VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui

School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

Evolving Clustering Similarity Functions

Hayden Andersen

Supervisors: Andrew Lensen, Bing Xue

Submitted in partial fulfilment of the requirements for
Bachelor of Science with Honours in Computer Science.

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 similarity functions such as Euclidean Distance. However, most predefined similarity functions, including these simple distance-based 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, a process that naturally lends itself to the evolution of functions. This project 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.

Acknowledgements

I would like to thank my supervisors Dr Andrew Lensen and Associate Professor Bing Xue for their constant support and guidance. Andrew has always been willing to lend an ear for ideas, bounce suggestions back at me, and has helped me structure my life in such a way that I didn't struggle anywhere near as much as I expected at the beginning of the year. Bing has been a massive font of knowledge and experience, guiding me along the long path of honours and ensuring I don't get too off track with offshoot ideas.

I would like to thank my friends and family for supporting me through this endeavour, helping to keep me motivated and energised. A special shout out to Nick, Greg, and Abby for bearing my constant complaining about workload and providing suggestions to improve my report.

A final special thanks to my partner AJ for somehow managing to put up with my long nights, bad moods, and stress throughout this project. I couldn't have done this without her support.

Contents

Figures

Chapter 1

Introduction

Clustering is a form of machine learning that falls under the category of unsupervised learning [1], grouping similar instances of unlabelled data into distinct subsets known as clusters. Unsupervised learning is where a model is created without any explicit feedback based on the label information present in the data [2]. Many clustering algorithms employ some form of predefined dissimilarity function to evaluate how different instances are, to see if they should be grouped together. One common category of dissimilarity functions are 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 the 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 can be very important as a feature, while day of the week is likely to have no influence on the patterns themselves. These distance measures can also be inflexible, as they use a pre-defined 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 (EA) method that evolves programs, 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.

This project explores using GP to evolve new dissimilarity functions for clustering problems, using the features of the two points being compared as input and producing an output that represents how dissimilar the two points 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 these in the function. This provides a form of feature selection, as not all features will be selected for use in the dissimilarity functions. 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. These tailored functions will allow clustering algorithms to create stronger clusters than the simple pre-defined functions.

Previous research [5] has applied the concept of using GP for clustering algorithms utilising graph theory, with 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.

1.1 Motivations

While previous research has shown that GP can evolve high performance dissimilarity functions, it is limited to graph based clustering [5]. However, the concept of evolving the functions can be extracted from the graph based clustering, and it should be able to be extended to a framework that can evolve a dissimilarity for any provided clustering algorithm.

In addition to this, the previous research employs a rather naive view of the GP process. First, they employ a fitness function that is a combination of what they describe as three competing metrics - multiobjective optimisation could be used to ensure that all three of these metrics are optimised at the same time. Also, the research uses a naive terminal set that is simply made of the base level features of the instances. This makes it possible for the algorithm to learn dissimilarities that are fully evaluated based on the features of only one of the points.

1.2 Goals

There are two major goals to this project:

1. Create and evaluate a framework that can be used to evolve dissimilarity functions for an arbitrarily provided dataset and clustering algorithm. This framework can interface with existing clustering software tools. This expands upon the idea introduced by [7] by removing the reliance on graph-based clustering and instead provides a more general approach. The framework can be provided any clustering algorithm that utilises a dissimilarity function, and will evolve custom tailored functions for that algorithm.

2. Based on the evaluation of the initial framework, extensions should be made to provide better performance. The extensions explored will be:

    - Apply transfer learning between different clustering algorithms to fine-tune a dissimilarity function evolved for a seperate algorithm;
    - Constrain the GP process so that it spends less time evaluating solutions in dead space;
    - Perform multiobjective optimisation to allow for better tradeoffs between metrics of cluster quality.

The dissimilarity functions evolved in this project will be evaluated against a selection of commonly used pre-defined functions.

1.3 Contributions

This project presents a novel GP based framework for evolving dissimilarity functions. These dissimilarity functions create clusters that demonstrate higher measures of clustering quality than clusters created using pre-defined dissimilarity functions. This is then improved upon again with multiobjective optimisation, evolving fronts of dissimilarity functions containing individuals with a high similarity to the gold standard clusters of the datasets.

Chapter 2

Background

2.1 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 [8], or hierarchies of clusters (clusters of clusters) can be created [1]. However, this project focuses on the simplest form of clustering, where each individual belongs to at most one cluster. The clustering algorithms explored in this project are discussed in Section 2.3.

2.2 Similarity and Distance Functions

Similarity functions are used in clustering to describe how similar two given instances (or points) are to each other. The higher the result of the similarity function when applied on two points, the more similar those points are [9]. The opposite of a similarity function is a *dissimilarity* function, which will be smaller the more similar the two instances are [10].

Most dissimilarity functions used in clustering are distance functions. These represent the distance between the instances in the feature space [3]. While there are a wide range of available dissimilarity functions through the literature, this research will focus on some of the most commonly used ones. These are the Euclidean, Manhattan, and Chebyshev distance functions [11]. The research will also evaluate against the cosine distance measure - a simple conversion from cosine similarity, the most commonly used non-distance similarity measure [12].

There are four properties that a full distance function must satisfy [9]:

1. **Non-Negativity**
   $d(x, y) \geq 0$
2. **Identity**
   $d(x, y) = 0 \iff x = y$
3. **Symmetry**
   $d(x, y) = d(y, x)$
4. **Triangle Inequality**
   $d(x, z) \leq d(x, y) + d(y, z)$

A similarity function must satisfy properties 1-3, but is not required to ensure the triangle inequality property [9].

2.2.1 Euclidean Distance

Euclidean distance is the most commonly used distance function in clustering [4]. It is used to measure the straight line (as the crow flies) distance between two points. The distance is defined by Equation 2.1 where $dim(I)$ is the number of dimensions in a single instance and F_{1i} and and F_{2i} are the ith features of the 1st and 2nd instances being compared [3]. Many clustering algorithms will use squared Euclidean distance as a metric rather than Euclidean distance in situations that only a ranking of the closeness of instances is required [13], as this removes the need to calculate an expensive square root and provides the same rankings.

$$D = \sqrt{\sum_{i=1}^{dim(I)} (F_{1i} - F_{2i})^2} \tag{2.1}$$

2.2.2 Minkowski Distance

Minkowski distance is a generalisation of Euclidean distance, raising each difference by a given power h and then taking the h root of the sum, where h is a real number ≥ 1. The absolute value of the differences of each feature is taken so that the symmetry property is preserved. The distance is defined by Equation 2.2 where $dim(I)$ is the number of dimensions in a single instance and F_{1i} and and F_{2i} are the ith features of the 1st and 2nd instances being compared [11]. Euclidean distance gives the Minkowski distance when $h = 2$. A Minkowski distance is also known as the L_h norm. As such, Euclidean distance is known as the L_2 norm.

$$D = \sqrt[h]{\sum_{i=1}^{dim(I)} |F_{1i} - F_{2i}|^h} \tag{2.2}$$

2.2.3 Manhattan Distance

Manhattan distance measures the distance between two points as if a grid pattern was followed, travelling at right angles along each axis. Because of the pattern of the distance function, it is sometimes referred to as "city block" distance. Manhattan distance gives the Minkowski distance when $h = 1$. The distance is defined by Equation 2.3 where $dim(I)$ is the number of dimensions in a single instance and F_{1i} and and F_{2i} are the ith features of the 1st and 2nd instances being compared [3]. Manhattan distance is also known as the L_1 norm.

$$D = \sum_{i=1}^{dim(I)} |F_{1i} - F_{2i}| \tag{2.3}$$

2.2.4 Chebyshev Distance

Chebyshev distance (also known as the supremum distance) is the maximum distance along a single axis of the two points. It can be seen as the Minkowski distance when $h \to \infty$. As such, it is sometimes referred to as the L_∞ norm. It is defined by Equation 2.4 where $dim(I)$ is the number of dimensions in a single instance and F_{1i} and and F_{2i} are the ith features of the 1st and 2nd instances being compared [11].

$$D = \lim_{h \to \infty} \sqrt[h]{\sum_{i=1}^{dim(I)} |F_{1i} - F_{2i}|^h} = max_{i=1}^{dim(I)}(|F_{1i} - F_{2i}|) \tag{2.4}$$

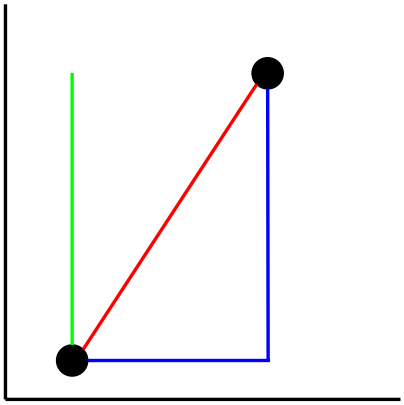


Figure 2.1: Visual representation of each Minowski distance. Red is Euclidean, blue is Manhattan, green is Chebyshev

Figure 2.2.4 shows the relationship between the three Minkowski distances. The red line represents the Euclidean distance, the blue line represents the Manhattan distance, and the green line represents the Chebyshev distance.

2.2.5 Cosine Similarity

Cosine Similarity is a widely implemented metric that measures the similarity between two points by the cosine of the angle between them in the feature space. It is defined by Equation 2.5 where $dim(I)$ is the number of dimensions in a single instance and F_{ni} is the ith feature of the nth instance [12]. As this gives a similarity metric in the range [-1, 1] where a similarity of 1 represents the highest possible similarity, we can easily convert it to a dissimilarity function using Equation 2.6 The dissimilarity version of the metric remains in the range [-1, 1].

$$S = \frac{\sum_{i=1}^{dim(I)} F_{1i}F_{2i}}{\sqrt{\sum_{i=1}^{dim(I)} F_{1i}}\sqrt{\sum_{i=1}^{dim(I)} F_{2i}}} \tag{2.5}$$

$$D = 1 - S \tag{2.6}$$

2.3 Clustering Methods

There are a vast range of clustering methods, that can broadly be split into several different categories [14]. For the purposes of this research, four of these categories will be evaluated against. They are as follows:

1. **Partitional clustering**, which treats the centre of the data points in a cluster as the centre of that cluster, and iteratively updates these cluster centres until the algorithm converges [14]. An example of this is the k-means algorithm [14].

   The k-means algorithm is a partitional clustering algorithm that first randomly selects k random instances on a uniform distribution from the dataset, and sets them to be the centre of each cluster. It then adds each instance to the cluster with the nearest centre based on the dissimilarity function. The centres are then updated to be the mean of

all the values in the cluster, and the algorithm reassigns clusters again. This continues until the cluster partitions no longer change [15].

One big issue with the k-means algorithm is that it is heavily reliant on the initial clusters chosen in order to reach a good convergence as it very easily falls into a local optimum. One proposed method to fix this problem, known as k-means++, changes the uniform distribution of initial cluster centre selection such that it is more informed by the distribution of the data. To do this, it first selects a random initial centre. For the remaining $k-1$ centres, it randomly selects an instance x' from the dataset X with probability given by Equation 2.7, where $D(x)$ is the shortest Euclidean distance from instance x to any of the already chosen centres [16]. This weights the probability of selection such that the distance between cluster centres will be maximised.

$$p(x') = \frac{D(x')^2}{\sum_{x \in X} D(x)^2} \tag{2.7}$$

2. **Hierarchy based clustering**, which constructs clusters based on the hierarchical relationship between clusters created when either combining clusters together from single instance clusters, or splitting from a single cluster containing every instance [14]. An example of this is agglomerative clustering [14].

   Agglomerative clustering is a hierarchical clustering method that initially starts with each instance in its own cluster, and then merges these clusters until the required number of clusters are formed. While there are many different 'linkage criteria' for agglomerative clustering, the main one focused on in this project is single-linkage clustering, which at each step will combine the two clusters with the most similar instances together into one combined cluster, where similarity is calculated by the chosen dissimilarity method [17]. Other linkage methods are often used. For complete linkage the two most similar clusters are combined, where cluster similarity is evaluated by the similarity of the two least similar instances between the clusters. For average linkage the two most similar clusters are combined, where cluster similarity is evaluated by the similarity between the centre of the clusters. These linkage methods are more likely to follow the intuitive notion of what a cluster should be, but single linkage clustering is much faster to compute [18] which is an important factor to consider in a wrapper based metric such as the one presented in this project [19].

3. **Density based clustering**, which works off of the idea that the data which is in a region with a high density is to be considered as belonging to the same cluster [14]. An example of this is the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm [14].

   DBSCAN is a density based clustering algorithm that takes two parameters ϵ and $MinPts$. A cluster is created by first connecting together each instance that has at least $MinPts$ other instances within ϵ similarity based on the provided dissimilarity function. These are marked as core points. Next, all remaining instances within ϵ of the core points are marked as border points. The cluster is then defined as the set of connected core and border points. This is repeated across the entire dataset, and any remaining unclustered instances are marked as noise. This allows DBSCAN to automatically discover the number of clusters, as the clusters are simply all the different densely connected groups [20].

4. **Clustering based on graph theory**, where each node is an instance in the data and the edges represent the relationships between the instances [14]. An example of this is the Highly Connected Subgraphs (HCS) algorithm [14].

A naive graph based clustering algorithm is to simply connect each instance node to its k-nearest neighbours using a given dissimilarity function. The clusterings are then taken as each distinct unconnected subgraph. This is a very simple algorithm, but can then be used as the basis for more complex graph based clustering methods [5].

One such clustering method is the HCS algorithm. Once the initial graph has been built, this algorithm calculates the "minimum cut", the minimum number of edges that can be removed from the graph to create two distinct graphs. If the number of edges in the minimum cut are more than half the number of edges in the graph then the graph is considered highly connected, and becomes a cluster. If the graph is not highly connected then the graph is split along the minimum cut, and the algorithm is run recursively on the two subgraphs created. Once all subgraphs have been marked as highly connected, they are returned as the generated clusterings [21].

2.3.1 OPTICS and HDBSCAN

An issue with DBSCAN is that it requires the ϵ parameter to be finely tuned to the scale of the dataset and dissimilarity function [22]. Because this can not be pre-set for arbitrarily defined dissimilarity functions such as those evolved in this project, two different density based clustering algorithms are used instead — OPTICS and HDBSCAN.

Ordering Points To Identify the Clustering Structure (OPTICS) is a density based clustering method which works as an extended method of the DBSCAN algorithm that checks an infinite number of ϵ_i such that $0 \leq \epsilon_i \leq \epsilon$. This works by assigning each instance a core-distance representing its *MinPts*' closest point when there are at least *MinPts* instances within ϵ radius. The reachability-distance from that instance to another instance is then defined as the maximum of the similarity between the two points or the core-distance. This represents the smallest distance such that the two instances are reachable from each other if one of them were to be a core point. The clusters are then generated based on "troughs" in a graph of the reachability-distance [22].

Hierarchical DBSCAN (HDBSCAN) is another density based clustering algorithm that extends the DBSCAN algorithm by converting it into a hierarchical clustering algorithm and then extracting clusters based on their stability. It does this by first assigning each instance a core-distance and calculates the reachability-distance in the same way as OPTICS, except it removes the need for the ϵ value. These reachability-distance values are then used to build a hierarchical single linkage graph, which is then split into the resulting clusterings [23].

2.4 Cluster Evaluation Metrics

Clustering performance metrics tend to measure at least one of three categories:

- Compactness, a measure of intra-cluster dissimilarity between each instance in that cluster. A common clustering method that utilises compactness as a metric is k-means [24]. The inverse of compactness is known as sparsity, with higher quality clusters exhibiting a lower sparsity [5].
- Separability, a measure of the inter-cluster dissimilarity between each cluster. This is not often used on its own as a metric, and is more often utilised along with one of the other two categories [24].
- Connectedness, the idea that neighbouring instances should be placed into the same cluster as each other. A common clustering method that utilises connectedness as a metric is DBSCAN [24].

These metric categories are general descriptors of cluster quality, and there are many ways to formulate each one of them.

2.4.1 Silhouette Function

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 [25]. While originally created as an aid for graphical representations of clusters, it is a good general measure of the quality of a clustering solution. One issue with using the silhouette as a metric, however, is that it operates under the assumption of spherical clusters. The silhouette of an instance in the solution is defined by Equation 2.8 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 definition of a_i and b_i are given by Equations 2.9 and 2.10 respectively where C_i is the cluster containing instance i and $d(i, j)$ is the Euclidean distance from instance i to instance j [25].

Equation 2.9 gives a measure of how similar an instance is to instances of its own cluster and Equation 2.10 gives a measure of how similar an instance is to instances of different clusters. Equation 2.8 then takes the difference between these values.

$$s_i = \frac{b_i - a_i}{max(a_i, b_i)} \tag{2.8}$$

$$a_i = \frac{1}{|C_i| - 1} \sum_{j \in C_i, i \neq j} d(i, j) \tag{2.9}$$

$$b_i = \min_{k \neq i} \frac{1}{C_k} \sum_{j \in C_k} d(i, j) \tag{2.10}$$

2.4.2 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 [26]. 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 2.11 shows how the Rand index is calculated [27].

$$RI = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.11}$$

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 2.12.

$$ARI = \frac{\sum_{ij} \binom{n_{ij}}{2} - [\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}] / \binom{n}{2}}{\frac{1}{2}[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}] - [\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}] / \binom{n}{2}} \tag{2.12}$$

2.5 Genetic Programming

Genetic Programming (GP) is a form of evolutionary algorithm. Evolutionary Algorithms (EA) are algorithms that simulate aspects of the natural world, usually through a simulation of some sort of population, to produce solutions to problems [28]. GP is represented by a population of randomly generated programs made up of functions and terminals. Functions take 1 or more inputs and return an output, and can be arithmetic operations, programmatic operations, domain specific functions, or more [29]. 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 criteria is reached. There are a number of considerations required in applying GP [6], [29]:

- Determining the set of terminals;
- Determining the set of functions;
- Determining the fitness function;
- Determining hyperparameters for controlling the run, e.g. population size, number of generations, individual size restraints, elitism amount, etc.
- Determining the method of designating a result from the population;
- Determining Initialisation, Crossover, and Mutation to use.

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 in the tree [6].

2.6 Evolutionary Multiobjective Optimisation

Often in complex optimisation problems an issue is encountered where it is desired to optimise for multiple objectives at once, but these objectives are conflicting. Multiobjective optimisation is the process of creating optimal solutions for these problems [30]. Because the objectives are conflicting, evaluating potential solutions is not as easy as simple single objective problems, where a single value can be compared. This is because as the performance of one objective improves the performance other objectives will perform degrade, so an optimal balance between the objectives is desired [30]. Evolutionary Multiobjective Optimisation (EMO) is the process of performing multiobjective optimisation using evolutionary algorithms such as GP. These are well suited to the task of multiobjective optimisation as the population based behaviour of EAs allows for easy derivation of a group of best solutions, rather than just a single solution [31]. This group of best solutions is know as a Pareto front.

A solution with objectives x is said to dominate another solution with objectives y if Equation 2.13 holds [32]. This equation states that a solution dominates another if it is not worse in any objective, but is better in at least one.

$$\forall i : x_i \leq y_i \wedge \exists j : x_j < y_j \tag{2.13}$$

Note that this equation is for the case where all objectives are minimisation problems. In cases where maximisation is required, the direction of the $<$ operators would be reversed.

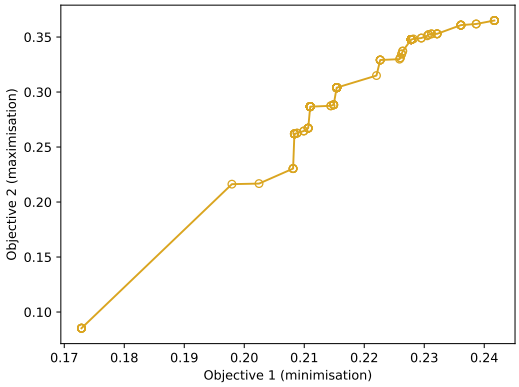


Figure 2.2: Sample evolved front from project

A solution that is not dominated by any other solutions is known as a Pareto optimal solution. The Pareto front is then defined as the set of all Pareto optimal solutions.

Figure 2.6 shows an example front evolved during this project, using one minimisation and one maximisation objective.

2.6.1 Hypervolume

There are a number of different metrics for evaluating the quality of a Pareto front. One of the most commonly used metrics is the hypervolume [33]. The hypervolume is a simple metric that calculates the volume inside the Pareto front. A higher hypervolume represents a more widely distributed Pareto front, which is regarded as a stronger front [33]. For a multiobjective problem with only two objectives such as the one in this project, the hypervolume can also be seen as the area behind the Pareto front.

2.6.2 NSGA-II

The EMO framework used in this project is the NSGA-II algorithm. NSGA-II is a widely used and well known algorithm for performing EMO that first sorts the solutions into ordered ranks of fronts, where each rank dominates all solutions in the next rank but is dominated by all solutions in the previous rank. Solutions in the same rank are then sorted based on a concept of crowding distance. The crowding distance is calculated by creating a cuboid with corners at the two nearest solutions, and then defining the distance as the average length of the sides of this cuboid.

When using NSGA-II to select a given number of solutions, individuals are taken from each rank in order, until a rank contains more solutions than are needed to create the desired selection size. At this point, solutions are taken from that rank in order of crowding distance, where the solutions with the greatest crowding distance are selected first [34].

2.7 Related Work

The most related work to this project (as well as the inspiration for this project) is the Genetic Programming Graph-based Clustering (GPGC) algorithm [5], [7]. 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. As in the naive

graph based method and HCS, the resulting clusterings are the different distinct subgraphs. The fitness function used in GPGC is a combination of sparsity (intra cluster distance), separation (inter cluster distance), and connectedness (a measure of how well points are clustered with other nearby points) metrics [7]. One important improvement made to GPGC was the introduction of a multi-tree approach, where each GP individual consisted of multiple smaller trees as opposed to a single larger tree. This was found to significantly improve performance results on a variety of datasets [5].

A similar piece of work to the idea presented in this project was recently published [35]. This paper evolved constructed features for use in the clustering domain, working as a wrapper method around the k-means++ algorithm. While the author found some great 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.

A well regarded paper for applying EAs to the clustering domain introduces an algorithm known as Multiobjective Clustering with Automatic K-determination (MOCK). The MOCK algorithm is split into two parts. First, an optimal 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 [36]. While this paper is similar to this project in that EAs are applied to multiobjective clustering, MOCK directly treats the clusters as the representation in the algorithm. In contrast, this project evolves a dissimilarity function, which is the used to create clusters using a wrapper style algorithm. This paper also introduced a simple algorithm for creating artificial clustering datasets with ellipsoidal clusters and arbitrary cluster orientations.

There is very little work in the literature on using GP for clustering. One proposed method uses multitree GP to evolve a set of "membership functions", where each tree corresponds to a specific cluster. Each instance is then placed into the cluster for which there is the highest output from the function tree that corresponds to that cluster [37]. One potential issue with this method is that it is unlikely to scale well as the number of clusters increases, as there is a tree evolved in every GP individual for every cluster. This is another algorithm applying GP to the clustering domain, however, it directly evolves the clustering solution while the one proposed here evolves the dissimilarity functions for use in the clustering solutions.

This is not the first research focused on the concept of evaluating similarity functions for the clustering domain. Previous research [38] has 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 some interesting theoretical underpinnings, there is no practical evaluation to prove that the theoretical concepts suggested are correct. This project 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.

2.8 Datasets

The datasets used in this project are sourced from two different pieces of work. These are all artificial datasets, as ground truth clusters are required to evaluate clusters using the ARI and it is very difficult to find real-world datasets for clustering that provide ground truth clusters. It has been suggested to use classification datasets and remove the class labels, but research has shown that this can result in poor quality datasets for clustering as there is no requirement in classification that classes correspond to well formed clusters [39].

The first set of datasets are those used in [35]. These are generated using HAWKS [40],

Table 2.1: Datasets used for evaluation

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

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 ground truth clusters. These generated datasets are rather simple, with each cluster being spherical in shape and easy to cluster according to the ground truth clusters. As the dimensionality of these datasets increase, they get easier to correctly cluster.

The second set of datasets are originally created for [41]. These datasets were generated using the well known generators used in the MOCK paper [36]. These datasets are used as they create clusters that are not completely spherical, and instead are ellipsoidal in shape. This presents a much more difficult clustering challenge. Additionally, these datasets have a higher number of dimensions than the previous set.

Table 2.1 shows the four datasets used in the evaluation shown of the framework.

Chapter 3

Initial Design

Work on the project began by creating and testing a framework for evolving dissimilarity functions for a number of different clustering algorithms.

Previous work [5] has created methods for evolving similarity functions, but the method to do so is intrinsically linked with the method of clustering performed. In order to decouple the evolution from the clustering, a framework must be created that allows a clustering method to be set as chosen by the user.

3.1 Proposed Method

The main idea of the proposed method is to evolve a population of dissimilarity functions that can be used in the place of general distance metrics in pre-existing clustering algorithms. The algorithms chosen are K-means++, HDBSCAN, OPTICS, Graph Clustering, and Agglomerative single link clustering. DBSCAN was considered, but the difficulty of choosing a good ϵ value made the algorithm difficult to work with when combined with dissimilarity functions of different scales. This range of algorithms gives at least one algorithm from each of the large categories of clustering methods, and allows for the flexibility of the system to be tested.

3.1.1 Representation

The evolved similarity functions are represented by GP trees, where the leaves are taken from the terminal set and internal nodes are taken from the function set. An example evolved tree that demonstrated good performance is shown in Figure 3.1. This particular tree was chosen as it provided reasonably strong results and has a smaller representation than most of the evolved trees. The trees are evaluated recursively in a top down manner, such that the input of each function is the evaluation of its children.

3.1.2 Terminal Set

As the evolved trees are used to evaluate the dissimilarity between two instances, they need to take both instances as input. Initial testing was performed with a simple terminal set consisting of every individual feature on both points. Mathematically, the terminal set T was defined by Equation 3.1, where $dim(I)$ is the dimensionality (number of features) of the data.

$$T = \{x_i | i \in \{1, 2, .., dim(I)\}\} \cup \{y_i | i \in \{1, 2, .., dim(I)\}\} \tag{3.1}$$

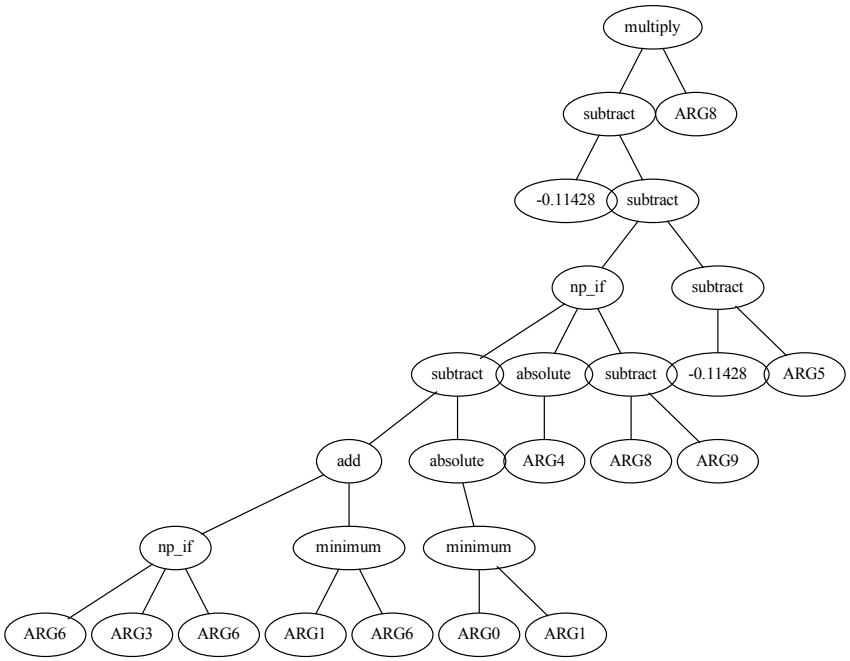


Figure 3.1: Sample GP Representation of Similarity

Initial testing using this terminal set produced very poor results, with ARIs well below the pre-existing similarities. This will be discussed further in Section 3.3 and Section 3.4. Another issue with this terminal set was that it didn't ensure the similarities would be symmetrical. For example, the simple similarity functions shown in Figure 3.2 would result in a different similarity between A and B than they would get from B and A.

Given these trees, imagine 2 instances with 5 dimensions, $a = [-0.4, -0.1, 0.3, 0.8, 0.6]$ and $b = [-0.1, 0.8, 0.9, 0.7, -0.5]$. Equations 3.2 and 3.3 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.

$$\begin{aligned} d(a,b) &= 0.3 + 0.7 = 1.1 \\ d(b,a) &= 0.9 + 0.8 = 1.7 \end{aligned} \tag{3.2}$$

$$\begin{aligned} d(a,b) &= if(-0.1)then(0.3)else(-0.5) = -0.5 \\ d(b,a) &= if(0.8)then(0.9)else(0.6) = 0.9 \end{aligned} \tag{3.3}$$

In order 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 highly assumed in all of the pre-existing similarity functions, as they will compare the same feature from each point. In order 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 3.4, where $dim(I)$ is the dimensionality (number of features) of the data.

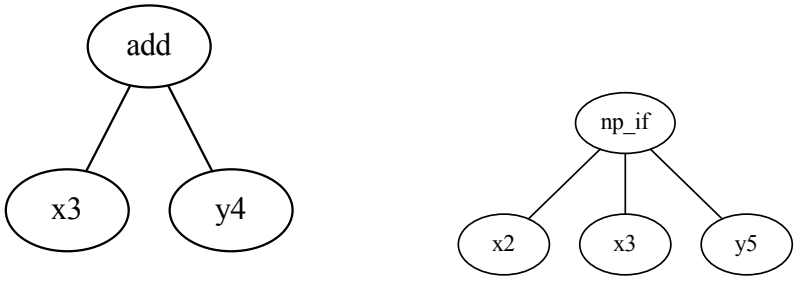


Figure 3.2: Example trees using naive terminal set

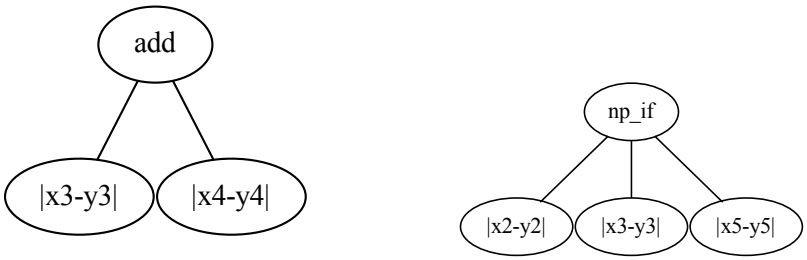


Figure 3.3: Example trees using improved terminal set

$$T = \{|x_i - y_i||i \in \{1, 2, .., dim(I)\}\} \tag{3.4}$$

The use of the absolute values in the definition of the terminal set ensures that the symmetry is preserved in the resulting dissimilarity functions. Figure 3.3 shows a similar tree to figure 3.2, 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 3.5 and 3.6 show the resulting dissimilarities calculated for these instances. Now the symmetry property is preserved.

$$\begin{aligned} d(a,b) &= |0.3 - 0.9| + |0.8 - 0.7| = 0.6 + 0.1 = 0.7 \\ d(b,a) &= |0.9 - 0.3| + |0.7 - 0.8| = 0.6 + 0.1 = 0.7 \end{aligned} \tag{3.5}$$

$$\begin{aligned} d(a,b) &= if(|-0.1 - 0.8|)then(|0.3 - 0.9|)else(|0.6 - -0.5|) = if(0.9)then(0.6)else(1.1) = 0.6 \\ d(b,a) &= if(|0.8 - -0.1|)then(|0.9 - 0.3|)else(|-0.5 - 0.6|) = if(0.9)then(0.6)else(1.1) = 0.6 \end{aligned} \tag{3.6}$$

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

3.1.3 Function Set

The function set is defined to be the usual arithmetic functions $\{+, -, \times, \div\}$ where $\div$ refers to protected division ($x \div 0 \equiv 1$), 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 positive, or the third input if the first input is negative. These operators are based on those used in previous research [5].

3.1.4 Fitness Evaluation

The fitness of the individuals is calculated as a wrapper around a supplied clustering algorithm. The evolved 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 3.7, where s_i refers to the silhouette of instance i, calculated according to Equation 2.8.

$$f = \frac{1}{|I|} \sum_{i=1}^{|I|} s_i \tag{3.7}$$

This is then returned as the overall fitness of the GP individual — we wish to maximise this fitness through the GP process.

3.2 Experiment Design

The proposed method has been implemented using DEAP [42], an evolutionary computation framework that is lightweight and easily extensible, allowing for future modifications to be made without many issues. A few modifications, however, have been made to the framework in order to make a more standard GP process [19] and to follow the example of previous research [7]. The main modification is that elitism has been added to the selection process so that a given number of individuals in the population move on to the next generation.

A diagram of the algorithm is shown in Figure 3.4. First, a random population of solutions is generated. Then, every individual in the population is evaluated according to Equation 3.7. A chosen number of the best solutions are put aside for elitism. Then, a child population is generated through selection, mutation, and crossover. Finally, the child population is appended to the best solutions that were put aside to form the new population. This process continues until a pre-set number of generations have passed, at which point the individual with the best fitness is returned.

3.2.1 Algorithm Implementations

All of the algorithms used except for the graph clustering algorithm were implemented using the PyClustering library [43]. In order to speed up the evaluation, the algorithms from the library were slightly modified to take the vectorized similarity rather than the inbuilt loops. Naive graph clustering was implemented by hand as no existing libraries handled the process of building cluster graphs.

During evaluation the evolved dissimilarity is used as a metric inside the K-means++ and agglomerative algorithms, and is used to build a pairwise dissimilarity matrix for the remaining algorithms.

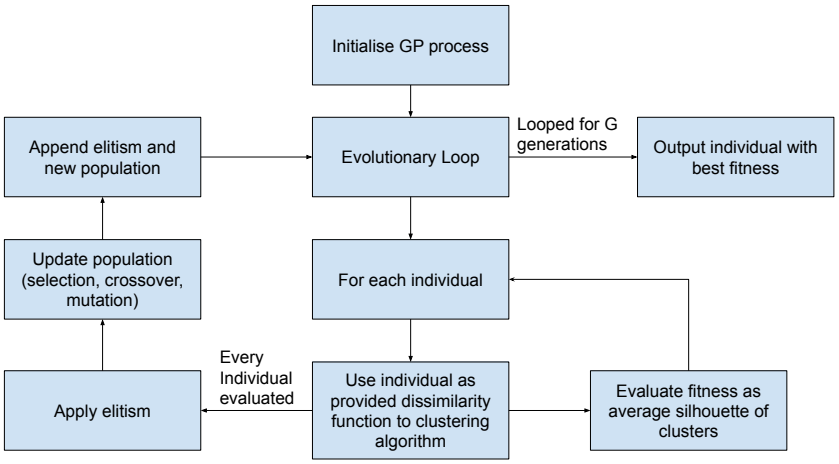


Figure 3.4: GP algorithm used

Table 3.1: Hyperparameters 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

3.2.2 Parameters

For the experiments each algorithm-dissimilarity pair is run 30 times on each dataset, with a different seed each time. The mean ARI and silhouette are then recorded as a measure of performance.

Table 3.1 shows the hyperparameters chosen for the GP process. These are standard GP parameters [19] with the exception of the population size, which is kept slightly lower to reduce computation time.

3.2.3 Datasets

Results are gathered on 2 of the simple HAWKS datasets and 2 of the complex MOCK datasets.

The HAWKS datasets have 10 dimensions and 10 ground truth clusters and 20 dimensions and 20 clusters respectively. Each of these datasets has 1000 instances. These were chosen as they are relatively easy clustering problems, and will provide a baseline for if the evolved functions can perform as well as pre-defined funtions. Initially higher dimensional datasets from this group were evaluated, but they were found to be too each to cluster with

a perfect ARI score.

The MOCK datasets have 50 and 100 dimensions each, both with 10 ground truth clusters. Each of these datasets have 2698 and 2892 instances respectively. They were chosen to provide a much more complex clustering problem with non-spherical clusters and a higher number of dimensions.

3.2.4 Evaluation Metric

To evaluate the performance of the dissimilarity evolution process against predefined dissimilarity functions for each of the clustering methods, both the ARI and the silhouette are reported. The ARI is used to give a clear indication of the performance of the method according to the gold standard clusters, and the silhouette is reported to ensure the GP process is producing results with high enough fitness.

3.3 Results

The results gathered on the chosen datasets are shown in Tables 3.2 and 3.3. The best ARI and average silhouette for each algorithm are shown in bold.

Of note are some of this missing values for the OPTICS algorithm. PyClustering has a bug where it will seemingly segfault for certain dissimilarity measures, which is what occured in a few cases during testing.

Table 3.4 shows the results obtained using the original terminal set on the 10d-10c dataset. It is worth noting that no results are obtained for OPTICS and agglomerative clustering. This is because the PyClustering implementations of these algorithms have fine-grained behaviour requiring that the properties of a distance function are met. As will be discussed in Section 3.4, this is not always guaranteed.

3.4 Discussion and Analysis

From the results, it is apparent that there is merit to evolving custom similarity functions. For the lower dimensional datasets, the evolved function outperforms all of the 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 pre-defined dissimilarity functions.

Chapter A in the appendix shows the best tree on each dataset evolved for the k-means++ algorithm. Notably, these trees contain very few constant values. This suggests that the dissimilarity functions are stronger when only considering the features without weighing or offsetting them. Each of the trees also has a wide spread of features, instead of using the same feature many times as leaves of the tree.

An issue with this method is that the good ARI performance observed in the low dimensional datasets disappears as the dimensionality increases. However, despite the ARI being significantly lower, the silhouette on all methods except for agglomerative is either the same as or higher than what is gathered using any of the pre-defined functions. We give more importance to the value of the ARI compared to the silhouette, as the ARI is based on the gold standard clusters while the silhouette is simply a measure based on the appearance of the clusters. The huge disparity between the high silhouette and extremely poor ARI implies that while the evolution is succeeding in finding good individuals based on the silhouette, the silhouette may not be the best indicator of clustering quality, especially in the higher

Table 3.2: ARI and silhouette on 10d-10c, 20d-20c, and 50d-10c datasets

10d-10c dataset

Dissimilarity	Measure	K-means++	HDBSCAN	OPTICS	Agglomerative	Graph
GP	ARI	**0.8952**	0.9315	0.8216	**0.9523**	0.9035
	SIL	**0.7489**	**0.8206**	0.8216	**0.7519**	0.7923
Euclidean	ARI	0.8402	0.9413	0.9083	0.8616	**0.9437**
	SIL	0.605	0.809	**0.822**	0.428	**0.807**
Cosine	ARI	0.8348	0.6965	0.0	0.01493	0.7812
	SIL	0.556	0.803	NA	-0.428	0.705
Manhattan	ARI	0.8370	**0.9432**	0.9386	0.8973	**0.9437**
	SIL	0.6069	0.808	0.821	0.484	**0.807**
Checbyshev	ARI	0.8306	0.9076	**0.9391**	0.7821	**0.9437**
	SIL	0.585	0.820	0.801	0.304	**0.807**

20d-20c dataset

Dissimilarity	Measure	K-means++	HDBSCAN	OPTICS	Agglomerative	Graph
GP	ARI	**0.9682**	0.9914	**0.9759**	0.4855	0.9351
	SIL	**0.7377**	0.7858	0.7846	0.1392	0.7519
Euclidean	ARI	0.8854	**1.0**	0.9485	**1.0**	**1.0**
	SIL	0.613	0.783	**0.790**	**0.783**	**0.783**
Cosine	ARI	0.9587	0.8748	0.5190	0.6175	0.6143
	SIL	0.700	**0.833**	0.722	0.299	0.401
Manhattan	ARI	0.8859	**1.0**	0.0554	**1.0**	**1.0**
	SIL	0.613	0.783	0.674	**0.783**	**0.783**
Checbyshev	ARI	0.8824	0.9778	0.9751	0.9624	**1.0**
	SIL	0.600	0.782	0.782	0.552	**0.783**

50d-10c dataset

Dissimilarity	Measure	K-means++	HDBSCAN	OPTICS	Agglomerative	Graph
GP	ARI	0.2040	0.0003	0.0011	0.0901	0.0021
	SIL	**0.5983**	0.6559	0.5584	0.3988	**0.5870**
Euclidean	ARI	0.3958	0.5886	**0.6011**	**0.4455**	0.9985
	SIL	0.5671	0.4521	0.3791	**0.4807**	0.4255
Cosine	ARI	**0.7186**	0.5415	0.5097	0.1458	0.0000
	SIL	0.3227	**0.6989**	**0.5832**	-0.3388	NA
Manhattan	ARI	0.4039	**0.5976**	0.0000	0.2675	**1.0000**
	SIL	0.5683	0.3631	NA	0.4511	0.4425
Chebyshev	ARI	0.4604	0.4407	0.4428	0.3659	0.5044
	SIL	0.5170	0.4612	0.2931	0.2912	-0.0313

Table 3.3: ARI and silhouette on 100-10c dataset

100-10c dataset						
Dissimilarity Measure		**K-means++**	**HDBSCAN**	**OPTICS**	**Agglomerative**	**Graph**
GP	ARI	0.2439	0.0005	0.0012	0.0817	0.0011
	SIL	**0.6100**	**0.6768**	**0.6269**	**0.4137**	**0.6508**
Euclidean	ARI	0.4769	**0.8668**	**0.8514**	0.2043	0.9869
	SIL	0.5904	0.4450	0.5623	0.3012	0.5807
Cosine	ARI	**0.8162**	0.8133	0.4670	**0.3072**	0.8428
	SIL	0.3765	0.5465	0.2087	-0.1896	0.4004
Manhattan	ARI	0.4703	0.8652	0.0180	0.2013	**1.0000**
	SIL	0.5862	0.4536	0.5518	0.3006	0.5731
Chebyshev	ARI	0.5221	0.7777	0.6735	0.1787	0.6539
	SIL	0.6070	0.4336	0.4687	0.2923	0.3414

Table 3.4: Results on 10d-10c dataset using original terminal set

10d-10c dataset				
Dissimilarity Measure		**K-means++**	**HDBSCAN**	**Graph**
Original	ARI	0.7631	0.5008	0.3772
	SIL	0.720	0.616	0.591

dimensional datasets. The silhouette can be seen as a combined measure of both similarity and separability of clusters - it may be a good idea to split these metrics into separate objectives and evaluate the clusters using EMO. This idea is explored in chapter 4.

The behaviour on the high dimensional datasets can also be explained by the non-spherical nature of the gold standard clusters. The silhouette assumes that the clusters will be spherical, meaning that it will assign a low fitness to a correct solution using these datasets.

3.4.1 Terminal Set Comparison

Comparing the results gathered between the two terminal sets on the 10d-10c dataset, there is an obvious improvement with the updated terminal set. Inspecting some of the individuals generated by the original terminal set, the evolution has the possibility to have a large proportion of terminals chosen from only one instance. This leads to issues clustering as the relationship between the points is never actually evaluated, only the straight attributes of the points are. Additionally, as mentioned in Section 3.1.2, the dissimilarity function is not symmetric. In order to get around this, the dissimilarity was evaluated in both directions, with the overall result being the sum of the two separate evaluations. This slowed down the overall computation, however. While this original terminal set in theory would be able to produce the same results as the new one, the search space was found to be too large to reliably evolve good quality clusters. It was observed that occasional runs achieved very high performance, but these were runs that had high performing individuals in the randomly generated initial population. However, runs that didn't have good performance initially were unable to check enough of the search space before the algorithm terminated.

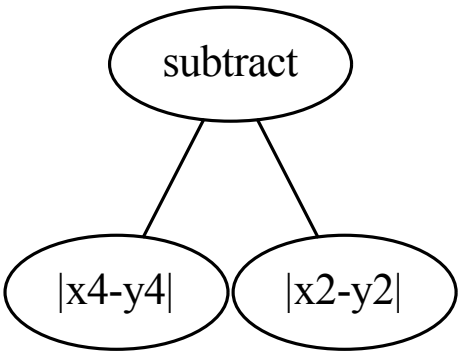


Figure 3.5: Possible tree that breaks non-negativity property

3.4.2 Issues With the Evolved Functions

During the evolution, it was found that the OPTICS algorithm would sometimes crash due to a segfault error. This did not only occur with the evolved functions, as it also occured on some datasets not shown in the report when using the Manhattan distance function. These were from the first set of datasets [35], and were not included as they were too easy to cluster, so most algorithms were getting ARIs of 1.0.

Looking at the function and terminal sets of the evolution, it can be seen that many of the possible evolved dissimilarity functions will not meet the four properties of a full distance metric. Analysing each property:

1. **Non-Negativity**
   It is possible to evolve a tree that will result in negative dissimilaries if a subtraction function is used and the value to the right of the function is larger. Figure 3.5 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 second feature on the two instances then the resulting dissimilarity will be negative.

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 3.6 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

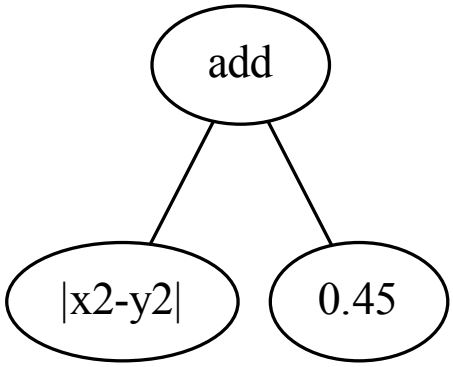


Figure 3.6: Possible tree that breaks identity property

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 exactly 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 3.7 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. The 3 dissimilarities are given according to Equation 3.8

$$\begin{aligned} d(a,b) &= if(|0.1-0.4|-0.35)then(0.9)else(0.1) = if(-0.05)then(0.9)else(0.1) = 0.1 \\ d(b,c) &= if(|0.4-0.5|-0.35)then(0.9)else(0.1) = if(-0.25)then(0.9)else(0.1) = 0.1 \\ d(a,c) &= if(|0.1-0.5|-0.35)then(0.9)else(0.1) = if(0.05)then(0.9)else(0.1) = 0.9 \end{aligned} \tag{3.8}$$

   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 $0.9 > 0.1 + 0.1$.

3.5 Computation Cost

Because the evolved functions require a large amount of numerical computations, they are slow to evaluate. To reduce some of the impact of this, the GP process builds *numpy* vectorized functions, however the computations still take a lot of time to compute. A single run evolving a function for k-means++ takes approximately 10 hours to complete, while the other algorithms take up to 40 hours. To allow for timely result collection and to prevent using too many grid resources, all grid experiments were set to terminate after 40 hours. This meant that occasionally an experiment would terminate before completion. To ensure

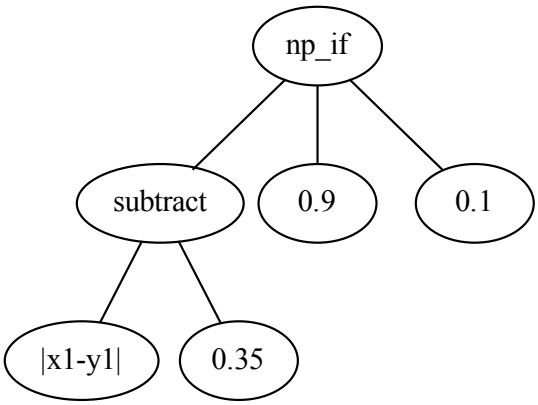


Figure 3.7: Possible tree that breaks triangle inequality property

results were still gathered, the best individual at each generation was used to produce clusters, which were then saved. K-means++ is quicker than the rest of the algorithms because a pairwise dissimilarity matrix is built for each of the other algorithms, while for k-means++ the dissimilarity is only evaluated from each instance to each cluster centre. Despite the cluster centres being moved and needing re-evaluation, the total number of computations required remains lower.

3.6 Summary

In this chapter, a framework allowing for dissimilarity functions to be evolved for a provided dataset and clustering algorithm has been introduced. After testing on multiple datasets and algorithms, it was found that in general the clusters created had better ARI and average silhouette score than pre-existing dissimilarity functions for simple low dimensional datasets. However, while the method evolved dissimilarity functions with a higher average silhouette score than the pre-existing functions, the ARI was considerably worse.

Three major issues with the framework were identified. First, the dissimilarity functions throughout evolution are often very poor, not fitting the assumed properties of distance metrics. Second, the fitness function used for evolution is shown to not be effective to evolve clusters according to the gold standard clusters. Finally, the algorithm takes a long time to run. Potential fixes for these issues are explored in the next chapter.

Chapter 4

Further Design

4.1 Motivations

In Chapter 3 we explored an initial design for evolving custom dissimilarity functions for clustering, using GP. However, a number of shortcomings were found with this design:

1. The evolved dissimilarity functions have no requirements to fit the assumed properties of distance metrics. It was found that this resulted in many solutions both throughout the learning process and in the final solutions with a number of clusters that were considerably different from the gold standard. This means that the evolution is spending a lot of time in "dead space", evaluating and passing on dissimilarity functions that have no real hope of creating a good number of clusters for the dataset.

2. The fitness function of the average silhouette may not be the best indicator of cluster fitness, as despite evolving higher silhouette values than the predefined functions, the evolved functions have a lower ARI. The hypothesis for why this is occurring is that the silhouette is a metric combining the concepts of compactness and separability. However, these are often conflicting objectives as increasing the average compactness of clusters will have the effect of decreasing the average separability, and vice versa. This makes the silhouette an inflexible metric, as it only considers one specific trade-off between these conflicting metrics.

3. The algorithm takes a long time to run, reducing viability as a commonly used method.

With experimentation of different evolved fitness functions, it was found that individuals that strongly broke the triangle inequality property resulted in very poor performance. In k-means++, where the number of clusters is preset, it was even resulting in empty clusters due to the algorithm expecting the triangle inequality to hold as an implicit requirement. This behaviour of finding very low numbers of clusters was also shown in all of the other algorithms with5 the exception of agglomerative. Agglomerative clustering manifested this issue in a slightly different form, with a few clusters containing most of the instances and the remaining clusters with only one or two instances.

4.2 Transfer Learning

In an attempt to alleviate the third shortcoming an attempt at a form of transfer learning was applied, with the dissimilarity function being initially evolved using k-means++ before being fine-tuned on a different algorithm. Figure 4.1 shows the modified algorithm used in

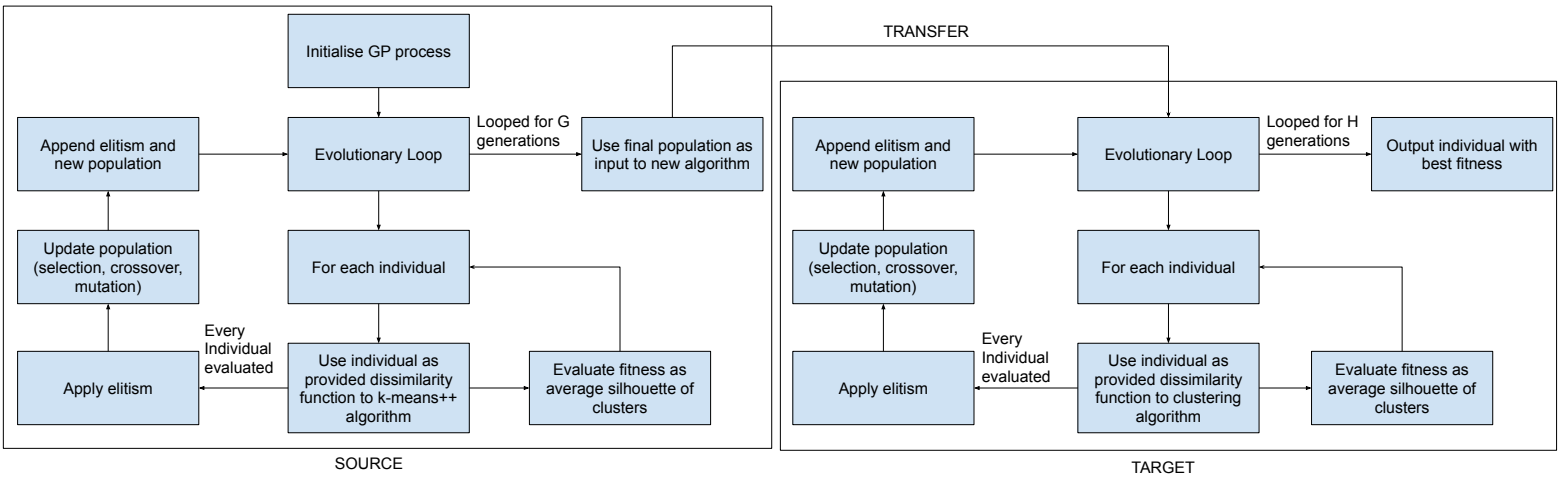


Figure 4.1: Transfer learning algorithm

this version of the method. This is very similar to the original method, except that the population is always trained using k-means++. The final population is then transferred to a new evolutionary loop, which fine-tunes the dissimilarity function to the provided clustering algorithm.

While this change did reduce the overall computation time, it was found that it also reduced the quality of the dissimilarity functions produced. This points to the fact that the evolved dissimilarity functions are not generalisable between algorithms, as this method can be seen as a form of wrapper feature selection/construction, learning features for the specific algorithms rather than the dataset in general.

4.3 Proposed Method

There are two distinct extensions to the initial method proposed in this section. First, a form of constraint is added to the algorithm such that only individuals that produce a good number of clusters will be selected. Second, EMO is used in order provide to a more accurate fitness evaluation than can be provided by the single fitness function of the silhouette.

In addition to these extensions, the logical functions if, min, and max were removed from the function set. This is due to the fact that these logical functions very blatantly break the requirement of the triangle inequality, as described in Section 3.4.2. However, removal of these functions does reduce the ability of the trees to represent non-linear decision boundaries.

4.3.1 Individual Constraints

A number of different constraint methods were considered. These are all ways of encouraging the GP process to evolve individuals with close to the gold standard number of clusters.

Of these, the four that were tested were:

1. **Hard Constraint**
   This is the simplest constraint. It compares the number of clusters created with the preset gold standard number, and discards all individuals that don't result in that amount. This ensures that all individuals will create the perfect number of clusters.

2. **Soft Constraint**
   This is similar to the hard constraint, but instead discards all individuals that fall outside a given margin of error around the gold standard number of clusters. In the experiments in this report the margin of the soft constraint was set to be $c \pm 0.1c$ where c is the preset number of clusters. This ensures close to the correct number of clusters is created, with some room for difference for the sake of better cluster quality.

3. **Objective Constraint**
   This constraint ties in with the EMO methods discussed in the next section, and adds the absolute error from the correct number of clusters as an additional objective. This allows for an innate tradeoff between cluster quality and the gold standard number of clusters. While this isn't technically a constraint, it is included in this section for conciseness.

4. **Tiebreaker Constraint**
   This constraint is made to work with the NSGA-II algorithm for EMO. It adds an extra ordering metric between the rank and the crowding distance. Within each rank, the individuals are now sorted in terms of absolute error from the gold standard number of clusters, before finally being sorted by clustering distance. This in theory should allow for a wide range of strong solutions along the Pareto front, with only weaker solutions having the constraint applied to them.

Of note is that all of these constraints require the "correct" number of clusters to be known in advance. This does reduce the effectiveness of methods such as OPTICS which can operate without being told how many clusters to create.

It was decided in initial testing on k-means++ that only the hard and soft constraints would be explored further in the project. This was because both the objective constraint and tiebreaker constraint resulted in large numbers of solutions that had strong separability and sparsity but very low numbers of clusters, meaning that the constraint was not controlling the cluster numbers strongly enough. Another issue with the tiebreaker constraint is that it only worked for NSGA-II, so could not be applied to single objective experiments.

Initial results using the hard and soft constraints found the population size shrinking by a large amount after the initial population was evaluated. This is due to the fact that not many of the randomly generated solutions were producing close to the gold standard number of clusters. Because of the reduction in population size, this resulted in extremely poor performance. From this, the idea emerged to repeatedly generate a random population, adding individuals that pass the constraints until the population is the desired size.

4.3.2 Multiobjective Based Method

The previous results show that using average silhouette as a fitness function doesn't evolve dissimilarity metrics that correctly cluster complex higher dimensional data. This is hypothesised to be due to the fact that silhouette is a combination of metrics of sparsity and separability, without any weighting to one or the other. This can cause problems in evolving strong dissimilarity functions as a higher separability will result in a higher sparsity, while a low sparsity will result in a low separability. These conflicting metrics naturally lead to the idea of performing EMO to evolve the fitness functions, using the sparsity and separability as two separate metrics. This defines the problem as a multiobjective optimisation problem with two conflicting objectives.

In order to use these metrics, we must define a mathematical interpretation of the conceptual ideas of sparsity and separability. The definitions used are taken from [7].

The sparsity of a cluster is defined by first finding the shortest distance between every pair of instances in the cluster, then finding the maximum of these distances. This gives us the distance from the most isolated member of the cluster to the rest of the cluster. The sparsity is defined in equation 4.1, where C_i refers to the cluster currently being evaluated and $d(a, b)$ refers to the Euclidean distance between a and b. This is the first objective for the multiobjective optimisation.

$$\text{Sparsity} = \max_{I_a \in C_i} \min_{I_b \in C_i} d(I_a, I_b) | I_a \neq I_b \tag{4.1}$$

The separability of a cluster is defined by finding the shortest distance between any instance in the cluster and any instance not in the cluster. The separability is defined in equation 4.2, where C_i refers to the cluster currently being evaluated and $d(a, b)$ refers to the euclidean distance between a and b. This is the second objective for the multiobjective optimisation.

$$\text{Separability} = \min_{I_a \in C_i} \min_{I_b \notin C_i} d(I_a, I_b) \tag{4.2}$$

These metrics are both calculated for all clusters, then the two objectives are set to the mean sparsity and mean separability. In evolution it is desired to minimise the sparsity and maximise the separability.

The NSGA-II algorithm was selected to perform the EMO evolution as it is a well regarded EMO algorithm with good results across the existing literature. A decomposition based EMO algorithm, MOEA/D [44], was considered, but was ultimately decided against due to the behaviour of the evolved fronts. MOEA/D

After initial testing on k-means++, it was found that most of the solutions in the evolved Pareto fronts were creating very low numbers of clusters. This is likely due to the separability metric, as having only a few clusters that are entirely separated from each other will create a lower mean separability than the correct solution. An example of this is shown in Figure 4.2, with a ground truth of seven clusters. Four of the ground truth clusters have a relatively small separation between them, and a large separation between them and the other group of three clusters. The ground truth clusters are shown in the blue ovals. However, the clustering solution shown in the red ovals will have a much higher average separability as the small distances between different clusters will not lower the average. This means that if one was to evolve a dissimilarity metric for this data then solutions that encourage the red ovals will be encouraged.

Because of this behaviour, the EMO method is always run with one of the individual constraints discussed in Section 4.3.1.

4.4 Experiment Design

The experiment design is very similar to that used in the initial implementation. The DEAP library is still used, and the only major difference is that the learning algorithm has been changed from a basic evolution algorithm to NSGA-II. In addition, the algorithm has been modified to allow for the individual constraints to be applied. The new algorithm used for both the single objective and EMO methods is shown in Figure 4.3. The overall evolutionary loop is the same, but has two major modifications. First, the population is repeatedly initialised and evaluated, before having the constraints applied. The solutions that pass the constraints are added to the starting population. This is repeated until the starting population is full. The second modification is that the algorithm now appends the child and parent

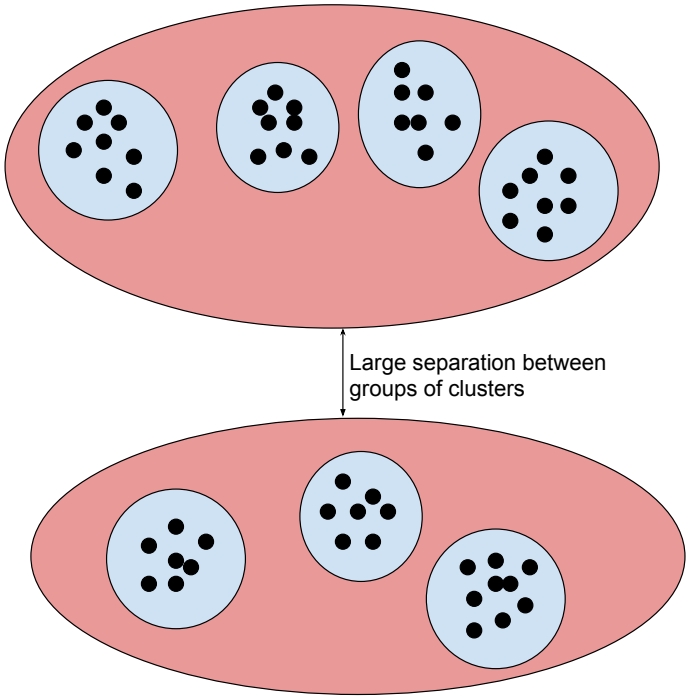


Figure 4.2: Example where separability metric performs poorly. Blue ovals indicate ground truth clusters, red ovals indicate separability optimised clusters.

populations, using either NSGA-II or tournament selection to create the new population for the next generation from the combined population.

The main difference between the single objective and EMO algorithms is the selection process after the constraints are applied. EMO applies NSGA-II while the single objectives apply tournament selection. Additionally, the EMO method outputs the entire Pareto front while the single objective methods output only the fittest individual.

The experiments are run on the same datasets as the initial model. However, results were not collected on the 100 dimensional dataset due to the memory requirements of creating a pairwise dissimilarity matrix with a larger number of instances than the simple datasets. While this would be feasable, time requirements involved with the honours project made it not possible to perform the number of experimental runs that would have been required.

The parameters used are the same as in the previous section, but are reiterated in Table 4.1.

For the 10d-10c dataset, 6 different experiments are performed. First, the basic algorithm using silhouette with both hard and soft constraints (Hard-Sil and Soft-Sil). Then, the experiment is repeated again with a basic combination of the defined sparsity and separability metrics in order to get a good comparison with EMO (Hard-Comb and Soft-Comb). In this version, the fitness function of an individual is given by $\frac{\overline{sparsity}}{\overline{separability}}$. As the multi-objective method seeks to minimise the sparsity and maximise the separability, this single objective function is a minimisation task. Finally, EMO is performed with both hard and soft constraints (Hard-EMO and Soft-EMO).

It was discovered from the 10d-10c dataset that the hard constraints resulted in the algorithm taking too long to generate a valid initial population. Due to this, the 20d-20c and 50d-10c datasets were only experimented on with the soft constraint.

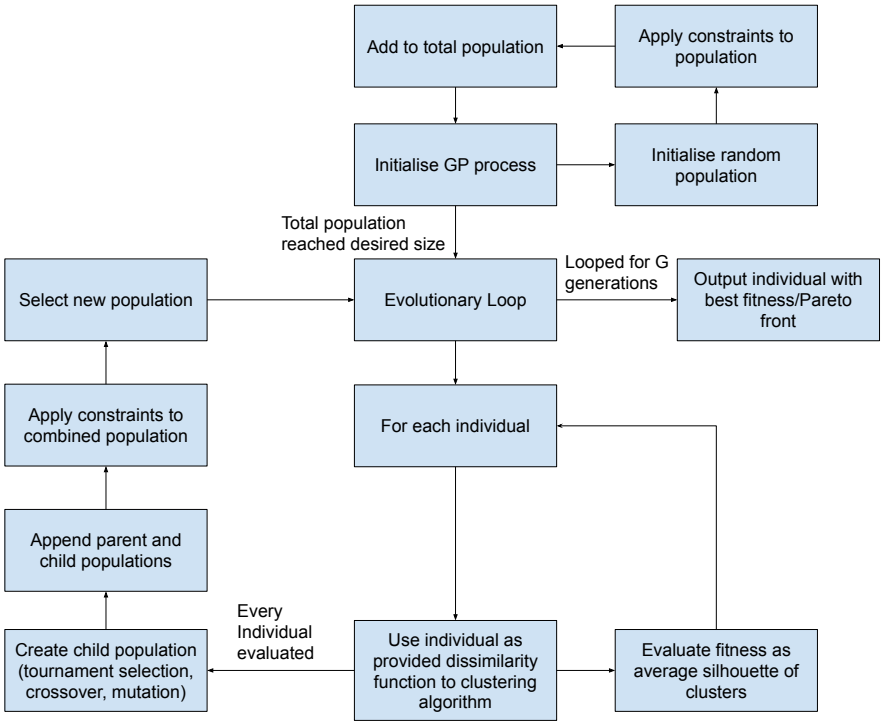


Figure 4.3: GP algorithm with constraints

Table 4.1: Hyperparameters 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

4.5 Results

The results gathered on the single objective methods are shown in Table 4.2. The best ARI, average silhouette, sparsity, and separation for each algorithm are shown in bold.

For the EMO algorithms, the hypervolume of the generated front across all 30 runs is calculated. The front with the median hypervolume is then selected and plotted. Each individual is coloured according to the ARI of the clusters produced using that function. Additionally, the fronts with the highest and lowest hypervolumes are plotted with dotted lines.

The plotted fronts are shown in Figures 4.4, 4.5, and 4.6.

Results are not shown for the MOCK datasets other than k-means++ and agglomerative for the 50d-10c dataset. The reason for this is discussed in Subsection 4.6.5.

4.6 Discussion and Analysis

4.6.1 Comparison With Basic GP

Looking at the results on the single objective algorithms, it is apparent that despite the expected beneficial effect of the constraints they generally result in worse solutions. A good way to show this is by comparing the GP results from the previous section with the silhouette results from this section, as the only differences between the two are the evolution algorithm used and the application of the constraints.

In the basic 10d-10c dataset, both the hard and soft constraints improve the average ARI for k-means++. The ARI is reduced, however, for the remaining clustering algorithms. While for the base GP method, for example, HDBSCAN produces results with an average ARI around 0.03 better than k-means++, it has 0.12 worse ARI for the hard constraint and 0.07 worse for the soft constraint.

These results get worse for the 20d-20c dataset. While k-means++ still produces a reasonable ARI, it is now slightly (but not statistically) worse than that given by the basic GP method. Looking at the other algorithms, it can be seen that the results produced are significantly worse. Using HDBSCAN as an example again, the ARI in the base GP method of 0.9914 (almost perfect to the ground truth) has now been reduced to 0.2357.

The only algorithm not directly effected by the constraints is agglomerative clustering, as that is guaranteed to always produce the ground truth number of clusters to begin with. However, it can be observed that the results gathered for agglomerative in the higher dimensional datasets are still worse than those from the original GP method - this points to the idea that the modifications made to the evolution process to make the constraints possible have a negative impact on the evolution themselves. The main difference to the algorithm here is that the parent and child populations are combined. If the children don't perform as well as the parents, this means that the new population has the possibility of largely consisting of the original data. This can hinder the ability of the algorithm to create new solutions and perform exploration.

4.6.2 Single Objective Comparisons

Comparing between the results of the silhouette and the combined $\frac{sparsity}{separability}$, it is clear that the more complex metric of the silhouette gives a better singular measure of cluster quality. In the 10d-10c dataset, there is no case where the combined metric produces a better ARI than the corresponding silhouette method. For the 20d-20c dataset, however, both OPTICS and agglomerative demonstrate a higher average ARI for the combined metric than the

Table 4.2: ARI on 10d-10c, 20d-20c, and 50d-10c datasets

10d-10c dataset

Algorithm Variant		K-means++	HDBSCAN	OPTICS	Agglomerative	Graph
Hard-Sil	ARI	0.9194	0.7988	NA	0.9549	**0.4601**
	SIL	0.7311	0.7310	NA	**0.7575**	0.0613
	SPARS	0.3744	0.3278	NA	0.2889	0.4153
	SEP	0.2556	0.1821	NA	0.3114	0.0857
Soft-Sil	ARI	**0.9286**	**0.8596**	**0.8435**	**0.9570**	0.3999
	SIL	**0.7380**	**0.8117**	**0.6707**	0.7544	**0.1290**
	SPARS	0.3782	0.2828	0.3213	0.2966	0.3740
	SEP	0.2409	**0.2608**	0.1880	0.3062	0.0858
Hard-Comb	ARI	0.7012	0.5023	0.2619	0.9490	0.2819
	SIL	0.6142	0.6435	-0.0284	0.7132	-0.1287
	SPARS	**0.2754**	0.2765	0.6905	**0.2657**	0.3054
	SEP	0.3893	0.1824	0.1160	**0.3360**	0.08691
Soft-Comb	ARI	0.6877	0.7229	0.5684	0.9312	0.3621
	SIL	0.6018	0.6782	0.3367	0.6749	0.0475
	SPARS	0.2755	**0.2602**	**0.2745**	0.2696	**0.2775**
	SEP	**0.3981**	0.2571	**0.1967**	0.3175	**0.1538**

20d-20c dataset

Algorithm Variant		K-means++	HDBSCAN	OPTICS	Agglomerative	Graph
Soft-Sil	ARI	**0.9506**	**0.2357**	0.2116	0.4338	**0.2762**
	SIL	**0.7189**	**0.2635**	-0.0026	0.0645	**-0.0482**
	SPARS	0.4375	0.7747	0.6244	0.9111	0.8572
	SEP	0.5493	**0.2428**	0.2342	0.2055	0.1970
Soft-Comb	ARI	0.9046	0.1120	**0.3109**	**0.6273**	0.1869
	SIL	0.6563	-0.0236	**0.0612**	**0.2253**	-0.1890
	SPARS	**0.3867**	**0.5470**	**0.5220**	**0.4640**	**0.6282**
	SEP	**0.5742**	0.2128	**0.2924**	**0.3713**	**0.1996**

50d-10c dataset

Algorithm Variant		K-means++	Agglomerative
Soft-Sil	ARI	**0.3671**	0.0530
	SIL	**0.6093**	**0.4241**
	SPARS	**0.6046**	**0.6789**
	SEP	0.1071	0.0986
Soft-Comb	ARI	0.2657	**0.0678**
	SIL	0.1803	0.0082
	SPARS	0.7274	0.7189
	SEP	**0.1612**	**0.2548**

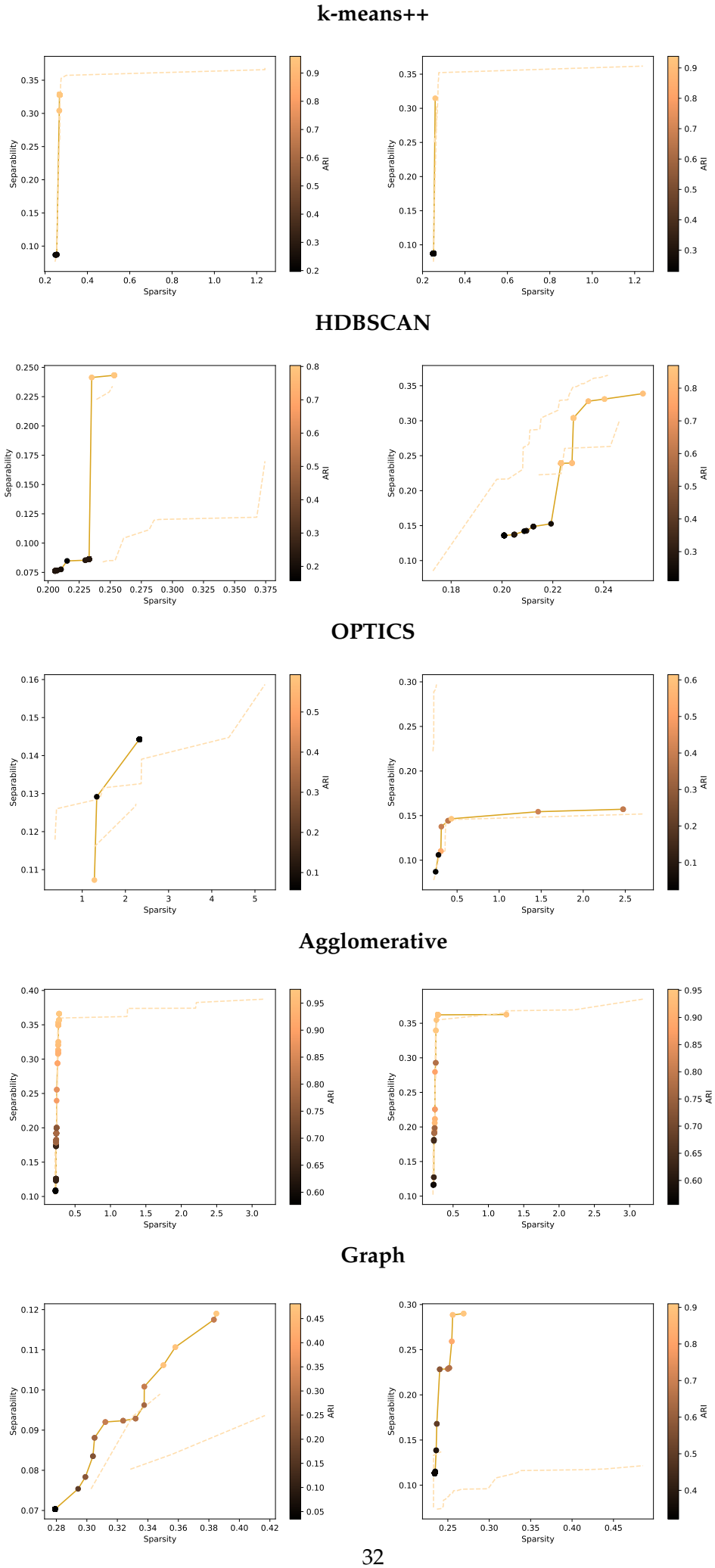
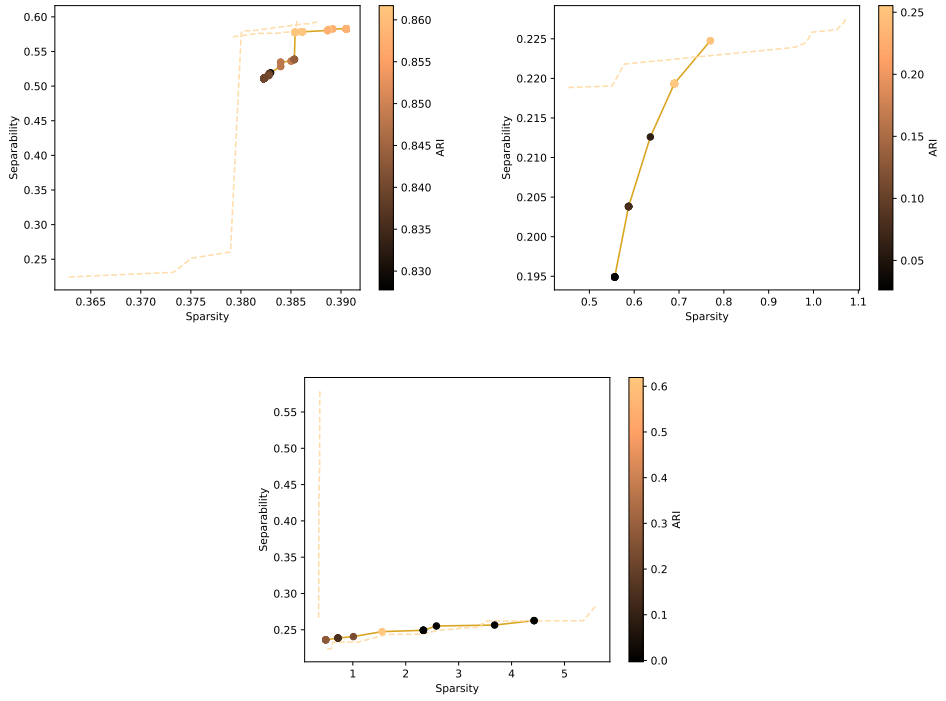


Figure 4.4: Fronts for 10d-10c dataset. Left column uses hard constraint, right column uses soft constraint.

k-means++, HDBSCAN, OPTICS



agglomerative, graph

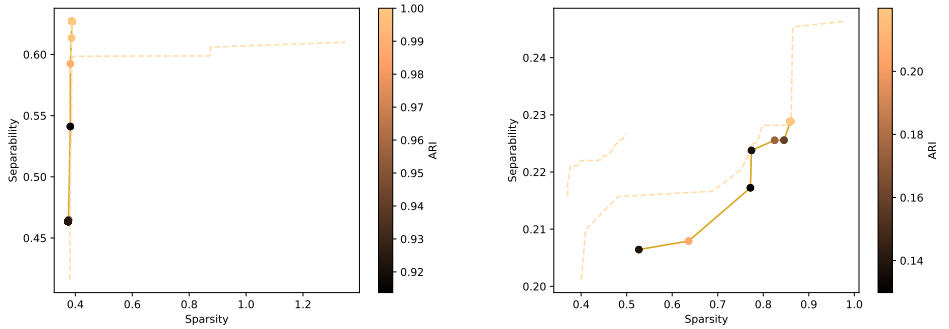


Figure 4.5: Fronts for 20d-20c dataset

k-means++, agglomerative

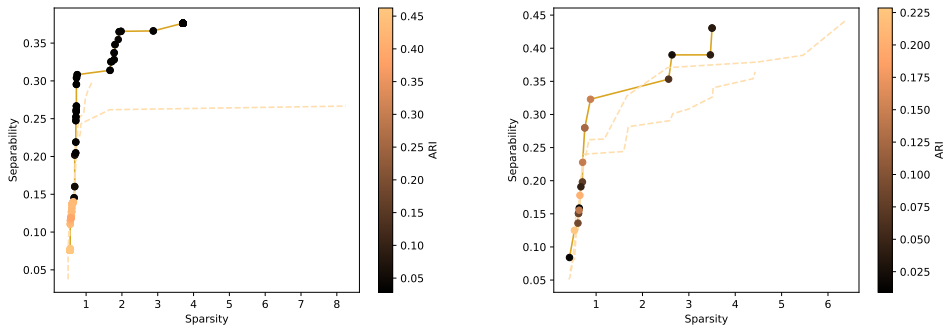


Figure 4.6: Fronts for 50d-50c dataset

single objective metric. This is an especially interesting result for agglomerative clustering, as the constrained silhouette results were similar to the original GP results. Agglomerative also demonstrated slightly higher ARI for the 50d-10c dataset, but the difference is small enough that it is not significant.

The behaviour on the 20d-20c datasets shows that for agglomerative clustering the combined metric may be a better indicator of cluster quality due to the single linkage behaviour of the algorithm. This allows for a stronger overall dissimilarity function to be evolved for the algorithm.

While the ARI is lower for the combined fitness function, in almost every case the combined function does result in a lower mean sparsity and higher mean separability. However, the mean silhouette is also considerably lower. Again, the exception to this appears to be the 20d-20c agglomerative clustering - the mean silhouette derived from the combined method beats the one derived from the silhouette method. It is worth noting that due to starting from the same random seeds, the methods both start with the same initial populations of functions. In addition to this, both methods share the exact same structure with the single difference between the two being the fitness function. This means that in almost all situations the silhouette is a better fitness for guiding the evolution through the search space than the combined function, while the combined function appears to be better for guiding the evolution for agglomerative clustering.

4.6.3 Multiobjective Based Method

For the HAWKS datasets, the multiobjective results show the ARI increasing as separability increases despite the sparsity also increasing. This demonstrates that the separability is much more important as a clustering metric than the sparsity for these datasets, as clustering solutions with a low sparsity but low separability result in every case but 10d-10c hard OPTICS with the worst ARI in the front.

Interestingly, this behaviour is flipped for the more complex MOCK generated 50d-10c dataset. This is most apparent in the results for k-means++. The results with low sparsity and separability demonstrate a high ARI, while the ARI sharply drops as the separability increases. This points to the fact that the importance of each of sparsity and separability as cluster fitness metrics changes depending on the dataset that is being clustered.

This lack of consistency between the optimal tradeoff of sparsity and separability shows that EMO is useful, if not required, to produce the optimal results using the sparsity and separability. In fact, for the more complex 50d-10c dataset it can be seen that individuals are produced with vastly better ARI than the other GP methods, including the non-constrained methods.

There are some interesting shapes present in the fronts. For example, both median fronts for k-means++ in the 10d-10c dataset have a very small front, with a sharp jump up in separability but only a small improvement in sparsity. However, all of the results with the high separability also produce a high ARI, which is consistent with the other algorithms. It appears that most of the fronts exhibit this behaviour, where the front will sharply increase in one direction while not increasing much in the other direction. Despite this, most fronts still have a clear increase in ARI in one direction along the front.

The maximum and minimum fronts also exhibit some interesting behaviour. For around 8 of the experiments, the maximum front only exhibits the maximum hypervolume because the sparsity of the front has a sharp increase, often only associated with a very small increase in the separability. This is not particularly desired behaviour — it can be intuitively known that clustering solutions with such high sparsity will not demonstrate very high quality clusters. The minimum fronts have the opposite behaviour, with many of them only containing

a few similar solutions. While this is not shown in the diagrams, the gathering of results showed that the maximum and minimum fronts often had a maximum ARI close to that from the median front.

Comparing the sparsity and separability gathered in the combined single objective methods and the multiobjective methods, the two methods result in similar values for both objectives. Despite this, the multiobjective optimisation manages to consistently achieve individuals with a higher ARI. The EMO methods manage to evolve a much more diverse population than the single objective methods by allowing a variety of tradeoffs between sparsity and separability, while the single fitness functions are limited to a set tradeoff.

4.6.4 Effects of Constraints

As discussed in 4.6.1, the constraints on the individuals resulted in much worse overall results. In retrospect, the constraints on which individuals are accepted are bound to negatively effect the evolution process.

One of the major strengths of GP is that it allows for partial solutions to be evolved, which are close to a solution with high performance but require one or more changes [45]. However, in removing individuals that don't have close to the correct number of clusters we are encouraging the evolution away from these partial solutions. For example, consider the clusters presented in Figure 4.7. This represents a dataset with eight ground truth clusters, represented by the blue circles. There could be a hypothetical partial solution that produces the clusters represented by the red circles. While this only has four clusters, and so would be rejected by the constraints, it perfectly contains the ground truth clusters. The only change the function would need is to differentiate the distances between close clusters more. This essentially means that the constrained methods lose the ability to exploit these strong partial solutions. In addition to this, the behaviour of ignoring all solutions that don't give the ground truth number of clusters will cut out large portions of the search space, heavily reducing the exploration ability of the algorithm.

The constraints as implemented also limit the ability of the evolution to travel through the search space. This is because the algorithm combines the parent and child populations and then applies the constraints, before finally performing selection. All of the parent individuals are guaranteed to pass the constraint check as they are not re-evaluated, while only a portion of the children will pass the constraint check. This means that a higher proportion of the combined population is made up of the parent individuals, and so parents will be more likely to be chosen by the selection algorithm. This greatly diminishes the exploration ability of the algorithm, as a non-insignificant number of individuals being passed down the generations are likely just the parents from multiple generations ago.

4.6.5 Computation Cost

Another important aspect to note about the methods proposed in this section is the impact on runtime caused by the constraints. While required in order to obtain any kind of decent performance, generating the initial population has the effect of causing the algorithm to take much longer to terminate. This had a massive effect on my gathering of results. Each experiment on the grid was set to terminate after 40 hours, with the results being saved after each generation in case of it running out of time before termination.

This issue is caused by the iterative generation of the initial population. The random initial populations can contain a very low number of individuals that pass the constraints, meaning that it takes a large number of loops before a full initial population can be generated. In some cases in the results gathered, this turned out to take upwards of 50 iterations

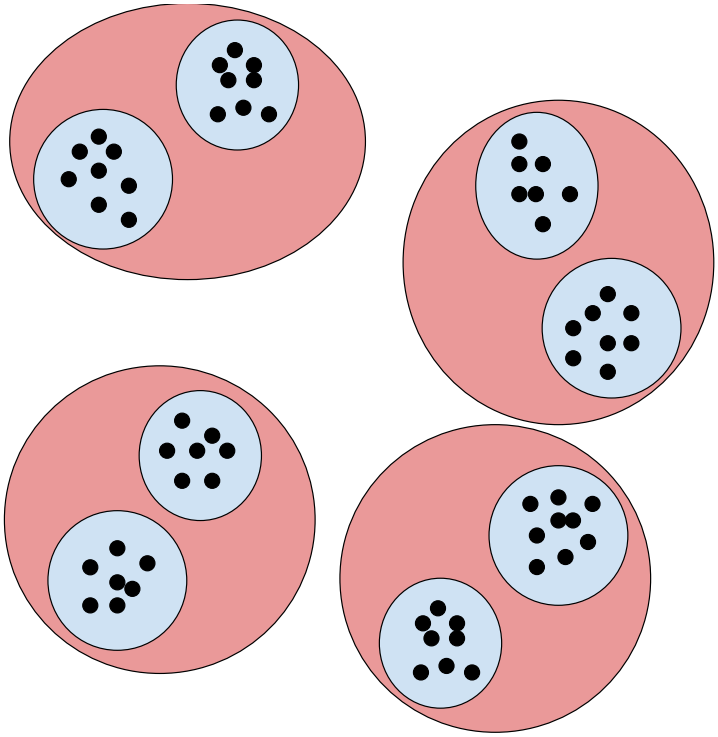


Figure 4.7: Example clusters produced by a dissimilarity measure with strong behaviour. This solution will be removed by constraints due to number of clusters.

before a full sized population was generated. The problem with this is that it reduces the amount of time available on the grid to actually evolve the population. This lead to some even greater issues where for some algorithms no results were gathered, as not a single run passed population generation stage before the 40 hours had passed. This initially occurred for the Hard-Sil method for OPTICS and graph clustering on the 10d-10c dataset, as well as for all results (Soft-EMO, Soft-Sil, Soft-Comb) for OPTICS, HDBSCAN and graph clustering on the 50d-10c dataset. These algorithms were then run again with a 70 hour limit, but still terminated near the end of the initial population generation. This further points to issues with the constraints as it is currently applied — this massive increase in runtime doesn't make the algorithm as viable for real world use.

4.7 Summary

In this chapter, the GP framework was extended with two new contributions. First, a method for constraining solutions through the evolution was introduced that ensured a number of clusters close to the gold standard. Next, a method utilising the EMO algorithm NSGA-II was introduced, with metrics of sparsity and separability as the two objectives. While the constraints alone negatively effected the performance of the GP optimisation, the combination of EMO and the constraints (specifically the soft constraint) was able to produce higher quality solutions than the single objective methods.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

This project presented a novel GP based framework to evolve tailored dissimilarity functions for a provided dataset/clustering algorithm pair. Seven variations on this were explored - the base GP method, Hard-Sil, Soft-Sil, Hard-Comb, Soft-Comb, Hard-EMO, and Soft-EMO. The single objective constraint methods did not perform as well as the base GP method, however it was found that the constrained EMO methods were able to generate individuals with similar clustering metrics but higher ARI than the base GP.

On the low dimensional hyper-spherical datasets, it was found that in most cases the GP method performed as good as or better than the pre-defined dissimilarities. However, on the higher dimensional ellipsoidal datasets the GP method failed to produce results anywhere near as good as the pre-defined functions.

Across all of the methods the algorithm that consistently has the best performance using this framework is k-means++. This could be due to the simplicity of the method, and the fact that partitions can easily be defined in any transformation of the feature space. It also raises an interesting point about other work in a similar domain to this project. Taking [35] as an example, the authors only evaluate the method on k-means++. However, it has been shown here that the k-means++ algorithm is the one that most benefits from this work, manipulating the feature space with GP. It would be interesting to see if other papers would retain their results if the generalisability of their methods were tested.

5.1.1 Fitness Function

There are some very apparent issues with all fitness functions used in this project. Most notably, despite the fact that the evolved dissimilarity functions can be seen as a transformation of the feature space, we still evaluate the fitness based on the original feature space. The silhouette, sparsity, and separability metrics all explicitly use the Euclidean distance in the original feature space to evaluate the distances. This seems counter-intuitive to the entire idea of evolving a new dissimilarity function, if you simply then use the pre-existing one to evaluate how good it is.

This, however, is not an easy thing to fix. One great difficulty with unsupervised learning is that there are no ground truth values to evaluate against during model training. Because of this, we have to define some way of evaluating results in a way that is consistent across all models and gives a good indication of the performance of the model. If we were to use our evolved dissimilarity functions in the place of Euclidean distance for our fitness evaluation, all of the results would show high fitness. As the exact same algorithm is used with the one

difference being the dissimilarity function, it is likely that the results would be near identical if the dissimilarity function of each individual was then used to evaluate the fitness.

5.2 Future Work

There are a number of different directions that could be explored leading on from this project

- Previous work in this area [5], [35] has shown that the use of multi-tree GP rather than single-tree GP produces much better results. This could be explored for the current research. One concern with this idea is that a naive implementation of multi-tree GP could lead to even more opportunities for the evolved functions to break the distance metric assumptions.
- Despite the poor results gathered using transfer learning during this project, it is still a possible avenue for future study. As it increases the overall speed of the method by utilising the computational benefits of k-means++, transfer learning could be applied to allow for a much larger population size without a loss in time taken to run the algorithm. If the transfer between the two algorithms is more complex, for example by seeding the initial population of the new algorithm with subtrees of the old one, then potentially the transfer will send more generalisable information.
- The constraint methods used in this project are rather crude, as if using a sledgehammer to crack a nut. Future work could investigate the concept of constraining the individuals one by one such that they better fit the properties of a distance metric, ensuring that the population will only contain reasonably valid solutions. This could give the benefits of the constraint methods presented in this project, without the large downsides.
- The algorithm presented has currently only been tested on a rather small sample of algorithms and datasets. It would be interesting to try the algorithm on non-ellipsoidal datasets, as it may be that the feature space modifying properties of this method will produce high performance results.

5.3 Effects of COVID-19

Much of the impact of COVID-19 on this project has been in the intangible effect. All of the code could be written from my home computer and was run remotely on the NeSI computation grid, so I wasn't physically effected by the pandemic. However, the university has handled the pandemic in a suboptimal way, resulting in higher physical and mental workload throughout the year.

Due to the absence of trimester 2 exams, course coordinators have taken it upon themselves to add extra assessment to courses throughout the trimester. However, I have found that a lot of this added assessment is extremely bottom heavy, and has all been added to the tail end of courses. This, compounded with the fact that the "exam period" has been shortened to a single week, means that the workload in the last few weeks of trimester is far too large for what should be a time to finish off 489 for most of the 400 level cohort. Where usually the exams would be spread out, I now have 3 full day tests in a row during the exam period when I should be working on a final project and preparing for the 489 presentation.

An additional effect of the requirement for students to be able to work from home is that a lot of tests have been turned into more difficult 24 hour versions of what would have been

a single 1 hour test. However, I have found that this has added a lot of mental toll onto the tests. Where previously I could spend half a day studying, take the test, and still be fresh for more work (such as on 489), the added difficulty due to 24 hour tests means that I have been spending entire days just working on tests. This not only means less time to be spent on other work, it is a much larger drain on mental health than a straight test or exam would be.

Bibliography

[1] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, 2nd ed. Springer series in statistics, Feb. 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," Jan. 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," Oct. 1997, pp. 314–319.

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

[8] 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.

[9] S. Ontañón, *An overview of distance and similarity functions for structured data*, 2020.

[10] S. Theodoridis and K. Koutroumbas, "Chapter 11 - clustering: Basic concepts," in *Pattern Recognition (Fourth Edition)*, S. Theodoridis and K. Koutroumbas, Eds., Fourth Edition, Boston: Academic Press, 2009, pp. 595–625.

[11] J. Han, M. Kamber, and J. Pei, "2 - getting to know your data," in *Data Mining (Third Edition)*, ser. The Morgan Kaufmann Series in Data Management Systems, J. Han, M. Kamber, and J. Pei, Eds., Third Edition, Boston: Morgan Kaufmann, 2012, pp. 39–82.

[12] F. Rahutomo, T. Kitasuka, and M. Aritsugi, "Semantic cosine similarity," Oct. 2012.

[13] M. Hossain and S. Abufardeh, "A new method of calculating squared euclidean distance (sed) using ptree technology and its performance analysis," in *Proceedings of 34th International Conference on Computers and Their Applications*, G. Lee and Y. Jin, Eds., ser. EPiC Series in Computing, vol. 58, 2019, pp. 45–54.

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

[15] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, Berkeley, Calif.: University of California Press, 1967, pp. 281–297.

[16] 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, New Orleans, Louisiana: Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035.

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

[18] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. USA: Cambridge University Press, 2008.

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

[20] 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, Portland, Oregon: AAAI Press, 1996, pp. 226–231.

[21] E. Hartuv and R. Shamir, "A clustering algorithm based on graph connectivity," *Information Processing Letters*, vol. 76, pp. 175–181, Dec. 2000.

[22] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "Optics: Ordering points to identify the clustering structure," in *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '99, Philadelphia, Pennsylvania, USA: Association for Computing Machinery, 1999, pp. 49–60.

[23] R. J. G. B. Campello, D. Moulavi, A. Zimek, and J. Sander, "Hierarchical density estimates for data clustering, visualization, and outlier detection," *ACM Trans. Knowl. Discov. Data*, vol. 10, no. 1, Jul. 2015.

[24] J. Handl, J. Knowles, and D. Kell, "Bioinformatics computational cluster validation in post-genomic data analysis," *Bioinformatics (Oxford, England)*, vol. 21, pp. 3201–12, Sep. 2005.

[25] 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, Nov. 1987.

[26] 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.

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

[28] P. A. Vikhar, "Evolutionary algorithms: A critical review and its future prospects," in *2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC)*, 2016, pp. 261–265.

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

[30] K. Demir, B. H. Nguyen, B. Xue, and M. Zhang, "A decomposition based multi-objective evolutionary algorithm with relieff based local search and solution repair mechanism for feature selection," in *2020 IEEE Congress on Evolutionary Computation (CEC)*, 2020, pp. 1–8.

[31] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 4, pp. 577–601, 2014.

[32] A. Mukhopadhyay, U. Maulik, S. Bandyopadhyay, and C. A. C. Coello, "A survey of multiobjective evolutionary algorithms for data mining: Part i," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 1, pp. 4–19, 2014.

[33] Q. Liao, Z. Sheng, H. Shi, L. Zhang, L. Zhou, W. Ge, and Z. Long, "A comparative study on evolutionary multi-objective optimization algorithms estimating surface duct," *Sensors (Basel, Switzerland)*, vol. 18, no. 12, p. 4428, Dec. 2018.

[34] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[35] 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.

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

[37] 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.

[38] 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.

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

[40] 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, Prague, Czech Republic: Association for Computing Machinery, 2019, pp. 463–471.

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

[42] 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, Jul. 2012.

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

[44] Q. Zhang and H. Li, "Moea/d: A multiobjective evolutionary algorithm based on decomposition," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 6, pp. 712–731, 2007.

[45] N. Choudhary, B. Singh, and G. Bagaria, "Genetic programming: A study on computer language," 2014.

Appendix A

Best evolved dissimilarities for k-means++

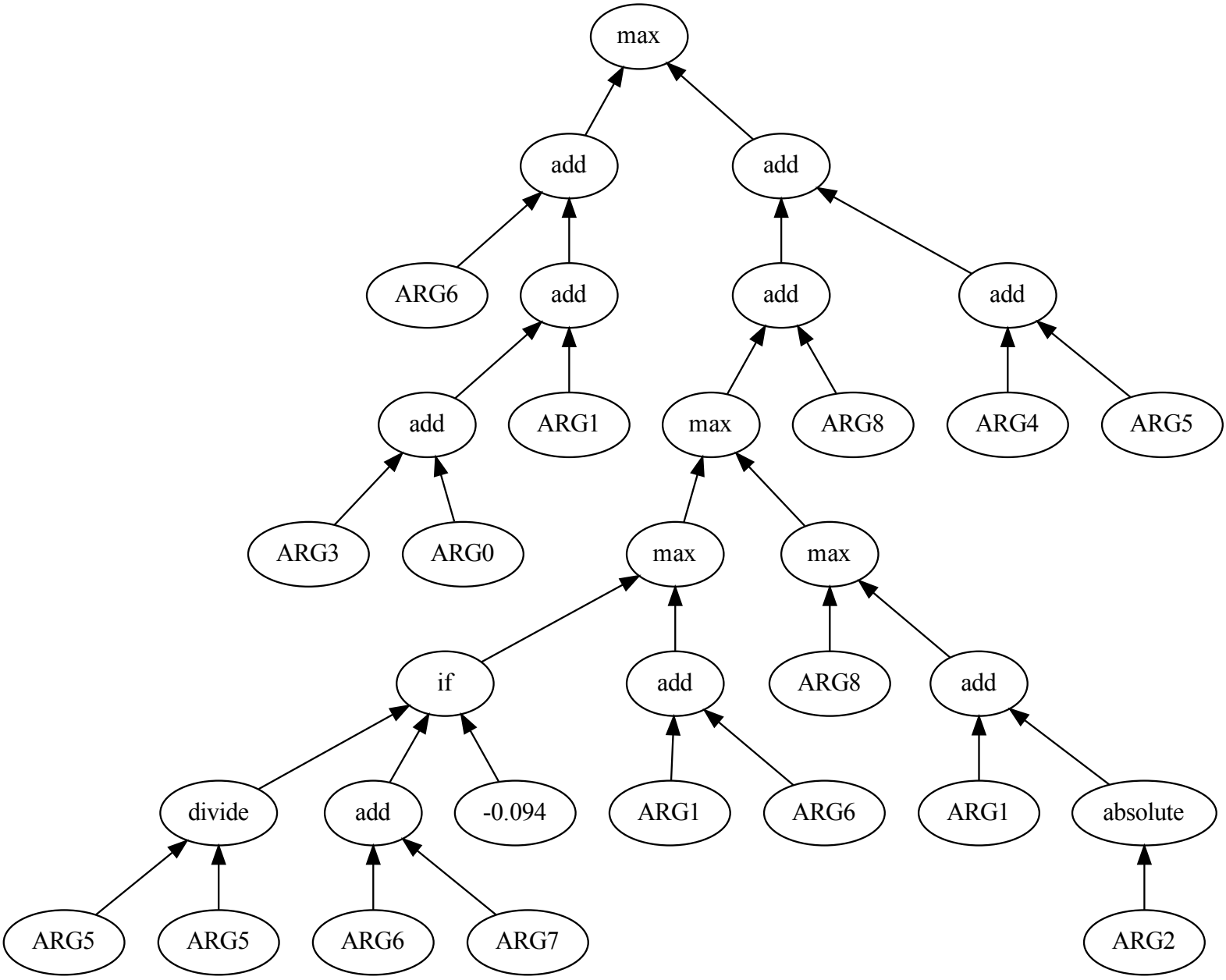


Figure A.1: Best dissimilarity function for 10d-10c dataset

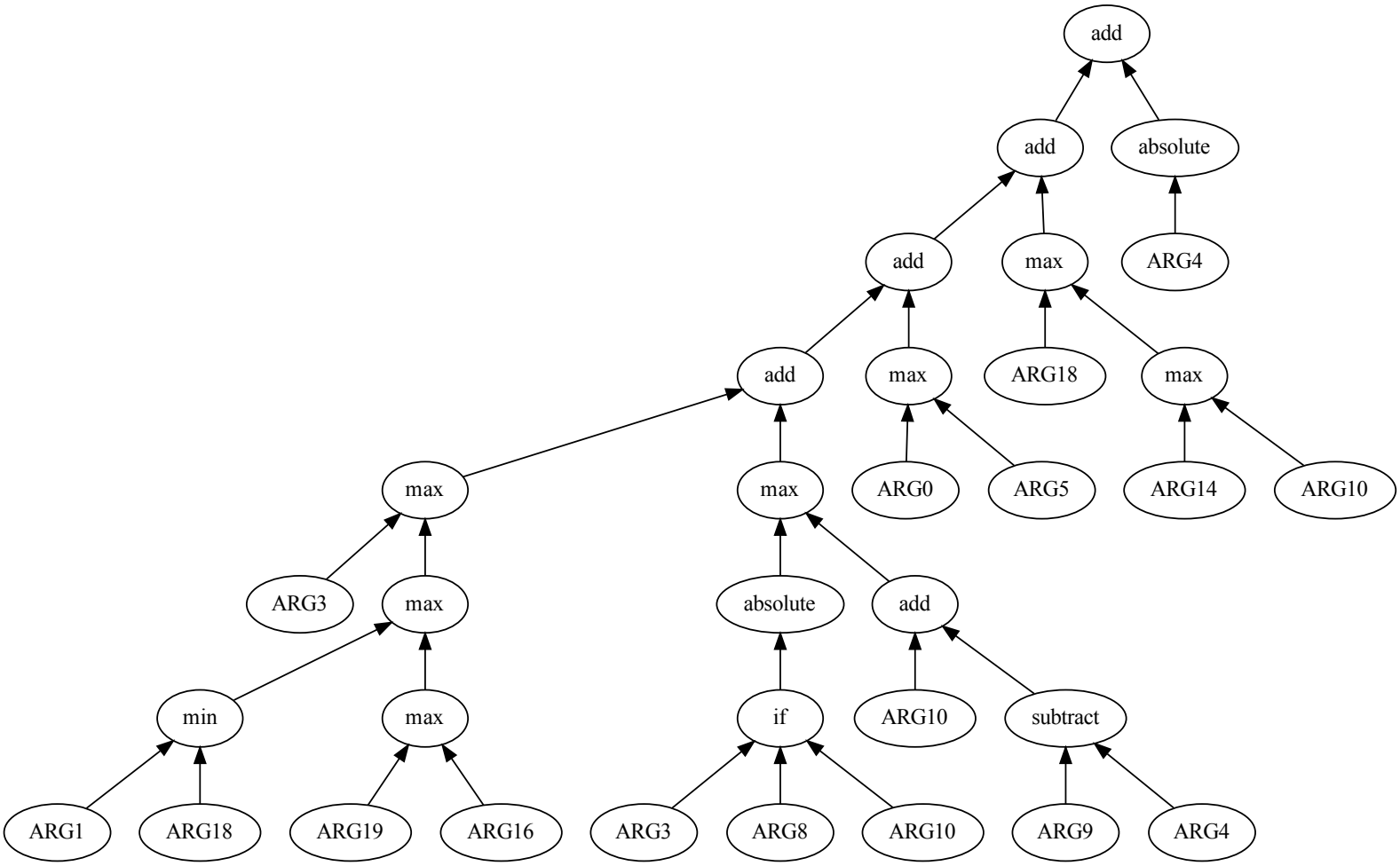


Figure A.2: Best dissimilarity function for 20d-20c dataset

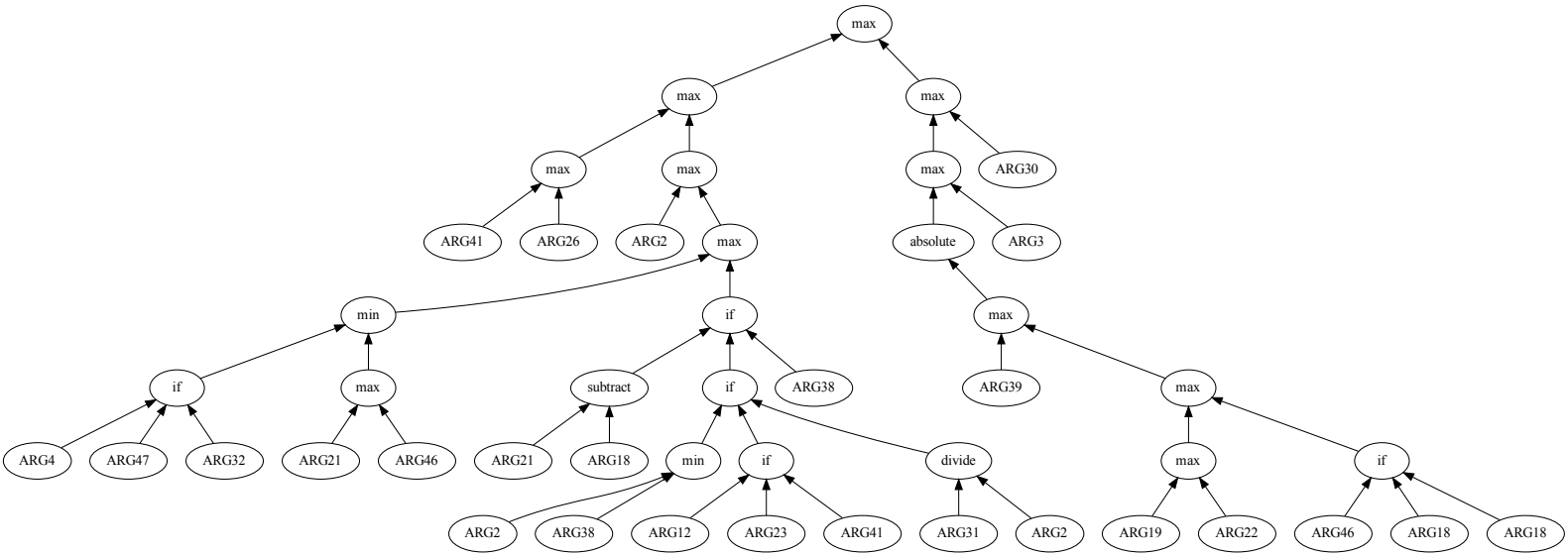


Figure A.3: Best dissimilarity function for 50d-10c dataset

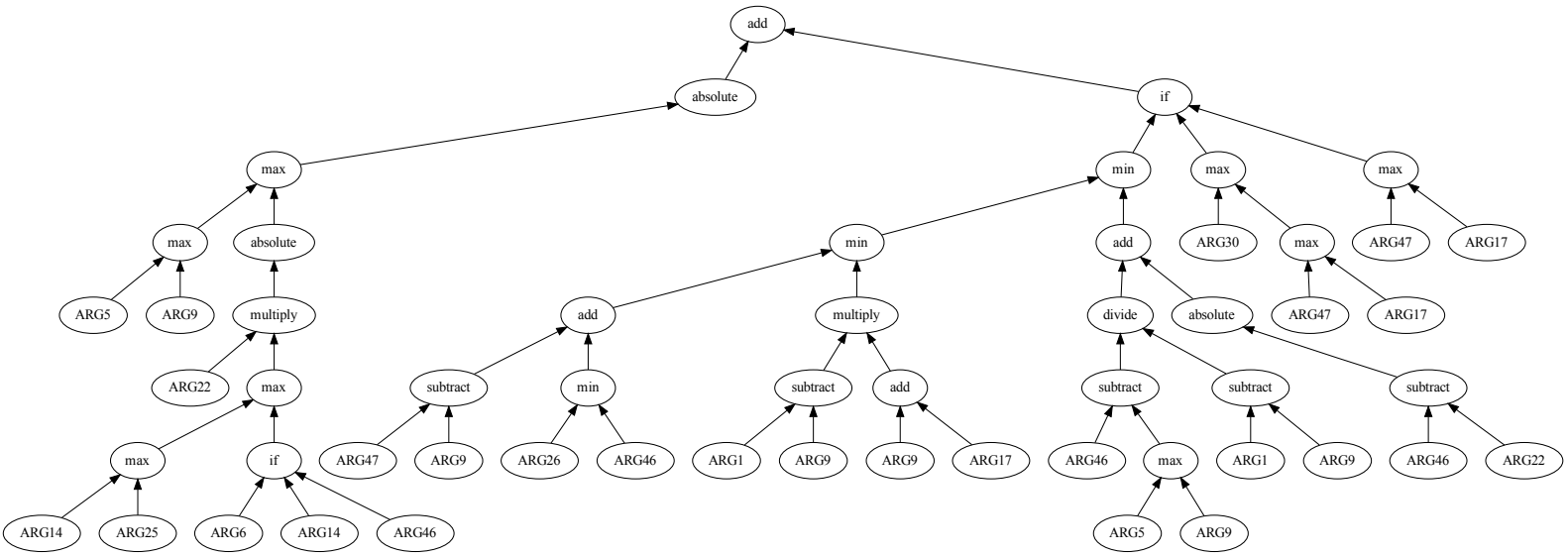


Figure A.4: Best dissimilarity function for 100d-10c dataset