# Fuzzy Multiple Sequence Alignment

Sara Nasser[1], Gregory L. Vert[1], Monica Nicolescu[1] and Alison Murray[2], [1]Department of Computer Science and Engineering, *University of Nevada Reno, Reno USA,* [2] Desert Research Institute, *Reno*

***Abstract*—DNA matching is a crucial step in sequence alignment. Since sequence alignment is an approximate matching process there is a need for good approximate algorithms. The process of matching in sequence alignment is generally finding longest common subsequences. However, finding a longest common subsequence may not be the best solution for either a database match or an assembly. An optimal alignment of subsequences is based on several factors, such as quality of bases, length of overlap, etc. Factors such as quality indicate if the data is an actual read or an experimental error. Fuzzy logic allows tolerance of inexactness or errors in sub sequence matching. We propose fuzzy logic for approximate matching of subsequences. Fuzzy characteristic functions are derived for parameters that influence a match. We develop a prototype for a fuzzy assembler. The assembler is designed to work with low quality data which is generally rejected by most of the existing techniques. We test the assembler on sequences from two genome projects namely, Drosophila melanogaster and Arabidopsis thaliana. The results are compared with other assemblers. The fuzzy assembler successfully assembled sequences and performed similar and in some cases better than existing techniques.**

**Index Terms— Bioinformatics, Sequence Assembly, Fuzzy Logic, Approximate Matching, Dynamic Programming**

## I. Introduction

DNA sequence assembly can be viewed as a process of finishing a puzzle where the pieces of the puzzle are DNA subsequences or strings. The main difference being that a puzzle has pieces that fit in very well with each other. The pieces of a DNA puzzle do not fit together precisely. It's a puzzle where the ends can be ragged, thus making it very difficult sometimes nearly impossible to complete the puzzle. Hence, we need methods or rules to optimally determine which piece fits with another piece. The following sections explain the steps involved in sequence assembly.

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 million base pairs. The whole genome can't be sequenced all at once because available methods of DNA sequencing can only handle short stretches of DNA at a time. Although genomes vary in size from millions of nucleotides in bacteria to billions of nucleotides in humans, the chemical reactions researchers use to decode the DNA base pairs are accurate for only about 600 to 700 nucleotides at a time [2].

Current techniques can read up to 800 base pairs (BP). So biologists chop up a sequence into smaller subsequences. The steps involved in this process are further explained in section I.

Sequencing of an organisms' DNA manually is a labor intensive task, but with recent advances in computational power this can be achieved. Several organisms' genomes have been sequenced. On a larger scale, the mouse, rat and chimpanzee genomes are all being sequenced and mapped to the human genome to better understand human biology, and multiple Drosophila species are being sequenced and mapped onto one another [4]. Other genome projects include mouse, rice, the plant Arabidopsis thaliana, the puffer fish, bacteria like E. coli, etc [6].

Even though computational power has made it possible to sequence genomes in limited time, several other problems exist. The sequence read from a machine is not always 100% correct. It may contain experimental errors. Human handling also causes errors in the input data. Some of the errors are due to low quality of the input. This adds additional problems to obtaining correct results. Another well known problem with these sequences is that they contain repetitive sections. In other words, certain sections of nucleotides, called repeats, are repeated in the whole sequences. We will elaborate on repeats in later sections.

### A. Genome Sequence Assembly

The problem of sequencing is not of an exact matching but obtaining approximate matches through consensus. Hence techniques try to obtain a consensus sequence through approximate matches by following an overlap and consensus scheme [5]. The process of reading chopped DNA

This work was supported in part by the EPSCoR-RING TRUE III-TASK 17 Fellowship (NSF EPSCoR Nevada).

S. Nasser is with the Department of Computer Science and Engineering, University of Nevada Reno, Reno, NV 89557. E-mail: sara@cse.unr.edu.
G. Vert is with the Department of Computer Science and Engineering, University of Nevada Reno, Reno, NV 89557. E-mail: gvert@cse.unr.edu.
M. Nicolescu is with the Department of Computer Science and Engineering, University of Nevada Reno, Reno, NV 89557. E-mail: monica@cse.unr.edu.
A. Murray is with the Desert Research Institute, Reno, E-mail: Alison@dri.edu

and creating a consensus sequence is shown in Fig 1.

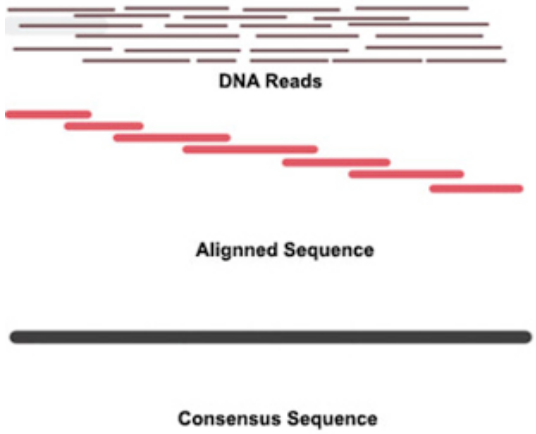


Fig. 1. Whole Genome Sequencing process is displayed in terms of reads from the DNA sequences.

### 1) *Whole Genome Sequencing*

The "whole-genome shotgun" method, involves breaking the genome up into small pieces, sequencing the pieces, and reassembling the pieces into the full genome sequence. This is the point where we are trying to put the puzzle together. Process of sequencing DNA using Shot-gun sequencing method was introduced in 1995 [3]. More details about whole genome shot-gun sequencing can be found in [1, 3].

### 2) *Sequence Alignment*

The process of DNA sequencing begins by breaking the DNA into millions of random fragments, which are then given to a sequencing machine. Since the process of selecting the fragments is random using the sequence one may not cover all the regions. Therefore multiple copies of original sequence are used to ensure that the entire sequence is covered. This is generally referred as a coverage of *'nX',* where n is the number of copies. Coverage of 8X is widely accepted to be able to generate the entire sequence. Next, a computer program called an *assembler* pieces together the many overlapping reads and reconstructs the original sequence [2].

The process of DNA sequencing can thus be divided into several steps, as shown in Fig 2.

a) Reading Sequences
b) Assembling
c) Finishing

#### *a) Reading Sequences*

Reading sequences is the process where a sequence reader reads raw sequences of DNA. The sequencer can only handle short sequences at a time. Hence the entire sequence cannot be fed into the sequencer. The output of the sequencer is sequences that are read into files called chromatograms. A Fourier transform on the chromatogram files can be done to figure out the bases from the chromatogram files. The results of this process are subsequences that are readable by human or any simple program. The output is now in the form of characters A, C, T, G, which represent nucleotides.

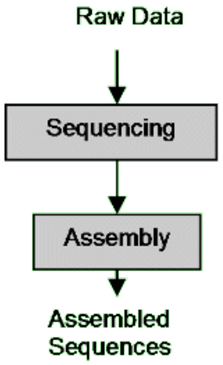


Fig. 2. The major steps involved in the process of assembling sequences.

#### *b) Assembly*

After sequences are read and converted to the data files, they need to be put together. We have pieces of sequences that need to be put in the right place. A huge amount work involved in sequencing lies in putting together these sequences in right place. This process involves creating consensus sequences as shown in Fig 1. The Contiguous or overlapping sequence of DNA is also known as a Contig. Sequences are searched for matching regions and if a significant portion of these sequences overlap they can be combined into a Contig. More commonly, this problem is of obtaining a longest common subsequence (LCS). A LCS is a common subsequence that belongs to two or more sequences.

The process of obtaining sequences is error prone, various problems occur such as errors in reading and flips. The process of assembly has to be robust to deal with these issues. The problems that occur during sequencing will be discussed later.

#### *c) Finishing*

Finishing is a process that is generally done at the end of the assembly process. This is a manual process where scientists go through the assembled sequences and fill in any missing gaps. This process some times requires repeating the previous steps to obtain the gaps or missing pairs in the sequence on a smaller scale. The process of finishing and the effort and time required depends on the assembly; if the assembly was good then finishing becomes easier.

## B. *Problems with Assembly*

In this paper we discuss problems with the sequence assembly. A lot of work has been done in sequencing genomes, but there are still challenges remaining. One of the reasons as we described above was due to the fact that precise alignment of subsequences is a hard problem. As genome sequences are huge in size even with the computational power available it is not possible to obtain best solution in a given time. The process of assembly becomes difficult because we have to look at several hundreds of subsequences before we can determine if they can be assembled together. The time complexity for a dataset of two subsequences of length k is $\Theta$ ($2^k$). Since this problem is NP-Hard, and we need to search through several hundreds or thousands of pairs, a brute-force force approach cannot be considered as a viable option.

The following section discusses the techniques and lists some of the tools available for sequence assembly. Section III provides details about dynamic programming, in Section IV the fuzzy approach and extensions to dynamic programming are proposed, Section V contains the results, followed by conclusion in Section VI.

## II. Techniques

The techniques or processes involved in sequence assembly can be divided into two steps, i) arranging the sequences in order to obtain consensus sequences, ii) recover the data lost during the experimental process.

The area of sequence assembly has been a very prominent area in computational biology or bioinformatics. Extensive work has been done to determine optimal methods for sequence assembly. Techniques such as Neural Networks, Hidden Markov Models, and Bayesian Networks have been used. Most of these techniques are computationally expensive and require high performance computing with huge training. Genetic Algorithms have also been used to perform sequence assembly [11]. Even though Genetic algorithms take a long time to run and converge they are computationally less expensive than Neural Networks. Several algorithms and issues with them are discussed in [2]. There are several tools for sequence assembly including Phrap, TIGR assembler, Celera assembler.

Currently most of the sequence applications do not tolerate any kind of inexactness or errors in sub sequence matching. String matching in nucleotide sequences is challenged by variation because there are few concepts in matching such as LIKE, NOT LIKE, or SIMILAR. Even though there have been methods where scores are calculated based on factors for similarity, these methods still try to find a crisp match.

Symbolic sequential data can be considered as either (1) exact matching or (2) approximate matching (most similar match). Quite often in real world data mining applications, especially in molecular biology, exact patterns do not exist and therefore, an approximate matching algorithm is required. An algorithm that performs a match to a certain degree is desired.

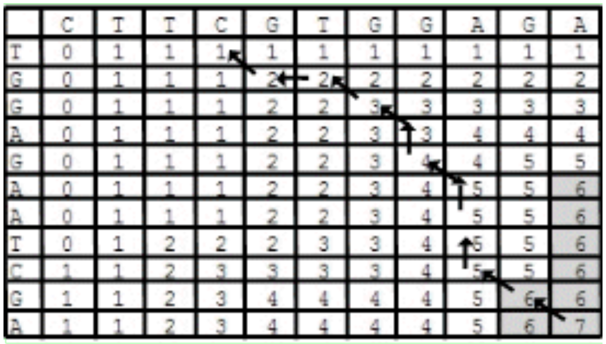
Another problem is with the existing tools for alignment. Majority of these tools are designed to work with good quality data. Data that is of low quality is not used in the consensus sequence. But if most of the data is low quality these tools fail to align the sequences and result in fewer Contigs. The quality of the data is measured as the probability of the correctness of a nucleotide read from the sequencer. The raw data is in form of chromatograms; these curves have different peaks which represent each of the 4 nucleotide bases, ‘A’, ‘C’, ‘T’ and ‘G’. Some times it is easy to determine a base from these peaks. Sometimes a base is not represented strongly. Generally, the quality of the bases is weaker at the ends and stronger in the middle of a subsequence. This is due to the behavior of the sequencing machine. Thus, there is a probability associated with each base read which indicates the strength of the read. This probability is called the quality of the base. The quality of the base is read during assembly of sequences. If the quality is low, the base is not considered in the alignment process.

For example *Phred* is software that can be used to generate base pair files [10]. *Phred* reads DNA sequence chromatogram files and analyzes the peaks, and determines the base from each peak; assigning quality scores ("Phred scores") to each base. *Phred* assigns quality within the range of 4-60 to the bases. A quality of 15 is generally considered as the lower bound. A quality of 20, which means 99% accuracy of the base, is universally accepted.

## III. Dynamic Programming

Dynamic programming, like the divide-and-conquer method, solves problems by combining the solutions to subproblems [15]. It can also be defined as a method for reducing the runtime of algorithms which exhibit the properties of overlapping sub-problems and optimal substructure [7]. Dynamic programming has been extensively used to determine the LCS. The reason for its popularity is that its time complexity is $\Theta(n^3)$. To assemble a genome we need to compare multiple sequences, thus the complexity for the assembly process can go up to $\Theta(n^4)$, unless modifications of dynamic programming are used. One such popular modification of dynamic programming is the Smith-Waterman algorithm with an $\Theta(n^2)$ time complexity.

Other techniques for finding the longest common subsequence include, suffix tree, KMS algorithm, greedy approaches. The *KMS Algorithm* identifies best matches of the longest substrings of the matches of many strings [8]. A greedy algorithm can be used for aligning sequences that differ by sequencing errors [12]. A greedy approach can be much faster than traditional dynamic programming but cannot be generalized. Fig. 3 shows the table constructed while using dynamic programming. The method of finding a LCS is to start from the end of the table and traverse along the direction indicated by the cell. The numbers in the cells indicate the length of the subsequence until that cell. The highest number in the table indicates the longest subsequence that can be found between the two sequences. Since the longest subsequence will most likely occur in the cells along the diagonal. Note that by simply traversing the table we can obtain the longest common subsequence. This may not be a contiguous subsequence. 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. This can be easily modified to find contiguous subsequences. In case of genome subsequences we would like to get the longest subsequence with few insertions or deletions (indels). One of common techniques used by assembly processes such as, Phrap is to search within a bandwidth along the diagonal. If the path is beyond the bandwidth the indels increase, and it is not a good match. The diagonal arrows within each cell indicate a match, the more diagonal arrows, we stay within the bandwidth, as mismatches increase we either go up or left.



| | C | T | T | C | G | T | G | G | A | G | A |
|---|---|---|---|---|---|---|---|---|---|---|---|
| T | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| G | 0 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| G | 0 | 1 | 1 | 1 | 2 | 2 | 3 | 3 | 3 | 3 | 3 |
| A | 0 | 1 | 1 | 1 | 2 | 2 | 3 | 3 | 4 | 4 | 4 |
| G | 0 | 1 | 1 | 1 | 2 | 2 | 3 | 4 | 4 | 5 | 5 |
| A | 0 | 1 | 1 | 1 | 2 | 2 | 3 | 4 | 5 | 5 | 6 |
| A | 0 | 1 | 1 | 1 | 2 | 2 | 3 | 4 | 5 | 5 | 6 |
| T | 0 | 1 | 2 | 2 | 2 | 3 | 3 | 4 | 5 | 5 | 6 |
| C | 1 | 1 | 2 | 3 | 3 | 3 | 3 | 4 | 5 | 5 | 6 |
| G | 1 | 1 | 2 | 3 | 4 | 4 | 4 | 4 | 5 | 6 | 6 |
| A | 1 | 1 | 2 | 3 | 4 | 4 | 4 | 4 | 5 | 6 | 7 |

Fig. 3. Table constructed using Dynamic Programming to find the Longest Common Subsequence.

The optimal subsequence could be one with perfect matches, or in some cases the users could tolerate indels more than in other cases. These criteria can depend on the use, the source of the data, quality of the data, etc. Almost all existing techniques provide thresholds where users can choose where to cut off. Sometimes the user is not clear on the ideal cut off point for a particular data set, and may need to determine it empirically. For example, if the cutoff value for the maximum gap allowed is 30 bases and there are fairly large numbers of sequences with a gap of 31 and 32, we will not be including these sequences, even though they are close. Due to the fact that these techniques allow for crisp matches only. On the other hand we can represent a match of 30 and lower with a fuzzy value of 1, which is for crisp matches. Matches those are very close 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. The user in this case does not have to look into the data and change parameters and run the program several times. Since there are several parameters the user may not even know which parameter needs to be altered? The main objective is to obtain the best consensus overall.

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.

The application of Fuzzy Logic has not been explored much in the area of approximate matching or similarity measures for genome assembly. Fuzzy Logic has been applied to classification problems in computational biology. Even though applications of fuzzy logic have not been done extensively, recently it started gaining popularity. A modified fuzzy k-means clustering was used to identify overlapping clusters of yeast genes based on published gene-expression data following the response of yeast cells to environmental changes [9].

## IV. Fuzzy LCS

The main objective of our method is assembling data by approximate matching using fuzzy logic. To achieve this we provide several matches of two subsequences. Then to pick the best match based on the criterion specified by the user. We use dynamic programming to demonstrate the use of fuzzy rules as it is the most commonly used method.

Fuzzy Logic has been used in approximate string matching using distance measures, etc. However, very little work has been done in the application area of building genomes from subsequences of nucleotides. Moreover this process becomes computationally expensive because multiple comparisons have to be performed for each possible string pair. The accuracy of any fuzzy matching system is partially determined by the error model used. An accurate system reflects the mechanism responsible for the variations in the match. Therefore a flexible error metrics is desired that is generic for any fuzzy matching. 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 by algorithms that otherwise may be important in deriving a genomic sequence and the metabolic characteristics of such a sequence.

### *A. Modifications to Dynamic Programming*

One of the problems as mentioned earlier with existing techniques is that they have crisp bounds. The user has to specify the parameters for the program. The parameters need to be changed by the user to suit the data, and then the program is run one or more times, until an optimal solution is found, since the user has to determine which parameters work best with the given sample.

Sometimes selecting the longest sequence is the optimal solution. Some applications prefer longer sequences; in some other cases a long sequence with higher quality is preferred. If this criterion is not satisfied the sequences are generally not selected. We propose a method where we select more than one subsequence and then based on several parameters select the optimal solution. If the sequence satisfies the aggregate overall requirement it is selected. This selection is based on fuzzy values. 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. The similarity is lesser if the indels were more.

Firstly, we would like to select several overlapping regions. We do not want to select every possible overlap. Since finding a fairly longer subsequence is better we select this based on length. Based on the longest possible subsequence that can be obtained we select every subsequence that is within a range 'x' of the LCS. The fuzzy value for each possibility is calculated and if it is satisfied the subsequence is selected.

In Fig 3, the dynamic programming table is illustrated. The arrows indicate the longest common sequence path. The darker shaded cells indicate all the cells that will be traversed to search for the optimal subsequence. Since we perform a non-banded search it's not ideal to search every cell. A cell is marked if it was already traversed, so we don't check it again. We add a new table that will keep track of cells that are traversed and everything under a threshold is selected. We select subsequences which have:

$$\mathit{length} >= \mathit{threshold}$$

Instead of selecting the longest common subsequence from the dynamic programming table, we select all the subsequences that satisfy a minimum length required. Then we determine if either of these subsequences will create an optimal match using similarity measures. Each of the matching regions has a similarity measure associated with them.

Cells that have at least 6 matching base pairs are selected for traversal. If we skip cells with less than 6 matching base pairs and it's a high similarity subsequence, then they will be picked up in later iterations and only the ends will be lost. Therefore this rule does not eliminate any possible good subsequences.

In case of multiple sequence alignment we need to compare sequences to determine which ones can form a consensus or Contig as shown in Fig 1. This process requires several comparisons and it is not easy to determine which sequences would yield the better Contigs. Hence we propose to calculate fuzzy values for each Contig, where higher value indicates a better Contig. The parameters for this would be same as the parameters to select a subsequence. The selection process is done in constant time; therefore the complexity of the algorithm is same as the complexity of Dynamic programming, which is Θ(mn) for any two subsequences of length *m* and *n*. The following section lists the characteristic functions.

### B. Fuzzy Similarity Measures

Fuzzy similarity measures are an important step in creating a Contig from two subsequences or finding an overlap between two sequences. We use the term match to refer subsequence.

(i) Length of Overlap ($\mu_{lo}$): The first parameter 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. The higher overlap is better.

$\mu_{lo} = f_{\mathrm{n}}$ (Overlap)

(ii) Confidence ($\mu_{qs}$): The confidence for each Contig is defined as, a measurement of the quality of the contributing base pairs. The quality of a base pair indicates if the read was strong and 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 it contributing bases. For simplicity, the sum of average quality scores is the confidence of the Contig. $\mu_{qs}$ is the qs for the overall overlap region, which we calculate as follows

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

$w_i$ is used to standardize the quality scores. The bases with high quality are given a weight of 1. Only the bases that are of lower quality are given weights between 0 and 1. uQS (δ) is the standard bound for threshold that was explained earlier, this is generally specified by the user, $\min_{qs}$ and $\max_{qs}$ are the minimum and maximum values for quality.

$$w_i = \begin{cases} 1, & \mathit{if} \quad \mu_{qs} \geq \delta \\ 0, & \mathit{if} \quad \mu_{qs} = 0 \\ \dfrac{\mu_{qs} - \min_{qs}}{\max_{qs} - \min_{qs}} & \end{cases} \tag{2}$$

At present a simple function is used for weights. We would like to update this with standard functions such as a Gaussian, Sigmoid, etc.

(iii) Gap Penalty ($\mu_{gp}$): This is the maximum gap that is allowed in a match. Gap is measured in terms of the number of bases.

$$\text{Gaps} = \sum(f_{\mathrm{n}}(\text{insert}) + f_{\mathrm{n}}\ (\text{delete}) + f_{\mathrm{n}}\ (\text{replacement})) \tag{3}$$

$$\mu_{gp} = (f_{\mathrm{n}}\ (\text{overlap}) - \text{Gaps}) / f_{\mathrm{n}}\ (\text{overlap}) \tag{4}$$

(iv) MinMatch ($\mu_{wl}$): This is the minimum number of matching bases that are required between the two sequences. Genomic DNA contains only 4 characters and they can be lot of overlaps with these 4 characters. Therefore we would like to have a cutoff value for matching sequences. We also refer to this as the word length.

$$\mu_{wl} = f_{\mathrm{n}}\ (\text{matchingBP}) \tag{5}$$

(v) MinScore: This is the minimum score of a match. A score is calculated from the number of matching bases, number of indels and replacements. MinScore is used as a threshold. Score can be calculated in different ways. For example:

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 value one the score is

$$score = f_n(\text{MatchingBP}) - f_n(\text{Inserts}) - f_n(\text{Deletes}) - f_n(\text{Replacements}) \quad (6)$$

$$score = 11-2-1-1 = 7$$

Some methods weight the matching base pairs higher than the indels, so the score might be different.

Once the fuzzy value for each of these parameters is calculated, we plug them in an overall fuzzy function. This function is the aggregate fuzzy match value. We currently use only 4 of the above parameters in the aggregate function. At this point minimum score is used for checking if a score is below a certain threshold. In the current implementation we fix the value of weights. Later we would like to calculate the weights.

$$fa(c) = \mu_{qs} w_{qs} + \mu_{wl} w_{wl} + \mu_{gp} w_{gp} + \mu_{lo} w_{lo} \quad (7)$$

$$afv = fa(c)/m \quad (8)$$

The subsequences that produce the highest fuzzy value are selected as optimal sequences. Depending on their position as a suffix or prefix a new Contig or consensus sequence is formed. The new Contig has new consensus quality scores. In some cases they are subsequences that overlap only at the middle. For example,

*tcaatgttactagtgaatatttctatgatgaactgaagaa*  
-------------*agtgaatatttct*----------------------

We cannot create a new Contig with these subsequences. For these sequences we update the confidence of the consensus or the Contig. This makes it possible for them to be used in further matches since their confidence is increased. For example, if sequence Y has low quality bases and matches with other sequences of higher quality then the confidence of Y increases.

## C. Repeats

Repeats or repetitive patterns in a sequence are quite common in genomic data. Repeats make the process of assembly complicated. The sequences not only need to be put together into a Contig but the sequences that are similar and in a Contig may belong to different regions of the genome. We also need to evaluate the sequences are matched if they may be from different repeats.

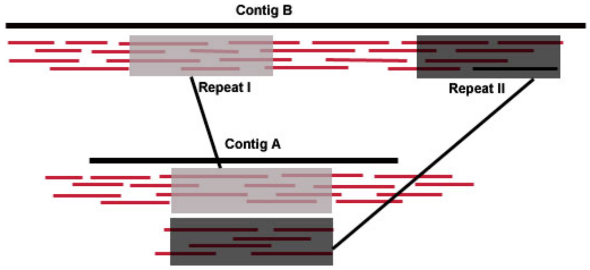


Fig. 4. Repeats in a Contigs can create mis assembly. As can be seen Contig A has repeats put together. Contig B would be the correctly assembled Contig.

Repeats are illustrated in Fig 4. Contig A is created by two repeats combining together and hence causing a mis assembly. Contig B is the actual Contig that is desired, where repeat I and repeat II are highly similar but still are different parts of the genome. There are ways to resolve issues of repeats. For example if a Contig has significantly large number of reads then it could be an incorrect assembly. This could be determined by the coverage of the genome.

The problem with repeats is that may lead to an incorrect assembly therefore we need to determine if the assembly contains repeats. Some assemblers count the depth of the match, if it is high then there are higher chances that it was a repeat. Since the sequences are sequenced X times, there should not be more than X occurrences of a segment. We use depth to refer to the number of subsequences that are contributing to the Contig. Counting the depth may not work since we are selecting the subsequences randomly, certain regions may be selected more times than the others. This process may work if some of these highly similar regions are removed. But the problem still occurs with repeats that are not detected because their depth was less and these regions were not represented very well.

Even though these repeats are highly similar they are very unlikely same. Small differences can be detected in these repeats to separate them. In the absence of sequencing errors, a single nucleotide difference between two copies of a repeat is enough to distinguish them [2]. Hence once these regions are detected we can try to find dissimilar ones among them.

In our approach we created Contigs by pair wise comparison of sequences or sequence and a Contig, and we also get multiple results, hence for the problem shown above we get both Contig A and Contig B. Now we have to determine which one of these Contigs is correct. Each Contigs is associated with a fuzzy similarity value. The fuzzy similarity value is an aggregate of several fuzzy values such as, the length of the Contig, the confidence of the Contigs, the number of mismatches, maximum length of words, etc. If the fuzzy similarity measure is taken for both Contig A and Contig B, there are more chances that Contig B will be chosen. The first reason is that the length of Contig

B has a higher value. Assuming that the repeats have slight dissimilarities the match fuzzy value would be lower for Contig A than Contig B. Finally, the confidence score of Contig A would be lower if we have even a single nucleotide that was different than the other. Hence all these factors would result in selecting the Contig with the higher fuzzy value that is selecting Contig B.

## V. EXPERIMENTS, RESULTS AND DISCUSSIONS

The Fuzzy Genome Sequence Assembler was implemented with modified version dynamic programming described earlier. The parameters selected for the assembly are, length of overlap, confidence, minscore, gap penalty, and minmatch as described in section IV. The assembler was tested on generated data sets and data from GenBank. GenBank is a publicly available database of nucleotide sequences. Artificially generated data sets were used to verify the algorithm and thus the assembly process. The experiments were run on a G5 with a 1.83GHz Intel Core 2 Duo processor and 4GB of RAM.

The first genome sequence tested was the Wolbachia endosymbiont of the Drosophila melanogaster strain wMel 16S ribosomal RNA gene, partial sequence, which can be obtained from GenBank. Wolbachia is a microscopic organism that has been used to test several alignment tools. The gene is "rpoBC", locus_tag="WD0024" and the GeneID is “2738525” [13]. This particular sequence codes for a protein. The sequence contains 8514 base pairs. The total bases read were 4X of the original sequence. We selected 300 random fragments or subsequences from this set. Each subsequence was in the range of 300-600bps. Fragment sizes less than 500bp are commonly used for assembly [11, 18]. Therefore we chose an average fragment/subsequence length 450bp. The results of assembly are shown in Table 1. In Table 1, MGS refers to an implementation of Smith-Waterman algorithm for multiple sequence alignment using Dynamic programming [16]. HGA-GS is a heuristically tuned GA, which used a 4X coverage and 500 subsequences with an average length of 400 bp [11], TIGR is a well-known assembler [19]. TIGR assembler assembled the data into a final consensus sequence. FGS is the fuzzy sequence assembly method that is described in the paper.

The second genome sequence we tested was the Arabidopsis thaliana, gene_id:F11I2.4. Detail of this sequence can be obtained from GenBank [14]. This sequence contains 36, 034 base pairs. We used 3X coverage of the original sequence. 300 sequences of length between 300-600 bps were randomly generated. The result of assembly on this data is shown in Table 2.

The results obtained from assembling both the genome projects showed a fairly high percentage of the genomes covered. This indicates that given random subsequences the algorithm was able to create a fairly large percentage of the original sequence. For the Wolbachia project 99.6% of genome was recovered which is similar to the TIGR assembler. For Arabidopsis thaliana 92% of the genome is recovered. Even though the results are better than the other techniques, we suspect the smaller percentage could be due to the smaller amount of initial coverage; only 3X was used in this case. We could not use larger coverage or test on larger genome projects due to limitations of available hardware.

## VI. CONCLUSION

This paper proposes use of fuzzy logic for approximate sequence assembly. Preliminary fuzzy characteristic functions are proposed which suggest one possible approach towards utilizing fuzzy logic in assembly. We tested the FGS assembler on published genome projects and compared the results with other assemblers. The results obtained clearly demonstrate that the FGS assembler can generate optimal assembly. The results show that FGS covered larger or same length of genome sequence as the other assemblers. FGS has the same run time as LCS with Dynamic programming. This technique addresses the problems caused by repeats and low quality data.

The functions proposed can be easily adapted in other assembly methods or techniques. Fuzzy logic can also be used in a similar fashion for database querying, since the approach proposed can be easily generalized for database search problems.

TABLE I
ASSEMBLY COMPARISONS ON RPOBC OF WOLBACHIA GENOME

| Assembler | Number of Contigs | Average Length | Percentage Genome Covered |
|---|---|---|---|
| MGS | 106 | 853 | 65% |
| *TIGR* | 150 | 501 | 99.6% |
| *HGA_GS* | 181 | 301 | 82% |
| *FGS* | 165 | 608 | 99.6% |

MGS = Multiple Genome Sequencing using Dynamic programming, TIGR=TIGR Assembler 2.0, HGA_GS =Heuristically tuned GA for assembly, result taken from [11], FGS= Fuzzy Genome Sequencing, Number of Contigs= total number of Contigs obtained after assembly, third column is the average length of the Contigs, fourth column is the percentage of original genome covered by the assembly.

The assembly can be further improved by applying techniques such as scaffolding. Another improvement would be 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

TABLE II
ASSEMBLY COMPARISONS ON ARABIDOPSIS THALIANA

| Assembler | Number of Contigs | Average Length | Percentage Genome Covered |
|---|---|---|---|
| MGS | 144 | 940 | 56.8% |
| *TIGR* | 102 | 502 | 88.8% |
| *FGS* | 190 | 842 | 92.135% |

MGS = Multiple Genome Sequencing using Dynamic programming, TIGR=TIGR Assembler 2.0, FGS= Fuzzy Genome Sequencing, Number of Contigs= total number of Contigs obtained after assembly, third column is the average length of the Contigs, fourth column is the percentage of original genome covered by the assembly.

can be achieved by encoding the data, using hash tables, indexing, etc.

## VII. Future Work

The work proposed in this paper is preliminary; there is room for enhancements and extensions. We would like to refine each of the parameter's fuzzy characteristic functions. Fuzzy matching calculates the degree to which the input data match the conditions of the fuzzy rule; this can be used to determine the degree of similarity [16]. One of our main goals is to describe each match in terms of degrees of similarity. Some other future goals include:

a) Enhancing fuzzy matching techniques and using them with other methods that are used for sequence assembly such as suffix trees.

b) Using the fuzzy approximate methodology for database searches of genomes.

c) Secondary structure prediction to enhance the assembly process, using fuzzy similarity matrix.

d) Creating an assembly tool for meta-genomic data that is based on fuzzy logic.

## Acknowledgment

The authors wish to thank Martin Gollery. Martin Gollery is supported by NIH Grant Number P20 RR-016464 from the INBRE Program of the National Center for Research Resources. This work was supported in part by a grant from NSF EPSCOR Nevada.

## References

[1] Gene Myers, Whole-Genome DNA Sequencing, IEEE Computational Engineering and Science 3, 1 (1999), 33-43.

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

[3] F. Sanger et al., "Nucleotide Sequence of Bacteriophage Lambda DNA," J. Molecular Biology, vol. 162, no. 4, 1982, pp. 729-773.

[4] Mihai Pop, Adam Phillippy, Arthur L. Delcher and Steven L. Salzberg, "Comparative Genome Assembly", HENRY STEWART PUBLICATIONS 1467-5463. BRIEFINGS IN BIOINFORMATICS. VOL 5. NO 3. 237–248. SEPTEMBER 2004.

[5] 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.

[6] Genome Wikipedia, http://en.wikipedia.org/wiki/Genome , Accessed, October 2006.

[7] Dynamic Programming- Wikipedia, http://en.wikipedia.org/wiki/Dynamic_programming, Date accessed Oct 2, 2006.

[8] K Kaplan. An Approximate String Matching Algorithm with Extension to Higher Dimensions. UMI Microfilm. 1995.

[9] Gasch, A. P. & Eisen, M. B. Exploring the conditional coregulation of yeast gene expression through fuzzy k-means clustering. Genome Biol 3, RESEARCH0059 (2002).

[10] Phred Quality Base calling, http://www.phrap.com/phred/#qualityscores, date accessed Oct 3 2006.

[11] Satoko Kikuchi and Goutam Chakraborty," Heuristically Tuned GA to Solve Genome Fragment Assembly Problem", IEEE Congress on Evolutionary Computation, Vancouver, Canada, 2006. pp. 5640-5647

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

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

[14] Arabidopsis thaliana genomic DNA, chromosome 3, BAC clone:F11I2, http://www.ncbi.nlm.nih.gov/entrez/viewer.fcgi?db=nucleotide&val=7209730, Date accessed Oct 2006.

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

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

[17] J. Yen, R. Langari, "Fuzzy Logic Intelligence, Control, And Information", 1999, Prentice Hall.

[18] K. Mita, et al., "The Genome Sequence of Silkworm, Bombyx mori," DNA Research 11(1), pp.27-35, 2004

[19] TIGR Assembler 2.0, http://www.tigr.org/software/assembler/, Date accessed Oct 2006.