sum_j \binom{b_j}{2}]/\binom{n}{2}}{\frac{1}{2}[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}] - [\sum_i \binom{a_i}{2}\sum_j \binom{b_j}{2}]/\binom{n}{2}} \quad (12)$$

D. Evaluation Approach

Each pair of algorithm and dissimilarity measure is run 30 times on each dataset, with a different seed each time. To evaluate the performance of the dissimilarity evolution process with predefined dissimilarity functions for each of the clustering methods, both the ARI and silhouette are reported. The ARI is used to give a clear indication of the performance of the method according to the ground truth clusters, and the silhouette is reported to ensure the GP process is producing results with high enough fitness.

V. Results

The results gathered across the datasets are shown in Tables III and IV, where the ARI and silhouette are the means across the 30 runs. The best ARI and average silhouette for each clustering algorithm are shown in bold.

VI. Discussions and Analysis

From the results, there is a clear benefit to evolving similarity functions on different clustering algorithms. For the lower dimensional datasets, the evolved function outperforms all predefined similarity functions for k-means++. While it doesn't quite reach the ARIs obtained by the predefined functions for the remaining algorithms, the average silhouette score produced for each algorithm is either approximately the same as or better than when using predefined dissimilarity functions.

At higher dimensionality, the ARI achieved by GP is often lower than when using other dissimilarity measures. However, the silhouette on the majority of methods is generally higher than when using any of the pre-defined functions. As the silhouette metric is what the GP algorithm uses to measure fitness, this signifies that GP can optimise effectively, but that silhouette is not always a good measure of cluster quality (that is, there is a low correlation between silhouette and ARI). For example, on datasets with non-hyper-spherical clusters, the silhouette cannot measure cluster quality as accurately. The silhouette can be seen as a combined measure of both similarity and separability of clusters — in the future, it may be fruitful to split these metrics into separate objectives and evaluate the clusters using evolutionary multiobjective optimisation.

TABLE III
Mean ARI and silhouette obtained by different dissimilarity measures across the datasets

10d-10c dataset

Measure		K-means++	HDBSCAN	Agglom.	Graph
GP	ARI	**0.8952**	0.9315	**0.9523**	0.9035
	SIL	**0.7489**	**0.8206**	**0.7519**	0.7923
Euclidean	ARI	0.8402	0.9413	0.8616	**0.9437**
	SIL	0.605	0.809	0.428	**0.807**
Cosine	ARI	0.8348	0.6965	0.01493	0.7812
	SIL	0.556	0.803	-0.428	0.705
Manhattan	ARI	0.8370	**0.9432**	0.8973	**0.9437**
	SIL	0.6069	0.808	0.484	**0.807**
Checbyshev	ARI	0.8306	0.9076	0.7821	**0.9437**
	SIL	0.585	0.820	0.304	**0.807**

20d-20c dataset

Measure		K-means++	HDBSCAN	Agglom.	Graph
GP	ARI	**0.9682**	0.9914	0.4855	0.9351
	SIL	**0.7377**	0.7858	0.1392	0.7519
Euclidean	ARI	0.8854	**1.0**	**1.0**	**1.0**
	SIL	0.613	0.783	**0.783**	**0.783**
Cosine	ARI	0.9587	0.8748	0.6175	0.6143
	SIL	0.700	**0.833**	0.299	0.401
Manhattan	ARI	0.8859	**1.0**	**1.0**	**1.0**
	SIL	0.613	0.783	**0.783**	**0.783**
Checbyshev	ARI	0.8824	0.9778	0.9624	**1.0**
	SIL	0.600	0.782	0.552	**0.783**

50d-10c dataset

Measure		K-means++	HDBSCAN	Agglom.	Graph
GP	ARI	0.2040	0.0003	0.0901	0.0021
	SIL	**0.5983**	0.6559	0.3988	**0.5870**
Euclidean	ARI	0.3958	0.5886	**0.4455**	0.9985
	SIL	0.5671	0.4521	**0.4807**	0.4255
Cosine	ARI	**0.7186**	0.5415	0.1458	0.0000
	SIL	0.3227	**0.6989**	-0.3388	NA
Manhattan	ARI	0.4039	**0.5976**	0.2675	**1.0000**
	SIL	0.5683	0.3631	0.4511	0.4425
Chebyshev	ARI	0.4604	0.4407	0.3659	0.5044
	SIL	0.5170	0.4612	0.2912	-0.0313

100-10c dataset

Measure		K-means++	HDBSCAN	Agglom.	Graph
GP	ARI	0.2439	0.0005	0.0817	0.0011
	SIL	**0.6100**	**0.6768**	**0.4137**	**0.6508**
Euclidean	ARI	0.4769	**0.8668**	0.2043	0.9869
	SIL	0.5904	0.4450	0.3012	0.5807
Cosine	ARI	**0.8162**	0.8133	**0.3072**	0.8428
	SIL	0.3765	0.5465	-0.1896	0.4004
Manhattan	ARI	0.4703	0.8652	0.2013	**1.0000**
	SIL	0.5862	0.4536	0.3006	0.5731
Chebyshev	ARI	0.5221	0.7777	0.1787	0.6539
	SIL	0.6070	0.4336	0.2923	0.3414

TABLE IV
MEAN ARI AND SILHOUETTE ON THE 100-10C DATASET

100-10c dataset

Measure		K-means++	HDBSCAN	Agglom.	Graph
GP	ARI	0.2439	0.0005	0.0817	0.0011
	SIL	**0.6100**	**0.6768**	**0.4137**	**0.6508**
Euclidean	ARI	0.4769	**0.8668**	0.2043	0.9869
	SIL	0.5904	0.4450	0.3012	0.5807
Cosine	ARI	**0.8162**	0.8133	**0.3072**	0.8428
	SIL	0.3765	0.5465	-0.1896	0.4004
Manhattan	ARI	0.4703	0.8652	0.2013	**1.0000**
	SIL	0.5862	0.4536	0.3006	0.5731
Chebyshev	ARI	0.5221	0.7777	0.1787	0.6539
	SIL	0.6070	0.4336	0.2923	0.3414

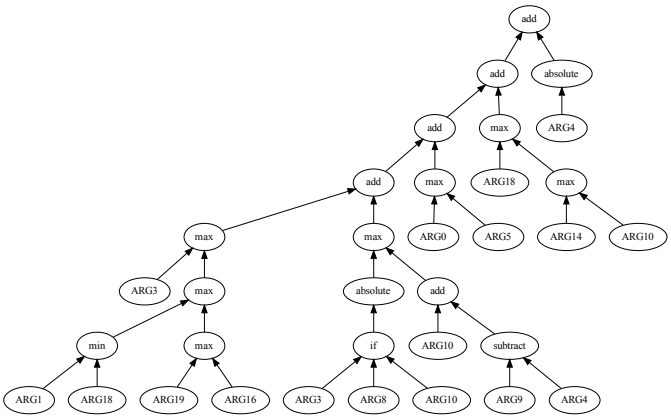

Fig. 5. Best dissimilarity function for 20d-20c dataset

A. Further Analysis

Figure 5 shows the best tree evolved for the k-means++ algorithm on the 20d-20c dataset. Notably, this tree contains very few constant values. This suggests that the dissimilarity functions are stronger when only considering the features without weighing or offsetting them. The tree also has a wide range of features, instead of using the same feature many times as leaves of the tree.

VII. FUTURE DIRECTIONS

One major finding of this work was the performance benefit of ensuring symmetry in the evolved dissimilarity functions. It follows that evolving functions that meet the other properties of a distance *metric* would further improve performance on clustering algorithms that depend on them. We consider each of these properties in turn:

1) **Non-Negativity:** It is possible to evolve a tree that will result in negative dissimilarities if a subtraction function is used and the value to the right of the function is larger. Figure 6 shows an example dissimilarity tree that will sometimes result in a positive and sometimes result in a negative value, depending on the two points. In this tree, if the distance between the second feature on the two instances is larger than the distance between the fourth feature on the two instances then the resulting dissimilarity will be negative.

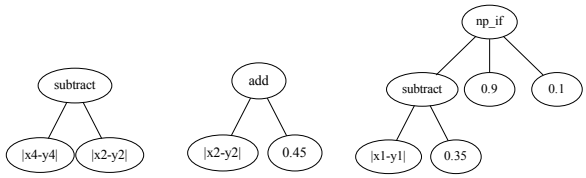

Fig. 6. Three sample trees that break the non-negativity, identity, and triangle inequality properties respectively

2) **Identity:** It is possible to evolve a tree where the dissimilarity from a point to itself will not be 0. This happens in the case where a constant is carried up to the top level of the function evaluation in the tree. Figure 6 shows an example dissimilarity tree that will always break the identity property. If the two instances being compared are the same then the distance between x_2 and y_2 will be 0. This means that the resulting dissimilarity will be 0.45 when the instances are the same.
This way of breaking the property, however, is not an issue for most clustering algorithms as all possible dissimilarities will be shifted by the same amount. This, in essence, means that it can be treated as a constant offset and ignored.
3) **Symmetry:** This property is correctly preserved by the evolved functions. As the terminal set is taken as the absolute difference between the two points, the terminal set from A to B and the terminal set from B to A will be the same.
4) **Triangle Inequality:** It is possible to evolve a tree for which the triangle inequality will not hold if the logical operators such as if, min, and max are used. For example, Figure 6 presents a very simple tree for which the triangle inequality will not hold due to the presence of the if operator. As an example, take 3 instances a, b, and c where the values for the first feature are 0.1, 0.4, and 0.5 respectively. Remembering that the triangle inequality requires that $d(a, c) \leq d(a, b) + d(b, c)$, we can see that this does not hold as $0.9 > 0.1 + 0.1$.

We expect that future work addressing each of these properties will further improve the feasibility of evolving dissimilarity functions with GP.

VIII. CONCLUSIONS

This work presents a novel GP-based framework that can evolve dissimilarity functions for a variety of different clustering algorithms. This can address the main drawbacks of existing dissimilarity functions. Experimental results show that the evolved dissimilarity functions can create clusters that demonstrate higher measures of clustering quality than clusters

created using predefined dissimilarity functions, particularly on lower-dimensional datasets.

This is one of the few works on evolving dissimilarity functions in clustering. As discussed in Section VII, several directions should be further explored. We will refine this approach further to better satisfy the properties of a distance metric. We also plan to adapt the fitness function to be less dependent on the datasets satisfying the assumptions of the Silhouette metric.

REFERENCES

[1] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, 2nd ed. Springer series in statistics, 2 2009.

[2] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Prentice Hall, 2009.

[3] J. Williams and Y. Li, "Comparative study of distance functions for nearest neighbors," 01 2008, pp. 79–84.

[4] N. Bouhmala, "How good is the euclidean distance metric for the clustering problem," in *5th IIAI International Congress on Advanced Applied Informatics, IIAI-AAI 2016, Kumamoto, Japan, July 10-14, 2016*. IEEE Computer Society, 2016, pp. 312–315.

[5] A. Lensen, B. Xue, and M. Zhang, "Genetic programming for evolving similarity functions for clustering: Representations and analysis," *Evolutionary Computation*, 2019.

[6] M. Willis, H. Hiden, P. Marenbach, B. McKay, and G. Montague, "Genetic programming: An introduction and survey of applications," 10 1997, pp. 314 – 319.

[7] H. Al-Sahaf, Y. Bi, Q. Chen, A. Lensen, Y. Mei, Y. Sun, B. Tran, B. Xue, and M. Zhang, "A survey on evolutionary machine learning," *Journal of the Royal Society of New Zealand*, vol. 49, no. 2, pp. 205–228, 2019.

[8] B. Tran, B. Xue, and M. Zhang, "Genetic programming for multiple-feature construction on high-dimensional classification," *Pattern Recognition*, vol. 93, pp. 404–417, 2019.

[9] E. H. Ruspini, J. C. Bezdek, and J. M. Keller, "Fuzzy clustering: A historical perspective," *IEEE Computational Intelligence Magazine*, vol. 14, no. 1, pp. 45–55, 2019.

[10] D. Xu and Y. Tian, "A comprehensive survey of clustering algorithms," *Annals of Data Science*, vol. 2, 08 2015.

[11] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '07. USA: Society for Industrial and Applied Mathematics, 2007, p. 1027–1035.

[12] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*. USA: Prentice-Hall, Inc., 1988.

[13] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "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*, ser. KDD'96. AAAI Press, 1996, p. 226–231.

[14] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992.

[15] A. Lensen, B. Xue, and M. Zhang, "Gpgc: Genetic programming for automatic clustering using a flexible non-hyper-spherical graph-based approach," 07 2017.

[16] F. Schofield and A. Lensen, "Evolving simpler constructed features for clustering problems with genetic programming," in *IEEE Congress on Evolutionary Computation, CEC 2020, Glasgow, United Kingdom, July 19-24, 2020*. IEEE, 2020, pp. 1–8.

[17] N. Boric and P. A. Estévez, "Genetic programming-based clustering using an information theoretic fitness measure," in *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2007, 25-28 September 2007, Singapore*. IEEE, 2007, pp. 31–38.

[18] J. Handl and J. Knowles, "An evolutionary approach to multiobjective clustering," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 1, pp. 56–76, 2007.

[19] M. Balcan, A. Blum, and S. S. Vempala, "A discriminative framework for clustering via similarity functions," in *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, C. Dwork, Ed. ACM, 2008, pp. 671–680.

[20] P. Rousseeuw, "Rousseeuw, p.j.: Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. comput. appl. math. 20, 53-65," *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, 11 1987.

[21] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, 7 2012.

[22] R. Poli, W. Langdon, and N. Mcphee, *A Field Guide to Genetic Programming*, 01 2008.

[23] A. Novikov, "PyClustering: Data mining library," *Journal of Open Source Software*, vol. 4, no. 36, p. 1230, 4 2019.

[24] U. von Luxburg, R. C. Williamson, and I. Guyon, "Clustering: Science or art?" ser. Proceedings of Machine Learning Research, I. Guyon, G. Dror, V. Lemaire, G. Taylor, and D. Silver, Eds., vol. 27. Bellevue, Washington, USA: JMLR Workshop and Conference Proceedings, 7 2012, pp. 65–79.

[25] C. Shand, R. Allmendinger, J. Handl, A. Webb, and J. Keane, "Evolving controllably difficult datasets for clustering," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 463–471.

[26] A. Lensen, B. Xue, and M. Zhang, "Genetic programming for evolving similarity functions for clustering: Representations and analysis," *Evolutionary Computation*, pp. 1–29, 10 2019.

[27] N. X. Vinh, J. Epps, and J. Bailey, "Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance," *Journal of Machine Learning Research*, vol. 11, no. 95, pp. 2837–2854, 2010.

[28] L. Hubert and P. Arabie, "Comparing partitions," *Journal of Classification*, vol. 2, no. 1, pp. 193–218, 12 1985.