University of Nevada
Reno

Fuzzy Methods for Meta-genome Sequence classification and Assembly

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science and Engineering

by

Sara Nasser

Dr. Frederick C. Harris, Jr./Dissertation Advisor

May 2008

University of Nevada, Reno
Statewide • Worldwide

THE GRADUATE SCHOOL

We recommend that the dissertation
prepared under our supervision by

SARA NASSER

entitled

Fuzzy Methods for Meta-genome Sequence classification and Assembly

be accepted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

Frederick C. Harris Jr, Advisor

Monica Nicolescu, Committee Member

Alison Murray, Committee Member

Sergiu Dascalu, Committee Member

Anna R. Panorska, Committee Member

Sami Fadali, Graduate School Representative

Marsha H. Read, Ph. D., Associate Dean, Graduate School

May, 2008

To my parents Ayesha Aziz and Hafeezuddin A. Nasser.

Acknowledgments

I would like to thank my advisor Dr. Frederick C. Harris Jr., it was his constant support and patience that made this journey easy for me. I hope to that in future I can be as encouraging and helpful to my students as he has been to me. Truly, he is a model advisor.

I would also like to thank my committee members Dr. Monica Nicolescu, Dr. Alison Murray, Dr. Sergiu Dascalu, Dr. Sami Fadali and Dr. Anna Panorska for their valuable time. Thanks to Alison and Monica for their insight into the problem and for guiding me. To Dr. Dascalu for meticulously reading my papers and presentations in the past. Each of the committee members has been valuable and has contributed in numerous ways.

Also, thanks to Adrienne Breland for working with me, and to Martin Gollery for introducing Bioinformatics to me. Thanks to Cindy Harris for reading my work and editing it. The Department of Computer Science and Engineering has been very supportive.

A very special thanks to my uncle Abdul K. Aziz and aunt Taneem Fatima Aziz, for their love and support that made it possible for me to get through the years in Reno. In the end, I would like to say that I cannot thank God enough for all the wonderful people and experiences at UNR.

March 17, 2008

Abstract

Traditional methods obtain a microorganism's DNA by culturing it individually. Recent advances in genomics have lead to the procurement of DNA of more than one organism from its natural habitat. Indeed, natural microbial communities are often very complex with tens and hundreds of species. Assembling these genomes is a crucial step irrespective of the method of obtaining the DNA. This dissertation presents fuzzy methods for multiple genome sequence assembly.

An optimal alignment of DNA genome fragments is based on several factors, such as the quality of bases and the length of overlap. Factors such as quality indicate if the data is high quality or an experimental error. Sequence assembly does not have crisp results and is based on degree of similarity. To address this challenge we propose a sequence assembly solution based on fuzzy logic, which allows for tolerance of inexactness or errors in fragment matching and that can be used for improved assembly.

Assembly of a single organism's genome is presented using a modified dynamic programming approach with fuzzy characteristic functions. The characteristic functions are used to select alignments of sequence fragments. Assembly of environmental genomes starts with the classification of mixed fragments from different organisms into homogeneous groups. Separating closely related species is a difficult task because the fragments contain many similarities. We propose fuzzy classification using modified fuzzy weighted averages to classify fragments belonging to different organisms within an environmental genome population. Our proposed approach uses DNA-based signatures such as GC content and nucleotide frequencies as features for the classification. This divide-and-conquer strategy also improves performance on larger datasets. We evaluate our method on artificially created environmental genomes to test various combinations of organisms and on environmental genomes obtained from acid mine drainage available at National Center for Biotechnology Information. The assembler is enhanced by a parallel version for high performance.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

DNA is the building block of all life on this planet, from single cell microscopic bacteria to more advanced creatures such as humans. In 1953 James Watson, Rosalind Franklin and Francis Crick cracked the DNA code and revealed the double helix structure, one of the most significant achievements of the century. Cracking the code made it possible to understand the structure of the DNA and, hence, lead the way for future research. Twenty years later, Frederic Sanger, a Nobel Laureate, invented the chain termination method of DNA sequencing, also known as the Dideoxy termination method or the Sanger method [56]. His research paved the way to a technique to obtain DNA sequences and to the first genome sequence assembly, Bacteriophage ϕX174 [55]. In 1990 the Human Genome Project was announced, which sought to sequence the billions of nucleotides present in human DNA. This was an ambitious project, projected to be completed in 2005. In 1993 The Institute for Genome Research (TIGR) developed an algorithm that could assemble thousands of expressed sequenced tags. TIGR decided to use the TIGR EST algorithm which is based on whole-genome shotgun sequencing method to assemble a microbial genome. Thus in 1995 at TIGR, *Haemophilus influenzae* was sequenced, becoming the first genome to be sequenced entirely [16]. Shotgun sequencing was first demonstrated to close a genome in this paper. The assembly of *H. influenzae* proved the potential of shotgun sequencing and thus lead to subsequent projects that would be based on shotgun sequencing. The Human Genome Project was finished in 2003, two years before its projected date. There were two major advancements in technology that lead the to complete sequencing of the Human Genome and the *H. influenzae*: shotgun sequencing and the use of computational techniques. This brief history gives insight into the advancements that were made in sequencing using computational techniques. For more history on the evolution of DNA sequencing and timelines of genome sequencing projects, refer to [11] and [44].

A DNA strand consists of four nucleotides: Adenine(A), Cytosine(C), Guanine(G) and Thymine(T). Any given DNA consists of these four nucleic acids. 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 proteins, and the building blocks of life. A genome sequence determines the characteristics of an organism. Genome sequences are large in size and can range from several million base pairs in prokaryotes to billions of base pairs in eukaryotes. For example, Wolbachia genome, a bacteria has 126 million base pairs (Mb) , Arabidopsis thaliana, a plant has 120Mb, and the human genome is 3.2 billion base pairs. The whole genome cannot be sequenced all at once because available methods of DNA sequencing can handle only 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 600 to 700 nucleotides at a time [46]. Current techniques can read up to 800 base pairs. The most recent techniques create much smaller sequence fragments with much duplication. Genomes are cut at random positions then cloned to obtain the smaller fragments, also known as shotgun sequences. Obtaining shotgun sequences has allowed sequencing projects to proceed at a much faster rate, thus expanding the scope of the realistic sequencing venture [44]. Sequencing DNA using the shotgun method led to the completion of several organism genomes, including human, mouse, fruit fly and several microbes, such as *Wolbachia* genome and *H. influenzae.*

Microorganisms live in communities, and their structure and behavior is influenced by their habitat. Most microorganisms genomes are known from pure cultures of organisms isolated from the environment, be it a natural organism-associated (i.e, human) or artificial system. Cultivation-based approaches miss majority of the diversity that exists however, such that development of cultivation-free methods has been implied. In the past, microbial DNA was sequenced by culturing microorganisms in a controlled environment. Cultivating these organisms did not reveal enough information about these communities of organisms. Invitro cultivation methods allow the extraction of DNA from only a limited selection of microbial species that can grow in artificial environments. These methods do little to characterize the properties of globally distributed microbes, because the vast majority of them have not been cultured.

New techniques in genomic sciences have emerged that allow an organism to be studied in its natural habitat as part of a community. Research has broadened from studying single species to understanding microbial systems and their adaptations to natural environments. These techniques have been achieved by developing methods that can sequence mixed DNA directly from environmental samples [3, 52].

This field of metagenomics (environmental genomics) involves the sampling of microbial DNA from natural environments rather than relying on traditional single-species cultivation techniques. This sampling, coupled with rapid developments in molecular biology, is changing our understanding of bacterial evolution and naturally existing microbial systems. Metagenomics is the application of modern genomic techniques to the study of microbial communities in their natural environments, bypassing the need for isolation and lab cultivation of individual species [9, 57, 14].

Whole-genome shotgun sequencing of environmental DNA gained attention as a powerful method for revealing genomic sequences from various organisms in natural environments [3, 57]. An organism's DNA was not only sliced into small fragments but also mixed with other organisms' DNA fragments, thus creating a huge population of fragments, initial efforts were with long fragments ranging from 40Kb - 150Kb in fosmid or BAC libraries. Even though DNA fragments from diverse populations can be gathered together at the same time, they need to be assembled in order for us to make meaningful conclusions.

There have been several approaches to assembling microbial genomes from pure cultures or microorganisms cultivated in isolation. These approaches are designed for examining a single organism, even though they have been used for assembly of microbial metagenomes. There is a need for tools that are specific for community-level analysis. In this dissertation, I propose methods of sequence classification and assembly for a metagenomic population. Sequence classification is a process of grouping genome fragments into classes based on their similarities. The proposed method aims to use an approximate method based on fuzzy logic to classify genome fragments into groups and then perform assembly.

The dissertation is structured as follows: Chapter 2 presents basic definitions and background information on computational biology and DNA sequence assembly, including a survey of the related literature. Shortfalls of previous algorithms are addressed that lead to the new algorithm based on fuzzy logic. Chapter 3 presents the fuzzy solution for sequence assembly. Chapter 4 presents taxonomical classification methods for metagenome fragments, leading to new ideas for the assembly of metagenomic fragments in Chapter 5. Improvements achieved due to signatures and a new technique of fuzzy classification, in addition to results attained, are included in their respective sections of Chapters 3, and 4 and 5 Conclusions and a look into future directions are presented in Chapter 6.

Chapter 2

Background

Computational biology is an expanding interdisciplinary field that applies computational sciences to solve biological problems. Several biological problems that were tedious or impossible to solve manually have been tackled or are being tried using computational techniques. Some of these problems include DNA sequence alignment, DNA sequence assembly, gene finding, protein structure prediction and protein visualization. Computational biology is divided into subfields for each of these biological research areas. Computational genomics is the subfield of computational biology that tackles problems related to DNA or expressed sequence tags extracted from RNA genome sequences. These problems include DNA sequence assembly, sequence alignment, and motif and pattern discovery. In this dissertation the genome sequence assembly problem is addressed with a focus on metagenomic data.

This chapter covers the background for genome sequence assembly, which is necessary for understanding the computational problems in genome sequencing and leads to solutions for these problems. An overview of genome assembly and the problems with environmental genome assembly are presented. This chapter starts with definitions of some of the key terms in Section 2.1, followed by a literature review in Section 2.2 and metagenomics in Section 2.3. Section 2.4 discusses taxonomical classification and DNA signatures in the context of this research, and in Section 2.5 an overview of fuzzy logic is presented.

2.1 Basic Definitions

Several concepts and terms from genomic sciences that will be used in this dissertation will be informally defined in this section. There are several books on computational biology that provide detailed explanations of the terms listed below [2, 72].

Definition 1 *Sequence Assembly: Reconstructing of a DNA sequence by combining smaller sequences.*

Definition 2 *Base Pair: Two nucleotides on a paired double-helix-structured DNA strand. These two nucleotides are complements of each other.*

Definition 3 *DNA Fragment/Subsequence: A section of the genome sequence of nucleotides that forms a DNA strand.*

Definition 4 *Contig (Contiguous Sequence): A consensus sequence created by overlapping two or more sub-sequences or fragments.*

Definition 5 *Nucleotide Frequencies: The measure of occurrences of nucleotide pairs of a specified length.*

Definition 6 *Metagenome: Genome sequences containing an unspecified number of microbial organisms directly obtained from the natural habitat.*

Definition 7 *Codons: A set of three nucleotides that code for amino acids. There are a total of 64 codons.*

Definition 8 *Dicodons or Hexamers: A set of six nucleotides or a pair of codons. All possible combinations of the six nucleotides result in 4096 dicodons.*

Definition 9 *Markov Chain: A Markov model is random process whose future probabilities are determined by its most recent values. A Markov chain is a discrete markov process. A zero-order Markov chain uses is based on a set of random variables that follows the property that the next state is dependent on the present state and independent of future and past states.*

2.2 Genome Sequence Assembly

The problem of sequence assembly is acquiring data and assembling the DNA fragments or sequences into an entire genome sequence. Available chemical technologies for sequences produce short fragments of DNA sequences (40 Kbp -1000 Kbp) depending on the technology. There are two important aspects to understanding the problems that arise in genome assembly: the genome is cut into smaller portions, and fragments or sequences are cut at random positions. To obtain the original sequence these fragments need to be combined. The fragments are combined by determining overlaps between fragments. Thus, to combine fragments together by findding overlaps portions of fragments need to appear more than once. Using sequence once cannot create fragments that can overlap. Multiple copies of original sequences are made to ensure that the entire sequence is covered. This process is generally referred to as coverage of nX, where n is the number of copies and X is the sequence. Coverage of 8X or 10X is widely accepted and it has been shown it is sufficient to reconstruct the entire sequence. Thus for a genome sequence of length 4(million)MB, if the sequence fragments of length around 500 bp are generated we need 80,000 sequences.

Following the sequencing process, an assembler pieces together the many overlapping bases and reconstructs the original sequence [46]. The process explained above is known as the whole-genome shotgun method. The main steps involved in the assembly of sequences are illustrated in Figure 2.1. The first step, Sequencing, breaks the genomic DNA into fragments by sonication, a technique which uses high-frequency sound waves to make random cuts in DNA molecules [6]. In the assembly phase the sequences are combined to form contiguous sequences. Generally assembly leads to longer sequences of contigs that do not overlap. The final phase is finishing, in this phase contigs are joined by closing physical gaps. This phase is the most time consuming phase, which can be improved by using more than one clone libraries are prepared using different vectors. As different vectors clone sequences differently, using more than one vector can help improve coverage. Fragments that could not be cloned by one vector could be cloned by the other. Thus gaps could be reduced as overall coverage increases when sequences are generated using different vectors.

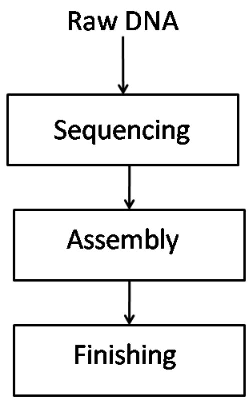


Figure 2.1: Whole Genome Sequencing Steps

Sequencing of an organism's DNA is a labor-intensive task, made possible by recent advances in automated DNA sequencing technology. On a large 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 [45]. There are several genomes that have been sequenced, some of the genome projects include rice, the plant *Arabidopsis thaliana*, the puffer fish, bacterial genomes such *Escherichia coli K12*, and *Acidiphilium cryptum JF-5*. Even though automated DNA sequencing technology made it possible to sequence genomes and computational techniques has made it possible to assemble genomes in reasonable time, several other problems exist. The sequence read from a machine is not always 100% correct; it may contain experimental errors. The process of acquisition of genome sequence data may lead to the insertion of certain discrepancies in the sequences, known as base-calling errors. The actual DNA sequence is read as a frequency signal, which is converted to represent the character sequence representing the four nucleotides as A,C, G and T. PHRED is a popular tool for reading signals and assigning quality scores can be found in [15]. In this scoring technique each nucleotide is given a score based on the strength of the base, a high score implying a higher probability of the base being correct. Low scores are assigned to bases that have less probabilty of being true. Sequences at the ends tend to have weak signals that make it difficult to identify them as A, C, G, or T. The base is assigned to the closest match and marked as a low quality base.

Base-calling errors create additional problems during assembly. These differences are categorized into three groups: insertions, deletions and replacements. Another well known problem with the sequences is that they contain repetitive sections. In other words, certain sections of nucleotides, called repeats, are repeated in the sequences. Repeats will be elaborated in later sections. All the parameters mentioned above make assembly an approximation problem.

The most popular approach to DNA fragment assembly has been to iteratively find the best overlap between all fragment pairs until an acceptable final layout is determined. If enough fragments are sequenced and their sampling is sufficiently random across the source, the process should determine the source by finding sequence overlaps among the bases of fragments that were sampled from overlapping stretches [33].

Computational complexity of problems is generally determined if they could be solved in polynomial time. Problems for which a polynomial time algorithm does not exist is known as NP (Non-deterministic polynomial time or NP). NP-Hard problems are a set of problems that could be NP or more complex than NP. 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 schemes have been explored to save computation time.

The earliest approach to finding solutions using the shotgun sequence approach was to find the shortest common superstring from a set of sequences. This approach becomes complicated with sequencing errors. Current approaches use pairwise sequence alignment as a method to align two sequences at a time. Instead of obtaining the shortest superstring, the longest common substring is used. To obtain the common substrings of two sequences, it is required to consider all possible substrings of the given sequences. The longest substring is then selected as the one with the longest overlap with the two sequences. This sequence is also known as the longest common sequence (LCS). Finding the LCS for all possible sequences becomes an NP-hard problem as the number of sequences grows. Thus, a brute-force approach is not feasible. Finding the LCS between fragments is the key to the process of sequence assembly. Dynamic programming solves problems by combining the solutions to subproblems to reduce the runtime of algorithms containing overlapping subproblems and optimal substructures [13]. Using dynamic programming, a polynomial-time solution for the LCS problem can be found. Therefore, dynamic-programming-based approaches are the most routinely used approaches in sequence assembly and alignment.

Other techniques for finding the longest common subsequence include suffix trees and greedy approaches. A suffix tree is a data structure that uses suffix information for fast processing of string problems. A suffix tree can be constructed in linear time using the Ukkonen algorithm [66]. Suffix trees allow a linear-time search for matching substrings. Even though suffix trees are a linear answer to sequence comparison problems, they are not good at storing and handling large data sets. Greedy algorithms are shown to be much faster than traditional dynamic programming in the presence of sequencing errors [78]. Greedy paradigms, applied in popular assemblers such as TIGR [59], Phrap [19] and CAP3 [22], are relatively easy to implement, but they are inherently local in nature and ignore long-range relationships between reads that could be useful in detecting and resolving repeats [46]. Greedy and hill-climbing approaches generally find a local optimal and thus the global solution could be missed. Additionally, these algorithms work with specific kinds of errors and cannot be generalized; they also become difficult to implement on larger data sets.

Unlike greedy approaches the overlap-layout-consensus mechanism considers all possible solutions before selecting the consensus overlap. An application of graph theory is found in [25] in which fragment reads are represented by nodes and an overlap between two fragments is represented by an edge. In such an implementation all possible contigs are considered. Paths are constructed through the graph such that each path forms a contig. The paths are then cleaned by resolving and removing any problems such as intersecting paths, and consensus sequences are constructed following the paths. One of the major problems of this approach to DNA sequence assembly is the extensive computation requirement. Fragment assembly performed with the overlap–layout–consensus approach becomes inefficient with an increasing number of fragments.

Even with the algorithmic improvements, additional reductions to the search space in fragment assembly problems are routinely employed. Techniques were presented to divide the entire genome space into groups to reduce the comparisons. For instance the simplest technique is to create two groups of data without any preprocessing. This comparison does not help assembly because it still needs to compare all the sequences if no prior knowledge is applied and the data are not related. 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 and artificial neural networks. A divide-and-conquer strategy for sequence assembly is described in [42]. A K-means clustering scheme was applied to fragments based on their average mutual information profiles, measuring the degree of closeness between fragments.

2.3 Environmental Genomics

Molecular biology has impacted microbiology by shifting the focus away from clonal isolates and toward the estimated 99% of microbial species that cannot currently be cultivated [9, 23, 50]. As an illustration, traditional culture and PCR-based techniques showed a bias of *Firmicutes* and *Bacteroides* as the most abundant microbial groups in the human gastrointestinal (GI) tract. Metagenomic sampling has revealed that *Actinobacteria* and *Archaea* are actually most prolific [31]. Microbial diversity studies often employs the retrieving 16S rRNA (ribosomal RNA) genes from mixed DNA fragments to identify which species are present in a sample. 16S rRNA is a subunit of an RNA gene, and its primary and secondary structure is conserved, which means that it can be used to identify organisms. Over the past 20 years microbiologists have used the polymerase chain reactions (PCR) techniques that allow making millions of copies of the gene from natural complex communities. The acquisition is done without the need for cultivation. This culture-independent approach to gene retrieval was first used more than two decades ago [41].

Metagenomic data can be ecosystem or organism associated: ecosystem associated metagenome contains DNA of microbes obtained from an environmental sample and an organism associated metagenome contains DNA from organisms. Examples of ecosystem associated metagenomes are soil, marine and mine metagenomes. Examples of organism associated metagenomes are mouse gut, human gut and fossil metagenomes. Another example is the Sargasso sea project [67]. Microbial samples collected through the filtering of sea water contained large amounts of novel genetic information, including 148 new bacterial phylotypes, 1.2 million new genes, and 782 new rhodopsin-like photoreceptors. Bacteriorhodopsin enables the capture of light energy without chlorophyll, and it had been previously unknown that this type of phototrophy was so abundant in marine waters. A similar metagenomic project giving new insight into naturally existing bacterial systems was the sampling biofilm from of an underground acid mine drainage [65]. Because this sample was from a system with low complexity, almost all DNA from present species were completely reconstructed, allowing the examination of strain differences and naturally forming lineages. It also enabled access to the full gene complement for at least two species, providing detailed information such as metabolic pathways and heavy metal resistance. Soil samples and the mouse GI tract are some other published metagenomic projects [52, 64].

Closely related organisms can contain remarkable genomic diversity, as was shown for some bacteria [70]. These variations, even though few, can result in different metabolic characteristics. Extracting these variations is one of the key ideas to further processing of the metagenomes. In this dissertation the genomic diversity between metagenome samples is extracted and used as a marker to separate data phylogenetically.

2.4 Phylogenetic Classification Using Signatures

The metagenomic approach of acquiring DNA fragments often lacks suitable phylogenetic marker genes, rendering the identification of clones that are likely to originate from the same genome difficult or impossible [63]. Separating the fragments in a metagenomic sample and reconstructing them is a complex process that involves the identification of certain features. These features can distinguish one genome from another in some circumstances. Organisms within a metagenome population can belong to different ranks in the taxonomy, for example they could belong to different domains or could be from the same species. For example, the acidmine drainage data consists samples that belong to archeal and bacterial domains. Acidmine drainage is an example of a community with low-complexity. A metagenome population can contain a large number of organisms that could either be very diverse, that is not closely related or it could be constrained to strains or species that are closely related. Thus complexity of a metagenome can also dictate the classification accuracy of DNA signatures.

Metagenome complexity can be measured with three different parameters taxonomic relation, evenness and richness. Visualization these is complex for example there are several ranks within taxonomy.

Figure 2.2 depicts the complexity of metagenomes with respect to taxonomy. The concept of the pyramid is that as you move down the pyramid the complexity of the metagenome increases. So if the metagemoes have organisms belonging to two different domains they would be at the top if the pyramid. If a metagenome is more complex and it had organisms that are same species or strains it will be at the bottom of the pyramid. The rectangular region will contain metagenomes and each level in the rectangle denotes the complexity. A depiction of a metagenome can be done on this pyramid as shown in Figure 2.3. Here the metagenome has organisms from two different domains, and within each domain there are different phylla, classes, families and species.

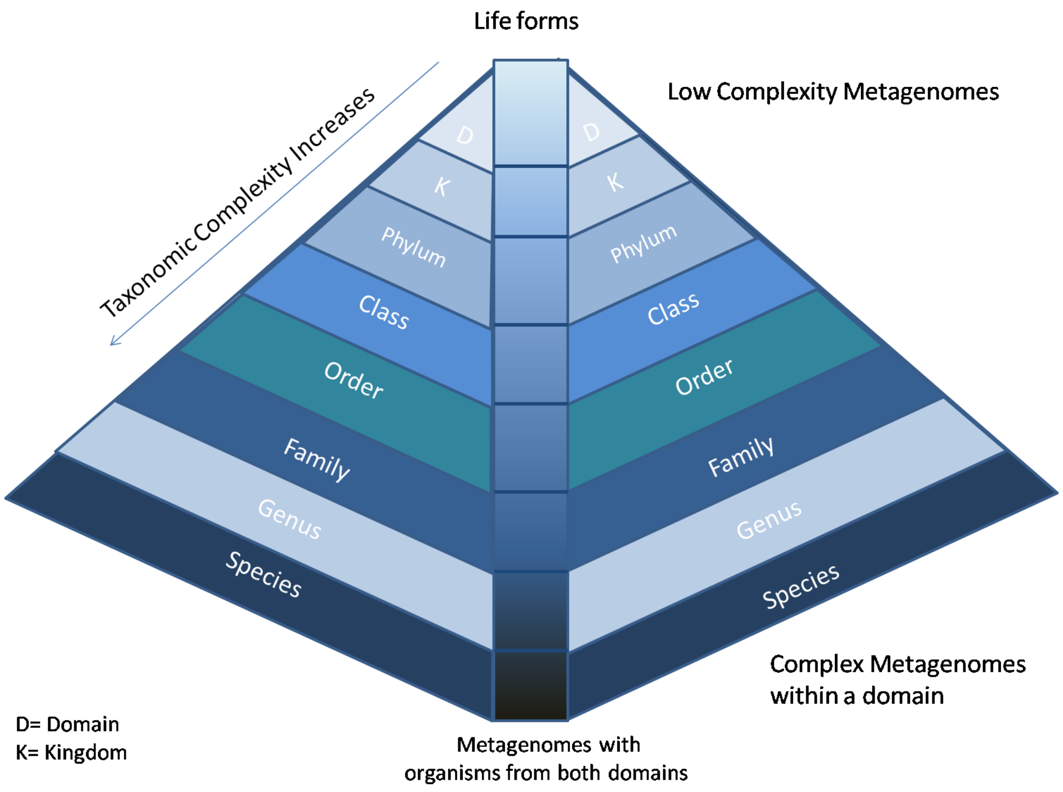

Life forms
Low Complexity Metagenomes
Taxonomic Complexity Increases
D
K
Phylum
Class
Order
Family
Genus
Species
D
K
Phylum
Class
Order
Family
Genus
Species
Complex Metagenomes within a domain
D= Domain
K= Kingdom
Metagenomes with organisms from both domains

Figure 2.2: Pyramid Representing Complexity of Metagenomes

This section describes the DNA signatures that can be employed in identification of bacterial and archeal strains(populations). A study on the performance of DNA signatures for classification of organisms at different ranks will be described in Chapter 4.

Biological sequences contain patterns that can lead to discoveries about the sequences. Oligonucleotide composition within a genome contains bias, making certain

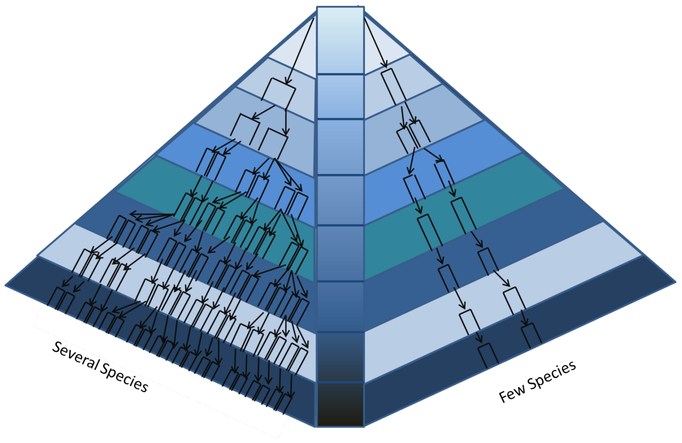


Figure 2.3: Example of the Complexity of a Metagenome

patterns appear several times within the genome. These oligonucleotide usage patterns can be specific to certain groups of organisms, thus certain oliginucleotides are over-represented and some could be under-represented [24]. The process of pattern discovery can be stated in three steps: (1) identify certain patterns, (2) find these patterns in an unknown sequence, and (3) group sequences based on these patterns. Sequence motif discovery algorithms can be generally categorized into 3 types: (1) string alignment algorithms, (2) exhaustive enumeration algorithms, and (3) heuristic methods [8]. DNA signatures are specific patterns that are observed within a DNA strand. These patterns can be observed in specific regions such as coding regions or can be observed throughout a genome including non-coding regions. Coding regions generally constitute more than 90% of a prokaryote(bacteria and archea) genomes. There have been several studies on the patterns found in DNA sequences. Patterns within a genome distinguish a coding region from a non-coding region. Coding regions are regions within a genome that code for a gene, the regions that do not code for genes are known as non-coding regions. Classifiers or neural networks are trained with a genome and used to identify coding regions of an unknown sequence from the same type. Two kinds of signatures are important to the discussion in this dissertation regarding metagenomes: GC content and oligonucleotide frequencies. A detailed discussion of the strengths and limitations of motif discovery algorithms can be found in [21].

2.4.1 GC Content

The four nucleotides of a DNA strand (A, C, G, and T) have hydrogen bonds between them. The nucleotide A bonds specifically with T and the nucleotide C bonds with G. AT pairs have two hydrogen bonds, as shown in Figure 2.4. GC bond pairs have three hydrogen bonds, as shown in Figure 2.5, making the bonds more thermostable. Thus, the GC content in an organism can sometimes be used to determine certain characteristics about that organism.

Organisms are generally biased in the distribution of A, C, G and T. This fundamental property of organisms is used in separating one genome from another. Certain organisms contain higher percentages of GC and are thus known as GC rich, while some other organisms are dominated by AT and are known as AT rich. Organisms belonging to the phylum *actinobacteria* are listed as GC rich [37]. On other hand, *Arabidopsis thaliana*, a very popular plant research organism has less than 40% CG content and thus has more AT content.

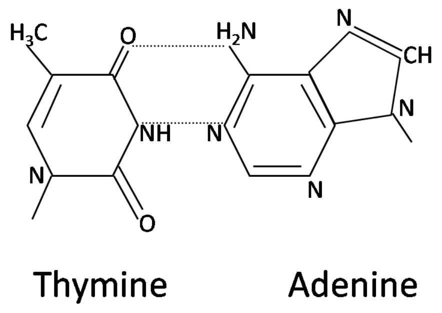


Figure 2.4: Hydrogen Bonds between A and T

2.4.2 Nucleotide Frequencies

Another signature that has been used frequently for analysis of genome sequences is

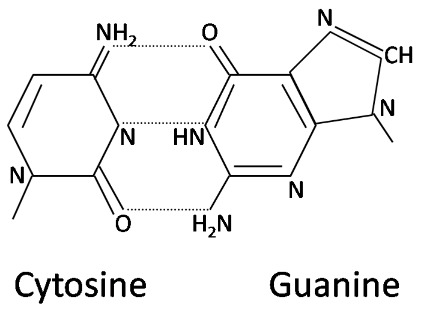


Figure 2.5: Hydrogen Bonds between G and C

the oligonucleotide frequencies, which is a measure of the occurrence of words of fixed sizes in the genomic sequence. Oligonucleotides are short sequences of nucleotides generally of length less than 20. An example of an oligonucleotide sequence of length 3 is 'ATC'. Nucleotide frequencies have been extensively used for grouping or for differentiation of organisms. Specific details about obtaining nucleotide frequencies will be covered in Chapter 4.

2.5 Fuzzy Logic

In 1965 Lotfi Zadeh published an article called "Fuzzy Sets" [76]. This article extended the concept of sets to a broader concept of continuous membership. The concept of fuzzy logic and approximate reasoning was introduced in 1975 [77]. Fuzzy logic formalizes an intuitive theory based on human approximation, which is by definition imprecise or vague. In human approximation, words such as *might*, *maybe*, and *could have* are commonly used. Fuzzy logic uses human linguistics to evaluate expressions, which are later assigned exact values. For example, temperature can be expressed as

$$\text{Temperature} = \ \{'Hot','Medium','Cold'\}$$

Fuzzy logic differs from traditional logic methods where exact results are expected. In traditional logic a variable takes a particular value from a set. In fuzzy logic the value of variable is not precise, as it can take one or more values with a certain degree of confidence. Fuzzy logic is used in problems where the results can be approximate rather than exact. Hence, the principles of fuzzy logic are well suited to real-world problems where approximate results and human reasoning fit the solution better than exact results or when exact results cannot be found. For example, consider the problem of assigning A, B and C grades to students in a class. These sets are defined as

$$91 \leq A \leq 100$$
$$81 \leq B \leq 90$$
$$C \leq 80$$

With a crisp grading system, a student with a grade of 79 is categorized as a C student; however a student with a grade of 79 clearly is closer to a B than to a C. Crisp sets fail to find the best possible solution in these kinds of situations. Fuzzy set theory allows classification of entities into categories by establishing degrees of weak or strong membership. A fuzzy set F is given by

$$\begin{aligned}
&F \ = \{\mu_F(x) \mid x \in X, \mu_F(x) \in [0,1]\} \\
&where: \\
&x = \ a\ given\ element \\
&\mu_F(x) \ = \ fuzzy\ set\ membership\ function \\
&X = \ the\ Universe\ of\ Discourse
\end{aligned} \tag{2.1}$$

The fuzzy set membership function, $\mu_F(x)$, returns a membership value between 0 and 1(inclusive) that signifies the degree of membership. Zero indicates a crisp non-acceptance to the fuzzy set, and 1 indicates a crisp acceptance. The values 0 and 1 in a fuzzy set are the equivalent of exact membership in a boolean set. All the other membership function values are real numbers between 0 and 1. A value of 0.9 indicates a high degree of membership, and a value of 0.12 indicates a low degree of membership. This function only states the degree with which certain object belongs to the set; it does not determine the object's acceptance or rejection from the set. A sliding acceptance standard, set later, determines final acceptance or rejection. This sliding standard is a value between 0 and 1 that acts as a threshold to accept a value as a likely or unlikely match. This process of getting back definite results is known as defuzzification. Information on different types of defuzzification functions can be found in [75].

Fuzzy logic has been used in several engineering applications. Fuzzy control has been used for complex mathematical models and non-linear systems. An extensive list of industrial applications of fuzzy control can be found in [58].

Fuzzy approaches to bioinformatics have been explored to some extent. Even though the application of fuzzy logic is not widely used, it has begun to gain popularity. An application to ontology similarity using fuzzy logic was presented in [74]. Fuzzy logic also been applied to classification problems in computational biology. A modified fuzzy K-means clustering was used to identify overlapping clusters of yeast genes [17]. A model for creating fuzzy set theory for nucleotides was proposed by Sadegh-Zadeh [54]. In this model a fuzzy polynucleotide space is made to measure the degree of difference and similarity between sequences of nucleic acids. The model proposes transforming a nucleotide chain into a unique point on a hypercube. The model can work for problems when an exact comparison of sequences is required or when sequences differ from each other by a certain hamming distance. For example, sequence alignment can be a good application of this model. To assemble sequences an overlap is required, and two sequences that are less similar to each other may form a better assembly than two similar sequences. This can be explained by the following example:

$$\begin{array}{ccccccccc} C & C & C & C & T & A & T & - & T \\ - & C & - & C & T & A & T & T & T \end{array}$$

$$\begin{array}{ccccccccccccc} - & - & - & - & C & C & C & C & T & A & T & A & T \\ A & T & G & T & C & C & C & C & T & - & - & - & - \end{array}$$

Sequence CCCCTATAT and ACTCTATTT differ by three base pairs. However, CCCCTATAT has a better overlap with ATGTCCCCCT even though they differ by 4 base pairs. Sequence assembly is performed by creating longer contiguous sequences by overlap of smaller sequences. In the example, sequence CCCCTATAT and ACTCTATTT are similar, but they do not create a contiguous sequence. Whereas, ACTCTATTT and ATGTCCCCCT can be used to generate ATGTCCCCCTATAT. Alignment of sequences has different specifications, and thus, alignment tools are not suitable for assembly purposes. Assembly of sequences is influenced by several factors besides the sequence chain. Encoding the genome into a single point or comparing hamming distances can overlook these factors. Therefore, there is a need for an approximation method that takes into consideration all the factors for assembling sequences. Specific fuzzy applications for sequence assembly and classification will be covered in Chapter 3 and Chapter 4.

Chapter 3

Fuzzy Genome Sequence Assembly

3.1 Introduction

DNA sequence assembly can be viewed as the process of finishing a puzzle, where the pieces of the puzzle are DNA subsequences. Although a puzzle has pieces that fit together well, the pieces of a DNA puzzle do not fit together precisely; the ends can be ragged and some pieces are missing, thus making it difficult, and sometimes nearly impossible, to complete the puzzle, patricularly when the complexity is increased in cases of multi-genome assembly. Hence, methods or rules to determine optimally which piece fits with another piece are required. The problem of sequence assembly is one of obtaining approximate matches through consensus. The consensus sequences are also known as contiguous sequences or contigs. A consensus sequence is constructed through approximate matches by following an overlap and consensus scheme [43] as illustrated in Figure 3.1.

Sequence assembly is an NP-hard problem, but with high-throughput automated capillary-based sequences it has become possible to sequence genomes at much faster rates than could be done manually. Automating the problem also leads to fewer human errors. Genome sequencing is generally done with thousands of fragments consisting of millions or billions of nucleotides. For example, the human genome is approximately 3 billion(Bb) long, assembling it would require approximately 40 million sequence fragments of length 700 bp each, a total of 30 billion base pairs are involved. An exhaustive method such as brute force requires several comparisons

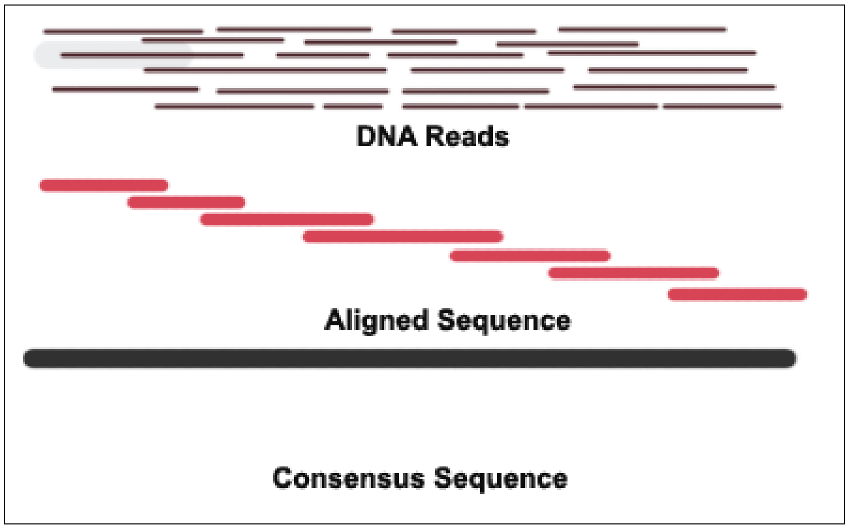


Figure 3.1: Whole Genome Sequencing Process of Creating Contigs from Fragment Reads

for every base at each position. Comparing every option to create an assembly has drawbacks and is not viable for huge data sets. Generally, heuristic approaches work better for such problems. Rules can be set to search only certain sequences or groups of sequences.

In this chapter, an approximate matching scheme based on fuzzy characteristic functions is presented. Several parameters are considered to create an optimal assembly. Then a divide-and-conquer strategy is presented to speed up the assembly by dividing the sequences into classes. This technique for dividing the data relies on the similarity of sequences with each other. Assembling sequences is accomplished by first grouping the sequences into clusters so that sequences in a cluster have high similarity with one another and sequences between two clusters are less similar.

Because this research focuses primarily on metagenomic (multiple microbial genomes) sequences, assembly tests are conducted on individual microbial genomes. To ensure that the assembler works on different types of data sets, few comparisons of assembly are conducted on different types of genomes. With recent advances in genomics, samples are collected from the environment. These samples are always composed of mixed species or mixed strains, though the complexity varies greatly. Therefore, a classification of data along with assembly is performed to so that organisms with high diversity can be separated.

The rest of this chapter is structured as follows: Section 3.2 discusses the previous work done in sequence assembly. Section 3.3 contains the approach. In Section 3.4, binning of sequences into groups is shown. Section 3.5 contains the results obtained from the assembly.

3.2 Previous Techniques

In general, the approach to sequence assembly has been to find iteratively 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 large, and the degree of computation required increases exponentially, many different approaches with varied parameters and matching schemes 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.

Dynamic programming has been extensively used to determine the LCS, using LCS for assembly reduces the NP-hard assembly problem to a time complexity of assembly $\Theta(n^2)$. The method is simple and useful in finding the LCS that may have mismatches or gaps. Thus, dynamic programming is well suited to assembly problems since not all subsequences found will be perfect and can be easily modified to find contiguous subsequences. The Smith-Waterman algorithm to find the LCS for multiple sequence alignment [61], which is one of the most prominent algorithms used in sequence assembly programs, is an application of dynamic programming. The Smith-Waterman algorithm maximizes the overall result, in this case the score, by comparing multi-lengthed segments, bypassing the need to look at the entire sequence. The maximum similarity is obtained using Equation 3.1.

$$H_{ij} = \; max\, \{H_{i-1,j-1} + S(a_i, b_j);\; H_{i-k,j} - W_k;\; H_{i,j-l} - W_l;\; 0\, \} \tag{3.1}$$

In Equation 3.1, S refers to similarity, H is a matrix that is set to measure the similarity, W is the weight assigned to a gap, and a and b are elements of the two given sequences. The Smith-Waterman algorithm gained popularity because it reduces the number of searches required; more details of the algorithm can be found in [61]. Several assemblers are based on the concept presented in the Smith-Waterman algorithm. Numerous software tools were built for sequence assembly such as PHRAP, AMASS, CAP3 and TIGR. PHRAP, an assembler based on a greedy algorithm, determines the best fragment matches by comparing only the highest quality pairs of reads [19], reducing search time and possibly increasing accuracy. This heuristic creates high quality results but limits the assembly to certain sets of the data. For example, if the data contains several low quality regions, then those poor quality reads or regions will not be assembled, and data is ignored, limiting the results to high quality reads only.

The AMASS algorithm limits searches to short, randomly selected sequences within fragments rather than comparing complete reads. This approach has shown a drastic reduction in assembly time [26]. Another approach to time reduction involves determining groups of fragments having more potential for alignment and comparing only these. For example, the assembler STROLL [10] 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 [22]. CAP3 has the capability to clip low-quality regions of reads at the ends. It uses base quality values in the computation of overlaps between reads, the construction of multiple sequence alignments of reads, and the generation of consensus sequences. The program also uses forward-reverse constraints to correct assembly errors and link contigs. CAP3 assembly produces stringent results, thus having fewer errors in the sequences. One of the earliest assembly schemes, SEQuencing AID (SEQAID) [43], examines ancillary fragment information to aid in the determination of fragment order. These approaches do not create a contig as a consensus of the parameters involved to determine best match, and hence some cases end up with resulting sequences that are very long but may not be correct or with chopped-off regions with low quality. Moreover, greedy approaches do not create globally optimal solutions.

These assemblers have crisp bounds and do not adapt to the data sets. For example, a data set can contain all, or a significant number of, low quality reads. If the assembler clips low quality regions, then most of the regions get clipped and are not used in assembly. However, if the assembler can adapt itself and allow a new threshold for low quality, this problem can be avoided. 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. Thus, approximate sequence assembly is a good candidate for fuzzy logic. In the next section a non-greedy approach is presented, based on approximate sequence matching using fuzzy logic.

3.3 Sequence Assembly

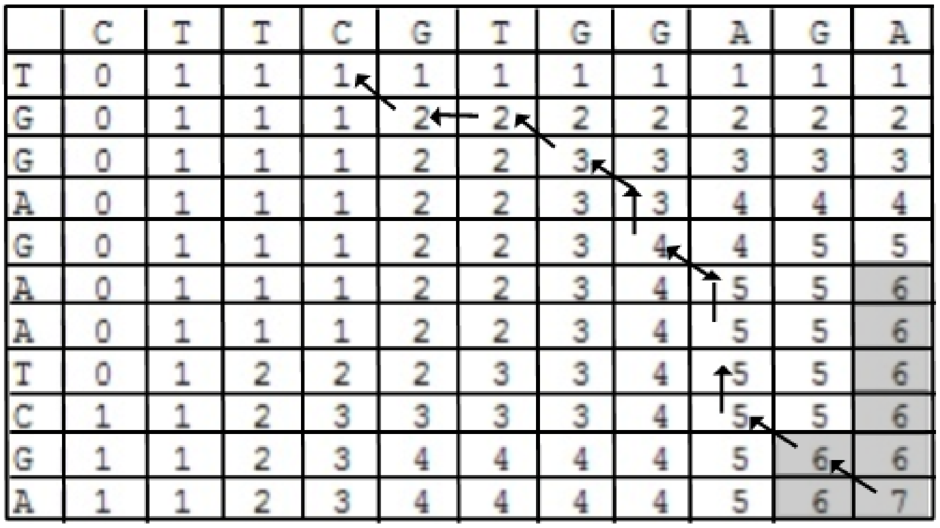
The sequence assembly problem is tackled using two different approaches the first module performs fuzzy sequence assembly, and the second module performs a fuzzy divide-and-conquer strategy for assembly. The divide-and-conquer strategy uses a fuzzy membership function to divide genome sequences into groups, reducing the number of comparisons and performing meaningful assembly. The fuzzy functions used in this section are a modified version of the fuzzy genome sequencing assembler proposed in [36].

3.3.1 Longest Common Subsequence with Fuzzy Logic

Sequence assembly requires creating contigs from DNA sequence reads. The longest subsequence (contig) with fewest insertions or deletions (indels) is ideal. Since an exhaustive search is not applicable for this problem, a time constraint is also placed on the solution.

As mentioned earlier, one of the problems with existing techniques for sequence assembly is that they have crisp bounds. The user has to specify the parameters for the program, for example in the program called Phrap, minimum score and minimum match are used. Minimum match and score are described in Section 3.3.3. The parameters need to be changed by the user to suit the data, and then the program is tested one or more times until an optimal solution is found, allowing the user to determine which parameters work best with the given data set. The longest sequence is not always the optimal solution. Some scientific applications prefer longer sequences while others may require higher quality or smaller gaps. The user has to balance the thresholds selected.

The approach developed here starts by acquiring the LCS of given sequence fragments using dynamic programming. Figure 3.2 shows the table constructed with dynamic programming. This method is simple and is useful in finding LCSs that may have mismatches in the sequence. This method is well suited to assembly problems since not all subsequences found will be perfect. It can also be easily modified to find contiguous subsequences.



	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

Two sequences 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 longest common subsequence for the indices of the cell. The highest number in the table indicates the longest subsequence that can be found between the two sequences.

Figure 3.2: Table Constructed Using Dynamic Programming to Find the LCS.

One of the common techniques used by assembly programs, such as PHRAP, is to search within a bandwidth along the best possible match. There is a higher probability of finding the LCS along the diagonal. As the path grows beyond the bandwidth, the gaps increase, and the subsequence shifts away from the optimal match. The diagonal arrows within each cell indicate a match: as mismatches increase the direction changes to either up or left. The optimal subsequence can be a perfect match, or the user may choose to tolerate indels. These criteria can depend on the user, the source of the data or the quality of the data. Almost all existing techniques provide user-defined thresholds for the number of indels allowed. For example, users can specify that they will allow gaps of size x, which implies that any overlap that has more than x gaps will be rejected. The user generally runs the program several times to obtain optimal results. In such cases it is better to determine empirically the ideal cut-off point or the threshold. 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 gaps of 31 and 32. Due to the fact that these techniques allow for crisp matches only, these potentially important sequences would be excluded. Alternatively, a gap of 30 and lower can be represented with a fuzzy confidence value of 1, which is for crisp results. Sequences with gaps that are very close to 30, like 31, can have a fuzzy value of 0.98. If the user chooses to allow all gaps greater than the value 0.8, these subsequences would be included. In this case, the user does not have to preprocess the data, change parameters and run the program several times.

There are several factors that determine if two subsequences 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. The sequences could be highly similar if they have a fewer number of inserts or deletes (indels) or less similar with more indels. The other parameters will be described later. A method is proposed in which multiple subsequences are selected and then, based on fuzzy parameters, select the optimal solution. For example, sequences that have any minimal length of overlap are selected and then the best consensus sequences are selected. The novelty of the proposed method is that it uses more parameters, confidence, gap penalty, and score of the sequence besides the length of overlap, and it is shown that these parameters can lead to a better assembled contiguous sequence. The sequence satisfying the aggregate overall requirement is selected. In other words, the fuzzy similarity of the given subsequences is measured. The proposed fuzzy matching technique can have crisp and non-crisp matches. The user can also obtain a fuzzy value that states how well the matching sequences fit the threshold. Fuzzy logic has been used to approximate string matching using distance measures; 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, leading to large amounts of data being rejected, or being treated as singleton reads. The main objective is to obtain the best consensus of the overall parameters.

Traversing in every possible direction in the LCS table gives all possible matches that occur between the two sequences. It is not ideal to try every possible subsequence combination. On the other hand, searching along the diagonal only may not give an overlap at all. Instead of selecting the LCS, all the subsequences that satisfy the minimum length required are selected as given in Equation 3.2. 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 it is not checked again. Track of cells that are traversed is maintained, and paths above the threshold are selected:

$$length >= threshold, \quad threshold = f(length(LCS)) \tag{3.2}$$

In Equation 3.2 length is the size of overlap, and threshold determines the minimum length required. The length of the overlap is obtained from the dynamic programming table. Threshold is determined from the LCS and can be a function of the length of the LCS. It needs to determined if either of these subsequences will create a match with each other. Any of the selected paths can generate an optimal solution. Therefore any possible good subsequences are not eliminated. The selection of the optimal subsequence is done using fuzzy similarity measures in constant time; therefore, the complexity of the algorithm is the same as the complexity of dynamic programming, which is $\Theta(mn)$ for any two subsequences of length m and n.

After the LCS is obtained the other factors that influence assembly need to be determined. At this point the different parameters that contribute to fuzzy assembly process are described. The following section lists the descriptions along with the characteristic functions for each of the parameters.

3.3.2 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 this approach for assembly.

Length of Overlap

The first similarity measure is the length of the match or length of overlap μ_{lo}, which is also the size of overlap when a contig is being created. This length includes indels and replacements. A higher overlap is better because it generates a contig with larger coverage; thus, this function aims at maximizing overlap. In equation 3.3, max—overlap(s1,s2)— represents an overlap where the shorter sequence is same as the overlap. The membership function for this measure is defined as:

$$\mu_{lo}(s1,s2) = \begin{cases} 1, & if|overlap(s1,s2)| = max|overlap(s1,s2)| \\ 0, & if|overlap(s1,s2)| = 0 \\ |overlap(s1,s2)|/max|overlap(s1,s2)|, & otherwise \end{cases} \tag{3.3}$$

Here, $|overlap(s1, s2)|$ is the length of overlap of sequences *s1* and *s2*. Given sequences *s1*, *s2* where no overlap occurs, the possibility of similarity does not exist.

Given sequences *s1*, *s2* where there is complete overlap or maximum overlap, the possibility is of maximum similarity, which is greater than partial overlap or no overlap. For all other cases the similarity is between 0 and 1. The logic of Equation 3.3 can be described in Equation 3.4 as follows:

$$\forall |overlap(s1, s2)| \uparrow \rightarrow \mu_{lo}() \uparrow \tag{3.4}$$

In the above equation the symbol $\uparrow$ represents increase in value. Equation 3.4 states that as overlap increases the fuzzy value of μ_{lo} increases until it reaches a crisp value of 1.

Confidence

The confidence μ_{qs} for each contig is defined as a measurement of the quality of the contributing base pairs [15]. The quality of a base pair indicates if the base was determined with a high confidence or low confidence. A strong signal indicates a correct read or less chance of an experimental error. Every base involved in the contig has a quality score, and the entire sequence can be a mix of low and high quality bases. The confidence of a contig is the aggregate quality score of its contributing bases. For simplicity, the sum of weighted average quality scores is the confidence of the contig. The weight can be calculated as shown in Equation 3.5. The bases with high quality are assigned a weight of 1. The bases that are of lower quality are given weights between 0 and 1, based on the cut-off value.

$$\mu_i = \begin{cases} 1, & if \quad q_i \geq \delta \\ 0, & if \quad q_i = 0 \\ (q_i - min_{qs})/(max_{qs} - min_{qs}), & otherwise \end{cases} \tag{3.5}$$

In Equation 3.5, δ is the threshold as explained earlier, generally specified by the user, and min_{qs} and max_{qs} are the minimum and maximum values for quality. The minimum and maximum values are obtained from the quality scoring algorithm. Equation 3.6 below describes the membership function:

$$w_{qs} = \frac{\sum_{i=0}^{n} w_i q_i}{n} \tag{3.6}$$

In Equation 3.6, 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, and q_i is the quality score of an individual base.

Gap Penalty

The LCS can be continuous or disruptive (or non-continuous). A continuous LCS is without any missing characters and is an exact substring of both the sequences under consideration. A disruptive LCS does not have be be continuous and is not generally a substring of both the sequences. In the case of genome assembly, exact matches rarely occur, and hence, an LCS with missing characters is used. Gaps refer to regions of a sequence that are missing when a sequence is compared to another sequence. These are divided into three categories: (1) Inserts are bases that are not present in the other sequence and thus create incontinuity in the LCS. (2) Deletes are the bases that are missing from the other sequence. Inserts and deletes can be used interchangeable for the two sequences under consideration. (3) Replacements are certain bases that have been replaced by other bases. In our context, gaps are generally of small size and cannot be greater than a specied word size. Gaps can be calculated as given in Equation 3.7:

$$Gaps = |insert| + |delete| + |replacement| \tag{3.7}$$

When a gap is encountered in the overlap, the LCS becomes smaller and less similar to the perfectly continuous overlap. Thus, as the LCS moves away from the ideal solution, a penalty is imposed. The gap penalty, denoted μ_{gp}, is the penalty that is imposed on gaps within an overlap region. A simple gap penalty can be calculated using the membership function in Equation 3.8 as follows:

$$\mu_{gp}(s1, s2) = \begin{cases} 1, & if \quad Gaps(s1, s2) = \Phi \\ 0, & if \quad Overlap(s1, s2) = \Phi \\ (Overlap(s1, s2) - Gaps)/(Overlap(s1, s2)), & otherwise \end{cases} \tag{3.8}$$

For the equation above, *Overlap (s1, s2)* is the amount of overlap of sequences *s1*, *s2* and Φ is an empty set. The logic behind Equation 3.8 can be explained by Equation 3.9 given below.

$$\forall\Delta(s1, s2) \uparrow \rightarrow \ \mu_{gp}() \rightarrow 0 \tag{3.9}$$

In Equation 3.9, the symbol ↑ represents increase in value, → implies approaches and Δ represents the difference, as $\Delta(s1, s2)$ increases, the value of μ_{gp} approaches zero.

Example of Equation 3.9: Given *Overlap (s1, s2)* = 10 and *Gaps* = 2, $\mu_{gp}(s1, s2)$ over [0,1] → 0.8 due to the fact that the overlap is higher than the gaps. The above definition of gap penalty is a simple method for calculating gaps. In this method, every kind of gap is given the same weight. In the above situation, 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 and thus can be counted as 1 instead of 10. Therefore, a weighed gap penalty is used instead of a simple penalty. An affine gap penalty is used which is given by

$$GapPen = GapOpening + Gaplength \times GapExtension \tag{3.10}$$

In the previous equation, *GapOpening* and *GapExtension* are scores for an opening or a continuation of a gap. The summation of Equation 3.10 gives the entire gap penalty *GapPen(s1, s2)*. The membership function for gap penalty is given as follows:

$$\mu_{gp}(s1, s2) = \begin{cases} 1, \quad if \quad GapPen(s1, s2) = 0 \\ 0, \quad if \quad Overlap(s1, s2) \leq GapPen(s1, s2) \\ 1 - (Overlap(s1, s2) - GapPen(s1, s2))/(Overlap(s1, s2)), \\ \qquad otherwise \end{cases} \tag{3.11}$$

For Equation 3.11, *Overlap(s1, s2)* is the length of overlap of *s1* and *s2*. The gap penalty is generally converted to a negative value or is subtracted from the overall fuzzy value.

Score

The score, denoted μ_{ws}, is calculated from the numbers of matching bases, indels and replacements. The score can be calculated by using different methods. For example:

$$score = fn(MatchingBP) - fn(Inserts) - fn(Deletes) - fn(Replacements) \quad (3.12)$$

In Equation 3.12, fn refers to the length. Given the following sequences:

A-CTCGCGAT-GCG

AGCTCG-GATTGAG

There are 10 base pair matches, two inserts(G and T in second sequence), one delete (C is not present in the second sequence), and one replacement (C replaced with A). The terms insert and delete can be used interchangeably. If all are given a value 1, the score is 6(10-2-1-1). 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.

A sample scoring matrix is given in Table 3.1 for each of the possible matches. If there is a match, a score of 3 is given. A replacement of A with G or C with T is attributed to evolutionary changes. Thus matches within Purine (A, G) or Pyrimidine (C, T) classes are given a score of 1. Finally, a score of -1 is given to matches between a Purine and Pyrimidine.

X	A	C	G	T
A	3	-1	1	-1
C	-1	3	-1	1
G	1	-1	3	-1
T	-1	1	-1	3

Table 3.1: Scoring Matrix

The fuzzy membership function for the score is defined as

$$\mu_{ws}(s1, s2) = \begin{cases} 1, & if tscore(s1, s2) = fmbp(s1, s2) \\ 0, & if fmbp(s1, s2) \leq 0 \\ fmbp(s1, s2)/tscore(s1, s2), & otherwise \end{cases} \tag{3.13}$$

where *fmbp (s1, s2)* is the score calculated using the matrix illustrated in Table 3.1 and *tscore (s1, s2)* is score of the overlap if there were no indels or replacements. The logic of Equation 3.13 is explained in Equation 3.14 below.

$$\forall \Delta tscore(s1, s2) = fmbp(s1, s2) \uparrow \rightarrow \mu_{ws}() \ \rightarrow \ 0 \tag{3.14}$$

In above equation, as the value of fmbp(s1, s2) increases, μ_{ws} approaches zero. **Example:** Given *tscore(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. These fuzzy membership functions can be also be found in [35] .

3.3.3 Fuzzy Thresholds

Minmatch

Minmatch is the minimum number of matching bases required. Minmatch is a term described in the Phrap assembler and it has been adapted into the proposed assembly. The calculating of minmatch however is not same as Phrap. It is not always possible to get a perfect overlap, and some amount of inexactness is tolerated. Therefore, a minimum match value for the overlap sequences is desired, which has a perfect match without any gaps. Generally, minmatch is used as a threshold to select or reject the contigs. A sigmoid membership function is used to select an optimal threshold for the minimum match required. The sigmoid function is given as

$$S(x, c) = \frac{1}{1 + e^{-(x-c)}} \tag{3.15}$$

In Equation 3.15, x is the minmatch value selected by the user, and c is the break point that determines a transition from membership to nonmembership.

Minscore

A score is calculated from the numbers of matching bases, indels and replacements as given previously in Equation 3.13. Minscore is a threshold which specifies the minimum allowable score of the overlap. Minimum score is a commonly used parameter that sets a limit on the minimum score value that must be satisfied to accept a sequence as a match.

3.3.4 Repeat Regions

Repeats(repetitive patterns) in a sequence are commonly observed in genomic data. These patterns can create misassemblies in the contigs. While assembling sequences, similar regions are sought. In some cases, these similar regions can originate from different sections of the genome. If these are assembled to form a contig, a misassembly is created. An example of repeats is illustrated in Figure 3.3. An assembler should incorporate techniques to address the problem of repeats, in order to ensure that overlapping sequences do not belong to different regions of the sequence but refer to the same section. The example illustrated in Figure 3.3 depicts contig A created by repeats. Contig B is the actual contig that is desired, where repeat I and repeat II are highly similar but still have slight variations.

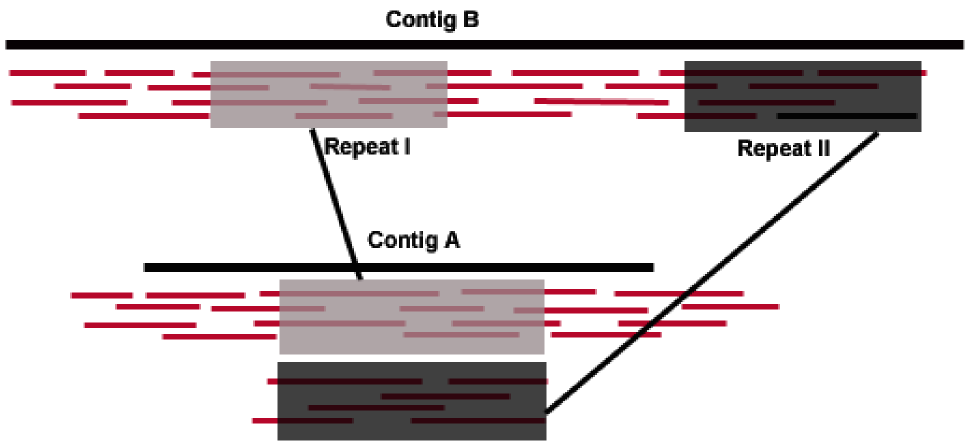


Figure 3.3: Repetitive Regions in Sequence that Can Create Misassemblies

There are techniques to resolve issues of repeats. For example, if a contig has a significantly large number of reads, then it could be an incorrect assembly. Because the sequences are duplicated x times, there should not be more than x occurrences of a segment. Depth is the number of subsequences that are contributing to the contig. Some assemblers count the depth of the match; if it is high for a contig, then there are higher chances that it was a repeat and the contig is discarded. Keeping an exact count for depth may not work since subsequences are cloned randomly and certain regions may be selected more times than others. An approximate value tends to work better in such cases. 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 smaller and these regions were not represented well. Even though repeats are highly similar, they are not exactly the same and contain gaps. Small differences can be detected in these repeats to separate them. In the absence of sequencing errors, a single nucleotide difference, with high confidence, between two copies of a repeat is enough to distinguish repeats [46]. Hence, once these regions are detected, dissimilarities between them can be detected.

In this approach contigs are created by pairwise comparison of sequences or of a sequence and a contig, and multiple results are obtain . For the problem shown in Figure 3.3 both contig A and contig B are obtained. Next step is to determine which one of these contigs is correct. Each contig is associated with a fuzzy similarity value. The fuzzy similarity value is an aggregate of the length of the contig, the confidence of the contigs, the gap penalty and the score. If the fuzzy similarity measure is taken for both contig A and contig B, the overall fuzzy value of contig B will be greater than that of contig A. The first reason is that the length of contig B has a higher value. Because the repeats have slight dissimilarities, the match gap penalty would be higher for contig A than for contig B. Finally, the confidence score of contig A would be lower if there is 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.

3.3.5 Aggregate Fuzzy Value

Once the fuzzy value for each of these parameters is calculated, it is plugged into an overall fuzzy function to determine the overall fuzzy value. To make a selection, this value needs to be defuzzified or converted to a crisp result. The aggregate fuzzy match value (AFV) acts as the defuzzification function. The center of average (COA) defuzzification function that uses weighted average values of the fuzzy members is employed. In a scenario of exact matching, perfect overlap can be defined as an overlap that satisfies the two thresholds, minmatch and minscore, is free of gaps, and satisfies the quality requirements. In a fuzzy system, this perfect match has a crisp value of 1. All matches that are closer to 1 than to 0 are known to be more similar. The fuzzy aggregate function of a contig is defined in Equation 3.16.

$$fa(c) = \mu_{qs}w_{qs} + \mu_{ws}w_{ws} + \mu_{gp}w_{gp} + \mu_{lo}w_{lo}, \tag{3.16}$$

$$where : w_{qs} + w_{ws} + w_{gp} + w_{lo} = 1$$

Each of the selected parameters has a weight associated with them. These weights w can be selected by the user to control the influence of an individual factor on the assembly. For example, to achieve a stringent assembly with the least gaps, w_{gp} can be set to a higher value (Note that μ_{gp} is a negative value if gaps are present and is zero for a contig with no gaps). To obtain longer contigs, w_{lo} can be set to a higher value. A weight can be assigned a zero value so that the factor does not influence the assembly.

$$afv = fa(c)/m \tag{3.17}$$

In Equation 3.17, m is the number of parameters, and *fa(c)* is given in Equation 3.16. Equations 3.16 and 3.17 give the overall fuzzy function and the aggregate fuzzy function for m parameters. The subsequences 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.

3.4 LCS Clustering

Genome sequence assembly is a rigorous task that performs comparisons of a genome with every other genome present in the population. As discussed in the background review, there have been techniques to divide the fragments into groups. These groups are intended to be small and to have high similarity between the fragments.

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 and artificial neural networks. A divide-and-conquer strategy for sequence assembly using average mutual information(AMI) was described in [42]. A K-means clustering scheme was applied to fragments based on their AMI measures. AMI profiles are used to measure the degree of "closeness" between fragments. In this research a classification based on the AFV of the LCS is performed. The idea of grouping based on the AFV derives from the fact that sequences that satisfy the overall requirement have higher similarity. These sequences have a higher chance of forming a consensus sequence. The process of assembly is named ClusFGS (Clustering with Fuzzy Genome Sequence), is described below and the pseudo-code in Figure 3.4

1. Select a few sequences as seeds (greater than the final number of contigs desired).
2. Initialize the selected sequences as the LCS.
3. Obtain a search sequence, and compare the AFV with the selected sequences.
4. Find the closest match to create a contig.
5. Update the LCS of the group.
6. Repeat for all sequences.

This technique improves the performance of assembly as shown in Figure 3.5. This simple selection technique suffers from a few drawbacks. If the initial sequences are selected randomly, they may not represent the entire population well. Instead of selecting arbitrary sequences, initial seeds can be set to sequences that have low similar with each other, allowing a wide range of contigs to be formed. Even though the method has drawbacks, there is still improvement in performance using a divide-and-conquer approach.

```
SelectRandomSeeds();
InitializeDefaultLCS();
for all sequences
{
  for all classes
  {
    CompareAFV()
    ObtainHighestFuzzyValue();
  }
  AssignFeaturetoClass();
  UpdateclassLCS();
  CreateContigsinClass();
}
UpdateContigs();
```

Figure 3.4: Algorithm for LCS Clustering (ClusFGS)

3.5 Experiment and Results

The primary objective of this work was to create an assembler using fuzzy logic. The secondary objective was to test the assembler on different microbial genomes and be able to differentiate genomes of these different species. As a preliminary step, differentiation of organisms at the highest level is performed. All living organisms can be categorized into three major groups based on the biochemistry and information processing abilities of the structure of their cells: bacteria, archea and eukaryotes. Bacteria and archea have the most primitive type of cell structure. Their nucleus does not have a membrane, and the structure is simple. An example of a prokaryote is *Escherichia coli (E. coli)*. Eukaryotes are organisms that have a membrane for the nucleus. Eukaryotic DNA is generally much longer in size and contains higher

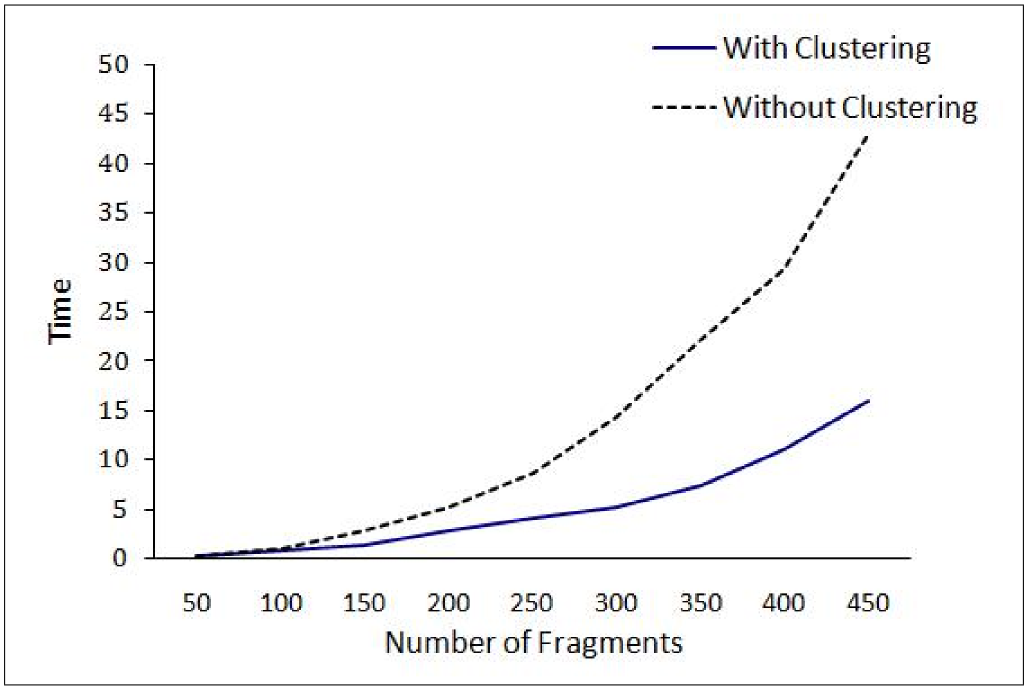


Figure 3.5: Comparison of LCS with and without Clustering

percentages of non-coding regions than bacteria and archea. Examples of eukaryotes include single celled algae, mammalian groups and plants. The fuzzy genome sequence(FGS) assembler is implemented with a modified version LCS along with the parameters listed above. The assembler was tested on artificially generated data sets and genome sequences obtained from GenBank. Artificially generated data sets were used to verify the algorithm and, thus, the assembly process. GenBank is a publicly available repository of nucleotide, protein, and EST databases. The experiments are divided into two major categories: assembly, and a classification of genomes. The results are compared with TIGR 2.0 [59].

The assembler was tested on artificially generated data sets and genome sequences obtained from GenBank belonging to different groups. Artificially generated data sets were used to verify the algorithm and, thus, the assembly process. GenBank is a publicly available repository of nucleotide, protein, and EST databases. The experiments for assembly are shown in Table 3.2. The results are compared with TIGR 2.0 [59]. Sequence fragments were generated from the genomes listed as follows: (1)

The Wolbachia endosymbiont of the Drosophila melanogaster strain wMel 16S ribosomal RNA gene containing 8,514 base pairs; (2) *Geobacillus thermodenitrificans NG80-2 plasmid pLW1071*, complete sequence, containing 57,693 base pairs; (3) *Yersinia pestis Pestoides F plasmid CD*, complete sequence, containing 71,507 base pairs; (4) *Arabidopsis thaliana genomic DNA, chromosome 3, BAC clone:F11I2*, GI:7209730, containing 36,034 base pairs; (5) *Ostreococcus tauri mitochondrion*, complete genome containing 44,237 base pairs; and, (6) *Phytophthora sojae mitochondrion*, complete genome containing 42,977 base pairs. The first three genomes belong to prokaryotes and the next three are eukarytoes. All these genomes can be obtained from GenBank [37]. The total base was covered 4 times, *4X* of the original sequence. Each subsequence is in the range of 300-900 base pairs. In Table 3.2, MGS is multiple genome sequencing using a simple LCS implementation, TIGR is the TIGR Assembler 2.0, FGS is the fuzzy genome sequencing described in this work, PHRAP is an assembler described in [19]. Since MGS did not perform well, it was not included in further experiments. Table 3.3 displays the performance of the FGS algorithm for the assemblies shown in Table 3.2.

	Percentage Genome Recovered			
Genome	MGS	PHRAP	TIGR	FGS
RPObc of Wolbachia genome	65%	93.94%	99.6%	99.6%
Yersinia pestis Pestoides		69.62%	93.9%	88.7%
Geobacillus thermodenitrificans		88.0%	77.1%	91.6%
Arabidopsis thaliana chromosome 3, BAC clone	56.8%	84.64%	88.8%	92.135%
Ostreococcus tauri mitochondrion		48.5%	77.7%	97.3%
Phytophthora sojae mitochondrion		86.02%	97.7%	97.2%

Table 3.2: Assembly Comparisons of Different Sequences

	FGS Assembly		
Genome	Time in Sec.	No. of Sequences	No. Contigs
RPObc of Wolbachia genome	146	100	15
Geobacillus thermodenitrificans	5800	650	195
Arabidopsis thaliana chromosome 3, BAC	466	200	61
Phytophthora sojae mitochondrion	1085	300	72

Table 3.3: Table shows time for assembly and number of contigs obtained for Assembly of Sequences using FGS, the experiments were conducted on AMD Turion 64 X2 dual-core processor, with 4GB of RAM.

The next experiment was to separate fragments from two organisms. Sequences from two organisms were taken and mixed with each other. The input data appears as if it is from a single organism. ClusFGS algorithm is used to classify the sequences into small classes followed by assembly of individual classes.

In Table 3.4, ClusFGS is the method described in Section 3.4 and is a modified version of FGS. Since TIGR does not perform a classification, comparison with TIGR is not included. Misclassification refers to the length of overall subsequences from genome 1 that were assembled incorrectly with contigs of genome 2. 61% of the genome could be recovered, the remaining 39% could not be recovered as the classification did not combine results from the classes into a final result. The results obtained in Table 3.2 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 3.2, FGS performed better than simple MGS. Assembly of *RPObc of Wolbachia* genome has the same results for both TIGR and FGS. *Yersinia pestis Pestoides* showed a slightly better recovery with TIGR. TIGR created larger contigs with less stringent parameters that can result in a better recovery. *Geobacillus thermodenitrificans* showed a much better assembly with FGS. Examining the results of Table 3.2, FGS performed better than 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 3.4 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 faster. The clustering and assembly for the sequences in Table 3.4 took approximately 3 hours to

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

Table 3.4: K-means Clustering and Assembly for Two Organisms with ClusFGS

run. The approach is based on full approximate matching of sequences which makes the algorithm slow. At this stage ClusFGS cannot recover a higher percentage of the genome because comparisons are done within a class. The performance is limited by factors such as selection of the seeds, no interaction between classes such as reassignment of sequences, and smaller classes not being merged. The assembler shows good classification results for the given data sets. The assembler is slower than current assembling techniques, this is due to the fact that approximate assembly is used. Other assemblers make use of high frequency fixed words. Using high frequency words to match certain portions of the genome can increase the performance of the assembly but may not obtain contigs that could be formed with low frequency words. This classification with some of it drawbacks is the inspiration for the new work that is presented in Chapter 4.

3.5.1 Paired End Assembly

Paired-ends are formed when libraries are constructed from DNA fragments using bacterial artificial clone (BAC) vectors or fosmids. The clone is generally a circular DNA, whose sequence is known and the unknown DNA is inserted on the clone. Clone inserts are sequenced from two ends of the circular DNA, thus paired sequence reads are obtained; the ends of the sequences inserted are called paired-ends. These pairs have a relationship to the other pair's orientation and distance. This relationship between the ends is used in assembly for the purpose of correcting assembly by examining the contigs. Examining the paired-ends is important to determine the accuracy of the assembled shotgun sequence contigs. When paired ends are known the distance between them can also be determined, as the insert size is known. A project can contain more than one library. Typical projects contain at least two insert libraries of sizes 2 to 3 kbp and 8 to 10 kbp, respectively, and may include BAC libraries of 100 to 150 kbp [46]. Using two libraries helps in obtaining maximum base pairs: base pairs not covered by one vector are covered by the other.

Assembly of sequences is generally enhanced by examining pairs of contigs. Paired end assembly is used order and orient the contigs. The end sequences are mapped to their location in the reference genome to determine their orientation against the main sequence. The library is checked to see if there were any clones whose two end sequences were located in different contigs. If this clone is be identified, then additional sequencing of its insert is done to close the sequence gap' between the two contigs [6].

Table 3.5 shows the assembly for RPObc of Wolbachia genome from the trace files using FGS assembler. The table indicates that the average contig length is small. This contig's length can be improved if a paired-end strategy is used. Also, it helps in removing incorrect contigs and orienting them. An incorrect contig can be determined by the distance between end-sequences. For example, if the distance between the ends is much larger than the average distance of the insert, then the contig is an incorrect assembly. Similarly gap length between sequences can also be estimated by the base pairs that are missing between contigs formed from pair-ends.

Genome	No. of Traces	No.of Contigs	Avg. Contig Length
RPObc of Wolbachia genome	118	13	1250
Influenza A virus (A/Melbourne/35(H1N1))	178	15	1129

Table 3.5: Assembly of Trace Sequences

The FGS could be enhanced by using pair-end assembly. A fuzzy based approach can be used to close gaps or determine if the contigs is not accurate. Fuzzy values can be used to determine the degree of accuracy of a contig based on the average distance of the insert, gap size of the contig. A gap that is greater than the average insert size will assigned a crisp value of 0 (incorrect assembly). If the contig obtains a contig size which is same as the insert fuzzy value of 1 given (best match). Gaps of other sizes can be assigned values between 0 and 1.

Chapter 4

Fuzzy Classifier to Taxonomically Group DNA Fragments within a Metagenome

4.1 Introduction

The metagenomic approach makes the acquisition of microbial genomic fragments easier; nevertheless, the approach suffers from limitations. Recall from Chapter 2 that the diverse genomes acquired together, separating these genomes can be beneficial to perform assembly. Classification of fragments into groups can yield information about the data present in the metagenome. For example, the data can be classified into distant clusters implying that the data sets belong to diverse and are not closely related. Taxonomical classification of genomic fragments is a vital problem in metagenomic approach. Nevertheless, closely related bacteria can contain remarkable genomic diversity [70]. These differences can be found by analysis of features of the DNA, which is referred as DNA signature.

Preassembly grouping of metagenomic fragments into classes can lead to faster and more robust assembly by reducing the search space required to find adjacent fragment pairs, because DNA from the same organism should be classified into the same taxonomic group. In this chapter a classification based on genomic DNA signatures is performed. The DNA signatures chosen are GC content, and tri- and tetra-nucleotide frequencies. The proposed method uses a fuzzy classifier and extracts signatures from given sequences and uses them as a feature set. The technique is verified with artificial shotgun sequences to measure correctness. The main purpose is to classify fragments of a community, which is depicted by classification of an acid mine drainage (AMD) environmental genome. Even though studies have successfully taxonomically differentiated full genomes or fragments of sizes greater than 1,000 base pairs [73, 30, 62], there is a lack of availability of applications that classify shorter (500-900 base pairs) shotgun fragments. The proposed approach is designed with a goal of classifying shotgun fragments. Earlier techniques have focused on using a single signature for classification [73]. A combination of different signatures is proposed for the classification.

The remainder of this chapter is laid out as follows: Section 4.2 presents background information on environmental genomics and DNA signatures. Section 4.3 gives an overview of K-means clustering algorithm, followed by the results in Section 4.4.

4.2 Background

Separation of domain-specific genomic fragments and reconstruction is a complex process that involves identification of certain features exhibited by entire taxonomic groups. These features are used to group the metagenomic sample into classes. The following section describes the DNA signatures that are employed in identification or classification of fragments.

4.2.1 DNA Signatures

Phylogenetically related groups of sequences show similar nucleotide frequencies either because of convergence or because they were inherited from a common ancestor [12]. For example, a study conducted on *Escherichia coli (E.coli)* revealed a non-random utilization of codon pairs [20] where some of the most frequent codon pairs found were: CTGGCG, CTGGCC, CTGGCA, CTGGAC, AACCCG, CTGGAA. This study and others have revealed that there is a non-random over-representation and under-representation of certain codon pairs within a species. Oligonucleotide frequency studies with short x-x bases have reported tendencies of under- and over-representation in Xmers [7]. This study brought to attention that certain oligonucleotides are rarely observed in certain species while certain other oligonucleotides have shown their dominance in a particular species. This also shows that nucleotide composition contains bias. There is also codon bias which shows a non-random utilization of codon-usage. Different organisms show different codons that encode for the same given amino acid, indicating that certain codons are preferred by certain organisms. A study on the codon bias of *Drosophila* revealed that *Drosophila* genes are as biased or more biased than those in microorganisms [48].

The key to the classification of individual genomes from a metagenomic data set is the presence of patterns in the sequences. Recall from Chapter 2, Section 2.4, that these patterns can be specific to certain organism groups. Thus, identification of these patterns provide a means of genome discrimination. Moreover, we propose using the patterns as signatures to distinguish organisms that are diverse from another. Now the discussion moves to the two groups of signatures that were tested herein.

The first signature is based on GC content present in the genome. GC content is found to be variable with different organisms; this variation is viewed to be the result of variation in selection or bias in mutation [5]. For example, coding regions within a genome code for genes and are less divergent within populations. Genes represent characteristics of an organism: the physical development and phenotype of organisms can be thought of as a product of genes interacting with each other and with the environment [39]. Studies have shown that the length of the coding sequence is directly proportional to higher GC content [40], thus showing a strong correlation between GC content and gene properties. The pre-assembly binning of a well-known metagenomic data set from acid mine drainage was performed by their GC content [65].

The second signature that was investigated were the oligonucleotide frequencies. Nucleotide frequencies are generally taken from a group of two, three, four, five, or six nucleotides. These are known as di-, tri-, tetra-, penta- and dicodon nucleotide frequencies, respectively. These prefixes indicate the presence or absence of certain words in a genome that have been used to separate certain species. Evaluation of frequencies of fragments and their correlations based on taxonomy was performed by Teeling, *et al.* [62]. In this paper it was shown that GC content is not sufficient for separating species and tetra-nucleotide frequencies showed better differentiation of species. This paper used fragments of size 40,000 base pairs for analysis. TETRA, a web-based application uses the above technique of using tetra-nucleotide frequencies for genome sequence differentiation [63] by generating correlation values of tetra-nucleotide frequencies. A grouping based on nucleotide frequencies resembles the phylogenetic grouping of the representative organisms [49]. In another approach, differentiation of bacterial genomes was performed using statistical approaches for structural analysis of nucleotide sequences [51]. Metagene, another example of a nucleotide analysis program, utilizes dicodon frequencies to find genes within a metagenome population, using frequencies along with other measures to alleviate the need for a training set [38]. As species present in metagenomic samples are usually unknown, a priori training data is hard to get and may add incorrect bias. Therefore, methods that do not require training are preferred.

Frequencies of larger word sizes such as tetra, penta and dicodon can be more reliable. Obtaining frequencies of a larger group is dependent on the size of genome fragment. For instance, there are a total of 4,096 dicodons. A sequence of length 10,000 base pairs contains 1,665 non-overlapping dicodons. Because this number is less than 4,096, the sequence cannot cover all the 4,096 dicodons. Most of the earlier approaches did not use larger words because there is not enough information to perform reliable statistical analysis. The same sample contains 3,332 tri-nucleotide frequencies, that can easily cover the 64 tri-nucleotides. Therefore, it is better to use tri-nucleotide frequencies in this case than dicodons.

4.2.2 Clustering

Clustering of fragments may be viewed as a more structured form of fragment thinning before assembly 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 or artificial neural networks.

K-means is an unsupervised learning algorithm to group objects into categories. It has been widely used in pattern recognition problems. The simplest K-means algorithm places N objects into K classes by using the minimum distance from the center of the class to each object. In the simple K-means approach, K is fixed a priori. Clustering problems generally derive some kind of similarity between groups of objects. K-means clustering is a simple and fast approach to achieve a grouping for data. A well-known approach to fuzzy classification is the fuzzy C-means algorithm [4]. An improvement of K-means using the fuzzy logic theory was presented [29], in which the concept of fuzziness was used to improve the original K-means algorithm.

A K-means algorithm starts with a large number of seeds (initial samples) for the potential clusters. It uses a set of unlabeled feature vectors and classifies them into K classes, where K is given by the user. From the set of feature vectors K of them are randomly selected as initial seeds. The seeds selected are uniformly distributed random numbers. Remaining samples are then assigned to a cluster based on their distance from the seed. The feature vectors are assigned to the closest seeds depending on their distance from it. 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.

Due to its simple method of using feature vectors as seeds and the arithmetic mean as the center for the clusters, the K-means algorithm suffers from drawbacks, such as convergence and mean is not always the best center for a cluster. An improvement to this approach was to start with a huge random population of seeds [69]. This method has been shown to find better seeds, since the initial seeds are more than K and are distributed in the data set. Even though this was an improvement on the simple K-means, it was limited in its ability to find better centers, since the mean does not always represent the center of a given data. A modified K-means was developed that uses a weighted fuzzy average instead of the mean to get new cluster centers. Using a fuzzy weighted average instead of a simple mean improved K-means and also leads to convergence [29]. In this research, a modification of the fuzzy K-means algorithm with fuzzy weighted averages is used for fragment clustering. This algorithm uses Expectation Maximum to enhance the clustering. Expectation Maximization is a statistical technique for maximum likelihood estimation using mixture models. It searches for a local maxima and generally converges very well. In this method these two algorithms are combined to generate optimum clusters which do not require a huge value of K and each cluster attains a more natural shape and guarantee convergence. The algorithm is described briefly in the next section and in detail in [34].

4.3 An Overview of Our Algorithm

Fragment classification divides entire data sets into smaller categories. The classes should represent two significant properties: (1) they contain fragments belonging to the same group or species present in the metagenomic data set, and (2) they have continuity and can represent adjacent regions of the genome. The first step to classification is the identification and extraction of the signatures. After the signatures are extracted the feature vector is initialized, and the K-means algorithm is implemented to create classes.

The operations carried out will be described in the following subsections.

4.3.1 GC Content

GC content is expressed as the percentage of C and G present in the fragment and is calculated as follows:

$$\frac{C+G}{A+C+G+T} \times 100 \tag{4.1}$$

In Equation 4.1, A, C, G and T refer to the frequencies of the occurences of the four base pairs. To analyze the GC content's power to separate fragments, the GC content of different organisms is observed. It starts by comparing the GC content of two genomes from the AMD environmental sample, as illustrated in Figure 4.1. The histogram of GC content is taken over a window size of 700 nucleotides (nt). The histograms measures the GC content of fragments and fragments are binned according to the GC content. Ranges of GC content can also be used instead of exact values. The bins are taken for the range of values in between 25-75%. GC content can never be 0 or 100%, thus lower and higher percentages are ignored in the analysis. The two classes shown belong to Archea and Bacteria domains and are phylogenetically distant. The histogram also indicates that there is a small region of overlap between these data sets, but most of the fragments have non-overlapping GC content.

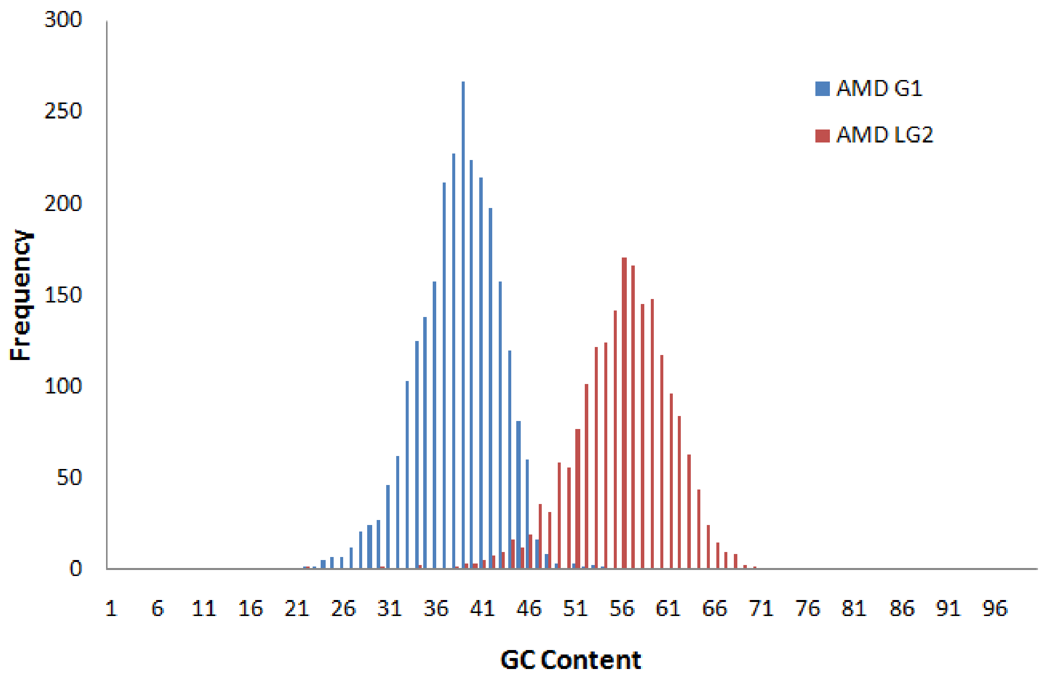


Figure 4.1: Histogram of Acid Mine Drainage Classes: AMD LG2 refers to *Leptospirillum sp. Group II* Environmental Sequence of length:960,150. AMD G1 is *Ferroplasma sp. Type II* Environmental Sequence of length: 1,317,076.

The next analysis is done on sequences that are close phylogenetically. Figure 4.2 shows the GC content of two *E.coli* sequences obtained from NCBI [37]. The first sample is *Escherichia coli O157:H7 EDL933*, RefSeq: NC_007414. This complete sequence contains 5,528,445 base pairs. The second sequence is *Escherichia coli HS*, RefSeq: NC_009800, containing 4,643,537 base pairs.

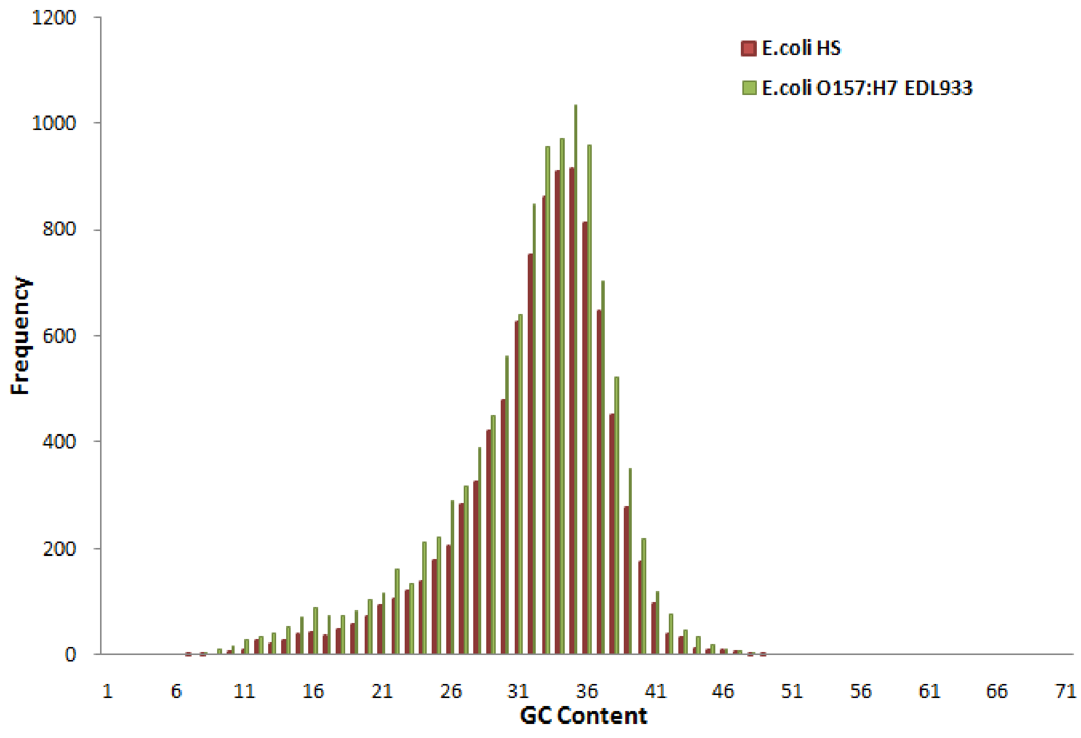


Figure 4.2: Histogram of E.coli Strains: *Escherichia coli O157:H7 EDL933*, RefSeq: NC_007414. and *Escherichia coli HS*, RefSeq: NC_009800

In both Figure 4.1 and Figure 4.2 the histograms of the genomes overlap one another. The histogram for two *E.coli* strains has a larger overlapping region. Both these strains belong to a bacteria group, and thus their GC content is similar, indicating that sequences belonging to similar groups tend to have similar GC content values.

GC content can sometimes prove to be a good parameter to separate fragments. However, certain factors need to be considered when using GC content. GC content is known to be more influential in coding regions. Shotgun fragments of metagenomic data do not contain information that reveals directly whether a certain fragment contains coding regions or the percentage of fragment region that can code for a gene. Analysis of GC content reveals that it is not sufficient to obtain a classification when closely related organisms are present in the data set. Thus, advanced signatures are required to obtain a good separation of groups within a metagenome.

4.3.2 Nucleotide Frequencies Using Markov Chain Model

Markovian models have been used in fields such as statistics, physics and queuing theory. Hidden Markov models are used in pattern recognition to represent unknown probabilities. Markov chain predictors have also been used to predict coding regions, thus finding genes. The simplest chain is the zero-order Markov chain which can be estimated from the frequencies of the individual nucleotides A, C, G, and T. The approach used to estimate the zero-order Markov chain is shown below. Consider the sequence GGATCCC, the nucleotide frequency is given in Equation 4.2, where p(X) gives the frequency of X:

$$p(GGATCC) = p(G)p(G)p(A)p(T)p(C)p(C) \tag{4.2}$$

Higher order oligonucleotides can be determined using the zero-order Markov method. It was shown that a zero-order Markov method yields greater inter-species distinction than higher order chains [49]. A zero-order Markov method removes biases only from mono-nucleotide frequencies, thus including other oligonucleotide frequencies. Given the di-nucleotide frequencies, the tri-nucleotide frequencies can be estimated by the product of the constituting overlapping di-nucleotide frequencies being divided by overlapping mono-nucleotide frequencies.

$$p(GGATCC) = \frac{p(GG)p(GA)p(AT)p(TC)p(CC)}{p(G)p(A)p(T)p(C)} \tag{4.3}$$

Higher order Markov chains can also be constructed using only the previous state frequencies. A maximal-order Markov chain removes biases from all the previous states except the immidiately preceeding state only. In this approach tri- and tetra-nucleotide frequencies are used. Expected values are directly calculated from the observed values using a maximal-order Markov chain as shown in Equation 4.4. In Equation 4.4 and Equation 4.5, O refers to the observed values, E is the expected value, and N_i refers to a nucleotide base pair. The observed values are the actual frequencies of the particular oligonucleotide and expected values are the frequencies that are computed based on the markov chain.

$$E(N_1N_2N_3) = \frac{O(N_1N_2)O(N_2N_3)}{O(N_2)} \tag{4.4}$$

$$E(N_1N_2N_3N_4) = \frac{O(N_1N_2N_3)O(N_2N_3N_4)}{O(N_2N_3)} \tag{4.5}$$

Nucleotide frequencies can be calculated as overlapping or non-overlapping. Non-overlapping words are not robust (or sensitive) to gaps or sequence errors within genomes. For instance, an insertion or deletion in a sequence can shift all the words following the insertion or deletion by 1. This also introduces change in the value of the all subsequent frequencies. Calculating frequencies with overlapping words is more robust to insertions or replacements and affects a single frequency value, while preserving the rest of the frequencies. Any bias introduced by overlapping words is consistent throughout the genome and does not affect the final values. Overlapping word frequencies are robust to insertions and deletions and also give more frequencies that are useful to perform any statistically relevant analysis. Therefore, overlapping word frequencies are used in this approach. Note that these frequencies are used only for classification purposes and not to identify over- or under-represented words.

4.3.3 Fuzzy K-means Clustering

Clustering for a metagenomic assembly problem has a two-fold purpose: to divide the data for performance improvement, such that assembly is performed for smaller groups, and to group genome fragments into classes such that each class has genome fragments from closely related organisms. The K-means algorithm uses a set of unlabeled feature vectors and classifies them into K classes. From the set of feature vectors K of them are randomly selected as initial seeds. The feature vectors, in this case DNA signatures, are assigned to the closest seed. The mean of features belonging to a class is taken as the new center.

Given N sequences, such that a sequence $S = \{C\}^i$,where C =$\{A, C, G, T\}$. K sequences are randomly selected as the initial seeds, where K is less than the number of sequences N. The nucleotide frequencies and GC content for all sequences are calculated. These frequencies form the p features to be used in classification.

The sequence is assigned to the class that has the highest fuzzy similarity. The fuzzy similarity is calculated using a weighted fuzzy average (WFA). Let $\{x_1, ..., x_P\}$ be a set of P real numbers. The weighted fuzzy average using the weight w_p for x_p is given as:

$$\mu^r = \sum_{p=1}^{P} w_p^{(r)} x_p, \ r \ = 0, 1, 2, \ldots \tag{4.6}$$

Here x is the parameter or feature and p the number of features. The number of the iteration is given as r. The mean is obtained for all the K initial classes. The next step is to assign feature vectors to each of the classes. A feature vector is assigned to the closest class by computing the distance of a feature vector from each of the classes. Given i=0,. . . ,N and j=0,. . . ,k, , where N is the number of sequences and k the number of initial classes, the distance $d_{i,j}$ of a sequence from each cluster can be calculated as follows:

$$d_{i,j} = min(\mu_j^r), for\ all\ j = 0, \ldots, k \tag{4.7}$$

Thus feature vectors are assigned to a class. Since a large number of classes was created initially, empty or small classes are eliminated. Classes that are close to each other (distance between their centroids is less than a threshold) are merged to form one class. This process is repeated until convergence by replacing the initial mean with the WFA, and feature vectors are reassigned by computing the distance.

The algorithm for the k-means thats extracts signatures and performs classification is depicted in Figure 4.3.

In the next section results obtained by classification are shown along with description of the genomes used to test the approach.

4.4 Clustering *via* Feature Extraction

4.4.1 Artificial Metagenome

To assess the performance of fuzzy clustering on genomic sequences, two experiments on artificial data were performed In the first experiment, two genomes from different domains of life are used for the first test case. These fragments were mixed with each other. The classifier first extracted the GC content and calculated the nucleotide frequencies and classifies the fragments into categories. The classifier is run until compact classes are generated. These K classes are greater than or equal to the actual number of genome classes used. Fragments of average size 750 base pairs are used in this test case. Table 4.1 shows the results obtained after classifying these two samples. The sequences are generated from *Renibacterium salmoninarum ATCC 33209*(Rs ATCC 33209) and *Aquifex aeolicus VF5*(Ae VF5).

In the second experiment, a data set that was created using *Adoxophyes orana granulovirus* and *Aeromonas salmonicida bacteriophage 25*. For this experiment not only classification but also assembly of the sequence using signature-based classification is conducted. Artificial shotgun sequences of average length 700 bps were generated from this mixed set. Recall results from Table 3.4, that classified genome fragments using an LCS-based approach. The results of the classification and assembly are shown in Table 4.2. These results indicate improvement in assembly using the signature-based method, even though there are few misclassifications. The reason for the misclassifications is that ClusFGS classifies based on LCS, which considers the entire sequence for classification (each base pair is compared), whereas signature-based classification uses signatures without creating an overlap. This also makes the

```
initializeValues();

getTriFrequencies();

getTetraFrequencies();

getGCContent();

standardize_feature_vectors();

//Draw a large number K of uniformly random seed vectors in the feature space
find_seed_vectors();

// Eliminate any seed vectors that are too close to other seed vectors and
// reduce K accordingly
reduce_seed_vectors();

// Assign each of the customers feature vectors to the nearest seed vector
assign_feature_vectors();

// Eliminate all seed vectors that are centers of empty clusters or have
// fewer than P vectors and reduce the seed vector number K
eliminate_seed_vectors();

// Compute the mean and variance for each cluster
compute_mean_and_variance();

//repeat until convergence
while(classesChange())
{
  //eliminate empty clusters
    eliminate_empty_clusters();

// Compute the weighted fuzzy average of each class as the new
  // class prototype with curent K
  compute_weighted_fuzzy_average();

    // Assign each of the feature vectors to the class with the nearest WFA
    assign_feature_vectors_WFA();

    // merge close clusters
   merge_nearest_clusters();
 }
```

Figure 4.3: Sequence Classification Algorithm

	# Fragments classified incorrectly			
Signature	Ae VF5	Rs ATCC 33209	Total %	Time in Seconds
GC	450	22	0.157	14
T_z	NA	NA	NA	22
TR_z	NA	NA	NA	68
$GC - T_z - TR_z$	14	27	0.013	106
T_m	22	9	0.01	39
TR_m	57	48	0.035	111
T_m- TR_m	12	0	0.004	146
$GC - T_m$	1	118	0.039	40
$GC - T_m - TR_m$	5	1	0.012	139

GC refers to clustering with GC content, T_z refers to tri-nucleotide and TR_z refers to tetra-nucleotide frequencies using zero-order Markov chains. TR_m refers to tetra-nucleotide frequencies using a maximal-order Markov chain, T_m indicates the tri-nucleotide frequencies using a maximal-order Markov chain. Combinations of different signatures are shown by hyphenating individual frequencies. A value of NA indicates that the signatures could not separate the fragments into groups and all the data was placed into one class. Time indicates the total time taken for the assembly in seconds.

Table 4.1: Separating 500 Fragments Belonging to Two Organisms Using Different Signatures

approach much faster than ClusFGS. The results indicate the classification accuracy and percentage of genomes that was recovered in assembly. This approach shows that classifying sequences based on DNA signatures is as good as classification based LCS that was proposed in Chapter 3. Signature based approach is much faster than LCS based approach and as the clustering algorithm was improved there is improvement in assembly.

Genome	Length	% Recovered	Miss-Classifications
Adoxophyes orana granulovirus	99,657	98.72%	0.12%
Aeromonas salmonicida bacteriophage 25	161,475	98.81%	0%

The two data sets are mixed with each other to create an artificial metagenome. The classifier is executed which groups them into classes based on the proposed algorithm and performs assembly. The percentage of genome recovered is measured from the contigs

Table 4.2: Clustering and Assembly of 3,500 Artificial Metagenome fragments

4.4.2 The Acid Mine Drainage(AMD) Metagenome

The AMD metagenome was obtained from the Richmond Mine at Iron Mountain, CA [65]. The acid mine drainage environmental genome was shown to contain two groups with five dominant microorganisms. Artificially created shotgun sequences of two genomes of AMD is used, namely the *Leptospirillum sp. Group II* (AMD LG2) environmental sequence and the*Ferroplasma sp. Type II*(AMD G1) environmental sequence. These sets are 960,150 and 1,317,076 nucleotide base pairs respectively. The first group belongs to the bacterial genus *Leptospirillum*; the second one is an archea from the genus *Ferroplasma.* These genomes are almost complete and are available at NCBI. These two sets represent the genomes that were completely assembled. Shotgun sequences of average size 700 base pairs were created from these genomes. These shotgun fragments are randomly combined with each other to create an environmental genome sample of an AMD metagenome. Figure 4.4 depicts the classification results on AMD data. A set of 3,000 samples was used for the display. Figure 4.4 indicates that the combination of GC content and trinucleotide frequencies was able to separate the two classes (Note: only one signature is shown in the picture, all 64 signatures were used for the classification). There are a few cases where fragments were classified incorrectly. Classification using GC content and three different nucleotide frequencies is shown in Figures 4.4, 4.5 and 4.6. The nucleotide frequencies are chosen based on their over or under-representation in the given sets respectively. The codon GAA was had a similar representation in both sets, whereas GCC was found to be represented well in the *Ferroplasma sp. Type II.* and was under-represented in the *Leptospirillum sp. Group II.* ATT was under-represented in *Ferroplasma sp. Type II.* Classification of actual traces obtained from the AMD sample show two dominant groups as shown in Figure 4.7, the clustering was performed on 500 traces. The result obtained may indicate the Bacteria and Archea domains that are found the AMD set.

Classification of 20,000 fragments with average length 700 base pairs from the AMD metagenome was performed using different combinations of signatures. Results of the classification for the proposed approach are given in Table 4.3. The final classification resulted in two groups, one with fragments from *Leptospirillum sp. Group II* and another with *Ferroplasma sp. Type II* fragments respectively. The results indicate that frequencies obtained using maximal-order Markov chain allowed a better classification than zero-order Markov chain. A combination of different signatures also

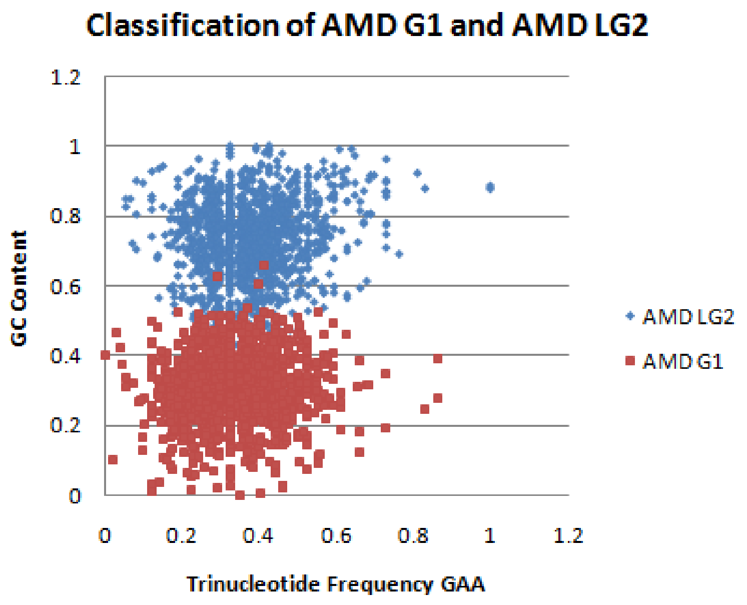


Figure 4.4: Classification using GC Content and Trinucleotide Frequency GAA for 3,000 Artificially Created Shotgun Sequence Fragments obtained from AMD G1 and AMD LG2

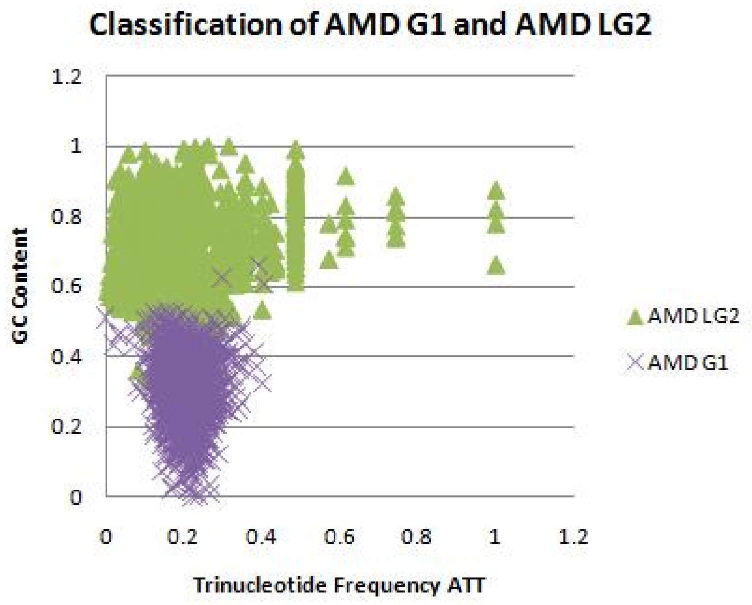


Figure 4.5: Classification using GC Content and Trinucleotide Frequency ATT for 3,000 Artificially Created Shotgun Sequence Fragments obtained from AMD G1 and AMD LG2

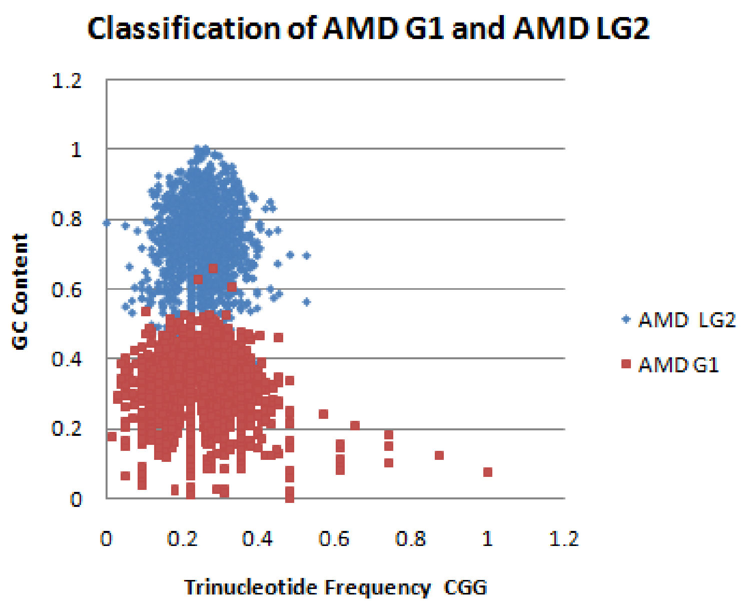


Figure 4.6: Classification using GC Content and Trinucleotide Frequency GCC for 3,000 Artificially Created Shotgun Sequence Fragments obtained from AMD G1 and AMD LG2.

resulted in fewer misclassifications.

	# Fragments classified incorrectly		
Signature	Lepto.	Ferrop. Type II	Total %
GC	500	27	0.026
T_z	NA	NA	NA
TR_z	NA	NA	NA
$GC - T_z - TR_z$	640	27	0.033
T_m	147	16	0.0081
TR_m	170	6	0.0088
T_m- TR_m	127	10	0.0068
$GC - T_m - TR_m$	129	11	0.007

GC refers to clustering with GC content, T_z refers to tri-nucleotide and TR_z refers to tetra-nucleotide frequencies using zero-order Markov chains. TR_m refers to tetra-nucleotide frequencies using a maximal-order Markov chain, T_m indicates the tri-nucleotide frequencies using a maximal-order Markov chain. Combinations of different signatures are shown by hyphenating individual frequencies. A value of NA indicates that the signatures could not separate the fragments into groups and all the data was placed into one class.

Table 4.3: Separating 20,000 Fragments from AMD into Two Classes Using Different Signatures

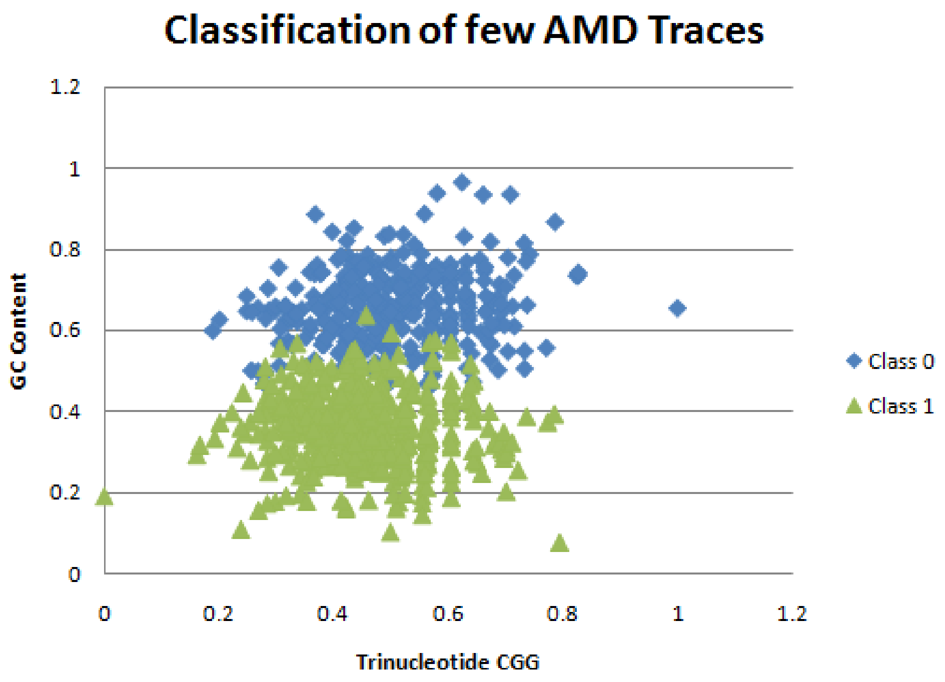


Figure 4.7: Classification using GC Content and Trinucleotide Frequencies for Shotgun Sequence Fragments obtained from unclassified AMD traces

4.4.3 Clustering at Levels within Taxonomy

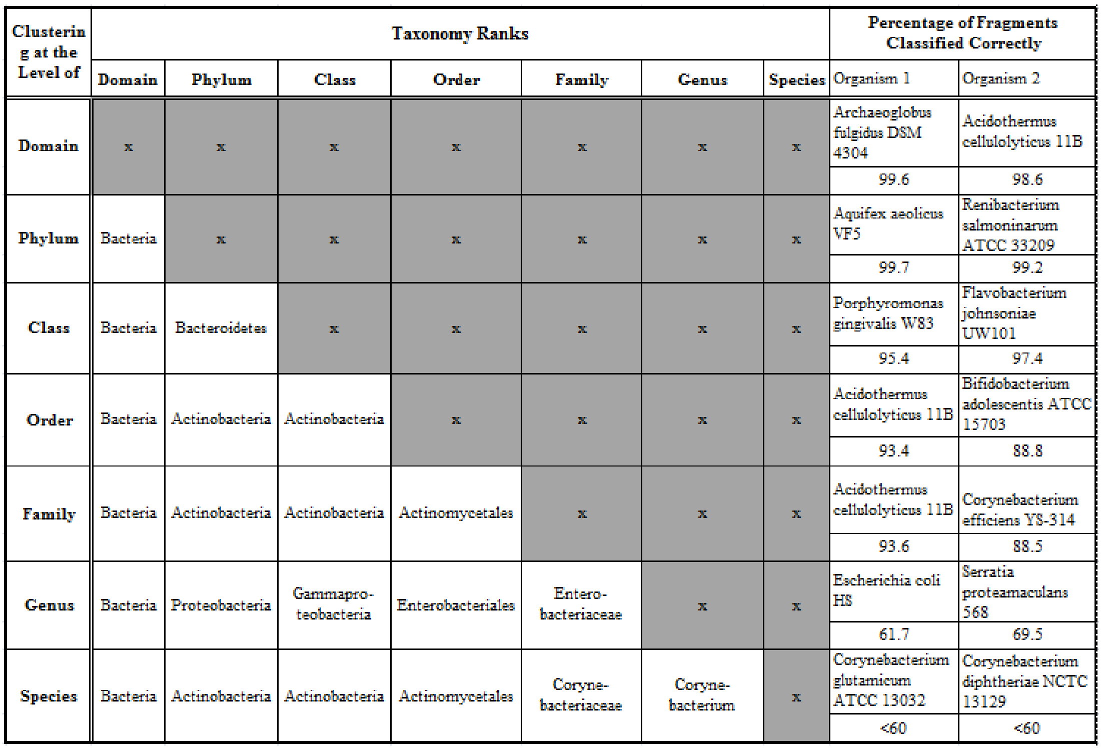
Taxonomy is a method of classifying organisms into broad types and further classifying types into subtypes to form a hierarchical structure. All species are classified into heirachical groups starting with *domain*, *kingdom*, *phylum*, *class*, *order*, *family*, *genus* and *species*. There are three *domains* that are further grouped into *kingdoms*. Each *kingdom* is divided into *phylum*, *phylla* are divided into *classes*, *classes* are divided into *families*. Organisms that belong to different domains can have genomes that are different. But as we go down the hierarchy the similarities increase, therefore organisms that belong to same species are highly similar. As similarities between organisms increase it becomes difficult to cluster them. In this section a study on the classification of fragments is performed. The study is intended to identify the accuracy of the classifier to classifying genomes that at different ranks in taxonomy.

The classification of genome fragments proposed in this dissertation can be evaluated by combining fragments belonging to different taxonomical ranks. Organisms related at seven different ranks are chosen for the study. The classification results are shown in Figure 4.8 The first row in Figure 4.8 indicates organisms belonging to two different *domain*. *Archaeoglobus fulgidus DSM 4304* belongs to archea and *Acidothermus cellulolyticus 11B* is a bacteria. The second row indicates classification of organisms belonging to the same domain bacteria but different *phyla*. The third row classifies organisms belonging to different *class* but same *phyla* or division. The forth row classifies organisms belonging to the same *class* but different *order*. The fifth row contains organisms from the same *order* and different *family*, the sixth row contains elements from same *family* different *genus* and finally the last row contains organisms from same *genus* different *species*. The 'x' in a cell indicates that organisms have different taxonomical grouping. The last two columns give the name of the organism and results of clustering.

The results of classification are shown in Figure 4.9. The results indicate that at higher ranks in the taxonomy the classifier works well and the classification gradually decreases until *genus* after which there is sharp increase in miss-classifications. It can be inferred from the graph that the classifier could not perform well for organisms from the different *genera* within the same *family* and different *species* within the same *genus*. There is a sharp decrease when sequences become more similar and thus this classifier could be used for seperaring sequences at higher levels in the taxonomy.

4.5 Discussion

This Chapter presented clustering of fragments to aid assembly of metagenome data sets. The results indicate that GC content and oligonucleotide frequencies can be used for clustering. Analysis of certain pairs also shows that there is over- and under-representation of certain oligonucleotide words. These words showed compact clusters as can be shown in Figure4.5 that frequency of ATT for the fragments belonging to Ferroplasma sp. Type II are in the range of 0.1 to 0.4. Thus ATT, that is under-represented in *Ferroplasma sp. Type II* can be useful for clustering the group.



Clustering at the Level of	Taxonomy Ranks							Percentage of Fragments Classified Correctly	
	Domain	Phylum	Class	Order	Family	Genus	Species	Organism 1	Organism 2
Domain	x	x	x	x	x	x	x	Archaeoglobus fulgidus DSM 4304	Acidothermus cellulolyticus 11B
								99.6	98.6
Phylum	Bacteria	x	x	x	x	x	x	Aquifex aeolicus VF5	Renibacterium salmoninarum ATCC 33209
								99.7	99.2
Class	Bacteria	Bacteroidetes	x	x	x	x	x	Porphyromonas gingivalis W83	Flavobacterium johnsoniae UW101
								95.4	97.4
Order	Bacteria	Actinobacteria	Actinobacteria	x	x	x	x	Acidothermus cellulolyticus 11B	Bifidobacterium adolescentis ATCC 15703
								93.4	88.8
Family	Bacteria	Actinobacteria	Actinobacteria	Actinomycetales	x	x	x	Acidothermus cellulolyticus 11B	Corynebacterium efficiens YS-314
								93.6	88.5
Genus	Bacteria	Proteobacteria	Gammaproteobacteria	Enterobacteriales	Enterobacteriaceae	x	x	Escherichia coli HS	Serratia proteamaculans 568
								61.7	69.5
Species	Bacteria	Actinobacteria	Actinobacteria	Actinomycetales	Corynebacteriaceae	Corynebacterium	x	Corynebacterium glutamicum ATCC 13032	Corynebacterium diphtheriae NCTC 13129
								<60	<60

The rows of the table indicate the level or rank within the taxonomy. The cells marked as 'x' indicate that the rank is different for the organisms and the first cell in each row indicates the level at which the organisms are different. Last two columns of the table indicate the name of the organisms and the percentage of sequences that were assigned correctly.

Figure 4.8: Classification of Fragments at Different Taxonomical Groups

An analysis of the misclassification shows a third of misclassifications using tri- or tetra-nucleotide frequencies occurred for sequences that were less than 700 base pairs(bps) long. Tests were conducted on sequences that had length in the range 500-700 bps. The software could not cluster if most of the sequences were shorter than 600 bps. A sequence of length 1024 contains 256 tetra-nucleotides and would be required if there is a random-utilization of tetra-nucleotides. Consider a sequence of length 600 bps, this sequence contains 150 tetra-nucleotide words, by taking overlapping frequencies we get total of 597 tetra-nucleotide words. If an exact utilization of all words was known then each word would be present two times. For sequences of length less than 600, tetra-nucleotides will be present less than two times, thus would not cover the 256 words well. Since words in DNA are known to have a non-random

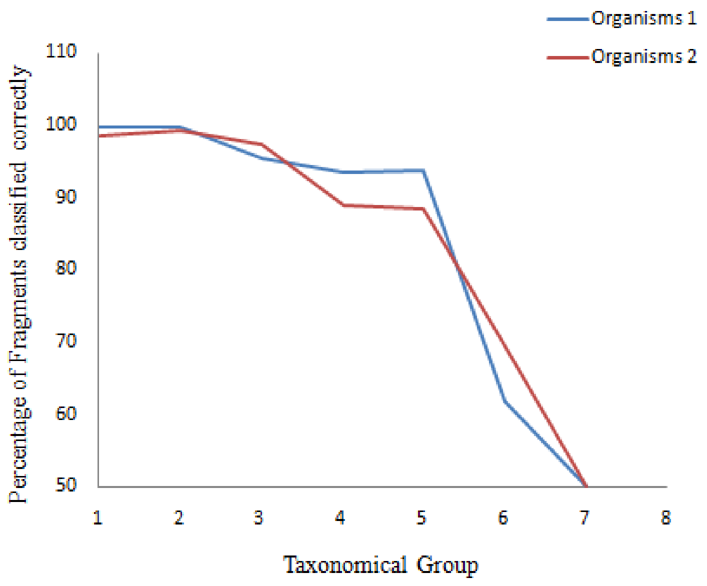


Figure 4.9: Classification Performance at different Taxonomical Ranks

utilization there will be some words that will be present more than two times and certain words that will be absent from the set or under-represented. The tri-nucleotide plots indicate that the frequencies of all the words are not spread out but rather stay in a small range. If 50% of frequencies are over-represented and other 50% under-represented then there will be words that wil be present more than three times and words that will be present less than two times. If a 75% to 25% ratio is considered the gap between present and absent frequencies increases and thus can provide a useful clustering. Longer sequences will result in more reliable frequencies.

Clustering at various ranks of taxonomy is presented in this Chapter to show the algorithm's performance. The results indicate that clustering becomes difficult or is not possible at lower ranks within the taxonomy using the signatures proposed. More advanced signatures or supervised clustering can be a potential approach for clustering organisms that are highly similar.

Chapter 5

Environmental Sequence assembly and parallel implementation

5.1 Introduction

The huge size of genome sequences and processing time complexity of assembly makes them a viable candidate for parallel processing. Parallel processing is a technique to divide a task into subtasks and to run the subtasks simultaneously on more than one processor. This makes task implementation faster. As processors have become less expensive, parallel systems became affordable and have popularity to solve some NP-Hard problems that could not have been approached earlier.

The applicability of parallel processing to a problem depends on how well the problem can be divided into subtasks. For example, Monte Carlo methods use repeated random sampling to solve a problem, each process is independent of other processes. Therefore, these methods can be parallelized easily [71]. Assembly of fragments is a task that can be easily divided into subtasks, thus parallel processing can be used to solve the assembly problem and to make it faster.

In this chapter we present a parallel implementation of the fuzzy sequence assembly, and discuss load balancing issues and the new approach in Section 5.3. In Section 5.3 we describe the sequential approach to dividing the process into modules. This modularization leads us to the parallel implementation of the algorithm in Section 5.4. Results and discussions are presented in Section 5.5.

5.2 Background

Utilization of parallel techniques have been sought to a limited extent in bioinformatics. Wide spread use of parallel processing in this domain has been limited by the availability of applications that exploit parallel architectures [18].

Bioinformatics databases have grown tremendously in the past few years. An example is the growth of sequences present in GenBank by 200% over the past 6 years. Techniques such as compression and hashing have been used in the past to address the issues of size and speed in assembly. Compression of DNA sequences was performed to reduce the size of the fragments and perform faster comparisons. Since the first method to compare compressed strings was proposed in [1], there have been several techniques to apply compression to DNA sequences. DNA compression results in increased speed, but is suitable for exact matching [27, 68]. Therefore, this technique is not well suited to approximation problems such as assembly. FASTA is a rapid heuristic search method for protein and DNA alignment that performs statistical analysis of data prior to alignment [28]. Hashing techniques are used to search for word patterns of high frequency which makes assembly faster and heuristics have been used to reduce number of comparisons required in assembly. These methods, although useful, have some drawbacks, such as, the need for decompression before assembly. Hashing techniques make assembly very fast but limit comparisons of overlaps to high frequency words.

To comply with the fast growth of these databases, high performance computing has been sought as an answer to increase performance. Application of parallel techniques can make the bioinformatics techniques faster without compromising the results. A parallel algorithm for DNA alignment, in which alignment is calculated on each node and then gathered into a single global alignment on the root, is presented in [53]. **mpiBLAST** is an example of an open-source, parallel implementation of NCBI BLAST to improve the performance of BLAST by several orders of magnitude while scaling to hundreds of processors [32]. *ParAlign* is another parallel approach for

DNA alignment and search within databases [60]. These tools with few others have led the way to parallel implementation of bioinformatics algorithms. This chapter presents a parallel implementation of the assembly algorithm proposed in Chapters 3 and 4. This parallel implementation is sought to make the assembly faster and allow larger genomes to be assembled.

5.2.1 Embarrassingly Parallel Computation

An embarrassingly parallel computing is a technique where a data set can be divided into completely equal independent tasks that execute simultaneously [71]. This also suggests that the only communication between the processors is when the process starts and ends. The problems that fit this category are easy to implement and require no special techniques. Examples of such problems are mandle-brot, Monte Carlo methods and some image processing transformations.

Simple implementation of sequence assembly fits the embarrassingly parallel computation technique. This simple parallel implementation is to divide the DNA fragments into groups of fixed size. The groups are then given to different nodes, where each node has access to the entire fragment data, but only assembles the sequences within the group. This strategy, though simple can increase the performance of assembly as n processes can now divide the work. A master-slave architecture works well for this kind of implementation. The master process performs initial grouping of fragments and distributes the jobs to the slaves. The slaves receive individual jobs and assemble the data given to them and return the assembled sequences to the master. The master makes a round through the contigs to eliminate any duplicate contigs and finalizes the assembly.

There are some drawbacks of the embarrassingly parallel implementation to assembly. This parallelization can improve performance, but still requires the same amount of memory on all the slaves, as all slaves need access to the entire data set for assemble. This assembly can create a large number of duplicate contigs as each slave works independently. Since slaves do not share information more processing needs to be done during assembly, which can be avoided by using simple heuristics.

A faster parallel version is to assign certain groups of sequences to each processor, the processors find contigs for that group and need only segments within the group. Even though process reduces redundant assembling, it performs assembly on limited number of sequences. A Master-Slave architecture is proposed for the parallel implementation based on this partitioning. To increase efficiency classes are created using the technique listed in Chapter 4, so that data is classified meaningfully.

There are several other parallel architectures to solve different kinds of problems, such as Master-Slave architecture for more information see [71].

5.3 Module Assembly Sequential Approach

The process of assembly starts with identification of DNA signatures to classify fragments, followed by fragment classification to divide the data sets into smaller categories. The classes represent two significant properties: 1) they contain fragments belonging to the same group or species present in the metagenomic data set and 2) they have continuity and can form contigs for the local regions of the genome. This grouping is followed by assembly, which is divided into three major steps. These steps of a sequential assembly are used to design the tasks for the parallel processing as shown in Figure 5.1. The steps start with assembly of the K individual classes (using the clustering algorithm presented in Chapter 4) to form local contigs and finally combining results into an overall assembly. The operations carried in the three modules is described in the following subsections.

5.3.1 Round 1: Divide-and-Conquer

This module is the first step of assembly. The sequences are classified using the approach presented in Chapter 4. There are two approaches to sequence classification: 1) forming classes of large size that represent a single organism or closely related organisms, 2) forming smaller classes that are compact, and, few of the small classes

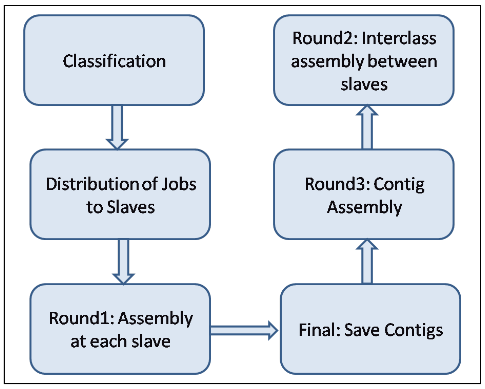

Classification
Round2: Interclass assembly between slaves
Distribution of Jobs to Slaves
Round3: Contig Assembly
Round1: Assembly at each slave
Final: Save Contigs

Figure 5.1: Module Assembly

represents one large group. We use the second approach as it creates smaller groups that can be assembled faster, and also helps to balance the load on a slave. Consider the AMD environmental genome, that contains a total of 8 million basepairs. A 10X coverage of this genome results in 80 million basepairs or over 100,000 sequences of length 700 basepairs. This metagenomic population contains two groups of organisms, thus a classification by groups will yield two classes. Assuming even distribution of the groups there will be two large groups of 50,000 sequences each for each slave. The situation gets worse if the data is not distributed evenly and can result in load balancing issues as illustrated in Figure 5.2. Thus, we choose to select smaller groups, so that the slaves get smaller classes to assemble, which also results in faster in-class assembly. Further details of load balancing will be covered in Section 5.4.3.

Each of the classes is assembled to form contigs within the class. Each class is an independent unit and does not interact with any other class. Two kinds of results are obtained from this module: 1) Contigs formed within the class, and 2) Singlets in the class. A singlet refers to a sequence that was not assembled with any other sequence. Singlets are found in a group if a sequence cannot be part of any contig. In this case, a singlet can also be found if a sequence was not classified correctly or even if it was classified correctly it does not have a good overlap with any of the sequences present in the class. Singlets are determined during the assembly process by identifying sequences that could not assembe. In such case the singlet can be

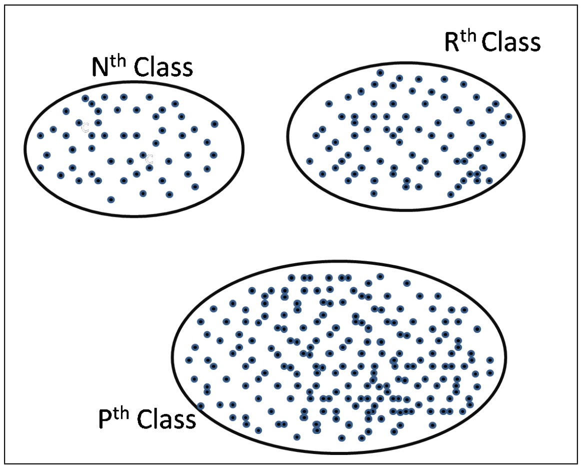


Figure 5.2: Unbalanced Classes

assembled with sequences from another class.

5.3.2 Round 2: Review-and-Regroup

The second module deals with correcting any mis-classifications made during the first module, reassigns singlet fragments between classes, and performs intra-class assembly. To avoid comparing all possible classes we use heuristics to determine whether the classes have any similarity, and only classes that are close to each other are used to perform intra-class assembly. The heuristics are described in Section 5.4. This round is required only if smaller groups of classes are created. If larger classes are formed, each class represents one group from a metagenome Round 2 is not required. Round 2 can be further enhanced by comparing singlets in all classes with each other. If a large number of singlets are found, this round can become a bottleneck for the entire approach.

5.3.3 Round 3: Collect-and-Combine

As the name of the module suggests this section of the algorithm collects the contigs assembled in Round 1 and Round 2. It combines the local contigs to form larger contigs and connects overlapping regions between classes. The contigs are cleaned to remove duplicates and longer contigs are created by further assembly.

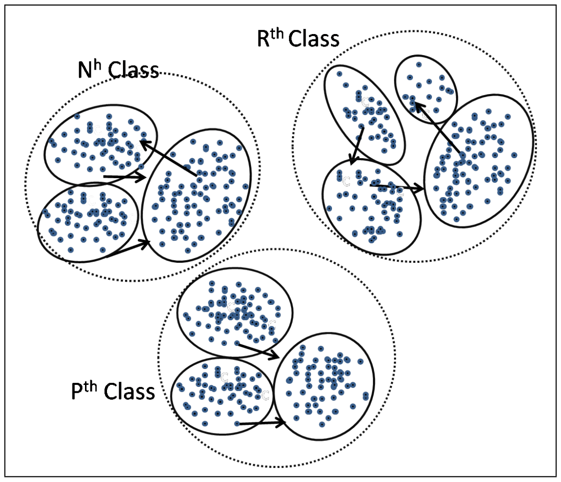


Figure 5.3: Round 2 Assembly: Review-and-Regroup

5.4 Parallel Design

5.4.1 Class Restrictions

Round 2 assembly which reviews sequence fragments and regroups data, becomes computationally intensive as the number of singlets and classes increases. Therefore, to keep the task from growing, restrictions are imposed on classes. Restriction on classes ensures that classes communicate with certain classes only and not all the classes present in the data set. Class restrictions are imposed by two techniques described in this section.

Class GC Content

In order to restrict comparisons of singlets with classes that are close, GC content of classes is used. Recall from Chapter 4, that each class has number of fragments that are grouped by similarity. GC content is one of the factors that was used to form these classes. Theoretically, GC content percentages of elements in a class are similar and within a small range. We compute the low and high GC content of each class. A singlet is compared with elements of this class if it's GC content is within this range. Along with GC content, entire basepair content (BPC) is also measured. BPC of a fragment is the number of A,C,T and G present in the fragments. BPC is calculated for the ends of the fragments, rather than the whole sequence. Using the entire fragment may include information that is not required for assembly. As assembly requires similarity at the ends of the fragments and if these are not similar then comparing the rest of the fragment cannot yield information important to assembly. If the BPC at the sequence ends is not similar then the sequences have less or no chance of assembly. BPC is calculated by simply counting the numbers of A, C, G and T.

Hierarchical Clustering

In a hierarchical clustering approach classes are created by combining smaller classes that are close. In this approach hierarchical clustering is used in Round 2 to restrict class comparisons to subclasses within the same larger group. Thus classes within a group are compared, whereas classes from different groups are not compared.

5.4.2 Master-Slave Architecture

Master-Slave architecture is a popular technique of dividing tasks in a message passing parallel implementation. In this technique, one processor is the master and the rest of the processors are slaves. The master performs the classification of sequences, and assigns tasks to the slaves. The slaves perform the assembly and return results back to the master. The master performs the final processing before finishing the task. The entire process can be described in the pseudo-code in Figure 5.4.

5.4.3 Load Balancing

Load balancing is an important issue in parallel implementation of bioinformatics algorithms. Load balancing aims to distribute jobs evenly across all process to maximize the overall performance. Several factors affect the balancing of a parallel system, such as different processing speeds of processors and different data sizes. If all processors are given equal size data sets and if everything else is fixed then maximum speedup can be expected. In reality, all slaves do not finish at the same time. As

```
//master
if(myrank == 0)
 {
      clusterSequences();
      loadBalance();
      sendJobs();
     //master waits for Round1 results from slave
     recvfromSlave();
     eliminateDupContigs();
     sendtoSlave();
     //wait for Round2 results
     recvfromSlave();
     eliminateDupContigs();
 }
 else //slave
 {
    round1();
    sendtoMaster();
    //wait for master to send contigs
    recvfromMaster();
    round2();
    eliminateDupContigs();
    sendtoMaster();
 }
 if(myid == 0) //master
 {
    round3();
    saveContigs();
 }
```

Figure 5.4: Master-Slave Architecture

sequences are classified based on nucleotide similarities, the classes formed are not necessarily the same size. Therefore, a class that is much bigger than other classes can be found.

A large class is not unusual in a metagenome sequence data set. This could be due to the fact that the data itself contains certain types of organisms that dominate in the population. Thus, these fragments will group into a single class. Figure 5.2 shows a pictorial representation of load balancing issues. In this example, after phylogenetic classification; one class gets most of the sequences and other classes are much smaller in size. Recall from Chapter 2 that assembly is an NP-Hard problem; this also makes assembly of *Pth* class slower, thus creating a bottleneck for the entire process.

Unbalanced classes can be avoided by imposing restriction during classification. For example, if classes are large then they cannot be merged. Nevertheless, if the signatures are quite similar then this restriction will not prevent fragments from belonging to a class.

To address this issue, a method for dividing large classes during assembly is presented. In a regular master-slave implementation of assembly, each slave gets a fixed number of classes. In the proposed load balancing approach, depicted in Figure 5.5, if a class is large in size then an embarrassing parallel technique is applied to divide the class between more than one processor. Each slave is responsible for assembling an assigned portion of the class, even though they have access to all the sequences in the class. This splicing does not affect the assembly and may add certain amount of overload to the slaves sharing the class. But overall is faster than one slave performing the entire assembly.

5.5 Results and Analysis

Parallel implementation generally aims at increasing the performance of a process, thus the speedup attained via the parallel processing is measured to analyze how well the parallel algorithm performed. The speedup factor *S(n)*, is defined as a measure of relative performance between a multiprocessor system and a single processor system,

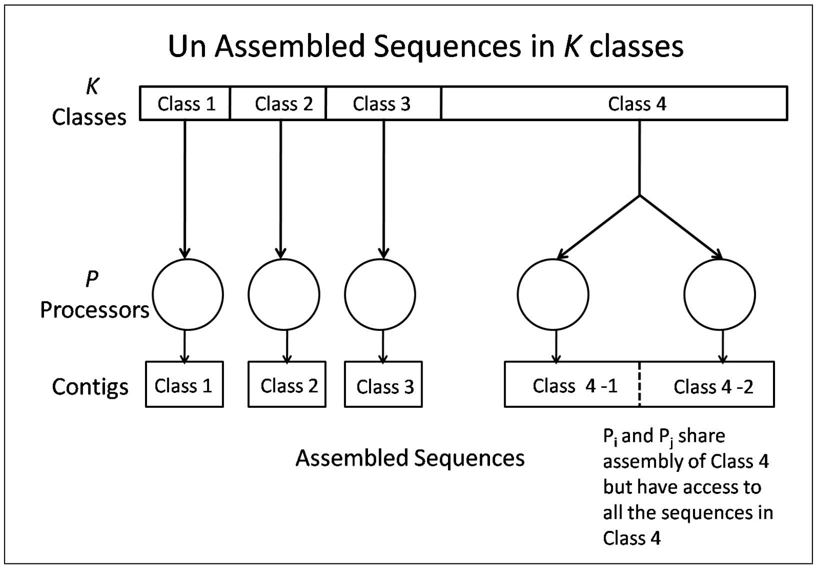

Un Assembled Sequences in *K* classes
K Classes
Class 1
Class 2
Class 3
Class 4
P Processors
Contigs
Class 1
Class 2
Class 3
Class 4 -1
Class 4 -2
Assembled Sequences
P_i and P_j share assembly of Class 4 but have access to all the sequences in Class 4

Figure 5.5: Static Load Balancing using Mixed Mode Simple Parallelization

and is given by

$$S(n) = \frac{t_s}{t_p} \tag{5.1}$$

Where, t_s is the execution time using one processor and, t_p is the execution time using a multiprocessor system.

Assembly of *Yersinia pestis Pestoides F plasmid CD* complete sequence, containing 71,507 base pairs, was performed are results are shown below. Figure 5.6 compares the speedup of Round 1 and Round 2 assembly processes. Because Round 1 is a simple parallel technique where master assigns jobs to the slaves and waits for the results, it has less communication overload. In Round 2, after slaves complete Round 1 the master collects all the contigs together and creates overall contig list and sends it back to the slaves. Master waits for the all slaves to complete Round 1 before it can start with next round. The slaves perform a second assembly with the singlets. Thus, Round 2 results in having more communication overload and processing. The speedup for assembly is given in Figure 5.6, which indicates that the parallel approach increases the performance of assembly.

Results of assembling two scaffolds obtained from the AMD environmental genome

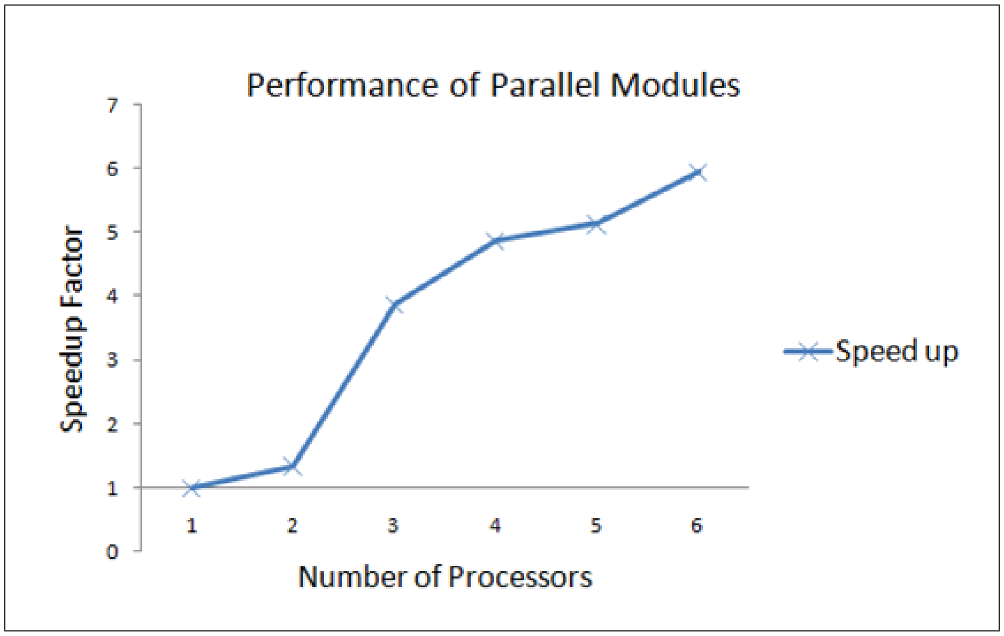


Figure 5.6: Speedup for Assembly of *Yersinia pestis Pestoides F plasmid CD* with Round1

sequences from NCBI are shown in Table 5.1.

	Percentage Genome Recovered
Genome	mpiCFGS
Ferrosplasma	96.57%
Leptospirillum Sp Type	94.02%

Table 5.1: AMD Scaffolds Parallel Implementation
In the table, mpiCFGS refers to the parallel implementation presented in this chapter. The shotgun sequences were created artificially. Contigs upto length 4000 bps were obtained.

Preliminary results of assembling a portion of AMD data belonging to *Ferroplasma* and *Leptospirillum* sets are displayed in Table 5.2. The assembly was run with 4X coverage. With a low coverage of 4X, 58.59% and 67.75% of the genomes could be recovered.

The results indicate that parallel implementation improves the speed of assembly. Parallel implementation also allows us to assemble larger data sizes at reasonable speeds, which was difficult earlier due to memory limitations. We compared different parallel implementations for assembly to analyze which implementation suits best. A

	Percentage Genome Recovered
Genome	mpiCFGS
Ferrosplasma	58.59%
Leptospirillum Sp Type	67.75%

Table 5.2: AMD Parallel Implementation
This table indicates results for the the two organisms of AMD data set listted above, total of 2244520 nucleotides are present in both the genomes.

simple load balancing technique is proposed, which can be improved by addition of dynamic load balancing.

Chapter 6

Conclusions and Open Questions

This body of work contributes an effective framework for assembly of sequences using fuzzy logic. The work was initiated to create an assembler that can work on metagenome fragments without pre-processing such as using tools to bin the data or performing a classification.

The process identifies parameters that influence assembly and constructs fuzzy characteristic functions for each parameter. Contiguous sequences are formed with optimal values of all the parameters. Fuzzy LCS, an extension of dynamic programming with fuzzy logic, is proposed.

The fuzzy assembly process can successfully assemble artificial shotgun sequences. The functions proposed can be easily adapted in other assembly methods or techniques and can be used for trace sequences. Sequences are not pre-processed or post-processed; thus, additional analysis is not required. I proposed using fuzzy thresholds for accepting or rejecting sequences. Sequences with low quality regions are not clipped but are saved for further processing; thus a sequence with a low quality region can be used if it has overlap with higher quality regions. All these parameters create an assembler that can work on low quality regions, is not dependent on user thresholds as it uses fuzzy values, and automates confidence values.

A classification scheme was also developed in which higher ranks in taxonomy were classified, and lower levels such as genomes within a order and family were classified and, finally, domains within an environmental genome were classified. The classifier, ClusFGS, based on an aggregate fuzzy value of the parameters, groups sequences having a higher aggregate fuzzy value in the same cluster. This grouping reduces the processing time of assembly, because instead of $n(n\text{-}1)/2$ comparisons per round, $m \times n$ comparisons are performed, where $m < n$ and m is the number of groups and n is number of sequences. If the value of m is significantly less than n assembly can be done in linear time. ClusFGS resulted in an improvement in time but did not yield a good recovery of the genome. ClusFGS depends on initial seed selection, singletons from one group were not reassigned to another group thus limiting the algorithm.

ClusFGS classifier was enhanced to use DNA signatures to perform a genome specific classification. A fuzzy clustering algorithm was proposed to classify shotgun genome fragments into taxonomic classes. This classifier uses a combination of signatures such as GC content, and tri-, and tetra-nucleotide frequencies.

An analysis of the signatures for Acid Mine Drainage metagenome was performed by comparing the frequencies of two fragments belonging to each of the classes. Over- and under-represented tri-nucleotides from two groups of AMD are shown in the plots along with the values. Figures 6.1 and 6.2 show the over-represented pairs from *Leptospirillum sp. Group II* (AMD LG2) and *Ferroplasma sp. Type II*(AMD G1), respectively. Comparing the plot and the results obtained in Chapter4 we can infer that: (1) Certain nucleotide pairs were dominant in each of the classes. (2) Certain nucleotides are avoided in both the groups consistently; for example, *taa* and *tta* is avoided by all the fragments in AMD LG2 and same with *cgg* for AMD G1 and (3) The frequencies of the fragments from two genomes are distinct from each other and much similar to the fragments that belong to the same genome. These properties have allowed us distinguish fragments and thus obtain the classification.

In Figure 6.1, four fragments from *Leptospirillum sp. Group II* (AMD LG2) are generated. The plot shows the over-represented trinucleotide frequencies. The plot indicates that all four fragments have the same three codons missing. In Figure 6.1, four fragments from *Ferroplasma sp. Type II* were taken and there frequencies are calculated, the plots indicate that all four fragments have the same codon under-represented (or missing). This indicates that fragments belonging to the same genome

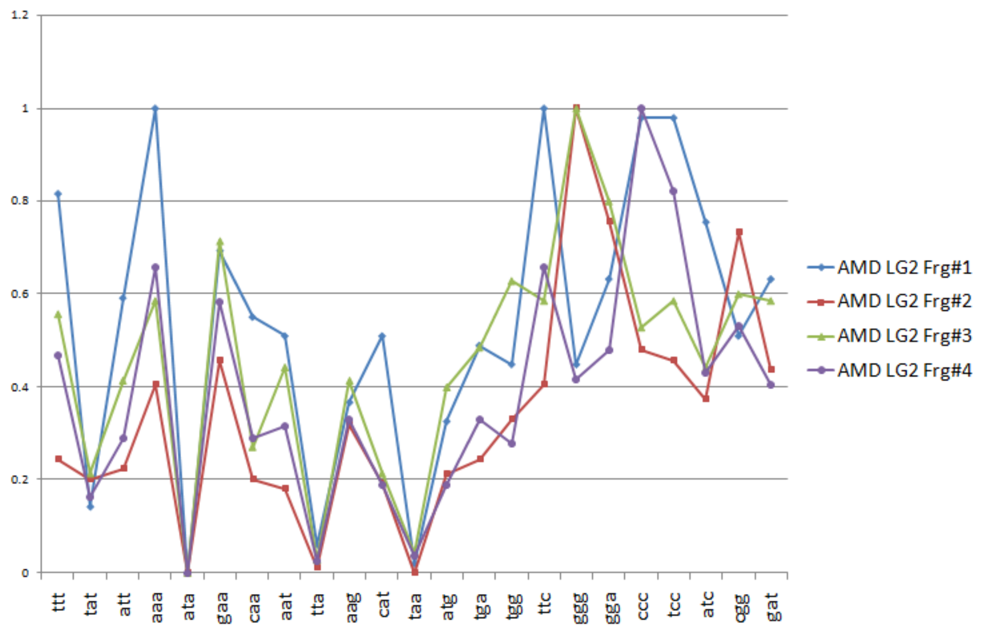


Figure 6.1: Over-represented Tri-nucleotide Frequencies in AMD LG2

show a similar pattern of codon usage. This is an important observation and can be used to improve classification.

To asses if few tri-nucleotides that are over- or under-represented can classify fragments a classification of AMD data set was performed using less than 64 tri-nucleotides. It was found that 12 tri-nucleotides were sufficient to classify the data set. Similarly, 100 randomly chosen tetra-nucleotides frequencies were sufficient for a classification.

In Chapter4 classification of fragments using different signatures and combination of signatures was done. Besides classifying a subset of the AMD metagenome, the full metagenome was used as a test case and was classified into two domains of bacteria and archea. The results indicate that Maximal-order Markov chains were the best separators and obtained the best classification. Zero-order Markov chain could not classify the data; this could be due to the fact the zero-order chain does not remove dependencies from previous frequencies. Using a combination of DNA signatures also showed improved classification results. In Chapter 4, Table 4.3 clustering using

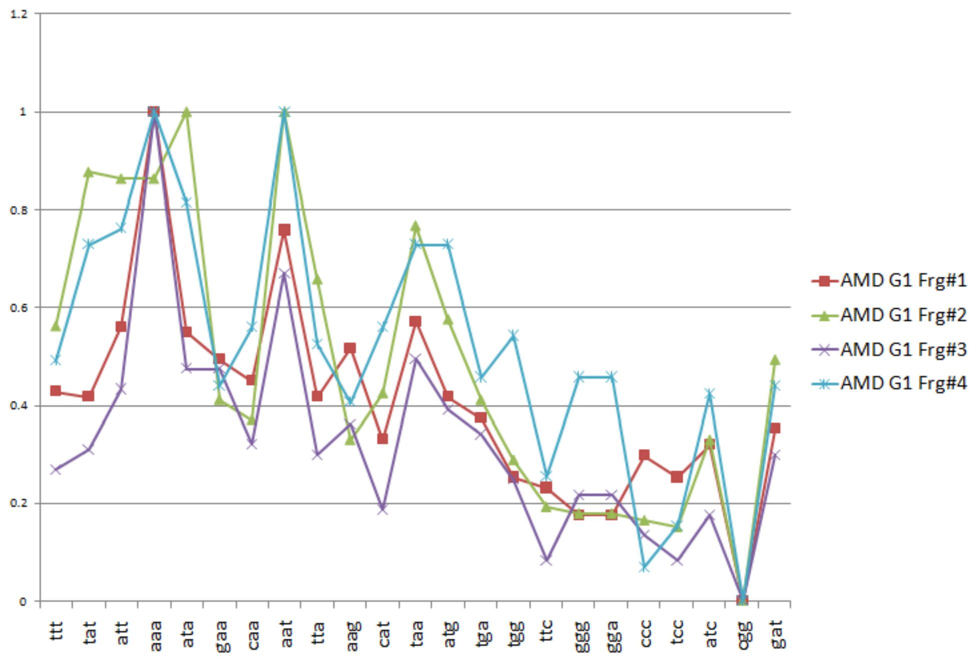


Figure 6.2: Over-represented Tri-nucleotide Frequencies in AMD G1

GC content was improved with addition of tri- and tetra-nucleotide frequencies, also combination of tri- and tetra-nucleotide frequencies showed an improvement over tri- and tetra-nucleotide frequencies individually. These results of clustering obtained for different types of genomic sequences, thus testing a wide range of input genomes as shown in Tables 4.3, 4.2 and 4.1. In Table 4.8, tests were conducted on seven different combinations of organisms for classification, the tests indicate where the classifier breaks and where it can be used for classification.

Prior to this work, classification was typically performed on full genomes or fragments that were longer than shotgun sequences. This work suggests that fragments of smaller length can also be classified into groups. This research also shows that combination of nucleotide frequencies produced better results.

The research builds an unsupervised classification that requires no training or identification of important signatures. An unsupervised method opens room for unclassified data to be classified. For example, an unidentified genome sequence can be added with known sequences before the classifier is run. The unknown sequence fragments group with sequences that are closer in phylogeny, thus some knowledge of the unknown sequence can be acquired. If the two groups fall into the same category they possibly are close related organisms.

A known limitation of the classification technique is that the classes have to be set by the user. If the classes are not set, the K-means algorithm determines final groups. The algorithm creates classes that are compact rather than classes that are large and dispersed. Thus, fragments from one genome can be present in more than one class, ensuring classes with minimal or no misclassifications. The technique can be improved by application of validity measures, using marker regions to identify and create groups that can represent a number of genomes within the sample.

6.1 Future Work and New Directions

This work put forth s question of using an adaptive assembler that can adapt itself to the input to generate the best possible assembly. The concept of an adaptive assembler is dependent on two factors: statistical analysis of data and the best approximation of the parameters.

The assembly can be further improved by data reduction before assembly, making it possible to run larger data sets at faster speeds. Current assemblers use different data reduction techniques such as encoding the data, using hash tables, and indexing. An improvement to the assembler would be to add a combination of exact and approximate assembly, the exact assembly can be hashed and can be assembler at a faster rates. The assembler can assemble contigs of smaller sizes. This can be improved by adding paired-end assembly strategies.

Proposed using parallel processing for the assembly, which can reduce the time for assembly. The assembler can run on different processes using the groups already created by classification.

The classification proposed can also be enhanced by generating signatures that are different from each other rather than selecting random signatures. This, again, is dependent on analysis of the data set provided, such as, patterns of oligonucleotide words.

The results indicate that we are able to group AMD shotgun sequences and several other simulated shotgun sequences from their frequencies and GC Content. The classifier could be enhanced by adding higher level frequencies such as penta-nucleotide and dicodons. Analysis of the DNA signatures can be done to find the best discriminatory words, enabling selection of features that suit the data set best rather than using all available frequencies. These will be unique for different organisms. We have conducted preliminary work in this direction and classified the AMD metagenome using 12 random tri-nucleotide frequencies with more than 98% accuracy. Similar results were achieved using 100 tetra-nucleotide frequencies. Tests showed that highly represented common words do not create a good classification, whereas words that were under-represented show a better classification. This result clearly demonstrates that all the frequencies are not required and that using a lower number of frequencies may also remove biases from words that are common to the genomes in the sample. I propose using Principal Component Analysis to reduce the dimensionality of the oligonucleotides that are required for classification. Principal components can be identified for any given data set. These components can be used for the clustering. The principal components can also be used for other purposes.

Genome databases are huge in size and are growing at a rapid rate. When a new DNA or protein sequence is acquired, a probe is run against the available nucleotide or protein databases to find close species to the probe genome. These tests perform a local alignment of the sequence, as was shown in Chapter 3. Even though there have been advancements in alignment techniques to make a fast probe, such a probe is a long process. Based on the taxonomical grouping presented in this work, we propose using a signature-based approach for alignment that can group the databases and performing a full local alignment only after the probe sequence is matched with a close group. This application can be used with BLAST and, thus, can improve the speed of BLAST by reducing the search time and the number comparisons required.

New sequencing technologies yield short sequences(30-40bps)such as

TTTTTGTTAATGAATGTAATTTCAAATGTTAGCTCA

This sequencing technique obtains sequences of shorter length and several sequences are obtained thus increasing the coverage. This kind of sequences creates a new challenge in assembly.

> "'Further difficulties arise because of the unavailability of paired-end reads, although limited forms of paired-end sequencing are just becoming available. The short read lengths and absence of paired ends make it difficult for assembly software to disambiguate repeat regions, therefore resulting in fragmented assemblies"'. [47]

To address this problem we propose an assembler that is depth based and not length based. We believe that a fuzzy application is an answer to such a problem because the data itself contains certain ambiguities. For instance, similarity of fragmented assemblies can be measured using fuzzy similarities.

One more direction would be analyzing the fragment lengths that are required for clustering. Fragment length is an important issue when using signatures based on oligonucleotide words. A statistical analysis of fragments of different lengths for oligonucleotides of different length can give insight into signatures that can be statistically relevant for a fragment of certain length. This analysis can also be used to determine fragment X-oligonucleotide Y pairs, such that, fragment of length X can be classified with oligonucleotide of length Y.

One of the main future directions is to make this assembler into a full function tool such as PHRAP, CELERA or JAZZ assemblers. Thus enhancements to make the tool work for larger data sizes and larger trace shotgun sequences are neccessary.

Moving out of the field of bioinformatics, the work proposed can also be used in other search problems. The fuzzy extension of LCS using dynamic programming can be used elsewhere for approximate string matching. New directions can be explored beyond those mentioned above.

Bibliography

[1] A. Amir and G. Benson. Two-dimensional periodicity in rectangular arrays. *SIAM Journal on Computing*, 27(1):90–106, 1998.

[2] A. D. Baxevanis and B. F. Ouellette. *Bioinformatics : A practical guide to the analysis of genes and proteins.* John Wiley, 1^{st} edition, 2005.

[3] O. Beja, M. Suzuki, E. Koonin, L. Aravind, A. Hadd, L. Nguyen, R. Villacorta, M. Amjadi, C. Garrigues, S. Jovanovich, R. Feldman, and E. DeLong. Construction and analysis of bacterial artificial chromosome libraries from a marine microbial assemblage. *Environmental Microbiology*, **2**:516–529, 2000.

[4] J. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms.* Plenum Press, 1981.

[5] J. Birdsell. Integrating genomics, bioinformatics and classical genetics to study the effects of recombination on genome evolution. *Molocular Biology Evolution*, **19**:1181–1197, 2002.

[6] T. Brown. *Genomes.* Garland Science, 3^{rd} edition, 2006.

[7] C. Burge, A. Campbell, and S. Karlin. Over- and under-representation of short oligonucleotides in DNA sequences. *Proceedings National Acaddemy of Science U S A*, 89(4):1358–1362, 1992 Feb 15.

[8] B. Chang and S. Halgamuge. Approximate symbolic pattern matching for protein sequence data. *International Journal of Approximate Reasoning*, **32**(2):171–186, 2003.

[9] K. Chen and L. Pachter. Bioinformatics for whole-genome shotgun sequencing of microbial communities. *PLoS Computational Biology*, **1**:106–112, 2005.

[10] T. Chen and S. S. Skiena. A case study in genome-lebel fragment assembly. *Bioinformatics*, **16**(6):494–500, 2000.

[11] S. Choudhuri. The path from nuclein to human genome: A brief history of DNA with a note on human genome sequencing and its impact on future research in biology. *Bulletin of Science Technology Society*, **23**:360–367, 2003.

[12] G. C. Conant and P. O. Lewis. Effects of nucleotide composition bias on the success of the parsimony criterion in phylogenetic inference. *Molecular Biology Evolution*, **18**:1024–1033, 2001.

[13] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. McGraw-Hill, 2nd edition, 2001.

[14] E. DeLong. Microbial population genomics and ecology. *Currrent Opinions Microbiology*, 5(5):520–524, 2002 Oct.

[15] B. Ewing and P. Green. Basecalling of automated sequencer traces using phred. ii. error probabilities. *Genome Research*, **8**:186–194, 1998.

[16] R. Fleischmann, M. Adams, O. White, R. Clayton, E. Kirkness, A. Kerlavage, C. Bult, B. Tomb, J Dougherty, and J. Merrick. Whole-genome random sequencing and assembly of Haemophilus Influenzae Rd. *Science*, **269**(5223):496–512, 1995.

[17] A. P. Gasch and M. B. Eisen. Exploring the conditional coregulation of yeast gene expression through fuzzy k-means clustering. *Genome Biology*, **3**(11):1–22, 2002.

[18] E. Gough and M. Kane. Evaluating parallel computing systems in bioinformatics. In *Third International Conference on Information Technology: New Generations*, pages 233–238, 2006.

[19] P. Green. *Documentation for Phrap*. Genome Center, University of Washington, 2006.

[20] G. Gutman and G. Hatfield. Nonrandom utilization of codon pairs in Escherichia coli. *Proceedings National Academy of Science USA*, **86**:3699–3703, 1989.

[21] J. Hu, B. Li, and D. Kihara. Limitations and potentials of current motif discovery algorithms. *Nucleic Acids Research*, **33**(15):4899–4913, 2005.

[22] X. Huang and A. Madan. CAP3: A DNA sequence assembly program. *Genome Research*, **9**(9):868–877, 1999.

[23] P. Hugenholtz. Exploring prokaryotic diversity in the genomic era. *Genome Biology*, **3**:reviews0003.1–reviews0003.8., 2002.

[24] S. Karlin, I. Ladunga, and B. Blaisdell. Heterogeneity of genomes: Measures and values. *Proceedings National Academy of Science USA*, **91**:12837–12841, 1994.

[25] J. Kececioglu and E. Myers. Combinatorial algorithms for DNA sequence assembly. *Algorithmica*, **13**:7–51, 1995.

[26] S. Kim and A. Segre. AMASS: A structured pattern matching approach to shotgun sequence assembly. *Journal of Computational Biology*, **6**(2):163–186, 1999.

[27] C. Lei, L. Shiyong, and J. Ram. Compressed pattern matching in dna sequences. In *Computational Systems Bioinformatics Conference*, pages 62–68, 2004.

[28] D. Lipman and W. Pearson. Rapid and sensitive protein similarity searches. *Science*, **227**:1435–1441, 1985.

[29] C. Looney. Interactive clustering and merging with a new fuzzy expected value. *Pattern Recognition*, 35:2413–2423, November 2002.

[30] A. McHardy, H. Martn, A. Tsirigos, P. Hugenholtz, and I. Rigoutsos. Accurate phylogenetic classification of variable-length DNA fragments. *Nature Methods*, **4**(1):63–72, 2007.

[31] E. Mongodin, J. Emerson, and K. Nelson. Microbial metagenomics. *Genome Biology*, 6(10):347, 2005.

[32] mpiBLAST. mpiblast: Open-source parallel blast. http://www.mpiblast.org/, NCBI, 2008.

[33] G. Myers. Whole-genome DNA sequencing. *IEEE Computational Engineering and Science*, 1:33–43, 1999.

[34] S. Nasser, R. Alkhaldi, and G. Vert. A modified fuzzy k-means clustering using expectation maximization. In *IEEE International Conference on Fuzzy Systems*, pages 231–235, 2006.

[35] S. Nasser, G. Vert, M. Nicolescu, and A. Murray. Multi sequence assembly for prokaryotes and eukaryotes using fuzzy logic. *International Journal of Information Technology and Intelligent Computing*, **2**(4), 2007.

[36] S. Nasser, G. Vert, M. Nicolescu, and A. Murray. Multiple sequence alignment using fuzzy logic. In *Proceedings of the IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology*, volume **7**, pages 304–311, 2007.

[37] NCBI. National center for biotechnology information. http://www.ncbi.nlm.nih.gov/, NIH, 2007.

[38] H. Noguchi, J. Park, and T. Takagi. Metagene: prokaryotic gene finding from environmental genome shotgun sequences. *Nucleic Acids Research*, **34**(19):5623–5630, 2006.

[39] M. A. Nowak. *Evolutionary Dynamics: Exploring the Equations of Life*. Belknap Press,, 1^{st} edition, October 2006.

[40] J. Oliver and A. Marn. A relationship between GC content and coding-sequence length. *Journal of Molecular Evolution*, **43**(3):216–223, 2004.

[41] G. Olsen, D. Lane, S. Giovannoni, N. Pace, and D. Stahl. Microbial ecology and evolution: A ribosomal RNA approach. *Annuals Reviews Microbiology*, **40**:337–365, 1986.

[42] H. H. Otu and K. Sayood. A divide-and-conquer approach to fragment assembly. *Bioinformatics*, **19**(1):22–29, 2003.

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

[44] E. Pillsbury. A history of genome sequencing. Technical report, Yale University Bioinformatics, 2001.

[45] M. Pop, A. Phillippy, A. L. Delcher, and S. L. Salzberg. Comparative genome assembly. *Briefings in Bioinformatics*, **5**(3):237–248, 2004.

[46] M. Pop, S. L. Salzberg, and M. Shumway. Genome sequence assembly: Algorithms and issues. *IEEE Computer*, pages 47–54, July 2002.

[47] M. Popa and S. L. Salzberga. Bioinformatics challenges of new sequencing technology. *Trends in Genetics*, **24**(3):142–149, 2008.

[48] J. R. Powell and E. N. Moriyama. Evolution of codon usage bias in drosophila. *Proceedings National Academy of Science*, **94**:7784–7790, 1997.

[49] D. T. Pride, R. J. Meinersmann, T. M. Wassenaar, and M. J. Blaser. Evolutionary implications of microbial genome tetranucleotide frequency biases. *Genome Research*, **13**(2):145–158, February 1, 2003.

[50] M. Rappe and S. Giovannoni. The uncultured microbial majority. *Annual Reviews Microbiology*, **57**:369–394, 2003.

[51] O. Reva and B. Tmmler. Differentiation of regions with atypical oligonucleotide composition in bacterial genomes. *BMC Bioinformatics*, 6(1):251, 2005.

[52] M. Rondon, P. August, A. Bettermann, S. Bradly, T. Grossman, M. Liles, K. Loiacono, B. Lynch, I. MacNeil, C. Minor, C. Tiong, M. Gilman, M. Osburne, J. Clardy, J. Handelsman, and R. Goodman. Cloning the soil metagenome: a strategy for accessing the genetic and functional diversity of uncultured microorgansims. *Applications Environmental Microbiology*, **66**:2541–2547, 2000.

[53] T. Royce and R. Necaise. A parallel algorithm for dna alignment. *Crossroads, AMC Student Magazine*, **9**(3):10 – 15, 2003.

[54] K. Sadegh-Zadeh. Fuzzy genomes. *Artificial Intelligent Medicine*, **18**(1):1–28, 2000.

[55] F. Sanger, A. Coulson, G. Hong, D. Hill, and G. Petersen. Nucleotide sequence of Bacteriophage Lambda DNA. *Journal Molecular Biology*, 162(4):729–773, 1982.

[56] F. Sanger, S. Nicklen, and C. AR. DNA sequencing with chain-terminating inhibitors. *Proceedings National Academy of Science USA*, **74**(12):5463–5467, 1977.

[57] J. Stein, T. Marsh, K. Wu, H. Shizuya, and E. DeLong. Characterization of uncultivated prokaryotes: isolation and analysis of a 40-kilobase-pair genome fragment from a planktonic marine archaeon. *Journal of Bacteriology*, **178**:591–599, 1996.

[58] M. Sugeno. *Industrial Applications of Fuzzy Control*. Elsevier Science Inc., 1985.

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

[60] R. T. Paralign: a parallel sequence alignment algorithm for rapid and sensitive database searches. *Nucleic Acids Research*, **29**(7):1647–52, 2001.

[61] S. T and W. M. Identification of common molecular subsequences. *Journal of Molecular Biology*, **147**:195–197, 1981.

[62] H. Teeling, A. Meyerdierks, M. Bauer, R. Amann, and F. Glockner. Application of tetranucleotide frequencies for the assignment of genomic fragments. *Environmental Microbiology*, **6**:938–947, 2004.

[63] H. Teeling, J. Waldmann, T. Lombardot, M. Bauer, and F. O. Glockner. TETRA: A web-service and a stand-alone program for the analysis and comparison of tetranucleotide usage patterns in dna sequences. *BMC Bioinformatics*, **5**:163, 2004.

[64] P. Turnbaugh, R. Ley, M. Mahowald, V. Magrini, M. ER, and G. JI. An obesity-associated gut microbiome with increased capacity for energy harvest. *Nature*, **444(7122)** :1009–10, 2006.

[65] G. W. Tyson, J. Chapman, P. Hugenholtz, E. E. Allen, R. J. Ram, P. M. Richardson, V. V. Solovyev, E. M. Rubin, D. S. Rokhsar, and J. F. Banfield. Community structure and metabolism through reconstruction of microbial genomes from the environment. *Nature*, 428:37–43, 2004.

[66] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, **14**(3):249–260, 1995.

[67] J. C. Venter, K. Remington, J. F. Heidelberg, A. L. Halpern, D. Rusch, J. A. Eisen, D. Wu, I. Paulsen, K. E. Nelson, W. Nelson, D. E. Fouts, S. Levy, A. H. Knap, M. W. Lomas, K. Nealson, O. White, J. Peterson, J. Hoffman, R. Parsons, H. Baden-Tillson, C. Pfannkoch, Y.-H. Rogers, and H. O. Smith. Environmental genome shotgun sequencing of the sargasso sea. *Science*, **304**:66–74, 2004.

[68] J. Wallace, G. Vert, and S. Nasser. An efficient method for compressing and searching genomic databases. In *High Performance Computing and Simulation*, pages –, 2007.

[69] N. Watanabe and T. Imaizumi. Fuzzy k-means clustering with crisp regions. *The 10th IEEE International Conference on Fuzzy Systems*, pages 199–202, 2001.

[70] R. Welch, V. Burland, G. Plunkett, P. Redford, P. Roesch, D. Rasko, E. Buckles, S. Liou, A. Boutin, J. Hackett, D. Stroud, G. Mayhew, D. Rose, S. Zhou, D. Schwartz, N. Perna, H. Mobley, M. Donnenberg, and F. Blattner. Extensive mosaic structure revealed by the complete genome sequence of uropathogenic Escherichia coli. *Proceedings National Academy of Science U S A*, **99**(26):17020–17024, 2002.

[71] B. Wilkinson and M. Allen. *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers.* Prentice Hall, 2^{nd} edition, 2004.

[72] L. Wong. *The Practical Bioinformatician.* World Scientific Publishing Company, 1^{st} edition, 2004.

[73] T. Woyke, H. Teeling, N. Ivanova, M. Huntemann, M. Richter, F. Gloeckner, D. Boffelli, I. Anderson, K. Barry, H. Shapiro, E. Szeto, N. Kyrpides, M. Mussmann, R. Amann, C. Bergin, C. Ruehland, E. Rubin, and N. Dubilier. Symbiosis insights through metagenomic analysis of a microbial consortium. *Nature*, **443**(7114):925–7, 2006.

[74] D. Xu, R. Bondugula, M. Popescu, and J. Keller. Bioinformatics and fuzzy logic. In *IEEE International Conference on Fuzzy Systems*, pages 817–824, 2006.

[75] J. Yen and R. Langari. *Fuzzy Logic: Intelligence, Control, and Information.* Pearson Education, 1998.

[76] L. A. Zadeh. Fuzzy sets. *Information and Control*, **8**:338–353, 1965.

[77] L. A. Zadeh. *Fuzzy logic and approximate reasoning*, volume **30**. Synthese, 1975.

[78] Z. Zhang, S. Schwartz, L. Wagner, and W. Miller. A greedy algorithm for aligning DNA sequences. *Journal of Computational Biology*, **7**:203–214, 2000.