Fuzzy Classification of Genome Sequences Prior to Assembly Based on Similarity Measures[*]

Sara Nasser, Gregory L. Vert, Adrienne Breland and Monica Nicolescu

Department of Computer Science Engineering
University of Nevada Reno
Reno, Nevada 89559, USA

sara@cse.unr.edu gvert@cse.unr.edu monica@cse.unr.edu abreland@cse.unr.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. However, the existence of insertions, deletions and substitutions can complicate the assembly of subsequences and confuse existing methods. What has been needed is an approach that deals with ambiguity in trying to match and assemble a genome from its sequenced subsequences. This research develops fuzzy similarity measures between subsequences that are then incorporated into an assembler based on fuzzy logic and fuzzy similarity measures. The research addresses the problem of extensive computation required by clustering data into meaningful groups. Preliminary evaluation of this approach in conjunction with K-Means clustering suggests that this approach is at least as good as standard approaches and in some cases better.**

I. Introduction

Genome shotgun sequencing for DNA alignment is a slow 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 [1]. Therefore DNA is chopped up into small subsequences.

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 subsequences 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 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 software called an *assembler* pieces together the many overlapping reads and reconstructs the original sequence [2].

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 [3]. More details about whole genome shot-gun sequencing can be found in [1, 3].

The problem of sequencing is not of exact matching but requires obtaining approximate matches through consensus. A consensus sequence is constructed through approximate matches by following an overlap and consensus scheme [5], this is illustrated in Fig. 1 below.

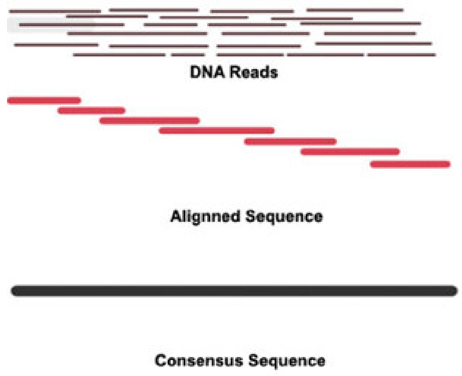


Fig. 1. Whole Genome Sequencing process is displayed in terms of reads from the DNA sequences.

Even though computational power has made it possible to sequence genomes, assembly is still an NP-Hard problem. An exhaustive search is not a viable option. Heuristic approaches that reduce the search space have been used.

The algorithm presented in this paper speeds up the original assembly by reducing search space. Assembling sequences can be done by first sorting the sequences into classes. Some sequences can have a higher similarity with each other and some other may have lesser or no similarity at all. We perform a meaningful partitioning of data, so that sequences in a cluster have high similarity with one another and sequences between

[*] This work is partially supported by NSF EPSCoR Fellowship (RING-TRUE III Award number: 0447416).

two clusters are less similar. A Divide-and-Conquer approach to consign data into similar groups is used.

The following Section discusses the previous work done in sequence assembly. Section III contains the approach. Section IV has the experimental results followed by conclusions in Section V and future work in Section V1.

II. Previous Techniques

In general, the approach to fragment 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 numerous 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 that can, among other things, save computation time. Finding the longest common subsequence between fragments is the key to the process of sequence assembly. Dynamic programming is used in the Smith-Waterman algorithm for multiple sequence alignment [6], one the most prominent algorithms used in sequence assembly programs. Dynamic programming solves problems by combining the solutions to sub problems [15], in this case substrings. It is a method for reducing the runtime of algorithms containing overlapping sub-problems and optimal substructures [7]. Other techniques for finding the longest common subsequence include suffix trees, the KMS algorithm and greedy approaches. Suffix trees allow a linear time search for matching substrings. The *KMS Algorithm* identifies best matches of the longest substrings of the matches of many strings [8]. Greedy algorithms can be much faster than traditional dynamic programming and work well with sequencing errors [12].

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[23], 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 [24]. 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 [22] 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 [25]. 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 [20]. 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 [17] is widely used for its simplicity. Another variation of K-means was proposed by Forgy [18]; this algorithm has been shown to converge to a local minimum [11].

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 [26]. Fuzzy Logic is used in problems where the results can be approximate rather than exact. Hence, the principles of fuzzy logic suit well to clustering problems. The results are determined by some degree of closeness to true or to 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 [27]. An improvement of K-means using the fuzzy logic theory was presented [14] in which the concept of fuzziness was used to improve the original K-means algorithm.

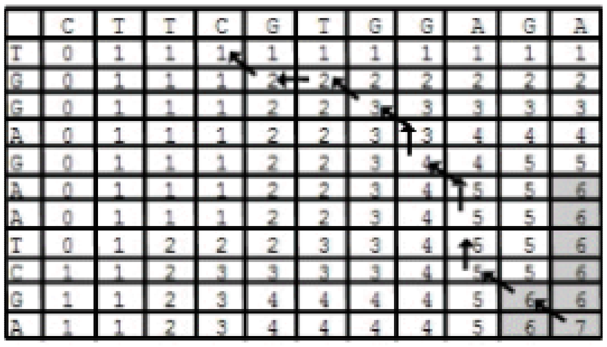
III. Clustering and Assembly

In this paper we present a clustering technique that uses a fuzzy membership function to divide fragments 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 [4].

A. Dynamic Programming 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 $\Theta(n^2)$.

Fig. 2 shows the table constructed while using dynamic programming. Two strings subject to comparison label the x and y axis. To find the LCS 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. 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 the case of genome subsequences we would like to get the longest subsequence with few insertions or deletions (indels). One of the 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. 2 Table constructed using Dynamic Programming to find the LCS

The optimal subsequence is either 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. 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, assume a 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. Due to the fact that these techniques allow for crisp matches only, these potentially important sequences would be excluded. 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 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. 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 parameters need to be altered. 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.

Fuzzy Logic has been applied to classification problems in computational biology. Even though applications of fuzzy logic have not been done extensively, recently it has begun to gain popularity. A modified fuzzy k-means clustering was used to identify overlapping clusters of yeast genes. Data was based on published gene-expression results following the response of yeast cells to environmental changes [9].

B. *Fuzzy LCS*

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. 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. 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 less gaps.

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 pick the best match based on the criterion specified by the user.

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 by algorithms that otherwise may 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 parameters select the optimal solution. 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.

We perform a non-banded search, it is neither ideal to search every possible subsequence or just use the diagonal. Instead of selecting the longest common subsequence from the dynamic programming table Fig. 2, 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:

$$\text{length} >= \text{threshold},$$
$$\text{where threshold} = f(n(LCS))$$

We need determine if either of these subsequences will create an optimal match. Any of the selected paths can generate the 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 Θ(mn) for any two subsequences of length *m* and *n*. The following section lists the characteristic functions.

C. Fuzzy Similarity Measures

Fuzzy similarity measures 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 we consider 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:

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

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

(ii) Confidence (μ_{qs}): The confidence for each Contig is defined as, a measurement of the quality of the contributing base pairs [10]. A high quality base pair indicates a strong read or a higher confidence in its accuracy. Noise and experimental error are often present in reads that makes the confidence lower. 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. μ_{qs} is the quality score (qs) for the overall overlap region, which we calculate as follows

$$\mu_{qs} = \frac{\sum_{i=1}^{n} w_i q_i}{n} \quad (2)$$

w_i is used to standardize the quality scores. 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. 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.

$$w_i = \begin{cases} 1, & if \quad \mu_{qs} \geq \delta \\ 0, & if \quad \mu_{qs} = 0 \\ \dfrac{\mu_{qs} - \min_{qs}}{\max_{qs} - \min_{qs}} \end{cases} \quad (3)$$

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

$$\text{Gaps} = \sum(f_n(\text{insert}) + f_n(\text{delete}) + f_n(\text{replacement})) \quad (4)$$

producing a membership function of:

$$\mu gp(s1,s2) = \begin{cases} 1 \mid \mid \text{overlap} \mid = \mid \text{mbp}(s1,s2) \mid \\ 0 \mid \mid \text{overlap}(s1,s2) \mid = \mid \text{diff}(s1,s2) \mid \\ [0,,1] \mid {}^{(1 - \mid \text{mbp}(s1,s2) \mid + \mid \text{diff}(s1,s2) \mid)} \big/ {}_{\mid \text{overlap}(s1,s2) \mid} \end{cases} \quad (5)$$

Here mbp(s1, s2) is the matching number of nucleotides in s1, s2 and diff(s1, s2) is "inserts(s1,s2) + deletes(s1, s2) + replacements(s1, s2)".

(iv) Score (μ_{ws}): This is the score of a match also known as similarity of a match. A score is calculated from the number of matching bases, number of indels and replacements. If all are given value one a using a simple scoring method, here i-inserts, d-deletes, r-replacements:

$$score = f_n(\text{MatchingBP}) - f_n(i) - f_n(d) - f_n(r)$$

We can weight the matching base pairs higher than the indels. The above equation leads to derivation of the following fuzzy similarity membership function.

$$\mu ws(s1,s2) = \begin{cases} 1 \mid \text{score} = \mid fmbp(s1,s2) \mid \\ 0 \mid \text{score} <= 0 \\ [0,,1] \mid {}^{(\text{score})} \big/ {}_{\mid \text{overlap}(s1,s2) \mid} \end{cases} \quad (6)$$

Here fmbp (s1, s2) is the score of matching base pairs.

D. Thresholds

(i) MinMatch: 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.

(ii) 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.

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 (afv). A perfect overlap refers to an overlap that satisfies the two thresholds above, is free of gaps, and satisfies the quality requirements.

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

$$afv = fa(c)/m, \text{ where m is number of parameters} \qquad (8)$$

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.

C. K-means Clustering

Clustering problems generally derive some kind of similarity between groups of objects. 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. 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.

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

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 can be calculated as follows:

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

IV. Experiments and Results

The Fuzzy Genome Sequence Assembler with Clustering (ClusFGS) was implemented and tested on generated data sets and several data sets from GenBank. We include results from one of the test cases below. The parameters selected for the assembly are length of overlap, confidence, score, gap penalty, minscore and minmatch as described in section IV. The 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 [18]. We selected k random seeds where k<<N. The results of assembly are shown in Table 1.

TABLE I
Assembly comparisons on RPObc of Wolbachia genome

Assembler	Number of Contigs	Percentage Genome Covered
MGS	106	65%
TIGR	150	99.6%
FGS	165	99.6%
ClusFGS	75	91%

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 percentage of original genome covered by the assembly.

In Table 1, MGS refers to an implementation of Smith-Waterman algorithm for multiple sequence alignment using Dynamic programming [16]. TIGR is a well known assembler [19]. FGS is the fuzzy sequence assembly method that is described in [4]. ClusFGS is the method described in this paper and is a modified version of FGS.

The results obtained 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. The clustering technique did not perform as well as the FGS with clustering but is still better than simple sequence alignment.

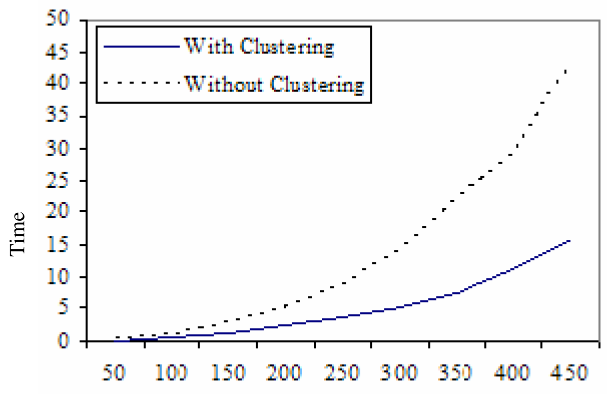


Fig. 3 Comparison of clustering on Wolbachia sequences

We compared the results with respect to time required for assembling. The results of clustering are shown above in Fig 3. The clustering technique is linear and hence can make the assembly much faster as indicated in the graph in Fig 3. As the value of k reaches the value of N the clustering technique becomes same as the FGS algorithm.

IV. Conclusions

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.

The results show that clusFGS was faster than FGS but could not recover the genome as much as FGS or TIGR assembler. ClusFGS can be used to get an estimate of assembly in short time, especially for large datasets where assembling the whole genome might be time consuming.

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.

The idea proposed can be used to group meta-genomic data into classes and provide a clean assembly for environmental sequences.

IV. Future Work

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. Future work will evaluate performance on eukaryotic and prokaryotic DNA which have different structural characteristics. 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.

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] Sara Nasser, Gregory Vert, Monica Nicolescu, Alison Murray, “Multiple Sequence Alignment using Fuzzy Logic”, Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB 2007), April 1-5, Honolulu, Hawaii, Vol, 07, pp 304-311.

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

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

[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. B. MacQueen, Some methods for classification and analysis of multivariate observations, Proc. 5th
Berkeley Symp. Probability Statistics, University of California Press, Berkeley, 1967, pp. 281–297.

[18] E. Forgy, Cluster analysis of multivariate data: efficiency versus interpretability of classifications, Biometrics, 21, 1965, 768–776.

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

[20] Hasan H. Otu, Khalid Sayood, “A divide-and-conquer approach to fragment assembly”, Bioinformatics 19(1): 22-29 (2003)

[21] Peltola H., Soderlund H., and Ukkonen E, “SEQUAID: a DNA sequence Assmebling problem based on a mathematical Model.” *Nucleic Acids Res*., 12, 307-321(1984)

[22] Chen and Skiena, “A Case study in genome-lebel fragment assembly”, *Bioinformatics*, 16(6):494-500(200)

[23] Green P. Documentation for Phrap. http//bozeman.mbt.washington.edu. Genome Center. University of Washington.

[24] Kim S, Segre AM, “AMASS: astructured pattern matchingapproach to shotgun sequence assembly”,*Journal of Computational Biology*, Summer;6(2):163-86 (1999).

[25] Huang X., Madan A, “CAP3:A DNA sequence assembly program”, *Genome Res*, Sep;9(9)868-77 (1999).

[26] L. A. Zadeh, Fuzzy logic and approximate reasoning. Synthese, vol. 30, 1975, 407--428.

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