MULTI SEQUENCE ASSEMBLBY FOR PROKARYOTES AND EUKARYOTES USING FUZZY LOGIC

Sara NASSER[1], Gregory L. VERT[1], Monica NICOLESCU[1], Alison MURRAY[2]

[1]*Department of Computer Science Engineering,*
University of Nevada Reno, Reno USA
Email: sara@cse.unr.edu, gvert@cse.unr.edu, monica@cse.unr.edu

[2]*Division of Earth and Ecosystem Sciences,*
Desert Research Institute, Reno, USA
Email: alison@dri.edu

Abstract

Nucleotide sequencing of genomic data is an important step towards building understanding of gene expression. Current limitations in sequencing limit the number of base pairs that can be processed to only several hundred at a time. Consequently, these sequenced substrings need to be assembled into the overall genome. Furthermore, the existence of insertions, deletions and substitutions can complicate the assembly of subsequences and confuse existing methods. What is needed is an approach that deals with ambiguity in trying to match and assemble a genome from its sub sequences. An assembler that can assemble Eukaryote and Prokaryote sequences alike is needed. Evaluation of this approach suggests that this approach is at least as good as standard approaches and in some cases better. Preliminary evaluation of this approach in conjunction with K-Means for clustering species shows good classification of and assembly Prokaryote and Eukaryote sequences.

Keywords: Computational Biology, Bioinformatics, Genome Sequencing, Fuzzy Logic, Eukaryote and Prokaryote Sequence Assembly

1 Introduction

All living organisms can be categorized into two major groups based on the structure of their cells, the prokaryotes and eukaryotes. Prokaryotes have the most primitive type of cell structure. Their nucleus does not have a membrane and the structure is simple. Bacteria are the most common and largest group of prokaryote. An example of the Prokaryote is Escherichia coli (E. coli). Eukaryotes are organisms that have a membrane for the nucleus. Eukaryotic DNA is much longer in size and contains bulk of non-coding regions. Examples of eukaryotes include mammalian groups, plants, etc.

Genome shotgun sequencing for DNA alignment is a process that requires building consensus sequences from small DNA fragments. DNA is composed of four nucleotides A, C, G, T. Genome sequencing is figuring out the order of DNA nucleotides, or bases, in a genome that make up an organism's DNA. These nucleotides and their order determine the structure of protein. Sequencing the genome is a very important step in Genomics. Entire genome sequences are very large in size and can range from several thousand base pairs to millions of base pairs. The whole genome can't be sequenced all at once because the chemical reactions researchers use to decode the DNA base pairs are accurate for only about 600 to 700 nucleotides at a time [10]. Therefore DNA is chopped up into small subsequences.

Sequencing DNA has several steps starting from acquiring data to assembling the DNA fragments or sequences into an entire genome sequence. The process of DNA sequencing begins by breaking the DNA into millions of random fragments, which are then given to a *sequencing* machine. Fragments or sequences are selected randomly; using a sequence once may not cover all regions. Therefore multiple copies of original sequences are used to ensure that the entire sequence is covered. This is generally referred to as coverage of *'nX',* where n is the number of copies. Coverage of 8X-10X is widely accepted to be able to reconstruct the entire sequence. Following the sequencing process, an *assembler* pieces together the many overlapping reads and reconstructs the original sequence [20]. The process explained above is known as "whole-genome shotgun" method, which involves breaking the genome up into small pieces, sequencing the pieces, and reassembling the pieces into the full genome sequence. Sequencing DNA using the shot-gun method was introduced in 1995 [8], [9a].

The problem of assembly is not of an exact matching but rather of obtaining approximate matches through consensus. The consensus sequences are also known as contiguous sequences or *contigs.* In contigs, mismatches can occur due to insertions or deletions (indels) or replacement of base pairs. A consensus sequence is constructed through approximate matches by following an overlap and consensus scheme [20a]. This is illustrated in Fig. 1. below.

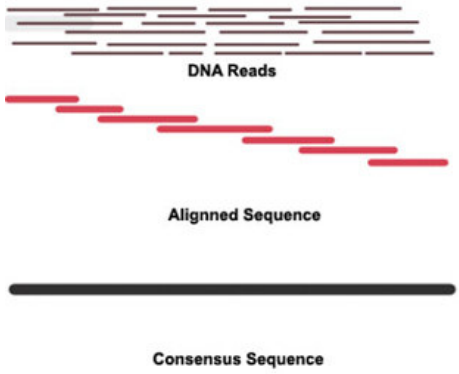

Figure 1. Whole Genome Sequencing process is displayed in terms of reads from the DNA sequences.

Even though computational power has made it possible to sequence genomes, assembly is still an NP-Hard problem. An exhaustive method such as brute-force is not a viable option. Heuristic approaches can work better for such problems.

The algorithm presented in this paper uses a divide-and-conquer strategy to speed up the assembly by dividing the sequences into classes. Some sequences can have higher similarity with each other and some others may have lesser or no similarity at all. Assembling sequences is accomplished by first sorting the sequences into classes. We perform a meaningful partitioning of data, so that sequences in a cluster have high similarity with one another and sequences between two clusters are less similar.

Prokaryotes and Eukaryotes are metabolically different. The main difference is in the coding regions. Eukaryotes contain large amounts of regions that do not code for proteins. As the basic structure of prokaryotes and eukaryotes are different we would like to build an assembler that can sequence organisms from both groups. With recent advances in science samples are collected from the

environment. These samples sometimes may have traces of mixed species or mixed strains. Therefore we perform classification of data along with assembly so that species can be separated.

The following Section discusses the previous work done in sequence assembly. Section 3 contains the approach. Section 4 has the experimental results followed by future work and conclusions in Section 5.

2 Previous Techniques

In general, the approach to sequence assembly has been to iteratively find the best overlap between all fragment pairs until an acceptable final layout has been determined. In current genome sequencing tasks, the number of fragments is usually very large and the degree of computation required increases exponentially. Being essentially an NP-Hard problem, many different approaches with varied parameters and matching schemas have been explored which can, among other things, save computation time. Finding the longest common subsequence (LCS) between fragments is the key to the process of sequence assembly. LCS for matching strings solves problems by combining the solutions to sub problems [27], in this case substrings. It is a method for reducing the runtime of algorithms containing overlapping sub-problems and optimal substructures [5a]. An application of dynamic programming is used in the Smith-Waterman algorithm to find LCS for multiple sequence alignment [11], one of the most prominent algorithms used in sequence assembly programs. Other techniques for finding the longest common subsequence include suffix trees, the KMS algorithm [17] and greedy approaches [13]. Suffix trees allow a linear time search for matching substrings. The *KMS Algorithm* identifies the best matches of the longest substrings of the matches for multiple strings [17]. Greedy algorithms can be much faster than traditional dynamic programming and work well in the presence of sequencing errors [28].

Even with these algorithmic improvements, additional reductions to search space in fragment assembly problems are routinely employed. For example, PHRAP determines best fragment matches by comparing only the highest quality parts of reads [12], both reducing search time and possibly increasing accuracy. The AMASS algorithm limits searches to short, randomly selected sequences within fragments rather than comparing complete reads. This approach showed a drastic reduction in assembly time [18]. Another approach to time reduction

involves determining which groups of fragments have more potential for aligning and only comparing those together. For example, the assembler STROLL [3a] significantly reduces the number of required comparisons by rejecting all candidate fragment pairs without exact matches of a threshold length. Similarly, the CAP3 program determines which fragment pairs have potential overlap before making comparisons [14]. Even one of the earliest assembly schemes, SEQuencing AID (SEQAID) [21] examines ancillary fragment information to aid in the determination of fragment order.

Pre-assembly clustering of fragments may be viewed as a more structured form of fragment thinning before alignment comparisons are made. Clustering is a process of grouping objects into like groups based on some measure of similarity. Clustering or classification can be achieved by several techniques such as K-means, artificial neural networks, etc. This divide-and-conquer strategy for sequence assembly was described in [13]. A K-means clustering scheme was applied to fragments based on their Average Mutual Information (AMI) measures. AMI profiles are used to measure the degree of 'closeness' between fragments.

K-means has been widely used in pattern recognition problems. Several variations and improvements to the original algorithm have been implemented. The K-means algorithm by MacQueen [15] is widely used for its simplicity. Another variation of K-means was proposed by Forgy [6a]; this algorithm has been shown to converge to a local minimum [25].

Fuzzy Logic formularizes an intuitive theory based on human reason of approximation. It differs from the traditional logic methods where crisp or exact results are expected. The concept of fuzzy logic was first put forth by Zadeh [19]. Fuzzy Logic is used in problems where the results can be approximate rather than exact. The results are determined by some degree of closeness to being true or false. Due to its applicability to problems that do not require hard solutions, Fuzzy Logic has been widely used in various fields to provide flexibility to classical algorithms. An earlier well known approach to fuzzy classification is the fuzzy C-means algorithm [1a]. An improvement of K-means using the fuzzy logic theory was presented [2a], in which the concept of fuzziness was used to improve the original K-means algorithm. Fuzzy approaches to bioinformatics have been explored to some extend. A model for creating fuzzy set theory uses for bioinformatics was proposed [16]. An application to ontology similarity using fuzzy logic was shown [4a].

Fuzzy Logic has been applied to classification problems in computational

biology. Even though application of fuzzy logic has not been done extensively, recently it started to gain popularity. A modified fuzzy K-means clustering was used to identify overlapping clusters of yeast genes [9]. Data was based on published gene-expression results following the response of yeast cells to environmental changes.

3 Sequence Assembly

In this paper we present a clustering technique that uses a fuzzy membership function to divide genome sequences into groups. This reduces the number of comparisons and performs meaningful assembly. The fuzzy functions used in this paper are a modified version of the Fuzzy Genome Sequencing Assembler described in [23].

3.1 Longest Common Subsequence with Fuzzy Logic

Dynamic programming has been extensively used to determine the longest common subsequence (LCS). The reason for its popularity is that reduces time complexity of assembly to $\Theta(n^2)$.

This method is simple and is very useful in finding longest common subsequence which may have mismatches in the sequence. This suits well to assembly problems since not all subsequences found will be perfect and can be easily modified to find contiguous subsequences.

In the case of genome subsequences we would like to get the longest subsequence with fewest insertions or deletions (indels). One of the common techniques used by assembly processes, such as PHRAP, is to search within a bandwidth along the best possible match. As the path grows beyond the bandwidth, the indels increase and the subsequence shifts away from the optimal match.

The optimal subsequence can either be a perfect match, or the user may choose to tolerate indels. These criteria can depend on the user, the source of the data, quality of the data, etc. Almost all existing techniques provide user defined thresholds for the number of indels allowed. The ideal cut off point or the thresholds for a particular data set generally needs to be determined empirically. For example, assume that a cutoff value for the maximum gap allowed is 30

bases and that there are fairly large numbers of sequences with a gap of 31 and 32. Due to the fact that these techniques allow for crisp matches only, these potentially important sequences would be excluded. Alternatively, we can represent a match of 30 and lower with a fuzzy confidence value of 1, which is for crisp matches. Matches that are very close to 30 like 31 can have a fuzzy value of 0.98. If the user selects to allow all matches greater than the value 0.8 then these subsequences would be included. In this case, the user does not have to look into the data, change parameters and run the program several times. There are several other parameters, which will be described later, that the user may have to alter to get a desired result. The main objective is to obtain the best consensus of the overall parameters.

This paper proposes a fuzzy matching technique where we can have crisp and non-crisp matches. The user could also obtain a fuzzy value that states how well the matching sequences fit the threshold.

3.2 LCS and Fuzzy Logic

As mentioned earlier, one of the problems with existing techniques is that they have crisp bounds. The user has to specify the parameters for the program such as minimum score and minimum match. Minimum match and score are described in Section 3.4. The parameters need to be changed by the user to suit the data, and then the program ran one or more times, until an optimal solution is found. This allows the user to determine which parameters work best with the given sample. Selecting the longest sequence is not always the optimal solution. Some applications prefer longer sequences while others may require higher quality or smaller gaps.

The main objective of our method is assembling data by approximate matching using fuzzy logic. Fuzzy Logic has been used in approximate string matching using distance measures, etc. However, there has been limited application to building genomes from subsequences of nucleotides.

Current sequencing methods tend to reject sequences that do not match with a high degree of similarity. This can lead to large amounts of data being rejected, which otherwise might be important in deriving a genomic sequence and its metabolic characteristics.

We propose a method where we select multiple subsequences and then based on several fuzzy parameters select the optimal solution. The novelty of our

method is that it uses more parameters of the sequence besides the length of overlap. These parameters can lead to a better sequence. The sequence satisfying the aggregate overall requirement is selected based on fuzzy parameters. In other words, we are measuring the fuzzy similarity of the given subsequences. There are several factors that determine if two subsequences can have an optimal overlap. These factors are used to measure their similarity. For example, two subsequences can form a contig if their overlap region is larger than a threshold. They could be highly similar if they have less number of indels, or less similar with more indels.

Since we perform a non-banded search, it is not ideal to try every possible subsequence combination. On the other hand it just searching along the diagonal may not give an overlap at all. Instead of selecting the longest common subsequence, we select all the subsequences that satisfy the minimum length required. The threshold is a function of the length of the LCS. The search is banded by using a threshold to prevent moving away from the diagonal. A cell is marked if it was already traversed, so we don't check it again. We keep track of cells that are traversed and paths above the threshold are selected:

$$\begin{aligned} &length >= threshold, \\ &threshold = f(n(LCS)) \end{aligned} \tag{1}$$

where:
length is the size of overlap, threshold determines the minimum length required. Threshold is selected based on the LCS.

We need determine if either of these subsequences will create an optimal match. Any of the selected paths can generate an optimal solution. Therefore we do not eliminate any possible good subsequences. The selection of the optimal subsequence is done using fuzzy similarity measures. The selection process is done in constant time; therefore the complexity of the algorithm is same as the complexity of Dynamic programming, which is $\Theta(mn)$ for any two subsequences of length m and n. The following section lists the characteristic functions.

3.2 Fuzzy Similarity Measures

Fuzzy similarity measures and the concept created by this research are an

important step in creating a contig from two subsequences or finding an overlap between two sequences. The following subsections describe the fuzzy functions utilized in our approach for assembly.

(i) Length of Overlap (μ_{lo}): The first similarity measure that we look at is the length of the match. This is also the size of overlap when a contig is being created. This length includes indels and replacements. A higher overlap is better as it generates a longer contig. The membership function for this measure is defined as:

$$\mu ol(s1, s2) = \begin{cases} 1, & |overlap(s1, s2)| = max|overlap(s1, s2)| \\ 0, & |overlap(s1, s2)| = 0 \\ [0,1], & |overlap(s1, s2) / max|overlap(s1, s2)| \end{cases} \quad (2)$$

where:
|overlap (s1, s2)| - length of overlap of strings s1, s2,
s1, s2 – strings being evaluated

Given strings s1, s1 where no overlap occurs, the possibility of similarity does not exist. Given string s1, s2 where there is complete overlap or maximum overlap, the possibility is of maximum similarity which is greater than partial or no overlap. For all other cases the similarity is between 0 and 1. The logic of Equation (2) can be described in Equation (3) as follows:

$$\forall \text{ |overlap (s1, s2)|} \uparrow \rightarrow \mu_{lo}() \uparrow \quad (3)$$

where:
$\uparrow$ - represents increase in value,

(ii) Confidence (μ_{qs}): The confidence for each contig is defined as a measurement of the quality of the contributing base pairs [21a]. The quality of a base pair indicates if the read was strong. A strong read indicates a correct read or less changes of noise or experimental error. Every base involved in the contig has a quality score. The confidence of a contig is the aggregate quality score of its contributing bases. For simplicity, the sum of average quality scores is the

confidence of the contig. Equation (4) below describes the membership function

$$\mu_{qs} = \frac{\sum_{i=1}^{n} w_i q_i}{n} \tag{4}$$

where:

μ_{qs} is the quality score for the overall overlap region, w_i is the weight used to standardize the quality scores, n is the number of bases, q_i is the quality score of an individual base,

The weight can be calculated as shown in Equation (5). The bases with high quality are assigned a weight of 1. Only the bases that are of lower quality are given weights between 0 and 1.

$$w_i = \begin{cases} 1, & if \quad q_i \geq \delta \\ 0, & if \quad q_i = 0 \\ \dfrac{q_i - \min_{qs}}{\max_{qs} - \min_{qs}} \end{cases} \tag{5}$$

where:

δ is the threshold as explained earlier, (this is generally specified by the user), $\min_{qs}$ and $\max_{qs}$ are the minimum and maximum values for quality.

(iii) Gap Penalty (μ_{gp}): This is the penalty that is imposed on gaps within an overlap region. Gap is measured in terms of the of bases and is given in Equation (6) as follows:

$$\text{Gaps} = |\text{insert}| + |\text{delete}| + |\text{replacement}| \tag{6}$$

A simple gap penalty can be calculated using membership function in Equation (7) as follows:

$$\mu_{gp}(s1, s2) = \begin{cases} 1, & Gaps(s1, s2) = \phi \\ 0, & overlap(s1, s2) = \phi \\ [0,1], & (|overlap(s1, s2)| - Gaps) / |overlap(s1, s2)| \end{cases} \tag{7}$$

where:
overlap (s1, s2) - amount of overlap of strings s1, s2,

The logic behind Equation (7) can be explained by Equation (8) given below.

$$\forall \Delta (s1, s2) \uparrow \rightarrow \mu_{gp} () \rightarrow 0 \tag{8}$$

where:
$\uparrow$ - represents increase in value, $\rightarrow$ - implies approaches, Δ - represents the difference.

*Informal proof: G*iven overlap (s1, s2) = 10 and Gaps = 2, μ_{gp} (s1,s2) over [0,1] $\rightarrow$ 0.8 due to the fact that the overlap is higher than the Gaps.

The above definition of Gap Penalty is a simple method to calculate gaps. In this method, every gap is given same weight. A gap of 10 bases is the same as 10 gaps of 1 base each. It can be argued that a gap of 10 bases is actually caused by single insert or delete. It can be counted as 1 instead of 10. Therefore a weighed gap penalty can work well in this case. We use an affine gap penalty which is given by

$$GapPen = GapOpening + GapLength \times GapExtension \tag{9}$$

Where, GapOpening and GapExtension are scores for an opening or a continuation of a gap. The summation of Equation (9) gives the entire gap penalty GapPen(s1,s2). The membership function for this is as follows:

$$\mu_{gp}(s1, s2) = \begin{cases} 1, \ GapPen(s1, s2) = 0 \\ 0, | \ overlap(s1, s2) | <= GapPen(s1, s2) \\ [0,1], \ (1 - \dfrac{(| \ overlap(s1, s2) | - GapPen(s1, s2))}{overlap(s1, s2)}) \end{cases} \tag{10}$$

where overlap (s1, s2) is the length of overlap of s1 and s2,

(iv) Score (μ_{ws}): A score is calculated from the number of matching bases, number of indels and replacements. Score can be calculated in different ways. For example:

$$score = f_n(\text{MatchingBP}) - f_n(\text{Inserts}) - f_n(\text{Deletes}) - f_n(\text{Replacements}) \quad (11)$$

A- CTCGCGAT- GCG
AGCTCG- GATTGAG

For the above subsequences there are 11 BP matches, 2 inserts, one delete, and one replacement. If all are given a value one the score is 7(11-2-1-1). Commonly, higher scores are given to matching bases and lower values are given to bases that don't match. A scoring matrix is used to assign these scores.

Table 1. Scoring Matrix

	A	**C**	**G**	**T**
A	3	-1	1	-1
C	-1	3	-1	1
G	1	-1	3	-1
T	-1	1	-1	3

In Table 1 the scores for each of the possible matches are given. If there is a match a score of 3 is given. Matches within Purine (A, G) or Pyrimidine (C, T) class are given a score of 1. Finally a score of -1 is given to matches between a Purine and Pyrimidine and vice versa.

The above equation leads to derivation of the following fuzzy similarity membership function.

$$\mu_{ws}(s1, s2) = \begin{cases} 1, \ t\_score(s1, s2) = fmbp(s1, s2) \\ 0, \ fmbp(s1, s2) <= 0 \\ [0,,1], \ fmbp(s1, s2) \Big/ t\_score(s1, s2) \end{cases} \quad (12)$$

where: fmbp (s1, s2) - is the score calculated using the matrix illustrated in

Table 1, t_score (s1, s2) - score of the overlap if they were no indels or replacements

The logic of Equation (12) can be explained in Equation (13) below.

$$\forall \Delta \text{ (t\_score(s1, s2)} - \text{fmbp(s1,s2)} \uparrow \rightarrow \mu_{ws} () \rightarrow 0 \quad (13)$$

where: $\uparrow$ - represents increase in value, $\rightarrow$ - implies approaches.

*Informal proof: G*iven t_score (s1, s2) = 10, and fmbp (s1, s2) = 9, μ_{ws}(s1, s2) = 9/10 = 0.9, because the relative percentage of differences to matches is high.

3.4 Thresholds

(i) MinMatch: This is the minimum number of matching bases that are required between the two sequences. It is not possible to get a perfect overlap and some amount of inexactness is tolerated. Genomic DNA contains only 4 characters and there can be several overlaps with these 4 characters. Therefore we would like to have a minimum match value for the overlap sequences.

(ii) MinScore: A score is calculated from the number of matching bases, number of indels and replacements as given above in Equation (12). MinScore is a threshold which specifies the minimum allowable score of the overlap.

Once the fuzzy value for each of these parameters is calculated, we plug them into an overall fuzzy function. This function is the aggregate fuzzy match value (afv). A perfect overlap refers to an overlap that satisfies the two thresholds above, is free of gaps, and satisfies the quality requirements.

$$fa(c) = \mu_{qs} w_{qs} + \mu_{ws} w_{ws} + \mu_{gp} w_{gp} + \mu_{lo} w_{lo} \quad (14)$$

$$afv = fa(c)/m, \quad (15)$$

where:

m is number of parameters

fa(c) is given in equation (14)

Equation (14) and (15) give the overall fuzzy function and the aggregate fuzzy function for m parameters. The sub-sequences that produce the highest fuzzy value for an overlap are selected as final sequences. Depending on their position as a suffix or prefix, a new contig or consensus sequence is formed.

3.5 K-means Clustering

Clustering problems generally derive some kind of similarity between groups of objects. Our approach uses K-means clustering to separate species. We use the technique of LCS and fuzzy logic proposed early in conjunction with K-means. K-means clustering is a simple and fast approach to achieve such grouping. The algorithm starts with a large number of seeds (initial samples) for the potential clusters. Remaining samples are then assigned to a cluster based on their distance from the seed. The centroid is recomputed for each cluster and the data points are reassigned. The algorithm runs until it converges or until the desired number of clusters is obtained.

Given N sequences, such that $S=\{C\}^i$, where C ={A, C, G, T}. We randomly select "k" sequences as the initial seeds, where k is less than the number of sequences N. The algorithm starts by performing LCS on each of the sequences with the k seeds with the method described in the previous section. The sequence is assigned to the class which has the highest fuzzy similarity. The fuzzy similarity is calculated as given below; it is also referred as the fuzzy weighted average. Equation (16) replaces Equation (15).

$$\mu^r = \sum_{(p=1,P)} w_p^{(r)} x_p, r = 0,1,2,..... \quad (16)$$

Here x is the parameter or feature and p the number of features. Given i=0,...,N and j=0,…,k, the distance $d_{i,j}$ for each cluster in this case, is the maximum fuzzy value and can be calculated as follows:

$$d_{i,j}=max(\mu^r_j), \text{ for all } j=0,\ldots k \quad (17)$$

4 Experiment and Results

The main objective of this research was to create an assembler using fuzzy logic. The second objective was to test the assembler on different organisms and be able to differentiate species.

The Fuzzy Genome Sequence Assembler was implemented with modified version LCS described above with all the parameters listed. The assembler was tested on generated data sets and data from GenBank. Artificially generated data sets were used to verify the algorithm and thus the assembly process. GenBank is a publicly available database of nucleotide sequences. The experiments were run on a 1.83GHz Intel Core 2 Duo processor and 1GB of RAM. The experiments are divided into 3 major categories, the prokaryotes assembly, the eukaryotes assembly and a classification of prokaryotes and eukaryotes. The results are compared with TIGR 2.0 [26]. TIGR 2.0 is a well-known assembler for assembling large shotgun sequence projects.

Table 2. Assembly Comparisons prokaryotic Data

Assembler	Genome	Percentage Genome Recovered
MGS	RPObc of Wolbachia genome	65%
TIGR	RPObc of Wolbachia genome	99.6%
FGS	RPObc of Wolbachia genome	99.6%
TIGR	Yersinia pestis Pestoides	93.9%
FGS	Yersinia pestis Pestoides	88.7%
TIGR	Geobacillus thermodenitrificans	77.1%
FGS	Geobacillus thermodenitrificans	91.6%

The first group of experiments is prokaryote assembly. First genome sequence tested was the *Wolbachia* endosymbiont of the Drosophila melanogaster strain wMel 16S ribosomal RNA gene, partial sequence. The sequence contains 8,514 base pairs. Second prokaryote genome is Geobacillus thermodenitrificans NG80-2 plasmid pLW1071, complete sequence. This genome is 57,693 base pairs. The third genome is Yersinia pestis Pestoides F plasmid CD, complete sequence. This genome contains 71,507 base pairs. All

these genomes can be obtained from GenBank [22]. The total bases read were 4X of the original sequence. Each subsequence was in the range of 300-900bps.

In Table 2 MGS = Multiple Genome Sequencing using simple LCS implementation, TIGR=TIGR Assembler 2.0, FGS= Fuzzy Genome Sequencing. The third column is the percentage of original genome recovered by the assembly process. Since MSG did not perform well we did not include it for further experiments.

The second set of experiment was performed on eukaryotic genomes. The first genome sequence is the Arabidopsis thaliana, gene_id:F11I2.4. This sequence contains 36,034 base pairs. The second genome is Ostreococcus tauri mitochondrion, complete genome containing 44,237 base pairs. The third genome is Phytophthora sojae mitochondrion, complete genome containing 42,977 base pairs. Details of these sequences can be obtained from GenBank [22].

Table 3. Assembly Comparisons Eukaryotic Data

Assembler	Genome	Percentage Genome Covered
MGS	Arabidopsis thaliana	56.8%
TIGR	Arabidopsis thaliana	88.8%
FGS	Arabidopsis thaliana	92.135%
TIGR	Ostreococcus tauri mitochondrion	77.7%
FGS	Ostreococcus tauri mitochondrion	97.3%
TIGR	Phytophthora sojae mitochondrion	97.7%
FGS	Phytophthora sojae mitochondrion	97.2%

In Table 3 MGS = Multiple Genome Sequencing using a simple LCS implementation, TIGR=TIGR Assembler 2.0, FGS= Fuzzy Genome Sequencing. The third column is the percentage of original genome recovered by the assembly.

The third set of experiments are to separate two species. Sequences from two organisms are taken and mixed with each other. The input data appears as if it is from a single organism. We use 4X coverage of the entire sample. A K-means clustering is performed to group sequences from the organisms into two different classes and then perform assembly.

Table 4: K-means clustering for two Organisms

Genome	Percentage Genome Recovered	Miss-Classifications
Phytophthora sojae mitochondrion	**61%**	**0**
Geobacillus thermodenitrificans	**61.1%**	**0**

In Table 4 ClusFGS is the method described in this paper and is a modified version of FGS. ClusFGS is described in Section 3.5. Since TIGR does not perform a classification we did test the results with TIGR. Misclassification refers to length of overall subsequences from genome 1 that were assembled incorrectly with contigs of genome 2.

The results obtained in Table 2 and Table 3 from assembling the genome projects showed a high percentage of the genome recovered while using FGS and TIGR. This indicates that given random subsequences, the algorithm was able to create a fairly large percentage of the original sequence. In Table 2 FGS performed better than simple MGS. Assembly of RPObc of Wolbachia genome has same results for both TIGR and FGS. Yersinia pestis Pestoides showed a slightly better recovery for TIGR. Geobacillus thermodenitrificans showed a much better assembly with FGS. Examining the results of Table 3, FGS performed better that a simple MGS and obtained better or similar results as the TIGR assembler. Some of small differences in results could be due to different thresholds being used.

Preliminary results from Table 4 show that Fuzzy K-means classification was successful in grouping these two classes separately. The clustering classified the data into two groups without any misclassification. The clustering technique is linear and hence can make the assembly much faster. At this stage it cannot recover a higher percentage of the genome as comparisons are limited to within class. The performance depends on the selection of the seeds. Currently we do not perform reassignment of clusters, merging, etc. Adding these methods can improve the classification and thus assembly.

5 Conclusions and Future Work

This paper proposes the use of fuzzy K-means for approximate sequence assembly. Fuzzy similarity measures were used and a fuzzy weighted average was created to perform the classification. We tested the assembler on published

genome projects and compared the results with other assemblers. A K-means classification is used to increase performance.

The functions proposed can be easily adapted in other assembly methods or techniques. The assembly can be further improved by data reduction before assembly; this would make it possible to run larger data sets and also make the process faster. Current assemblers use different techniques to achieve data reduction which makes assembly faster. This can be achieved by encoding the data, using hash tables, indexing, etc.

Preliminary results show that the assembler is capable of separating different genomes. The idea proposed can be used to group meta-genomic data into classes and provide a clean assembly for environmental sequences. We would like to extend the assembler to classify strains within a population.

While this work looks promising there are a number of research questions that remain to be answered. The first of these is the appropriate value to set the weights to in the fa(c) equation. It is thought that a mathematical relationship between fuzzy similarity measures and these weights should be derived. Additionally, this work has been evaluated on a small set of data from GenBank. To improve performance, further research can be done on reducing the amount of information to be processed via the use of data transformation schemes. Finally classification method proposed can be extended to classify environmental sequences and assemble them.

We would like to improve the assembly by creating longer contigs. Graph approaches to build possible outcomes of the genome can be used.

The assembly can be further improved by adding other parameters for clustering that can classify data using structural motifs such as AMI, G-C content etc. Our method uses a simple K-means without reassignment of objects between clusters, etc. An enhanced k-means can be used to improve the performance of the assembler. A supervised classification can be done so we can derive weights from the data set.

References

[1] Bezdek, J.C., 1981. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Plenum Press, New York.

[2] C. G. Looney, 2002, *Interactive clustering and merging with a new fuzzy expected value*, Pattern Recognition Lett., vol. 35, 187–197.

[3] Chen, Skiena, *A Case study in genome-lebel fragment assembly*, *Bioinformatics*, 16(6):494-500(200)
[4] Dong Xu Bondugula, R. Popescu, M. Keller, J., 2006, *Bioinformatics and Fuzzy Logic*, IEEE International Conference on Fuzzy Systems, July 16-21, 817- 824
[5] Dynamic Programming, 2006-Wikipedia, http://en.wikipedia.org/wiki /Dynamic_programming.
[6] E. Forgy, 1965, Cluster analysis of multivariate data: efficiency versus interpretability of classifications, Biometrics, 21, 768–776.
[7] Ewing B, Green P., 1998, *Basecalling of automated sequencer traces using phred. II. Error probabilities*. Genome Research *8:186*-194.
[8] F. Sanger et al., 1982, "Nucleotide Sequence of Bacteriophage Lambda DNA," J. Molecular Biology, vol. 162, no. 4, pp. 729-773.
[9] Gasch, A. P. & Eisen, M. B., 2002, Exploring the conditional coregulation of yeast gene expression through fuzzy k-means clustering. Genome Biol 3, RESEARCH0059.
[10] Gene Myers, 1999, Whole-Genome DNA Sequencing, IEEE Computational Engineering and Science 3, 1, 33-43.
[11] Genome Wikipedia, 2006, http://en.wikipedia.org/wiki/Genome, Accessed, October.
[12] Green P., 2006, *Documentation for Phrap*, http//bozeman.mbt. washingto - n.edu. Genome Center. University of Washington.
[13] Hasan H. Otu, Khalid Sayood, 2003, *A divide-and-conquer approach to fragment assembly*, Bioinformatics 19(1): 22-29
[14] Huang X., Madan A, 1999, "CAP3:A DNA sequence assembly program", *Genome Res*, Sep;9(9)868-77.
[15] J. B. MacQueen, 1967, *Some methods for classification and analysis of multivariate observations*, Proc. 5th Berkeley Symp. Probability Statistics, University of California Press, Berkeley, pp. 281–297.
[16] K. Sadegh-Zadeh. 2000, *Fuzzy genomes*, Artif Intell Med. Jan;18(1):1-28.
[17] K Kaplan, 1995, *An Approximate String Matching Algorithm with Extension to Higher Dimensions*. UMI Microfilm.
[18] Kim S, Segre AM, 1999, *AMASS: astructured pattern matchingapproach to shotgun sequence assembly*, Journal of Computational Biology, Summer;6(2):163-86.
[19] L. A. Zadeh, 1975, *Fuzzy logic and approximate reasoning*, Synthese, vol. 30, 407--428.

[20] Mihai Pop, Steven L. Salzberg, Martin Shumway, 2002, *Genome Sequence Assembly: Algorithms and Issues*.

[21] Peltola, H., Soderlund, H. and Ukkonen, E., 1984, *SEQAID: A DNA sequence assembling program based on a mathematical model*, Nucleic Acids Res., Vol. 12(1), pp. 307–321.

[22] rpoBC DNA-directed RNA polymerase, Wolbachia endoysymbiont of Drosophila melanogaster, 2006, http://www.ncbi.nlm.nih.gov/ entrez/query.fcgi? db=gene&cmd=Retrieve&dopt=full_report&list_uids= 2738525.

[23] Sara Nasser, Gregory Vert, Monica Nicolescu, Alison Murray, 2007, *Multiple Sequence Alignment using Fuzzy Logic*, Proceedings of the IEEE Symposium on CIBCB-2007, April 1-5, Honolulu, Hawaii, Vol, 07, pp 304-311.

[24] Smith T, Waterman M, 1981, *Identification of common molecular subsequences***.** Journal of Molecular Biology, 147:195-197.

[25] S. Z. Selim, M. A. Ismail, 1984, *K-means type algorithms: a generalized convergence theorem and characterization of local optimality,* IEEE Trans. Pattern Analysis Machine Intelligence, 6, 81–87.

[26] Sutton G., White, O., Adams, M., and Kerlavage, A. (1995) TIGR Assembler: A new tool for assembling large shotgun sequencing projects. *Genome Science & Technology* 1:9-19

[27] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, 2001*, Introduction to Algorithms*, Second Edition, pp 313-319.

[28] Zheng Zhang, Scott Schwartz, Lukas Wagner, Webb Miller, 2000, *A Greedy Algorithm for Aligning DNA Sequences*, Journal of Computational Biology, vol 7, pp. 203-214.