4-29-2022

Identification of Orthologous Gene Groups Using Machine Learning

Dillon Burgess
University of Nebraska-Lincoln, dillon.burgess@huskers.unl.edu

Identification of Orthologous Gene Groups Using Machine Learning

by

Dillon Burgess

A THESIS

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfillment of Requirements

For the Degree of Master of Science

Major: Electrical Engineering

Under the Supervision of Professor Hasan H. Otu

Lincoln, Nebraska

May, 2022

Identification of Orthologous Gene Groups Using Machine Learning

Dillon Burgess, M.S.

University of Nebraska, 2022

Adviser: Hasan H. Otu

Identification of genes that show similarity between different organisms, a.k.a orthologous genes, is an open problem in computational biology. The purpose of this thesis is to create an algorithm to group orthologous genes using machine learning. Following an optimization step to find the best characterization based on training data, we represented sequences of genes or proteins with kmer vectors. These kmer vectors were then clustered into orthologous groups using hierarchical clustering. We optimized the clustering phase with the same training data for the method and parameter selection. Our results indicated that use of protein sequences with k=2 and scaling the data for each kmer provided the best results. We employed Pearson's correlation as the distance metric and used complete linkage in the agglomeration step. The number of clusters are calculated based on four different approaches that evaluates optimum number of clusters. This algorithm was pitted against OrthoDB which is an orthologous gene grouping algorithm that has been proven to work well. The results show that when small datasets were used, our algorithm performed better than OrthoDB. When larger genome-level datasets were used, OrthoDB outperformed our algorithm as long as the input data to OrthoDB was divided based on organism count. Our algorithm has an advantage over OrthoDB in that the data doesn't have to be divided by organism; it can be given as one file. The proposed algorithm runs much faster than OrthoDB and is the first approach, to the best of our knowledge, that uses unsupervised machine learning techniques that does

not rely on sequence alignment or phylogeny to identify orthologues genes. Overall, our algorithm provides a novel solution that is fast, practical, and unlike existing approaches can be applied to data sets such as metagenomics where the underlying number of organisms is unknown.

Acknowledgements

I would like to thank Dr. Hasan Otu for all of his guidance throughout this whole process as well as Karla, Brad, and Keaton Burgess for all of their unconditional support throughout this thesis writing process. I would also like to thank the committee, Dr. Khalid Sayood and Dr. Andrew Harms for their input and for taking the time to read and listen to my thesis presentation.

Table of Contents

List of Tables

List of Figures

List of Equations

1 CHAPTER 1: Introduction

Life is all around us here on Earth. And what better way to gain a better understanding of life than to dive deep into the building blocks of life. This would primarily mean the deoxyribonucleic acid (DNA) molecule that contains the hereditary information for living organisms. DNA helps code for everything that goes on inside of an organism. It does this by having four distinct bases that make it up that serve as the pieces of the code. These four bases are adenine (A), guanine (G), cytosine (C), and thiamine (T). DNA is a double helix structure containing these four bases used to code for different processes and characteristics of the organism it is within. Functional parts of DNA, referred to as "genes," are copied onto the molecule called "messenger ribonucleic acid" (mRNA) through a process called "transcription." This single-stranded mRNA molecule is then processed to code for amino acids which are linked together to create proteins that carry out specific functions within an organism. Genes serve many functions from determining one's susceptibility to diseases to something as simple as how tall someone will grow or the color of their eyes. Studying genes and their regions that code for these traits in organisms can be vital to understanding life.

Through sequencing technologies, we have been able to identify the DNA content of different organisms and study them in detail. Such studies entail a wide range of topics that include gene identification, mutation detection, and motif discovery among others. One important question is identifying sequence similarity between different gene sequences [1]. These genes may come from the same organism, or they may belong to different organisms. The main motivation behind this has been the biological premise that

"forms fit function" [2]. The manifestation of this principle in DNA sequence analysis is that genes with similar sequences may have similar functions in the cell. Discovering such similarities may help us reconstruct evolutionary histories or understand biological function and mechanisms.

1.1 Orthology

Similarity in biological context is defined by the term "homology," which is the idea that similarity of genes can be linked to the fact that they are from a common ancestor. Orthologous genes are homologous genes that are inherited in different species but still have a common ancestor, while paralogous genes are homologous genes that are inherited in the same species and have a common ancestor. Studying homology may help us learn about unknown genes. If one knows about the make-up and function of a gene, and an unknown gene is known to be orthologous to the known gene, one can assume that the make-up and function of the unknown gene is similar to that of the known gene. This can help one learn about genes that they are unfamiliar with and their function.

In **Figure 1**, we show an example of ortholog and paralog genes. The early globin gene goes through a gene duplication and gives rise to alpha- and beta-chain globin genes. There are slight compositional and length differences between the two chains, which along with two other subunits, come together to form the hemoglobin protein [2]. As shown in the figure, the two alpha-globin genes in different organisms, e.g., frog and mouse, make an orthologous gene pair. Together with the alpha-globin gene in the chicken, we can consider these three genes forming an orthologous gene group.

Similarly, the three beta-globin genes in the three organisms form another orthologous gene group. Conversely, the alpha-globin and beta-globin genes in the same organism form a paralogous gene pair. For example, the alpha-globin and beta-globin genes in mice are paralogous.

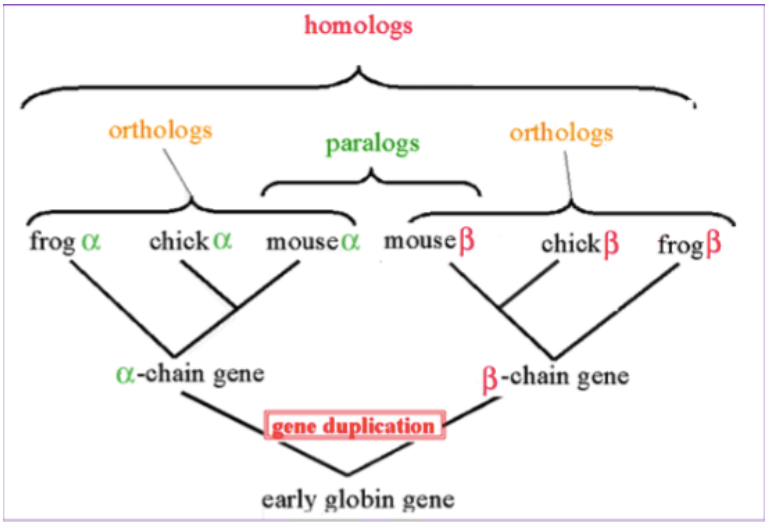

***Figure 1**: Example of homologs, orthologs, and paralogs.*

With the advent of sequencing techniques, we have a plethora of molecular sequences that can be used to develop computational tools to identify orthologous gene groups. This problem can be tackled in two ways. First, one can start building a database of orthologous genes, and update this as new genome or gene sequences become available. Although such an approach provides a big-picture view and often attracts non-computational scientists for its ease-of-use, it runs into the problems of being bulky, hard to navigate, and either miss or make it difficult to infer orthologous pairs inherited by distant species, e.g., thorough horizontal gene transfer [1].

Another way is to obtain the orthologous groups given a set of gene sequences "on the fly." Methods that follow this approach has the advantage of creating dynamic results for different sets of genes but suffer from huge computational time and space requirements when tried for large sets of sequences. As we describe in the following chapter, most existing approaches to define orthology follow the former approach with options to sift subgroups of interest. In this thesis, we develop a method in the latter class of approaches differing from existing ones by not relying on sequence alignment to find sequence similarity, which uses evolutionary assumptions and often produces ambiguous results [1].

1.2 Organization of Thesis

In this thesis we used alignment-free machine learning approaches to identify the orthologous gene groups given a set of molecular sequences. We first characterize the gene (or protein) sequences using mathematical modeling so that they can be used as the input to the employed algorithms. In Chapter 2, we provide a background by summarizing existing approaches for orthologous gene identification and describe the steps of the proposed approach. In Chapter 3, we will describe the tested sequence characterizations and run our models on known orthologous groups. This training phase enabled us to find the optimum model for the final workflow. In Chapter 4, we further tested the characterizations with various clustering approaches, similarity metrics, and data pre-processing methods. This resulted in separating a given set of gene (or protein) sequences into their respective orthologous groups and finding the optimum number of such groups. In Chapter 5, our proposed method is compared to other orthologous gene finding approaches using known orthologous gene group sets. Finally, Chapter 6 includes

discussion on obtained results, conclusive remarks, and points to future research directions.

2 Chapter 2: Background

Initial approaches to identify orthologous genes used the evolutionary tree represented by the species that the genes come from [3]. However, since phylogenetic trees rely on models that are often disputed and may not always be readily available for species under investigation, recent approaches use direct whole genome comparisons. These approaches often rely on best reciprocal hits (BRHs). Assume we are comparing gene or protein sequences between two genomes. Let $S_{1,1}$, $S_{1,2}$, ... , $S_{1,N1}$ denote the sequences in the first genome and $S_{2,1}$, $S_{2,2}$, ... , $S_{2,N2}$ denote the sequences in the second genome. Assume the sequence $S_{1,i}$ of the first genome is compared to all of the sequences in the second genome for similarity and the most similar sequence in the second genome is found to be $S_{2,j}$. Now, the procedure is reversed and the sequence $S_{2,j}$ in the second genome is compared to all of the sequences in the first genome for similarity. If $S_{1,i}$ makes the top of the list in the latter comparison, then $S_{1,i}$ and $S_{2,j}$ are called BRHs [4]. Following such all-against-all comparisons between two genomes, BRHs or "near BRHs" are identified to define orthologous gene groups.

2.1 OrthoDB

OrthoDB [5] performs an all-against-all comparison between the protein sequences in the two genomes using local alignment tools such as ParAlign [6] or BLAST [7]. Then, BRHs are clustered based on an e-value cut-off of 10^{-6}. The e-value is a parameter that tells the number of hits to expect to see by change when searching a database of a particular size. An e-value of 1 means that one might expect to see 1 match with a similar score simply by chance. The lower the e-value, the more significant the match [8]. If a

protein sequence did not have any BRHs but showed a similarity to a clustered BRH group with an e-value less than 10^{-10}, then this sequence is included in the cluster. An overlap of 30 amino acids is also required among the members of a cluster. If two proteins in a genome show similarity to each other that is more than their similarity to any sequence from other species and that shows over 97% identity, then they are included in the same cluster. Triangulation is applied to expand the clusters to other species and final orthologous groups are thus defined.

2.2 KEGG Orthology (KO) Database

Kyoto Encyclopedia of Genes and Genomes (KEGG) is a suite of bioinformatics tools and databases that is most popular for its pathway repository [9]. An important feature of the KEGG database is the so-called "K numbers" that define orthologous gene groups. First, a sequence similarity database (SSDB) is constructed, which is obtained by pairwise genome comparisons for gene and protein sequences and is stored as a graph. The edges represent best-hits and are weighted by the similarity scores between sequences. The KEGG Orthology And Links Annotation (KOALA) algorithm clusters this graph to identify "cliques" that are based on the weighted sum of Smith-Waterman (local alignment) scores, length of the overlap, ratio of sequence lengths, and common domains across the sequences. New genomes are added to the KO database using the BlastKOALA and GhostKOALA tools [10].

2.3 Cluster of Orthologous Groups (COG)

The COG approach relies on analysis of three genomes at a time with triangulation using BRHs [11]. If triangulated groups have overlapping gene or protein sequences, then the groups are converged. The main difference introduced in COG was the use of flexible similarity cut-offs that changed for different protein families [12]. If the final COGs are too large, a manual splitting is applied based on identification of conserved domains among the members of the COGs.

2.4 eggNOG Database

The eggNOG (evolutionary genealogy of genes: Non-supervised Orthologous Groups) database [13] treats a collection of protein sequences concurrently, without focusing on pairs of genomes. The sequence similarity is guided by known species similarity where the seed orthologous groups are initially formed in clades of similar organisms. This is extended in a hierarchical fashion where new subsets of organisms are either introduced or combined with more relaxed cut-off criteria.

2.5 Broccoli

The Broccoli algorithm is the most similar to what we are going to produce [14]. Broccoli performs orthologous gene grouping in four steps: kmer preclustering, similarity searches and phylogenetic analyses, identification of orthologous groups, and identification of orthologous pairs. For the first step of kmer preclustering, Broccoli converts the protein names into unique identifiers. The proteome of each species is clustered using kmers of amino acids. In each cluster, the longest is kept for further analysis. This is done to try to

reduce the number of proteins to be analyzed by removing allelic variants and dupicates. Kmer size is set to 100. This prevents paralogs being grouped between closely related species. Kmer size can be reduced when more distantly related species are analyzed. In step two, a phylogenetic tree is then constructed for each species by comparing its proteins against other proteomes and performing phylogenetic analyses in possible cases of gene duplications. Searches for similarity are individually performed using DIAMOND and the N best hits per species are given. DIAMOND is a high-throughput alignment program that compares a file of DNA sequencing reads against a file of protein reference sequences, such as NCBI or KEGG [15]. All hits are considered orthologs if no species have multiple hits. In step three, an orthology network is built from which orthologous groups are isolated using a machine learning algorithm. Orthologous groups are delineated using a relaxed species overlap approach. Two sets of leaves are orthologous if there is no common species between the two sets or there is only one common species an at least two unique species in both sets. For step four, a new set of orthologous relationships is built that considers gene duplication events within each orthologous group. The method is the same as step three but with a couple of differences: proteins not belonging to the orthologous group are removed, and orthologous and paralogous pairs are built at each tested node from the rooted trees. Then for each pair of proteins A and B belonging to the orthologous group, a ratio R(AB) is calculated by the following formula:

$$R(AB) = \frac{ortho(AB)}{ortho(AB) + para(AB)}$$

Equation 1: Ratio between the number of times two proteins A and B have been found as orthologs and paralogs

Where ortho(AB) and para(AB) indicate the number of times A and B are found as orthologs and paralogs, respectively. They will be considered orthologs if their ratio is higher than a predetermined threshold usually set to 0.5 [14].

2.6 Proposed Approach

In this thesis we introduce a new approach to find orthologous gene groups that utilizes methods rooted in machine learning. Unlike the method detailed for Broccoli or other similar algorithms that produce *de novo* orthology assignments, we do not use best-reciprocal-hits, sequence alignment, or phylogenetic approaches, which not only rely on evolutionary assumptions and produce ambiguous results, but also have huge time and space requirements that are not suitable for large data sets. We begin with characterizations of the sequences using mathematical models, which frees us from alignment or phylogeny-based similarity assessments that may be biased by the underlying evolutionary model assumptions. Unlike existing methods, our approach does not require whole genome representations but can identify orthologous gene groups for any given set of sequences.

Using classification accuracy as a metric, we found the best sequence characterization method based on training groups of orthologous genes. Then, clustering algorithms were employed to find the optimum method and measure that can separate the training orthologues gene groups. We employed approaches that can identify the optimum number of clusters automatically. Given a set of molecular sequences we utilized a two-

pronged approach. If the estimated number of clusters were small, we used each possible cluster number for evaluation. If the estimated number of clusters were too big, we used a step-size to evaluate different number of clusters. We used NCBI's HomoloGene database for small orthologous gene groups in our training and test phases. For large data sets, we used COG's organism-level orthologous gene groups in our test phase. Our results were compared with OrthoDB, the only database that provided a standalone tool to identify orthologous gene groups.

3 CHAPTER 3: Sequence Characterization

The first step in creating an algorithm to do orthologous gene grouping was to find a program to handle the task. MATLAB was decided on as it has a variety of toolbox features that could be used in building the algorithm. For the first phase, where we needed a classifier for evaluation, we opted for a neural-net approach. MATLAB's Deep Learning Toolbox ended up being perfect for this phase as it had the ability to choose a type of neural network, set the number of neurons in the hidden layer, take inputs and expected outputs, and train and test the neural network. The algorithm for this was done by reading in FASTA files either in the form of mRNA or proteins. These FASTA files were obtained from NCBI's HomoloGene database. This is an online database that contains homologous gene groups across numerous species. It has gene groups with varying numbers of genes in them.

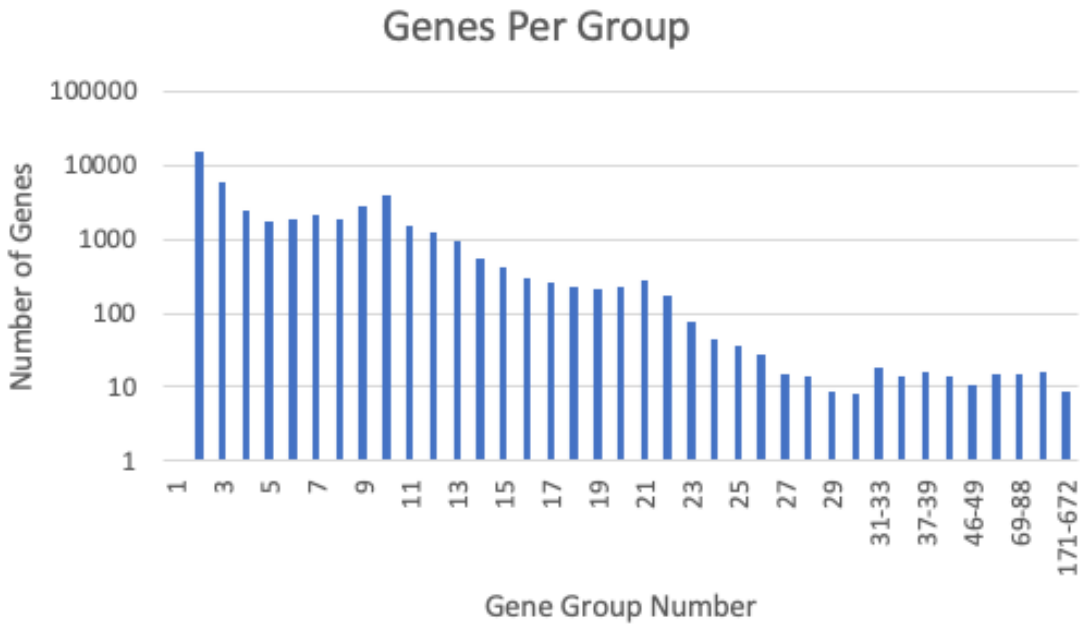

***Figure 2**: Distribution of the number of genes across the HomoloGene database.*

The distribution of the number of genes in the HomoloGene database gene groups is given in **Figure 2**. Out of the 44,233 gene groups that were queried, only 21 had more than 100 genes in them. The majority of the HomoloGene gene groups had four or fewer genes where 55 groups had more than 50 genes and there were 128 groups with more than 30 genes in them. Our first goal was to sample different sets of groups from the HomoloGene database and perform k-fold cross validation on the members of the gene groups using an artificial neural network (ANN) model. We used different characterizations of the sequences to see which one gave the best accuracy. We applied our test workflow on both the mRNA and protein sequences for the genes in the HomoloGene groups. For mRNA sequences, the HomoloGene database provided the coding DNA segment for the gene on the genome.

3.1 Artificial Neural Networks

Artificial neural networks (ANNs) are networks that can be used to predict the output of a function given a certain input. ANNs work by first taking an input and running it through a predetermined function controlled by weights and offsets. This function is in the hidden layer which contains a predetermined number of neurons. The inputs run through the function in the hidden layer controlled by the weights and offsets. This then gives an output which can be compared to the desired output. The difference between the output and the desired output can then be used to adjust the weights and offsets in the hidden layer function, and the inputs can be run through again. This continues until the weights and offsets are set in such a way that the desired output and the real output are as close as possible. This process of finding the correct weights and offsets is called training. Once training is complete, the weights and offsets are set to their optimal value, and the ANN is ready to be used to predict the output when certain input is put into the ANN. This can be used to find orthologous genes. The gene sequences can be used as the input, and the group they should be in can be used as the output. The ANN can then be trained to figure out what orthologous gene group certain genes should be in. The overall workflow used in this phase is summarized in **Figure 3**.

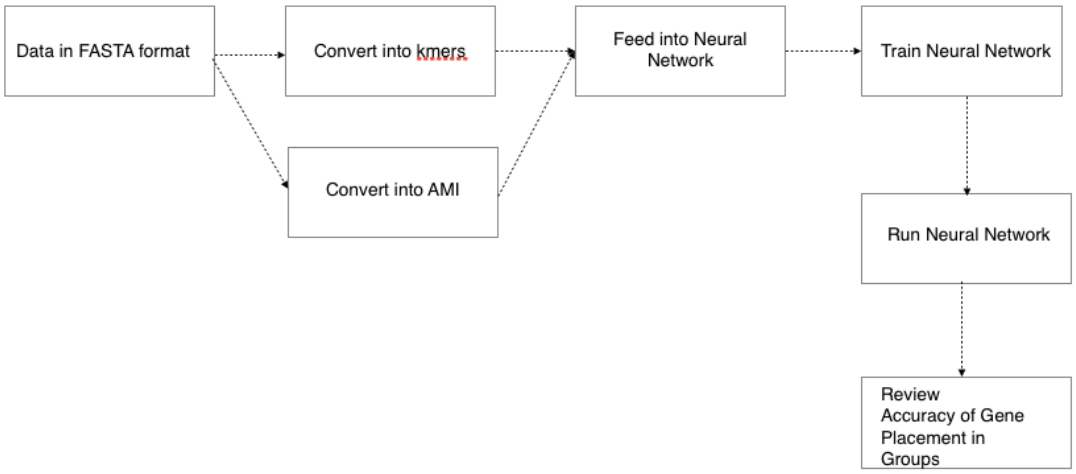

Figure 3: The track of the algorithm to test sequence characterizations.

3.2 Sequence Characterization Using kmer Vectors

Random HomoloGene groups with either the same or different numbers of genes in them were selected to test our approach. This was done to test the robustness of the sequence characterization methods. The test data sets were named based on the number of groups and the number of genes in each group. Our first characterization model was the kmer approach. Each gene in each group would be read in and analyzed to turn it into a vector of numbers that represented the numbers of different kmers in the gene. A kmer is a short string of nucleotides or amino acids (AA, ACG, CTGAT, etc.). The "k" in kmer is the number of nucleotides or amino acids in the string. The amount of each kmer found in each gene is represented as a vector and entered into the neural network as the main input. In **Table 1**, we show a sample kmer vector used as input in our test workflow. This example was obtained for an mRNA entry in the HomoloGene database. Since the database provides the corresponding coding DNA segment, we have the base 'T' instead

of ‘U’ that is found in RNA as the substitute for ‘T’. This example represents the case for k=2, and as there are 4 nucleotides, the vector has $4^k=4^2=16$ entries. The other input of the neural network is the expected output which is the original group number the gene belongs to. This group number is given in the form of a 1 in position of the group number and 0s in the other positions. For example, if there are five groups and the gene is expected to be in group two, it would be inputted into the neural network as a 01000.

GC	162
GG	150
CC	141
TG	120
CT	111
CA	108
AG	101
CG	81
GA	73
TC	71
GT	67
AC	66
AT	33
AA	29
TT	27
TA	20

***Table 1**: An example of a kmer vector that would be inputted into the algorithm. In the example, an mRNA sequence is used with k=2.*

After all the inputs were created they were passed to the neural network for 10-fold cross validation. This would then give a percentage accuracy which would give the user an idea of how successful the program was.

A test was run with different numbers of k and different numbers of neurons in the hidden layer. It was found that the optimal number of neurons in the hidden layer was 50 and the optimal values for k were 2 for protein kmers and 4 for mRNA kmers. A test was run with these parameters to find out the accuracy with 16 different datasets. The accuracy was described as the percentage of genes that were put in the correct groups. To get more accurate results, the algorithm was run 10 times and the average accuracy percentage was taken as the final accuracy. The results of the program are given in **Table 2** below. The next step is to do all of this again but with Average Mutual Information (AMI) profiles instead of kmers.

3.3 Sequence Characterization Using AMI Profiles

The average mutual information between two bases that are 'r' positions apart in a molecular sequence is defined as:

$$I(r) = \sum_{i,j} p_{ij}(r) log_2(\frac{p_{ij}(r)}{p_i p_j})$$

Equation 2: Average Mutual Information equation

where p_i is the probability of occurrence of symbol i and $p_{ij}(r)$ is the joint probability of observing symbol i and j separated by a distance r. I(r) is the amount of information symbol i carries about symbol j at a distance r. For an independent identically distributed sequence, I(r) = 0 for all r > 0. AMI has been used on DNA sequences to find coding regions of these sequences and to investigate statistical correlation of nucleotides [16]. AMI profiles have also been used to analyze evolutionary history of DNA sequences [17]. AMI vectors were in the form [I(1), I(2), … , I(R)]; we used R=16 to generate the

AMI vectors for the mRNA and protein sequences, which were subsequently run in the ANN classifier to produce 10-fold cross-validation accuracies for the 16 different HomoloGene test groups.

We also combined the two characterizations where we appended the kmer and AMI vectors for each sequence and applied the 10-fold cross validation approach using ANNs. The result for all three cases (kmer, AMI, kmer+AMI) are shown in **Table 2**.

Number of Groups	Number of Genes	mRNA/Protein	kmer	AMI	kmer + AMI
5	12-20	mRNA	**86.67%**	44.29%	61.90%
5	12-20	Protein	99.52%	77.14%	**100.00%**
10	5	mRNA	**85.33%**	69.33%	73.33%
10	5	Protein	**99.33%**	89.33%	98.00%
10	10	mRNA	**86.67%**	59.00%	73.67%
10	10	Protein	**100.00%**	87.33%	**100.00%**
10	12-20	mRNA	**82.92%**	39.38%	49.17%
10	12-20	Protein	**99.38%**	71.88%	**99.38%**
10	15	mRNA	**81.11%**	41.33%	59.33%
10	15	Protein	**99.33%**	84.44%	98.44%
10	20	mRNA	**78.17%**	32.50%	51.50%
10	20	Protein	**99.33%**	65.17%	98.67%
15	12-20	mRNA	**80.14%**	33.75%	46.11%
15	12-20	Protein	**99.17%**	72.50%	99.03%
20	12-20	mRNA	**72.40%**	34.48%	57.40%
20	12-20	Protein	**98.65%**	67.19%	61.31%

***Table 2**: Accuracy of groups using kmer, AMI, and kmer + AMI methods.*

As can be seen in **Table 2** above, the use of kmer vectors produces the best accuracy in the algorithm in all but three cases, two of which it tied with kmer and AMI together.

AMI by itself never produces the best accuracy. This shows that kmer vectors should be used moving forward for the input to the algorithm in order to obtain the best accuracy.

4 CHAPTER 4: Delineation of Orthologous Groups

The next step was to try to create an algorithm that would group the orthologous genes by using a clustering algorithm. Three different clustering algorithms were tested to see which one would work the best for the application: hierarchical clustering, k-means clustering, and Gaussian mixture model clustering.

4.1 Hierarchical Clustering

Hierarchical clustering works by making every data point its own cluster, then constantly repeating two steps over and over. It identifies the two most similar clusters and then merges them into the same cluster. Similarity between clusters is found by computing a distance between clusters and the two with the smallest distance between each other are considered to be the most similar. These distances are often put into a distance matrix where each data point is placed along the first row and first column of a matrix and the distance between two points is placed in the intersection point in the matrix. Distance can be defined in many different ways depending on the application. It could be defined as city block distance, Euclidean distance, etc. The other aspect to consider in hierarchical clustering is linkage. This helps determine from where distance is computed. It could be single-linkage where distance is computed between the two most similar parts of the clusters. Another linkage option is complete linkage where the two least similar parts of the clusters are used to compute the distance. One could also do average linkage where the distance is computed based on the center of each cluster. A good default linkage when one cannot be determined is Ward's linkage. Ward's linkage works by reducing the sum

of squared distances of each observation from the average observation of each cluster and hence unfortunately only works with the Euclidean distance. All of these things put together make hierarchical clustering a viable option for this application [18].

4.2 K-Means Clustering

K-means clustering is another option for clustering in the algorithm. K-means clustering works by setting a number "k" which will act as the number of centroids. A centroid defines the center of a cluster. Often centroids will be randomly generated to start with. Each data point is placed in a cluster based on which centroid it is closest to. Then the average of each cluster is taken to find a new centroid for each cluster. Points are placed into clusters again based on which centroid it is closest to now, then the average is taken again and the centroids are moved again. This continues until some threshold is met or until the data points no longer change clusters after the average is taken. Once this is done, there will be k distinct clusters [19].

4.3 Gaussian Mixture Model

Gaussian mixture model (GMM) clustering is similar to k-means clustering in that it chooses k starting points for clusters and assigns points to them. Then it updates them until clusters converge. The difference is that the clusters are not defined by a sphere but by a Gaussian model [20]. The premise in GMMs is that the data is generated using a mixture of k different Gaussian distributions, k being the number of clusters. First, a model is fit to the observed data points such that the weight, mean, and variance of each Gaussian distribution as well as the covariance between different Gaussian distributions

are estimated. Then, each data point is considered to be generated by the mixture of the Gaussian distributions with different weights. The data point is assigned to the cluster represented by the Gaussian distribution that gets the largest posterior probability (weight or coefficient) for that observation.

4.4 Testing and Choosing Clustering Algorithm

All of these clustering algorithms were tested in order to see which one would work best for the application at hand. They were tested in MATLAB on the same data sets that were used in the previous chapter. For each dataset, the correct clusters were entered as a variable and an adjusted Rand index (ARI) was used to compare them to the clusters the algorithm put them in. The ARI outputs a score on a scale of 0 to 1 where 1 is perfect clustering. Each algorithm was tested with various different settings to find the one with the best ARI. Here, we describe the settings, which also appear in the column headers of the results that are shown in **Table 3**, **Table 4,** and **Table 5** below.

kmeansE: kmeans clustering with Euclidean distance

kmeansC: kmeans clustering with correlation distance

HCEW: hierarchical clustering with Euclidean distance and Ward linkage

HCCC: hierarchical clustering with correlation distance and complete linkage

HCCA: hierarchical clustering with correlation distance and average linkage

HCCS: hierarchical clustering with correlation distance and single linkage

GMM_PCA: Gaussian mixture model clustering with principal components (PCA).

The input to the clustering algorithms was a matrix where the kmer sequence characterizations (k=4 for mRNA and k=2 for protein) were the rows. For example, when we used 10 Homologene groups with 10 protein sequences in each, we used a kmer of size 2 so that each sequence was represented by a vector of length 400. Putting this in a matrix form implied that the input to the clustering algorithm was a 100x400 matrix where the rows were to be clustered into ten groups.

One problem we had was with the GMM clustering as this method cannot accept data that has more features (400 in the above example) than the data points (100 in the above example). Therefore, in the case of GMM, we used PCA dimension reduction with 2 PCs so the data matrix that was used as input was (number of sequences)x2 (100x2 in the above example).

We also applied row or column normalization to the data matrix used as input to the clustering algorithm. This is denoted as (RN) and (CN), respectively. In each case the data matrix is scaled either along its rows or columns such that the vectors (rows or columns) are transformed to have zero mean and unit variance.

Data Set	kmeansE	kmeansC	HCEW	HCCC	HCCA	HCCS	GMM_PCA
10GroupsOf10Protein	0.447	0.867	0.691	0.698	0.698	0.259	0.259
10GroupsOf10mRNA	0.309	0.332	0.315	0.231	0.187	0.003	0.356
10GroupsOf12-20Protein	0.486	0.839	0.824	0.678	0.661	0.182	0.338
10GroupsOf12-20mRNA	0.127	0.126	0.089	0.093	0.048	-0.001	0.075
10GroupsOf15Protein	0.689	0.814	0.777	0.838	0.678	0.220	0.422
10GroupsOf15mRNA	0.280	0.172	0.261	0.126	0.050	0.000	0.180
10GroupsOf20Protein	0.353	0.638	0.480	0.539	0.288	0.030	0.222
10GroupsOf20mRNA	0.156	0.093	0.182	0.063	0.026	0.000	0.119
10GroupsOf5Protein	0.454	0.538	0.360	0.712	0.641	0.454	0.286
10GroupsOf5mRNA	0.324	0.515	0.409	0.626	0.471	0.175	0.376
15GroupsOf12-20Protein	0.643	0.585	0.723	0.454	0.369	0.111	0.316
15GroupsOf12-20mRNA	0.129	0.099	0.130	0.062	-0.005	0.000	0.057
20GroupsOf12-20Protein	0.501	0.691	0.584	0.530	0.249	0.056	0.354
20GroupsOf12-20mRNA	0.139	0.084	0.151	0.067	0.020	0.000	0.103
5GroupsOf12-20Protein	0.791	0.683	1.000	0.737	0.737	0.187	0.675
5GroupsOf12-20mRNA	0.208	0.249	0.239	0.164	-0.005	-0.002	0.170

Table 3: ARI for each dataset with different clustering algorithms. Protein data sets are highlighted in yellow.

kmeansE (RN)	kmeansC (RN)	HCEW (RN)	HCCC (RN)	HCCA (RN)	HCCS (RN)	GMM_PCA (RN)
0.818	0.845	1.000	0.698	0.698	0.259	0.511
0.373	0.400	0.486	0.231	0.187	0.003	0.222
0.620	0.665	0.854	0.678	0.661	0.182	0.360
0.113	0.101	0.152	0.093	0.048	-0.001	0.060
0.859	0.764	0.778	0.838	0.678	0.220	0.486
0.137	0.186	0.231	0.126	0.050	0.000	0.089
0.627	0.736	0.942	0.539	0.288	0.030	0.280
0.055	0.087	0.102	0.063	0.026	0.000	0.037
0.655	0.792	1.000	0.712	0.641	0.454	0.484
0.511	0.450	0.768	0.626	0.471	0.175	0.484
0.633	0.611	0.739	0.454	0.369	0.111	0.311
0.077	0.093	0.149	0.062	-0.005	0.000	0.043
0.328	0.694	0.591	0.530	0.249	0.056	0.241
0.109	0.063	0.142	0.067	0.020	0.000	0.033
1.000	0.709	1.000	0.737	0.737	0.187	0.715
0.213	0.232	0.221	0.164	-0.005	-0.002	0.089

Table 4: ARI for each dataset with different clustering algorithms. Datasets have been row normalized. The data set order follows that of Table 3. Protein data sets are highlighted in yellow.

kmeansE (CN)	kmeansC (CN)	HCEW (CN)	HCCC (CN)	HCCA (CN)	HCCS (CN)	GMM_PCA (CN)
0.760	0.878	0.691	1.000	1.000	0.878	0.670
0.342	0.434	0.292	0.441	0.327	0.091	0.320
0.665	0.953	0.761	0.903	0.903	0.764	0.187
0.092	0.094	0.097	0.130	0.086	0.000	0.085
0.531	0.751	0.777	1.000	1.000	0.839	0.405
0.176	0.110	0.277	0.232	0.153	0.000	0.220
0.315	0.978	0.706	0.978	1.000	0.722	0.311
0.167	0.074	0.162	0.098	0.025	0.000	0.133
0.630	0.711	0.845	1.000	0.914	0.914	0.483
0.405	0.355	0.377	0.646	0.548	0.403	0.357
0.596	0.870	0.713	0.917	0.981	0.768	0.308
0.122	0.108	0.125	0.138	0.070	0.000	0.062
0.536	0.730	0.455	0.994	0.911	0.795	0.267
0.180	0.156	0.161	0.128	0.067	0.000	0.085
0.746	1.000	1.000	1.000	1.000	1.000	0.680
0.140	0.185	0.266	0.210	0.192	0.004	0.050

Table 5: ARI for each dataset with different clustering algorithms. Datasets have been column normalized. The data set order follows that of Table 3. Protein data sets are highlighted in yellow.

As can be seen in the tables above, the clustering algorithm that does the best is hierarchical clustering with correlation distance and complete linkage. Notice however that this only does noticeably well for protein sequences and not so much for mRNA sequences. This means moving forward the algorithm being built will use hierarchical clustering with correlation distance and complete linkage on protein sequences with 2-mer characterization of the sequences. The only challenge left is that the way the algorithm is currently, one must enter the number of clusters that the algorithm must put the data points into. This is not ideal as in the real world one may not know how many orthologous gene groups a set of genes needs to be divided into. We used four different evaluation metrics that are used to identify the appropriateness of the number of clusters. Each metric returns the optimal number of clusters for the application based on its

criterion. The different kinds of evaluation approaches used were Calinski-Harabasz, Davies-Bouldin, Gap, and Silhouette.

4.5 Calinski-Harabasz

$$CH_k = \left(\frac{BCSM}{k-1}\right)\left(\frac{n-k}{WCSM}\right)$$

Equation 3: Calinski-Harabasz Index where k is the number of clusters, n is number of records in data, BCSM calculates separation between clusters and WCSM calculates compactness within clusters

The Calinski-Harabasz index is a measure of how good clusters are. There will be a number of clusters that maximizes this value. The index is calculated by dividing the variance of the sums of squares of the distances of individual objects to their cluster center by the sum of squares of the distance between cluster centers. As the index increases, the clustering model gets better. As stated before, there is an optimal number of clusters that will maximize this value, which is what is being searched for [21].

4.6 Davies-Bouldin

$$DB = \frac{1}{n}\sum_{i=1}^{n} max_{j\neq i}\left(\frac{\sigma_i + \sigma_j}{d(c_i, c_j)}\right)$$

Equation 4: Davies-Bouldin Index where n is the number of clusters and σ_i is the average distance of all points in cluster i from the cluster center c_i

Davies-Bouldin is similar to Calinski-Harabasz, but the main difference is that the clustering algorithm gets better as the Davies-Bouldin index decreases. There will also be

an optimal number of clusters that will minimize this index value which is what is being searched for [21].

4.7 Gap

$$Gap_n(k) = E_n^*\{logW_k\} - logW_k$$

Equation 5: Gap statistic where Wk is the measure of compactness of the clustering based on the Within-Cluster-Sum of Squared Errors in Equation 5.

$$W_k = \sum_{x_i \epsilon C_k} \sum_{x_j \epsilon C_k} ||x_i - x_j||^2 = 2n_k \sum_{x_i \epsilon C_k} ||x_i - \mu_k||^2$$

Equation 6: Within-Cluster-Sum of Squared Errors (WSS)

Calculating the gap statistic involves the following: Cluster the observed data on various number of clusters and compute compactness of clustering. Generate reference data sets and cluster each of them with varying number of clusters. Calculate average of compactness of clustering on reference datasets. Calculate gap statistic as difference in compactness between clustering on reference data and original data. Again, there will be an optimal number of clusters based on this data [21].

4.8 Silhouette

$$S(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

Equation 7: The Silhouette Coefficient where b(i) is the smallest average distance of a point i to all points in any other cluster and a(i) is the average distance of i from all other points in its cluster.

The silhouette coefficient tells if points are assigned to the correct clusters. Using **Equation 7** above, the following rules can be used when using silhouette coefficient: S(i)

close to 0 means the point is between two clusters. If it is closer to -1, it would be better to assign it to other clusters. If it is close to 1, it is assigned to the correct cluster [21].

4.9 Picking Best Evaluation Method for Number of Clusters

Tests were run to see which method would deliver the best results for how many clusters the algorithm should cluster the data into. The results from these tests can be seen in **Table 6** and **Table 7** below.

Data Set	TRUE	CalinskiHarabasz	DaviesBouldin	Gap	Silhouette	Average
10GroupsOf10Protein	10	9	12	14	10	11.25
10GroupsOf12-20Protein	10	9	12	14	15	12.5
10GroupsOf15Protein	10	9	11	14	15	12.25
10GroupsOf20Protein	10	5	15	15	11	11.5
10GroupsOf5Protein	10	5	15	15	15	12.5
15GroupsOf12-20Protein	15	10	20	20	20	17.5
20GroupsOf12-20Protein	20	19	25	23	25	23
5GroupsOf12-20Protein	5	5	9	10	9	8.25

Table 6: Table showing actual number of clusters required versus how many each evaluation type returned.

Data Set	CalinskiHarabasz	DaviesBouldin	Gap	Silhouette	Average
10GroupsOf10Protein	1	4	16	0	1.5625
10GroupsOf12-20Protein	1	4	16	25	6.25
10GroupsOf15Protein	1	1	16	25	5.0625
10GroupsOf20Protein	25	25	25	1	2.25
10GroupsOf5Protein	25	25	25	25	6.25
15GroupsOf12-20Protein	25	25	25	25	6.25
20GroupsOf12-20Protein	1	25	9	25	9
5GroupsOf12-20Protein	0	16	25	16	10.5625
TOTAL	79	125	157	142	47.1875

Table 7: Table showing squared error of the number of clusters returned by each evaluation type as well as the average of all of them.

As can be seen above in **Table 7**, the evaluation type with the lowest squared error is the average of all of the evaluation types, so that is what will be used. To recap, the new algorithm will use 2mer characterization of protein sequences with hierarchical clustering with correlation distance and complete linkage, and in order to determine the number of clusters, the average of Calinski-Harabasz, Davies-Bouldin, Gap, and Silhouette methods will be taken.

5 CHAPTER 5: Comparative Application on HomoloGene and Whole Genome Data Sets

The next step was to compare the results from our algorithm to that of an already established algorithm out in a real world-like application. The selected algorithm to test against was OrthoDB (described in Chapter 2). OrthoDB is a little more specific about its data input than our algorithm. OrthoDB requires all data that is inputted into it to be separated into groups by organism. This is not the case for our algorithm. For ours, any data can be inputted in one big file and results will still be obtained.

5.1 HomoloGene Test

We first focused on the 8 HomoloGene protein data sets described and used in Chapters 3 and 4, which comprised of 1,290 total protein sequences. When the data was rearranged into groups based on organism and fed into OrthoDB, it grouped the data almost perfectly. The only issues it had were that certain proteins were not placed into a group at all. This happened on 7 different proteins out of 1290. Other than that, the groupings by OrthoDB were perfect. In order to be comparable to our results, we then applied OrthoDB to each HomoloGene data set separately. Since we observed that dividing the protein sequences by organism provides an unfair advantage to OrthoDB, we put all of the sequences in a HomoloGene data set into a file and input it to the software. For example, for the “10GroupsOf10Protein” data set, this meant combining all 100 proteins in one file, just like the input used for our algorithm. However, OrthoDB resulted in an error message reporting it needed multiple FASTA files as input.

At this point, for each HomoloGene data set, we tested OrthoDB using two different input files. One, where all of the sequences in the data set were divided into two FASTA files, another where all of the sequences in the data set were divided into N FASTA files. N represented the number of HomoloGene groups in the data set. In **Table 8** we show the number of clusters, number of unassigned proteins, and the adjusted Rand index results when OrthoDB was applied to two and N randomly generated FASTA files for each data set.

Data Set	OrthoDB (2)			OrthoDB (N)		
	ARI	NumC	NumNA	ARI	NumC	NumNA
10GroupsOf10Protein	0.0966	17	61	0.8489	11	9
10GroupsOf12-20Protein	0.0851	31	86	0.6108	24	12
10GroupsOf15Protein	0.091	30	77	0.7434	21	11
10GroupsOf20Protein	0.1241	35	95	0.6014	28	11
10GroupsOf5Protein	0.2448	10	26	0.8533	9	7
15GroupsOf12-20Protein	0.0832	44	131	0.7114	31	7
20GroupsOf12-20Protein	0.1008	60	170	0.8049	37	10
5GroupsOf12-20Protein	0.1304	12	36	0.7611	10	5

Table 8: Results of OrthoDB when data is randomly divided into 2 sets or N sets. NumC: number of identified orthologous gene clusters. NumNA: Number of proteins that are not assigned to any clusters.

For our algorithm, the ARI results are shown below in **Table 9**. The number of clusters used in our algorithm was decided as described in Chapter 4. Briefly, all four approaches to find the best number of clusters were applied to the dataset and their average was used as the final number of clusters. As can be seen in the table, our algorithm performed

better than OrthoDB, having ARI values mostly in the 90% range for accuracy. OrthoDB, while completely failing in the case of 2 FASTA files, provided reasonable ARI values in the case of N FASTA files. However, both the number of clusters and the number of unassigned proteins showed the inferior performance of OrthoDB on the HomoloGene data sets.

Data Set	Number of Clusters	ARI
10GroupsOf10Protein	11	0.9889
10GroupsOf12-20Protein	12	0.9572
10GroupsOf15Protein	12	0.9531
10GroupsOf20Protein	11	0.9534
10GroupsOf5Protein	12	0.9309
15GroupsOf12-20Protein	17	0.9527
20GroupsOf12-20Protein	23	0.956
5GroupsOf12-20Protein	8	0.852

Table 9: Number of clusters and ARI for each data set using the proposed method

5.2 COG Test

We also wanted to test the two approaches on whole genome datasets. For this purpose, we used the COG database described in Chapter 2 as the ground truth. The 8 organisms picked from the COG database picked are shown in **Table 10**. These are whole genome sets for different types of bacteria that show evolutionary similarity.

Assembly ID	Organism name
GCF_000010825.1	Acetobacter pasteurianus IFO 3283
GCF_000020425.1	Bifidobacterium longum sub infantis ATCC 15697
GCF_000020225.1	Akkermansia muciniphila ATCC BAA-835
GCF_000338115.2	Lactobacillus plantarum ZJ316
GCF_000016825.1	Lactobacillus reuteri DSM 20016
GCF_000011045.1	Lactobacillus rhamnosus GG ATCC 53103
GCF_000092505.1	Leuconostoc kimchii IMSNU 11154
GCF_000253395.1	Streptococcus thermophilus JIM 8232

Table 10: Assembly IDs and organism names for the 8 organisms used for the COG test.

The COG database identifies the COG ID the proteins in these bacterial genomes belong to. In order to obtain their sequences, we matched the protein accession numbers used in COG to their NCBI RefSeq assemblies, which are represented by GCF files. These GCF files were obtained from the NCBI whole genome database.

Organism name	#of Proteins with a COG ID	# of COGs	Total size of COGs	#of Proteins in the Assembly	Overlapping Proteins	# of Overlapping COGs	Size of Overlapping COGs	# of Represented COGs
Acetobacter pasteurianus IFO 3283	2096	1443	2234	2813	1912	1360	2044	1322
Bifidobacterium longum sub infantis ATCC 15697	1624	1053	1708	2446	1329	910	1402	878
Akkermansia muciniphila ATCC BAA-835	1514	1128	1606	2124	1116	893	1179	860
Lactobacillus plantarum ZJ316	2253	1269	2354	2981	2107	1226	2200	1205
Lactobacillus reuteri DSM 20016	1515	1061	1591	1868	1397	1017	1468	1001
Lactobacillus rhamnosus GG ATCC 53103	2023	1251	2114	2656	1962	1232	2045	1211
Leuconostoc kimchii IMSNU 11154	1575	1092	1635	2023	1522	1070	1579	1047
Streptococcus thermophilus JIM 8232	1368	1069	1425	1690	1262	1028	1315	1006

Table 11: Detailed statistics for the COG data sets..

Table 11 shows information about each bacteria's genome. The second column shows the number of proteins in the bacteria's genome with a COG ID. The next column shows the number of groups the proteins are divided into. The total size of the COGs is in the next column. Note that this number is larger than the number of proteins with a COG ID because the proteins can be in more than one group. Number of proteins in the assembly is the number of proteins in the GCF file. Overlapping proteins are the proteins that

overlap between the proteins with a COD ID and the proteins in the GCF file. Number of overlapping COGs is the number of COGs the overlapping proteins will be divided into. When the number of proteins in each group is added, the size of overlapping COGs column is achieved. Since some proteins are in more than one COG, when running the algorithm, one COG had to be chosen. With this in mind, the number of represented COGs is the number of COGs after only one COG is chosen per protein.

Table 12 shows the results after our algorithm was run on different numbers of organisms. As can be seen based on the ARI, our algorithm did not perform as well as we had hoped it to. The ARI was mainly in the 10-25% range which is not very good.

Organisms	#of Proteins	# of COGs	# of Clusters	ARI
1_2	3241	1621	1000	0.1217
1_2	3241	1621	1500	0.1359
1_2_3	4357	1819	1500	0.1372
1_2_3_4	6464	2137	1500	0.1112
1_2_3_4	6464	2137	2000	0.1225
1_2_3_4_5	7861	2199	2000	0.155
1_2_3_4_5_6	9823	2276	2000	0.1752
1_2_3_4_5_6_7	11345	2309	2000	0.1919
1_2_3_4_5_6_7_8	12607	2353	2000	0.1992
1_2_3_4_5_6_7_8	12607	2353	2250	0.2081
1_2_3_4_5_6_7_8	12607	2353	2500	0.2149
1_2_3_4_5_6_7_8	12607	2353	2750	0.2208
1_2_3_4_5_6_7_8	12607	2353	3000	0.2265
1_2_3_4_5_6_7_8	12607	2353	3250	0.2285
1_2_3_4_5_6_7_8	12607	2353	3500	0.2353
1_2_3_4_5_6_7_8	12607	2353	3750	0.2282
1_2_3_4_5_6_7_8	12607	2353	4000	0.2219
1_2_3_4_5_6_7_8	12607	2353	4250	0.221
1_2_3_4_5_6_7_8	12607	2353	4500	0.2194

Table 12: Results of the proposed algorithm on different organism combinations. Numbers 1-8 refer to the order of the organisms listed in Table 11. Best ARI results when all 8 organisms were used is highlighted.

Table 13 shows OrthoDB's results for the 8 bacterial genomes. It can be seen that it does a better job than our algorithm when the data is divided up by organism and when it is divided up into 8 random files. It does worse than ours when it is divided up into 2 random files.

Individual Organism Files	2 Randomly Divided Files	8 Randomly Divided Files
2078 OGs	2121 OGs	2265 OGs
2367 Unassigned Proteins	7465 Unassigned	2464 Unassigned
ARI: 0.5129	ARI: 0.1010	ARI: 0.4288

Table 13: OrthoDB's results for the COG data set involving 8 bacterial genomes.OG: Orthologous Groups.

Our results from **Table 12** show that the optimum number of clusters using our algorithm was achieved at 3,500. We then tried to see if the approaches that estimate the optimum number of clusters can be used to attain this value. **Table 14** shows the criterion for each clustering method at each number of clusters. This was used to try to find the proper number of clusters for the COG test. For Calinski Harabasz, Gap, and Silhouette, the larger the criterion value the more optimal the number of clusters. For Davies Bouldin, the smaller criterion value implies the more optimal number of clusters. It would not be reasonable to try to find the optimal number of clusters like it was done for the HomoloGene test. For the HomoloGene test, the number of clusters was small, and it was easy to check a small number of potential number of clusters to see which one was the best. However, for the COG test, a very large number of clusters would have to be tested, and this is not practical. Instead, a few values were tested for each clustering method, and the results are shown in **Table 14**.

# of Clusters	CalinskiHarabasz	DaviesBouldin	gap	silhouette
2000	3.6284	2.5694	2.8013	-0.1469
2250	3.4823	2.4376	2.8099	-0.1342
2500	3.3282	2.3276	2.8131	-0.1257
2750	3.2425	2.2249	2.8233	-0.1106
3000	3.1305	2.1357	2.8280	-0.0988
3250	3.0599	2.0604	2.8362	-0.0865
3500	2.9966	1.9677	2.8450	-0.0761
3750	2.9222	1.8970	2.8501	-0.0620
4000	2.8683	1.8265	2.8575	-0.0455
4250	2.8289	1.7523	2.8665	-0.0279
4500	2.7841	1.6888	2.8731	-0.0031

Table 14: Criterion values for each cluster evaluation method for different number of clusters. The optimum criterion value is highlighted.

Our results indicated that unfortunately the Silhouette method failed to produce reasonable results (as it was not supposed to generate negative values) and the other three methods showed a monotonic behavior. Nevertheless, the average optimum number of clusters indicated by the three methods turned out to be 3,667. When the COG data set (all 12,607 proteins coming from 8 organisms) were clustered into 3,667 orthologous groups using our approach, we attained an ARI value of 0.2298.

Although we did not perform as well as OrthoDB on the whole genome datasets, it should also be noted the time that each algorithm took to run ends up being a major advantage for our algorithm. Our algorithm runs in a matter of a couple of minutes while OrthoDB takes hours to run and return results for the tested data sets. For example, on a Linux server with four Intel Xeon E7-8870 processors per node and 32 GB allocated RAM, our approach took 85.23 seconds while OrthoDBtook 5.62 hours to complete the 8 organism COG data set with 12,607 proteins on a single node.

6 CHAPTER 6: Conclusions and Future Directions

In this thesis, we developed an algorithm that identifies groups of orthologous genes using techniques rooted in machine learning. It was compared to a popular orthologous gene identification algorithm, OrthoDB, and as long as the datasets were small, the proposed algorithm was able to keep up with and even outperform OrthoDB. The proposed algorithm ran faster than OrthoDB on the tested data sets by over 200 folds.

There are many directions we could pursue in the future. First of all, we could turn the proposed algorithm into a web application, which would open it up to be used by the public. We could also look into different clustering techniques, in addition to the three methods tested in this thesis, to cluster the genes into groups. The algorithm could also be used in the field of metagenomics and metatranscriptomics. Our algorithm would work great for such applications because it accepts a single file of molecular sequences as input and it does not matter what organism the sequences come from. This is important because in metagenomics and metatranscriptomics, it is often not known how many different organisms are being analyzed and it is often not known how big the genomes of these organisms are. Our algorithm would be perfect to group this kind of data into orthologous groups. It would be a much better hit than OrthoDB as OrthoDB needs the data split up by organism, and as previously stated, this is often not known for this kind of data. We could also try different sequence characterizations instead of kmers or AMI. Other ways to represent the molecular sequences could be tried with the proposed algorithm to see if they yields better results. Despite the room for improvement, we believe the workflow

proposed in this thesis provides a robust framework to find orthologous genes given a collection of molecular sequences.

References

[1] J. Pevsner, Bioinformatics and Functional Genomics, Wiley-Blackwell, 2015.

[2] L. Urry, M. Cain, S. Wasserman, P. Minorsky and J. Reece, Campbell Biology, Pearson, 2016.

[3] H. Li, C. Avril, J. Ruan, L. James Coin, J.-K. Heriche, L. Osmotherly, R. Li, T. Liu, Z. Zhang, L. Bolund, G. Ka-Shu Wong, W. Zheng, P. Dehal, J. Wang and R. Durbin, "TreeFam: a curated database of phylogenetic trees of animal gene families," *Nucleic Acids Res.,* vol. 34, no. Database Issue, pp. D572-80, 1 January 2006.

[4] N. Ward and G. Moreno-Hagelsieb, "Quickly finding orthologs as reciprocal best hits with BLAT, LAST, and UBLAST: how much do we miss?," *PLoS One,* vol. 9, no. 7, 11 July 2014.

[5] E. V. Kriventseva, D. Kuznetsov, F. Tegenfeldt, M. Manni, R. Dias, F. A. Simao and E. M. Zdobnov, "OrthoDB v10: sampling the diversity of animal, plant, fungal, protist, bacterial and viral genomes for evolutionary and functional annotations of orthologs," *Nucleic Acids Res.,* vol. 47, no. D1, pp. D807-D811, 8 January 2019.

[6] P. E. Saebo, S. M. Andersen, M. Jon, J. K. Laerdahl and T. Rognes, "PARALIGN: rapid and sensitive sequence similarity searches powered by parallel computing technology," *Nucleic Acids Res.,* vol. 33, no. Web Server Issue, pp. W535-9, 1 July 2005.

[7] S. F. Altshul, W. Gish, W. Miller, E. W. Myers and D. J. Lipman, "Basic Local Alignment Search Tool," *J Mol Biology,* vol. 2, 5 October 1990.

[8] N. L. o. Medicine, "BLAST Frequently Asked Questions," [Online]. Available: https://blast.ncbi.nlm.nih.gov/Blast.cgi?CMD=Web&PAGE_TYPE=BlastDocs&DOC_TYPE=FAQ. [Accessed 28 April 2022].

[9] M. Kanehisa, M. Furumichi, Y. Sato, M. Ishiguro-Watanabe and M. Tanabe, "KEGG: integrating viruses and cellular organisms," *Nucleic Acids Res.,* vol. 49, no. D1, pp. D545-D551, 8 January 2021.

[10] J. Mol Biol, "BlastKOALA and GhostKOALA: KEGG Tools for Functional Characterization of Genome and Metagenome Sequences," *J Mol Biology,* vol. 428, no. 4, pp. 726-731, 22 February 2016.

[11] M. Y. Galperin, Y. I. Wolf, K. S. Makarova, R. V. Alvarez, D. Landsman and E. V. Koonin, "COG database update: focus on microbial diversity, model organisms, and widespread pathogens," *Nucleic Acids Res.,* vol. 49, no. D1, pp. D274-D281, 8 January 2021.

[12] M. Y. Galperin, D. M. Kristensen, K. S. Makarova, Y. I. Wolf and E. V. Koonin, "Microbial genome analysis: the COG approach," *Brief Bioinformatics,* vol. 20, no. 4, pp. 1063-1070, 19 July 2019.

[13] L. J. Jensen, P. Julien, M. Kuhn, C. von Mering, J. Muller, T. Doerks and B. Peer, "eggNOG: automated construction and annotation of orthologous groups of genes," *Nucleic Acids Res.,* vol. 36, no. Database Issue, pp. D250-4, January 2008.

[14] R. Derelle, H. Philippe and J. K. Colbourne, "Broccoli: Combining Phylogenetic and Network Analyses for Orthology Assignment," *Molecular Biology and Evolution,* vol. 37, no. 11, pp. 3389-3396, 2020.

[15] B. Buchfink, C. Xie and D. H. Huson, "Fast and sensitive protein alignment using DIAMOND," *Nature Methods,* 2014.

[16] G. Newcomb and K. Sayood, "Use of Average Mutual Information and Derived Measures to Find Coding Regions," *Entorpy (Basel),* vol. 23, no. 10, p. 1324, 2021.

[17] M. Bauer, S. M. Schuster and K. Sayood, "The Average Mutual Information Profile as a Genomic Signature," *BMC Bioinformatics,* vol. 9, no. 48, 25 January 2008.

[18] T. Bock, "What is Hierarchical Clustering?," [Online]. Available: https://www.displayr.com/what-is-hierarchical-clustering/. [Accessed 15 March 2022].

[19] D. M. J. Garbade, "Understanding K-means Clustering in Machine Learning," 12 September 2018. [Online]. Available: https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1. [Accessed 15 March 2022].

[20] J. VanderPlas, "In Depth: Gaussian Mixture Models," [Online]. Available: https://jakevdp.github.io/PythonDataScienceHandbook/05.12-gaussian-mixtures.html. [Accessed 15 March 2022].

[21] n. D. Baruah, "Cheat sheet for implementing 7 methods for selecting the optimal number of clusters in Python," 25 October 2020. [Online]. [Accessed 15 March 2022].