Master’s Thesis
submitted in partial fulfilment of the
requirements for the course “Applied Computer Science”

Algorithms for Efficient Maintenance and
Comparison of Multiple Sequence Alignments

Uche Ukeh Oteh

Institute of Computer Science

Bachelor’s and Master’s Theses
of the Center for Computational Sciences
at the Georg-August-Universität Göttingen

29. May 2015
Georg-August-Universität Göttingen
Institute of Computer Science

Goldschmidtstraße 7
37077 Göttingen
Germany

☎ +49 (551) 39-172000
✉ +49 (551) 39-14403
✉ office@informatik.uni-goettingen.de
✉ www.informatik.uni-goettingen.de

First Supervisor: Dr. Lena Wiese
Second Supervisor: Dr. Mehmet Gültas
I hereby declare that I have written this thesis independently without any help from others and without the use of documents or aids other than those stated. I have mentioned all used sources and cited them correctly according to established academic citation rules.

Abstract

The alignment and analysis of sequences is a commonly occurring task in the bioinformatics field. Multiple sequence analysis serves as a means to identify and infer similarities both structurally and functionally among protein families. Results for this analysis can be stored and compared against new data so as to provide a quick means of comparison. A means to store and analyze pre-computed files is therefore necessary so as to avoid re-computation of files by theoretical methods which are time-consuming. In this paper, we present an approach to compare multiple sequence alignment files stored in a local database. Applying data technologies to identify pre-computed files, which include exact file match and propose a string similarity method to identify files that do not match 100% but fulfill our similarity criteria. Our approach combines different methodologies to achieve the end result and also provides an application that has much room for further improvement.
Contents

1 Introduction .................................................. 3
   1.1 Biological Preliminaries and sequences ................... 3
   1.2 Biological databases ..................................... 4
   1.3 The MSA Problem ......................................... 5
   1.4 Motivation ................................................ 5
   1.5 Aim ....................................................... 5
   1.6 Structure of the Thesis ................................... 6
   1.7 Chapter Summary ......................................... 7

2 Foundations and Literature Review ................................... 9
   2.1 Sequence Alignments ....................................... 9
   2.2 Multiple Sequence Alignment ................................ 11
      2.2.1 Sequence Alignment analysis ......................... 11
      2.2.2 Alignments and scoring ................................ 13
   2.3 Hash Trees ............................................... 16
   2.4 Bloom Filters ............................................ 18
      2.4.1 Working methodology .................................. 19
      2.4.2 Extensions and variations of bloom filters .......... 21
      2.4.3 History and application ............................. 22
   2.5 Database and Database Index .............................. 22
      2.5.1 Data and Databases .................................... 22
      2.5.2 Database Indexes ...................................... 24

3 Application of Functions to File Comparison ......................... 29
   3.1 Input Format .............................................. 30
      3.1.1 Fasta Method ........................................ 31
   3.2 Hash Functions ............................................ 31
      3.2.1 The message digest hash algorithm .................. 32
   3.3 String Similarity and Hamming Distance ................... 34
### CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4</td>
<td>Database Organization and Structure</td>
<td>37</td>
</tr>
<tr>
<td>3.4.1</td>
<td>SQL Overview</td>
<td>37</td>
</tr>
<tr>
<td>4</td>
<td>Working Design and Implementation strategy</td>
<td>39</td>
</tr>
<tr>
<td>4.1</td>
<td>General</td>
<td>39</td>
</tr>
<tr>
<td>4.1.1</td>
<td>System and Development Tools</td>
<td>39</td>
</tr>
<tr>
<td>4.1.1.1</td>
<td>PHP</td>
<td>40</td>
</tr>
<tr>
<td>4.1.1.2</td>
<td>MySQL database</td>
<td>41</td>
</tr>
<tr>
<td>4.2</td>
<td>Overview</td>
<td>41</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Sequence Algorithm</td>
<td>43</td>
</tr>
<tr>
<td>4.3</td>
<td>Query Execution</td>
<td>44</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Receive Query and Extract Parts</td>
<td>44</td>
</tr>
<tr>
<td>4.3.1.1</td>
<td>Query Sequence</td>
<td>46</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Data Aggregation</td>
<td>48</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Obtain Result and Answer Query</td>
<td>48</td>
</tr>
<tr>
<td>4.4</td>
<td>Application and Results</td>
<td>48</td>
</tr>
<tr>
<td>4.5</td>
<td>Query Example</td>
<td>50</td>
</tr>
<tr>
<td>4.6</td>
<td>Performance</td>
<td>51</td>
</tr>
<tr>
<td>4.6.1</td>
<td>Experimental Testing</td>
<td>52</td>
</tr>
<tr>
<td>4.6.2</td>
<td>Reliability and Functionality</td>
<td>52</td>
</tr>
<tr>
<td>4.6.3</td>
<td>Responsiveness and Time consumption</td>
<td>52</td>
</tr>
<tr>
<td>5</td>
<td>Conclusion</td>
<td>55</td>
</tr>
<tr>
<td>5.1</td>
<td>Summary</td>
<td>55</td>
</tr>
<tr>
<td>5.2</td>
<td>Implications for practice and discussions</td>
<td>56</td>
</tr>
<tr>
<td></td>
<td>Bibliography</td>
<td>63</td>
</tr>
</tbody>
</table>
List of Figures

1.1 Structure of the Thesis ........................................... 6
2.1 Divisions of Sequence Alignments .................................. 11
2.2 An MSA of a Protein Family ...................................... 12
2.3 A simple MSA sequence with imbedded dashes ..................... 13
2.4 Figure of a Hash Tree ............................................... 17
2.5 An empty Bloom filter .............................................. 19
2.6 A Bloom filter with 4 Bits enabled ................................. 20
2.7 Bloom filter with the 2 keys embedded ............................. 20
2.8 Diagram of database types ......................................... 23
2.9 Diagram of an index file ............................................ 24
3.1 Fasta Format .......................................................... 30
3.2 Hash Sequence ........................................................ 31
3.3 Hash Function ......................................................... 32
3.4 Hamming Distance cube ............................................. 36
3.5 Hamming String Comparison ....................................... 37
4.1 Three Tier Architecture ............................................. 40
4.2 Task flow sequence .................................................. 42
4.3 Data flow diagram ..................................................... 45
4.4 Bloom filter of sample file ......................................... 47
4.5 Home Page ........................................................... 49
4.6 Prediction Page ....................................................... 49
4.7 Contact Page ........................................................ 50

1. Database Structure .................................................... 65
2. Database Content ...................................................... 66
3. Send Query ............................................................. 67
4. View Query ............................................................. 67
List of Listings

4.1 Hamming Distance .................................................. 46
1 Query .............................................................. 68
2 System Initialization .................................................. 70
3 Hamming distance .................................................... 72
4 Bloom Filter .......................................................... 76
5 Hash Value ........................................................... 77
6 Fasta Format .......................................................... 81
Chapter 1

Introduction

1.1 Biological Preliminaries and sequences

In the biological field, there are basically two types of biopolymers which can be represented as sequences. They are Nucleic acids and proteins.

- Nucleic Acid Sequences: A nucleic acid is a polymer based compound that contains numerous nucleoids residues intertwined with phosphate and sugar to form long chains of complementary strands that have the structure of a double helix. Nucleic acids carry and store the genetic information of life. By this, we mean direct activities of cell reproduction and growth. They are usually linear polymers connected by base pairs resembling the shape of a ladder. Nucleic acids are among the most important biological macromolecules alongside proteins and contain all necessary genetic information. In other words, they convey nucleic acid sequences through all living organisms and this is the reason for the connectedness between all living organisms as this flow of information is basically the same in all organisms. Nucleic acids were discovered by Friedrich Miescher in 1869 [1]. Nucleic acids are composed of two closely related nucleic acid namely Deoxyribonucleic acid also known as DNA and Ribonucleic acid, RNA. The major component of a nucleic acid is called a nucleotide and a DNA has four nucleotide bases; A for Adenine, C for Cytosine, G for Guanine and T for Thymine and each base can pair with only one base. DNA molecules contain several million nucleotides and RNA may contain up to several thousand [2]. Usually, DNA molecules are double strand molecules and RNA are single strand. DNA encode, contain and carry instructions needed for the functionality of living organisms. RNA, on the other hand is responsible for the conversion of this genetic information into protein sequences. Nucleic acid sequencing is the determination of the order of nucleotide bases in a DNA or RNA molecule. This is important as it may represent vital biological information.

- Proteins Sequences: Proteins are natural polymers made up of chains of amino acids. They
contain at least one long peptide and many polypeptides. They are the fundamental blocks of cells and are involved in varied biological functions which are essential to living organisms. Proteins are made up of 20 amino acid residues with each protein polymer assuming a specific 3-D structure by folding into chains and linking its amino acids \[3\]. DNA controls how these amino acids are linked together. The process by which this genetic code is translated is known as transcription or translation. This order by which the amino acid are linked dictates the structure which the protein compound assumes \[4\]. Protein sequencing is therefore referred to as a technique to determine the amino acid sequence in a protein compound. The sequencing of protein is necessary as it help for the exosition of the function and structure of protein compounds which directly influences understanding of the cellular process. There are primarily two methodologies involved in the sequencing of proteins. They are the Edman degradation reaction and by mass spectrometry.

1.2 Biological databases

A biological database can be described as a collection or aggregation of biologically related data organized in a particular format and accessible from a stored location. Recent trends in biological advances have given rise to more biological sequences being annotated. Biological databases are important for handling, managing and making this data available and accessible to a wide variety of users. They offer scientist and researchers the ability to access structure data and sequence information of numerous sequences ranging across a large number of organisms. These databases contain a wide variety of information spanning; protein sequences, its variations and structure, DNA variations, gene sequences, etc. They are large well-ordered bodies of persistent data associated with biological information as well as the computer infrastructure to manage, update and query the system. They provide easy access to information needed to answer biological queries. With the recent trends in bioinformatics and the application of information technology to analyze, manage and organize biological data, the need for biological databases has grown over the years and hence has led to its growing importance and use in bioinformatics and molecular biology.

Biological databases can be classified broadly into structure and sequences databases. With sequence database responsible for the management of protein and nucleic acid sequences and structure database catering for proteins. Biological databases help make biological data available to scientist and researchers in a computer readable form. Example of biological databases include; GenBank, EMBL and DDBJ \[5\]. They are mostly developed and maintained by a number of private and government organizations.
1.3 The MSA Problem

Sequence alignment is an important topic in bioinformatics and it aims to analyze and compare DNA or protein sequence. Sequence analysis is concerned with the comparison of two or more sequences by analyzing the individual characters in a given sequence and checking if they are aligned. In this thesis report we are concerned with the type of sequence alignment known as multiple sequence alignment. Multiple sequence alignment is a sequence alignment type in which three or more sequences are analyzed. It is the primary method used to infer homology structurally and functionally. Its primary aim is to generate a concise collection of sequenced data so as to be able to obtain the relationship of sequences to a gene family. By evolution, there exist relationships between all living organisms which infers seminaries between organisms. If the alignments between sequences are strong, it can be used to infer characteristics. However in multiple sequence analysis, it is major difficult to effectively establish a universal algorithm [6], which would deliver the best multiple sequence score.

1.4 Motivation

A large amount of research has been done on multiple sequence alignments, lots of algorithms have been developed to effectively compare the structure and alignment of multiple sequences to examine the relationships between input sequences. While there are well known programs and algorithms for the comparison of these multiple sequences, a local database would be beneficial to help in the speedy comparison of already aligned MSA files. This local database would aim to reduce the time spent on aligning new sequences by comparing them against existing files in the database thereby drastically reducing the time consumed on new analysis or the computing overhead leading to a more efficient means of MSA processing. That being said, a key challenge in this field is on methodologies that would support fast and accurate database search on local MSA files.

1.5 Aim

The overall aim of this thesis is to examine the applicability of already existing data structure technologies to MSA comparison and storage. In more detail, it is the aim of this thesis to:

1. Provide an analysis of the current state and application of data structure technologies relevant to this topic.

2. Analyze the applicability of these technologies to MSA comparison and storage and propose a system that utilizes them for effective MSA comparison and storage.
3. Create or adopt a storage back end that satisfies the above requirements and provides needed support for MSA files while also providing a PHP front-end that interfaces and enables communication between system.

1.6 Structure of the Thesis

This thesis is structured into five chapters beginning with an introductory chapter. The second chapter takes a look at the foundations of the study and the literature review of the necessary topics. We analyse sequence alignments, hash trees, bloom filters and database indexes taking a look at them generally and investigating their uses. In chapter 3, a study analysis is done on figuring out a methodology that combines some of the technologies mentioned in chapter 2 and others, while also investigating their applicability to MSA comparison. We examine properties, types and functionality of these methodologies. Chapter 4 discusses the working design and implementation of the proposed system. Chapter 5 concludes with a discussion and summary of the paper.

Figure 1.1: Structure of the Thesis
1.7 Chapter Summary

This chapter has laid the foundation for the rest of the thesis. It introduced the research background firstly and then pointed to the current state, importance and necessity to perform alignments. It also examined quickly the types of sequence alignments and finally outlined the research framework steps that are needed to be able to satisfy the requirements of this thesis.
Chapter 2

Foundations and Literature Review

This chapter examines and presents the current state of knowledge by taking a literature review of the research objectives and primary goal of this thesis outlined in section 1.5 of the first chapter. The proper understanding of multiple sequence analysis is necessary in this study as well as that of database structures, their use and application. As such we will critically examine multiple sequences and the database structures that are relevant in the fulfillment of this thesis.

To begin with, the following sections present an overview of multiple sequence algorithms as well as principles of hash trees which is a data structure that can be used in the storage, management and verification of data [7]. We then consider its current state and application. This is then followed by a thorough review of Bloom filters and its working methodology. Lastly, database systems are analyzed and the importance of database indexes to data sets highlighted.

2.1 Sequence Alignments

Alignments in biological terms can be described as the supposition of positional homology between Amino acids with homology being a proof of origination from a common ancestor both in terms of structure and evolution. Sequence alignment is a procedure used in bioinformatics or molecular biology for analyzing and visualizing relationships that may exist between structurally or evolutionarily related amino acids or proteins [8]. Sequence alignments can be done manually or with the aid of computers. Alignments involves arranging the sequences to be compared in a single line format which allows for easier analysis. It helps in the visualization of structural, functional or evolutionary characteristics in a visual concise form.

The study, analysis and organization of sequences is used to reveal several important information in the relationships that exist between protein compounds or families specifically and living organisms in general [9]. Given that all living organism are related to each other through evolution,
we all have some common ancestors from which we originated. Sequence analysis helps to discover and infer the relationships in living beings by aligning and analyzing homologous sequences. It also helps in the transfer of information and the properties derived from that information from a previously known sequence and attributes some or all of those properties to a relatively unknown sequence if it can be determined that similarities exist when their patterns are aligned and analyzed. Sequence alignments are also useful as they help identify portions in protein structure which are important functionally or structurally [10].

Sequence Alignment is generally done by arranging two or more DNA or protein sequences next to each other so that their elements collocate thus making it easier to examine structural or functional relationships between these sequences. Matching or similar sequences can then be deduced as emanating from the same or similar families and expected to share similarities or traits.

Simple short sequences can be aligned by hand with little effort. However for complex, lengthy or numerous sequences, human effort is certainly not sufficient and as such algorithms to perform these alignments need to be developed. To compare two sequences, a method known as 'pairwise' alignment can be utilized. The Pairwise alignment system is composed of three different methods; word, dynamic programming and dot-matrix methods. For higher amounts of sequence combinations, algorithms are necessary and needed to produce high-quality alignments and a method of analysis called multiple sequence analysis is used [9]. Using a computer or software based approach to sequence alignments requires one to follow either of two distinct methods outlined below:

- **Global alignment**: This is an alignment methodology that takes into consideration the overall and entire sequence structure of the sequences to be compared. It is an end-to-end alignment method. Global alignment is generally characterized by several elements which may include:
  - **Gaps**: A certain number of dashes or null characters can be embedded into any of the sequences.
  - **Substitution scores**: These are the scores given to alignment of pairs of letters. They can be match or miss-match scores depending on the alignment of the letters.
  - **Gaps scores**: When there is an alignment between a letter and a dash or null character, it is known as a Gap score.
  - **Alignment scores**: The total of scores for the letters that are aligned or paired and scores for the letters paired with null characters is known as the Alignment score.

- **Local Alignment**: This is an alignment type that aims to match or compare a portion of a sequence with its reference. It is an investigation of the best match between sequences, useful when used for the comparison of sequences that share similarities in some parts by may differ somewhere else.
2.2 MULTIPLE SEQUENCE ALIGNMENT

A multiple sequence alignment can be in the simplest of terms described as the optimal alignment of three or more homologous sequences. It aims to reflect the relationships between three or more sequences. These sequences are mainly protein, RNA and DNA originating majorly from the same family and as such may share traits and characteristics. MSA is an efficient and effective method of aligning, analyzing and detecting similarities and sometimes differences that may exist among sequences. It helps in the discovery of new members of a family by comparing and analyzing new sequences against pre-known sequences and as such could help in the creation of models which could be used to benchmark, test hypothesis or in certain cases even to illustrate the dissimilarities between groups of sequences. Multiple sequence alignment is a very useful tool in the bioinformatics, molecular and biological fields as accurate and good alignments can have very significant impact, produce useful information or generate results that are vital for the identification of new protein members or the relationships between existing ones.

2.2.1 Sequence Alignment analysis

The main aim of multiple sequence alignment is to be able to identify the homology between three or more protein and nucleic acid sequence patterns as shown in the figure below. A good alignment is characterized as one that is able to align the characters of the sequence so that they
are homologous and this is clearly visible by character alignment when the sequences are lined up one beneath the other to produce a biological correct alignment.

![An MSA of a Protein Family](image)

Figure 2.2: An MSA of a Protein Family

But the question arises, how is a computer program which is used for the analysis and alignment of the sequences supposed to be able to read sequences of protein and nucleic acids and produce biologically correct alignment given that a standard computer program understands nothing about sequences or biology? To be able to get a computer program to properly analyze these sequences and produce meaningful alignments from input sets of data, we need to develop a scoring scheme for the program which would assign a quality score to all the possible sequences alignments from the input data, analyze this and output or select the one with the best quality. In other words we would require that high scores are assigned to alignments that are meaningful and biologically okay and that lower scores are assigned to alignments which are not.

As the alignment of sequences have been established to have a direct relationship with the evolution of the sequences, it is safe to assume that when there is an alignment between two sequences, they are therefore related and as such have similar origins. A gap however is taken to correspond to a deletion or a character in the wrong position. Based on this we can establish a quality score for the sequences, with higher scores going to sequences that are homologous. i.e.: The higher the degree of similarity, the higher the quality. It should be noted that methods such as a substitution matrix are used to assign similarity scores to all the possible alignments of the protein or nucleic acid sequences and this scoring is based on values of the substitution scores and the penalties caused by gaps in the algorithm. The quality score of an alignment in this sense therefore would be derived by analyzing the aligned sequences, summing its similarity values and subtracting this from the penalty score caused by gaps in the algorithm.

These scores are important for multiple sequence analysis as they give various insights in molecular biology or bioinformatics and help determine quality algorithms which are important as:
2.2. MULTIPLE SEQUENCE ALIGNMENT

- They are beneficial for the design of PCR primers
- They are useful for predicting or determining secondary and tertiary structures
- They help in the discovery of similarities between known sequence families and unknown sequences.
- They help in the motif characterization of protein families.

It should be noted that alignment of sequences in simple terms is a process that is done by arranging the sequences on top of each other and then adding a certain number of spaces represented by dashes in all the arranged sequences if need be to get them to be of equal length.

\[ S1: \quad T\text{T}A\text{G}\text{G}\text{A}\text{C}\text{G}\text{TGA}\text{A}T\text{C}\text{G} \]
\[ S2: \quad A\text{C}\text{G}\text{T}\text{A}\text{G}\text{A}\text{G}\text{G}---\text{C}\text{T}\text{G}\text{A} \]
\[ S3: \quad T\text{T}---\text{C}\text{T}\text{G}\text{A}---\text{T}\text{A}\text{GCA}\text{T}\text{G}\text{A} \]

Figure 2.3: A simple MSA sequence with imbedded dashes

2.2.2 Alignments and scoring.

For calculating global multiple sequence alignments, a popular method can be used. This method is known as the progressive approach. In this approach, multiple sequences are computed by performing a couple of pairwise alignments on sequence units, until the list of sequences to be aligned is exhausted. However this method has its short comings. An example being that if there are misalignment in the beginning phases, it cannot be corrected in the later parts. This progressive method of alignment follows the ‘once a gap, always a gap principle’ \([11]\).

In the example below we would examine two sample sequences and go through steps necessary to align them. This is to give a simple overview of alignment.

Take a simple sequence say S1: a b c d e f which we want to align with another sequence S2: a b d g f. We align this by writing the latter before the former to get:
CHAPTER 2. FOUNDATIONS AND LITERATURE REVIEW

The next step would be to move the sequences to be able to have the maximum match between the both of them and the link characters that have a match with a vertical bar.

We can see that there still exist the possibility for further alignment between characters of the sequences. To take advantage of this and increase the alignment. For this we would have to insert dashes in the form of spaces or gaps in the below sequence to help the other characters align.

As shown:

Since alignment we need to maximize base to base matches and if needed then insert gaps between any of the sequences. However, gap-to-gap matches are invalid or not allowed. The base order of the sequence should be preserved also.

That being said, the question arises, how do we evaluate the alignment of these sequences? How do we determine one alignment is better than the other or is the best alignment as there are so many possible alignment that could be generated? A scheme to score the alignments is therefore necessary and this scheme is based on giving a score for each possible character placement a system called substitution score and also a penalty for gaps. The summation of this substitution score and the penalty for the gap is referred to as the alignment score and is a direct reflection on the quality of the algorithm.

In the below example, we analyze the scoring scheme and see how the substitution score and gap penalty are computed. For a DNA sequence, we would construct a substitution matrix which
assigns a positive figure (+1) for every match and a negative (-1) as a penalty for every mismatch. This substitution matrix should ignore gaps.

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>T</th>
<th>A</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>+1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>T</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>A</td>
<td>-1</td>
<td>+1</td>
<td>+1</td>
<td>-1</td>
</tr>
<tr>
<td>G</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>

Bearing in mind that the gap penalties as previously stated is 0, we evaluate the above to yield:

\[
\begin{align*}
S1: & \quad A \ T \ G \ G \ C \ G \quad = \quad \text{Query Sequence} \\
S2: & \quad A \ T \ G \ - \ A \ G \quad = \quad \text{Score:} \ +1 +1 +1 +0 -1 +1 = 3 \\
S3: & \quad A \ - \ T \ G \ A \ G \quad = \quad \text{Score:} \ +1 +0 -1 +1 +1 +1 = 1 \\
\end{align*}
\]

In the above, it can be seen that the sequence S2 has a greater score and as such would be the better sequence when aligned. However DNA sequences are much less complicated than protein sequence scoring, and for protein sequences we would need to use the point accepted mutation (PAM) or the Blocks substitution matrix (BLOSUM) instead of the substitution matrix to score sequences.

Once the scoring scheme is determined, it is possible to evaluate the alignment either by taking a score of all possible alignment from the sequences and then selecting the one with the best score or by the Needleman-Wunsch algorithm which uses the trace back matrix and score matrix functions to operate.
CHAPTER 2. FOUNDATIONS AND LITERATURE REVIEW

2.3 Hash Trees

In computing, a tree can be described as a data structure that has a set of linked nodes and emulates a hierarchical tree structure with node parents and children. Nodes that do not have children are referred to as terminal nodes, and the top-most node is known as the root node and as such has no parents. Nodes that have children are known as inner nodes [7].

Hash trees are data structures where leaves are hashes of the individual keys and parent nodes higher in the tree are hashes of their respective children. They are data structures that can be used to manage, verify or protect any kind of data transferred in and between computer systems or networks. For instance in the below, H1 is a result of hashing H0,0 and H0,1

\[ H1 = \{H0,0 + H0,1\} \]

Hash trees are primarily used to verify authenticity of data blocks [7]. They were invented by Ralph Merkle [7] in 1979 to handle lamport one-time signatures. They are trees which possess a summary of information about a large piece of data and provides a means for its verification. Hash trees are primarily binary trees with specific bit values associated to its constituent node so that the value of each interior node is a direct function of the value of its children [12]. Describable in simple terms as a hash function whose leaves are hashes of data blocks in a file with a hash function being a mathematical function which assigns a smaller integer value to another larger block of data [12] so that it performs the function of an index, pointing to the larger block of data. In a hash tree, the tree summary information is a data structure that simply contains information about a larger block of data and can be used to verify the larger data block [13]. It should be noted however that hash trees ideally take on binary representations which means that there are usually only two child nodes under each node. However this is not conclusive and as such there can be more than two child nodes for each node. Hash trees are an optimal method for maintaining and verifying data as data comparison and checks can be quickly done to examine parts of the file as having a valid top hash ensures or proves that the lower part of the tree is valid. Top leaf nodes are concatenations of their children [14]. At the top of a hash tree is the master or root hash. Hash trees usually are implemented by SHA-1, SHA-2 or SHA-3 algorithms, which are cryptographic based functions. However they can also be implemented by check-sums, this however is not a secure means of hashing.
In the figure 2.4 above, the validity of the data block can be easily deduced from the top hash or root node. To verify the validity of a data block when the whole tree is available, say the root hash, you only need hash 0 and hash 1. First you combine hash 0.0 with hash 0.1 in order to compute hash 0, and then combine hash 0 and hash 1 to get the root or top hash. When the Top or root hash is available and confirmed, the rest of the structure can be verified accurately, the same for each top node.

In Network applications, hash trees can be applied and used to ensure network traffic integrity for instance a hash tree can be constructed with packet data as hashes and signed with a cryptographic signature from the hash of the tree. This packet can then be sent together with the root signature and the necessary amount of tree hashes needed to get the root has reconstructed. The recipient of this packet can then verify using the root signature and reconstruct the hash tree [15]. Using hash trees would also help to cut down on the number of cryptographic functions needed for the verification of the integrity of the packet helping to reduce the amount of computation needed. Hash trees can also be used in network applications such as P2P networks for file distribution. For instance, when a peer wants to download a file, a trusted source is first queried to obtain the top hash of a needed file. Then the rest of the hash tree can be obtained from any other peer in the network, trusted or not. This received hash is then checked against the initial trusted top hash. This is used to verify the hash tree. If fake, invalid or damaged, it is easily identified. The program then queries another source, until a source is found who’s values match this top hash.

Data values in hash trees are verified based on publishing the hash value of the top hash after authenticating it with a digital signature. To verify the authenticity of data values, the sender simply provides to the receiver the data value and the path from the root to this value [16].

![Figure 2.4: Figure of a Hash Tree](image)
2.4 Bloom Filters

A bloom filter can be described as a simple randomized data structure for testing membership in a set. They are data structures optimized for quick and easy set membership queries [17]. They can be denoted as a set of S elements where \( D = \{a_1, a_2...a_S\} \) and serves as a means to check if a value is contained in a collection or not. Their usage and application extend into many areas including databases, security and social networks.

Bloom Filters were invented by Burton Bloom in 1970 [18] and was a data structure designed primarily for spell checking purposes. They are highly space efficient, used to represent a set and are able to perform set membership queries (i.e.: Queries that ask: “Does set X contain element Y?”) [19]. Bloom filters are error prone and may in some situations return incorrect answers to queries a situation known as false positives [20] which is mostly as a result of its space efficient trade-off. However its space saving advantages are usually more advantageous and out-weigh its drawbacks.

Bloom filters are beneficial for their time saving characteristics and can be easily run on a browser and used to check if an item does not belong to a collection, avoiding database access time and probably save bandwidth also. Bloom filters utilize bit array functionality and require hash functions. Its efficiency therefore is directly related to the number and quality of the hash function used as well as the number of bits in the array.

Bloom filters provide a system that offers a highly efficient storage using 10 bits or less per element in a given set regardless of the set size or number of elements contained in the set, with a 1% false positive probability [21]. A False positive is a situation that arises when the filter indicates that the element is a member of a set, when in reality it does not belong to that set or exist in it. False negatives means reporting an inserted element to be absent from the set. The main properties of a bloom filter include:

- It is impossible to get a false negative
- False positives do occur, however there can be controlled and their frequency is determined by the space/time efficiency system.
- The time required to verify if an element belongs to a given set does not depend on the number of elements contained in the given set.
- Bloom filters take up a very little amount of space as compared to the data contained in the set.
- Bloom filters can be halved which allows for shrinking of contained data [22].

The efficiency of a bloom filter can be determined by considering the amount of bits needed to represent all sets in a manner that allows for false positives at a fraction of the universal set, but no
false negatives [19].

2.4.1 Working methodology

Bloom filters only allow for the insertion of elements and did not previously support deletion of elements from the set. The accuracy of a bloom filter is usually a trade-off and depends on certain factors which include; the number of elements contained in the set (with large number of elements increasing the chances of having a false positive), the filter size and the hash functions used in the filter. These three factors influence the accuracy for the filter greatly [23].

Bloom filters generally supported only two kinds of operations:

- **Test operation:** This is used to test if an element is a member of a set or not. It has two output options
  - True: This means that the element ‘maybe’ contained in the set (taking into consideration the false positive factor).
  - False: This means that the element is certainly not a member of the set (Bloom filters are not affected by false negatives).

- **Add operation:** This is used to add elements to the set. Bloom filters did not originally allow for the deletion of elements from set. However in recent times, extension have been developed that allow for deletion of elements in Bloom filter sets e.g.: Counting filters [21].

Generally Bloom filter are composed of two elements; a bit vector with given length and a set of hash function say K. The length of the bit vector and the number of hash function is chosen based on the number of keys we require in the set and the allowable number of errors. Hash function ranges are computed to match the bit vector length. For example, given a vector that has is 480 bits long, the hash function should return a value that is in the range from 1-480. This is vital to distribute the output over all the possible range. To store a key in a bloom filter, we iterate it through the hash functions say K and treat the result as a bit vector, turning on whatever bit was previously at that position. E.g.: If it was a 0 or off, it is turned to a 1 or on. If it was a 1 or on, we leave it that way as bloom filters cannot be turned off because of false negative issues. For an illustration, consider a bloom filter with 4 hash function and a bit vector length of 16. In the figure below, an empty bloom filter is initialized with all the bits turned to 0.

![Figure 2.5: An empty Bloom filter](image-url)
Assume spaces to be the OFF or 0 state and X to be ON or 1. To insert elements to the bloom filter, take for example the word “computers”, we proceed to hash this string through the 4 hash function and aggregate the result.

Hash 1 (“Computer”) = 4  
Hash 2 (“Computer”) = 6  
Hash 3 (“Computer”) = 15  
Hash 4 (“Computer”) = 9  

These correspond to the appropriate location on the bloom filter with range till 16, we then turn this on accordingly as shown in the figure below:

![Figure 2.6: A Bloom filter with 4 Bits enabled](image)

To insert another key say “Routers”, we repeat this process

Hash 1 (“Routers”) = 4  
Hash 2 (“Routers”) = 6  
Hash 3 (“Routers”) = 16  
Hash 4 (“Routers”) = 3  

Assigning this to the appropriate bits, we get the image represented in the figure:

![Figure 2.7: Bloom filter with the 2 keys embedded](image)

From the above diagram we observe that some keys overlap and as such are stored at the same location. While this is efficient as it makes for greater space utilization, it makes it difficult to delete a key, as there is no guarantee of which bits belong to the key being removed. To check if a key is stored in the filter, we perform a similar procedure to the one above, iterating the key through the set of hash function. A check is then made to determine if the bits is turned on which would signify that there is a probability that the key may be in the filter. However if it is off, we are certain that the key is not contained in the filter as bloom filters do not allow for false negatives.
Taking a look at an example scenario in which Bloom filters might be used in practice; assume an application receives high input data and needs to perform a check to verify if the data is already contained in a data set on the database to avoid re-processing it. A bloom filters can be utilized in this case to quickly determine if the file is contained in the database as it takes less space, is fast and efficient and can determine if the said data is already contained in the data set.

For this thesis project, a bloom filter would be constructed using a cryptographic hash function. Cryptographic hash functions such as the MD5 algorithm is well distributed and spreads its output uniformly across any given hash space [24], unlike other hash function which are mostly concerned with only speed. Using a Bloom filter in this project implementation eliminates the need to access the database for every query. The Bloom filter can accurately determine if the query item is not available locally and as such would lead to a more efficient system.

2.4.2 Extensions and variations of bloom filters

Since Bloom filter were created by B. Bloom in the 1970’s several modifications and extensions have been developed. Below are a couple of them:

- Counting Bloom filters: This is a variant of the standard bloom filter developed by Fan et al [25] to tackle the issues of changing set members in a set of bloom filters. It was designed to be able to allow for deletion of items in a set as the original standard bloom filter did not allow for this. Counting bloom filters makes use of a counter which replaces the bit array. The counter is incremented or decremented on insertion of elements or deletion of elements. The counter value gives the number of elements it has hashed to it. However counter overflows do occur and this happens at a value of $2^n$ [26].

- Scalable Bloom filters: These are filters that are made up of two or more standard bloom filters, primarily designed to allow for expansion. New filters are added when one gets filled allowing for arbitrary growth [26]. Scalable bloom filters have the ability to dynamically adjust to the number of elements required to be stored while still maintaining a maximum false positive rate which can be pre-chosen at initialization [27].

- Compressed Bloom filters: This is another variety of bloom filters introduced by Mitzenmacher [28]. It serves to improve the performance of bloom filters when they are transmitted as messages. Compressed bloom filters allow for compression of much larger bitmaps into smaller messages for transmission and thus can achieve a small false positive rate over uncompressed bloom filters.
2.4.3 History and application

Bloom filters were originally introduced by B. Bloom in conjunction with an application to a hyphenation program \cite{29}. Bloom suggested a bloom filter to maintain a dictionary of words requiring look ups as certain percentage of words could not be hyphenated appropriately and needed a table look up. Bloom filters were also used for spell checking purposes on early UNIX systems \cite{50}. In this implementation, dictionary implementations were stored in Bloom filters instead of storing the dictionary itself \cite{19}. This was a means used to determine if a word was a valid word in its language by creating a bloom filter of all words in specific language and checking the filter for any query or word. This space saving advantages were very vital back then because of the low memory systems of the UNIX era and even till date. Another application for it was in the databases to speed up semi-join operations.

In recent times, Bloom filters are widely being uses in a number of applications ranging from network applications to security and database management to text and word analysis. Even Google chrome up until recent times utilized bloom filter technology to identify malicious URLs \footnote{malicious URLs}.

2.5 Database and Database Index

As the amount of information continues to rise with the recent trends in technology and computing, organizations and individuals are beginning to recognize the value and impact of data as it affects our daily lives. Ways to manage this enormous amounts of data and retrieve useful information are paramount. To take advantage of the benefits offered by this vast data and get the most out of data sets, tools have been developed to efficiently process data sets. Database systems evolved from traditional file management systems and is an efficient means to manage data. In the below sections we take a look at the integral parts of a database system and an in-depth review of database indexes which are data structures designed to improve retrieval of data from data sets, making for efficient and fast access to indexed items.

2.5.1 Data and Databases

In practically all fields of life, which includes education, law, culture, business and science some form of data is needed to enable the proper functioning of activities. In science, data and databases are very important and play a significant role in the advancement of science. Technical data, scientific data and factual domain data are vital in all parts of science and scientific work. The role of data and databases in everyday life cannot be overestimated. In this section, the basics of data, DBMS and database indexes are analyzed. Their evolution, differences as compared to conventional systems and structure are also discussed.
Data is the root of every database system. However data does not always convey useful information, as imagined by some. Useful information is derived from processed data. A database on the other hand can be described as a shared collection of logically related data and a description of this data designed to meet the information needs of an organization.

Robert Robbins described a database as being a persistent, logical and coherent collection of meaningful data that possess relevance to aspects of the real world \[31\]. Databases play an ever increasing role in computers and the human life in general. There are critical in virtually all areas where computers are utilized spanning from medicine to law, engineering and even commerce. They play a vital role in the everyday modern society and affect, impact or influence our lives daily. A database usually consists of tabs which are two dimensional representation of data consisting of columns and rows. It can be in the simplest of terms be described as a data repository designed for the efficient storage, maintenance and retrieval of data. With data being composed of facts and figures that have a clear and precise meaning.

Databases have different forms in which they can store data. They may be hierarchical, graphical or even tabular. When data is stored in tabular form which consists of rows and columns, it is known as a relational database. Databases may also be hierarchical; assuming a tree structure or graphical; pointing or highlighting relationships or connections between objects and called a network or graph database \[32\]. A database is generally managed by a database management system which is a software system that enables users to create, define and maintain the database and which provides controlled access to this database.

Databases consist of inter-related tables in which each table represents objects as records and these records contain a number of fields which are data items specifying the characteristics of the records. Database systems do not only provide storage processing and retrieval functionality for the data they contain, but also provide extra information such as their definition and description. This description is known as meta-data and is simply data about other data. Data on the other hand

![Diagram of database types](image)

**Figure 2.8:** Diagram of database types
is a representation of figures, facts or information from concepts which are created in computer readable form or translated into such a form [33]. Meta-data helps provide a self-describing nature to database systems. This helps for data independence, which implies that if new structures or already existing structures are added or modified in the database, the programs that utilize this data are not affected by this change as long as they don’t depend directly on the modified data [34].

2.5.2 Database Indexes

An index is a data structure that facilitates the efficient answering to a set of queries. Searches and queries performed by applications in a data set can be time and resource consuming. Indexes are a means to circumvent this problem and improve query performance [35]. Indexes can also be utilized in answering certain queries without having access to the data file [36].

An index stores the values for a specific column in a table [37]. Indexing simply means highlighting a particular column within a particular database table to be stored for use later [38]. It is a pointer to a data in a table, and allows for efficient and fast access to stored data. Indexes are usually comprised of a search key which points to a specific data block.

![Diagram of an index file](Figure 2.9: Diagram of an index file)

Index files are usually much smaller than the original file and database indexes occupy only a minute part of memory as they are only composed of key fields and not the rest of the data in the table. Indexing is an ideal method to reduce the number of logical input/output operations and helps to provide a structured search mechanism instead of row by row search [39].

Indexing methods can generally be classified into two, they are: primary and secondary indexes or clustered and non-clustered index respectively:

- Primary/ Clustering Indexes: This is an indexing system where index entries are sequential in an ordered file. The index whose search key specifies the sequential order of the file. It is an index that has a physical ordering of the datable data [40]. Here the search key of a primary index is usually but not necessarily the primary key.

- Secondary/ Non-clustering indexes: This is in indexing type where the sequential order of the record has an order different from that of the search key. They begin from a root index node and include intermediate index needs and leaf nodes [40]. In a non-clustering index,
the data page at the leaf level only have pointers which connect to more data pages that contain the actual data being searched [41].

Clustering indexes are efficient for multi-point queries where there exist equal access to non-unique fields [41]. For instance, assuming a look-up is to be performed on names in a paper phone-book, a clustering index would be useful because the pages of the phone-book contains the same last names on consecutive pages. This is in contrast to non-clustering index which perhaps aims to find the first four digits of telephone subscribers in the phone book. For example, a query to identify all the subscribers whose number starts with 1756 might require a look-up on all the pages of the phone book.

The main function of an index is to improve the performance of the system in terms of data retrieval and should be able to allow for creation and deletion without affecting the data. However the performance of data retrieval might be affected when indexes are deleted, and it should be remembered that using or creating an index where it is not needed might also affect system performance negatively.

When a database index is created, it stores the physical/logical location of data values in a table which is associated with the column against which it is indexed. When new data is added to the table, new entries are created. In situations when a database does not utilize an index, it performs what is basically called a full table scan. This is a process whereby the whole database table is scanned until the query result is found.

Creating an index of a data table does not alter the table data it simply creates a data structure which points or refers to a database table. In a real life example, a database index can be likened to the index found at the end of a book which is used to point to specific information at various points in the book. It takes up some space and increases the volume size of the book, but can also be beneficial in locating items quickly as all entries are well-defined and arranged orderly. A database index however is more practical than a book index as it allows for updating index entries or even deleting them [42]. Database indexes however do not provide advantages all the time. For instance, having an index on all field entries in a database would consume too much memory and reduce the efficiency of the system. It would also require a lot of maintenance to keep this accurate and update it when needed.

Indexes can also be classified on the properties that influence the effectiveness of searches. Using this criteria, we identify two types of indexes:

- Ordered Indexes: This is an indexing type in which records of files are sorted in a specify order and stored in physical memory location. The values contained in ordered indexes are sorted and ordered to that it is easy to perform binary searches on the index.

- Hashed Indexes: This an another indexing type that does not require indexes to be ordered. It makes use of a hashing function to avoid accessing index structures. A hash index is made
up of two fields, with the first field contained the value of the search key attributes and the second field composed of the pointer to the hash file structure.

Indexes are very important to database sets and makes for needed speedup when data retrieval is important. However indexes take up additional storage space and must be updated each time there is an addition to an existing table, posing an additional overload. There are different types and implementations of indexes, which include Bitmap index, sparse index and dense index. Below we examine them in detail.

- **B+ Trees Indexes:** A B+ tree index is basically a balanced tree where data structures are contained in the leaf nodes and internal nodes direct searches. B+ tree indexes have equal path length from the root of the trees to the leaves and have a fixed number of children for any given tree. The tree structure in B+ trees grow and shrink dynamically and references to data are made from the leaves alone. The leaves are linked sequentially which results in a sequence set that displays results in an ascending order when the list of leaves are scanned. Leaves of B+ trees have different structures than that of the other node and each node in sequence contains a minimum of $n/2$ elements and a maximum number of $n$ elements [43]. B+ tree is a widely used data structure and it maintains its structure regardless of insertion or deletion of data. The leaf nodes contain data entries while the internal nodes direct the search queries. Searching for a record on a B+ tree entails a traversal from the root down to the appropriate node and each node contains at least two sub-trees unless it is a leaf. All the leaves however are on the same level. Searches are faster in B+ trees and the performance level for direct access is good. However performance for range and sequential searches are excellent. B+ trees do have some disadvantages, such as insertions being more complex and the possibility to have duplicate search keys which could result in high memory consumption.

- **Bitmap Index:** A bitmap index is an indexing type that maintains a bitmap for every key value. Bitmap index utilizes bitmaps to indicate the column value being indexed. Bitmap indexes are supported by only some DBMS packages and as such might solve a limited number of problems. It however does provide excellent performance. Bitmap indexes are mostly relevant for heavy tables which are infrequently modified and handle lots of queries [41]. Bitmap indexes are advantageous in cases where a small number of distinct values are in columns that should be indexed such as sex or Boolean values. Bitmap indexes are popular data structures used in the speed up of retrievals of matching rows in tables as they posses performance and space advantages over other data structures that could be used of the purpose. They handle queries by conducting bit-wise logical operations on bitmaps, are simple and can be implemented using bits or strings of ones and zeros. Here a separate string of bits or ones and zeros are stored for each key value. To elaborate this we consider an example using a bitmap index defined for the column ‘sex’. Assume a bitmap index created on an employee table against the column sex which contains nine rows. Assume that the
bitmap index for the sex column has three strings; one for male, one for female and the last used to represent unknown, with each of them containing nine bits. The string therefore in this case would be positional and have a bit turned on or set to in the sex column in that row to signify the contained value of that particular string. Examine the below bitmaps:

<table>
<thead>
<tr>
<th></th>
<th>00110010</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNKOWN</td>
<td>11000010</td>
</tr>
<tr>
<td>MALE</td>
<td>00001100</td>
</tr>
<tr>
<td>FEMALE</td>
<td></td>
</tr>
</tbody>
</table>

The above strings indicate that rows 3, 4 and 7 are unknown. 1, 2 and 8 are male and 5, 6 and 9 are female.

- Sparse Index: A sparse index can be described as an index which is created for few values where each of the indexes contains a value and has a pointer to the very first record that contains that value. Sparse indexes are special types of partial indexes proposed by Stonebraker [44]. It is a data structure which contains the first record entry of each block of a data file and has a pointer to the location on disk of that respective block. To perform look-ups in a sparse index, a search is performed for a value which is equal or less than the value contained in the index which we are looking. Once the first record is gotten, a linear search can then be performed to locate the desired record as sparse indexes allow for prefix look-ups [45]. Sparse indexes can be advantageous in that they save memory space especially in large files because of their small index table size. However they result in slower searches as two searches have to be performed in total. The index table is searched firstly then another linear search is conducted inside the secondary memory.

- Dense indexes: A dense index is an index that contains an index entry on all of its search key values and therefore all the records in the data file. Records that have the exact value of search key attributes are sequentially stored after the initial record. In other words, the data entry order in the index is different from the order of data records. Dense indexes make for efficient and fast look-ups and can be effectively used for small and medium files [46]. However they take up larger amounts of space as compared to others as the index table is large and therefore not so efficient for large files. Deletion and insertions are also difficult to handle.

However, not all of these data technologies mentioned above were implemented and used in the final version of the proposed system. The reason being that some of them have been inefficient and would not help in the achievement of the overall aim of this thesis. For instance, in the course of development it was discovered that hash trees being a reliable means to verify authenticity of data or identify if a particular file does indeed stem from another, it would be inefficient when a new sequence file is to be compared to previously computer files. This is because while hash trees could be used to verify if one block of data originates from another, it fails when it is used to determine
percentage similarity between files. Also, database indexes while providing a structured method with a fast and efficient way to identify data stored in a database thereby improving performance, does not provide the needed functionality for the accurate file comparison.
Chapter 3

Application of Functions to File Comparison

The proposed algorithm in this thesis is made after investigating different database structure implementations which were analyzed and their applicability to the comparison of MSA files determined. A choice, which would be effective for the proposed system was then made and the methods used to achieve this is outlined below.

The following sections of this thesis examines the application of appropriate data technologies to achieve the primary aim of this thesis. A system to compare MSA files is devised and serves to identify similar files so as to avoid re-computation. For this, a methodology that combines Bloom filters, hash functions and hamming distance is utilized. This methodology is drawn from the inference that is more efficient to check if the query item is not contained in the database to avoid iterating though the database when the file may not even exist in the database. Hash functions are also applied to track and verify that the file searches are 100% accurate. This is also important and runs in parallel to Bloom filter technology so as to reduce the impact of false positive hits on the system.

In our implementation, the strings are passed through a Bloom filter, this acts like a verifier and only proceeds to the next step to check for 100% match accuracy if it is determined that the file ‘maybe’ contained in the set. These strings are then passed to the hash engine which performs a hash comparison and ensures a 100% match accuracy. In the event there is no 100% match, the hamming distance is used to identify files that have the percentage similarity determined by the user. The choice of the methodologies used to achieve the end result of this project is greatly influenced by certain factors each of which is peculiar to the individual technologies used. For bloom filters it is important to have a system that can easily and efficiently determine if a file exist in the database without having to consult the database directly. This makes for a faster execution of queries albeit in a probabilistic environment. This shortfall is however addressed by the hash value comparison in the event that there is a database hit. The MD5 hash algorithm being utilized in this case as it provides a quick and easy way to verify if files are indeed the same. Although
the MD5 algorithm has a higher collision risk than some other cryptographic functions, it still provides the needed functionality for this project and does not compromise the accuracy of our results in any way as it is still very suitable for basic file verification purposes even though it is not recommended for encryption and security purposes. A simple comparison of individual hash values of files identifies if they are similar.

File comparison using the above method only serves to identify exact (100%) files. However in this thesis project, the acceptable percentage to classify files as being the same has been set at 90% match accuracy. Therefore other methods to achieve this threshold are investigated. This gives rise to a string matching inquisition, in which we intend to match individual characters in strings of a file hoping to discover files with at least 90% string similarity. For this, two methods are considered; the levenshtein distance and the hamming distance. Both being an effective means to obtain the sequence similarity. The hamming distance is preferred in this project as it considers only strings of equal length which is adequate as we deal only deal with strings of the same length. In the below section we consider a literature on these implementations.

### 3.1 Input Format

The input files for this project is in the FASTA format, which is a format used to represent DNA and protein sequences. A FASTA file is a text based file which contains one or more sequences represented with single letter codes [47]. Every sequence in the FASTA format begins with a description which is a single line containing the sequence name. This identifier description is usually preceded by a ‘greater than’ (>) symbol in the first column and sequence data then follows on the next line terminating at the beginning of another ‘>’ character.

```text
>SequenceA  Name/description( this can be optional)
Text lines of Sequences (CCATCTCGAGGA)
>SequenceB  Name/description( this can be optional)
.
.
>SequenceC  Name/description( this can be optional)
Text lines of Sequences(AGTCTTNNFLIRPE)
```

Figure 3.1: Fasta Format

Text lines in FASTA files are usually recommended to be shorter than 80 characters in length and spaces are not allowed in sequences. These text lines are composed of plain text characters of the sequence. Protein sequences usually have the amino acid codes(IUPAC) and DNA sequences have
single letter nucleotide codes with both of them exempt of gaps and numeric digits. Sequence lines can either be in upper or lower case character [47].

3.1.1 Fasta Method

The Fasta method uses a methodology referred to as hash coding which involves constructing a look-up table which contains details on the positions of individual sequence words of K-length known as the K-tuple which is constructed for all sequences. The positions of characters in different sequences is thereby computed by finding the difference in the positions between the first and the second in that order. Alignments are guaranteed in offset positions in sequences. The first step in the Fasta method algorithm involves the identification of all the exact possible K-tuples (each of K length or greater) represented by diagonals in a dot plot set. The alignments of these diagonals are then re-scored using a similarity scoring method; only tuple regions above the set threshold are retained with all the gaps excluded. The third step involves rejoining all the tuples regions, these are the longest diagonals. Lastly, sequences with highest scores are aligned to initial regions.

3.2 Hash Functions

A hash function according to Hossan et al can be described as a set of tools used to verify the integrity of messages, create and append digital signatures and time stamps [48]. They are functions that accept arbitrary string length as input and compress this arbitrary string to another string of fixed length. They are mostly used to operate on digital data to produce a fingerprint of any given block of data and thereby help ensure its security. In hash functions, little modification to the input data provide extreme variations in the output data and as such can be utilized to verify or check if a block of data has been altered or tampered with.

![Diagram of Hash Function](image)

Figure 3.2: Hash Sequence

Hash functions are used to produce a digital signature from any given data object and are mostly
used in security applications including cryptography, data look-ups and retrieval systems. Hash functions have two key characteristics; ease of computation and compression.

This means that a hash function should be able to take arbitrary bit-length string and compress this easily to a fixed bit length string. For hash functions to be categorized as being strong, it should have some characteristics which include:

- It should be extremely difficult to have different blocks of data or messages that have the same hash values and thereby should be collision resistant.

- The hash value should not yield the original data block if reverse engineered. It should not be feasible to retrieve the data block if only the hash is known.

- It must possess an attribute known as the avalanche effect which means that a very small change or modification in the input data should result in a large clearly noticeable difference in the output.

These above characteristics are fundamental to a good hashing function and helps provide integrity to any given block of data that is to be hashed.

![Figure 3.3: Hash Function](image)

### 3.2.1 The message digest hash algorithm

A message digest algorithm is a type of cryptographic hash function. Message digest outputs are usually made up of short fixed length values referred to as hashes. They are usually 32-bit hexadecimal numbers and are the result of passing a piece of data, message or file through a one-way hashing formula. Message digest hash algorithms are one-way hash algorithms and this simply means that it is not possible to use a hash value produced from hashing any given string or data to retrieve the original string or data. It is only used to verify the integrity of the string.

The MD5 message digest algorithm was designed by Prof. R Rivest in 1991. It was developed as a replacement of the MD4 algorithm. It has the following differences when compared to the MD4 algorithm.

- 64 steps and 4 rounds
- Each step combines results of previous step
3.2. HASH FUNCTIONS

- Round functions are significantly less symmetric than MD4
- Optimization of shift amount done in each round.

MD5 algorithms as a type of hash function accepts an input message of arbitrary length and outputs 128 bit message digest of the input called its fingerprint. These fingerprints are digital identifiers of each digital document that is hashed and the MD5 algorithm is designed to produce a unique digital fingerprint for each unique document. Very little change of even a bit in the input document results in a large change in its fingerprint but similar documents always have the same fingerprint. The MD5 algorithm however has a weaknesses in that because its fingerprints are much smaller than the original input data, there can sometimes exist collisions for different data sets and this is as a direct result of the MD5 compression ability which is made in Davies-Meyer mode. MD5 was created to be used in different applications including integrity checking, security (SSL or digital signatures, though it has been proven to be ineffective for this).

In the MD5 algorithm, every small change in the input produces a very visible difference in output fingerprints. This is known as the avalanche effect (for obvious reasons).

For example:
MD5("Checking this string")
= bb5b9725ef69406c0ac14ba1369c80d6

and

MD5("Checking this string.")
= 93c50cdf34b1386b56ef1e61e9643348

In the above, it is very visible the effect a single punctuation mark has on the output hash.

MD5 algorithms are still widely used for file integrity and verification purposes even though they have been proven to be ineffective for security purposes. They still generate high levels of interest because of their platform independent nature, with a large number of platforms providing in-built support for MD5 check-sum generation. They are also fast as they were designed to be quick on 32-bit systems. The MD5 algorithm accepts input messages and generates 128-bit output hashes in five steps described by the Internet RFC 1321. These steps are outlined below [50]:

- Append padding bits: padding bits are appended to the message to make the bit-length equal to 448 mod 512. This padding step is always done even if the message length is congruent to 448 mod 512. This brings the total bit-length of the padded message to 512X for a positive integer X.

- Append length: A 64bit representation of the message length before it was padded or added to the result of the first step (append padding bits). The resulting message from this addition
would have a length that is a multiple of 512 bits, having a length that is a multiple of 16 (32-bit) words.

- Buffer Initialization: For the buffer initialization, a buffer consisting of four words (A, B, C, D) where each of A, B, C, D is a 32-bit register is used to compute the message digest.

- Process message in 16-word blocks: This step is concerned with defining four auxiliary functions that each take three 32-bit words as input and output a single 32-bit word in elaborate steps using a 64-element table constructed from a Sine function.

- Output: The output produced is a message digest beginning with a low-order byte of A and ending with a high-order byte of D in the for A, B, C, D.

It should be carefully noted that while the MD5 hash algorithm is not ideal for security and cryptographic purposes, it is still a reliable method to verify file integrity or to detect changes in files and as such can be used to achieve the intended aim of this thesis project. Comparing a hash value of an MSA file against pre-computed hashes stored in a database could be used to check if the new file has been computed before efficiently and effectively. This is done by computing the hash value of any given MSA file or the top line of the MSA file and checking if an identical match exists in the database.

In the development of this thesis, an implementation of hash functions is utilized specifically, the MD5 hash algorithm mentioned above. The MD5 algorithm in our program is used to generate a fingerprint of the first line of each sequence file stored in the database. The hash algorithm accepts MSA characters as input sequences and computes the hash values of the set of characters and produces a 128-bit hash value which is then stored in its appropriate column in the database. Hash values of new files which are uploaded to the system are compared against the saved values in the hash column. Using the hash values, little variations in the input characters would produce a very noticeable change in the output digest. This is important for comparing files in order to verify if there are any 100% matches among files. Files with the same characters when hashed would always have the same hash value, and as such becomes easily identifiable and therefore provides an easy method for exact match verification.

### 3.3 String Similarity and Hamming Distance

Nearest Neighbour and similarity query issues frequently occur in databases applications in so many different contexts. Similarity search is concerned with identifying the closest in relation or most similar item or object amongst a collection of items using certain attributes common to all of them. The main aim being to obtain data that fulfils the similarity level requirements in the database. In this thesis project, a similarity search to identify similar files to the query file is performed. This is necessary to match files that do not get identified by the 100% match accuracy
filtration performed by the MD5 hash algorithm but may still be very similar to the required results we hope to obtain. This is important as MSA files that have a 90% and above matching ratio can be considered as being the same file. In order to detect file similarity in this paper, the hamming distance similarity algorithm is chosen as it can be properly implemented in databases for similarity search.

Bit strings such as zeros and ones are often used in the representation, transmission and manipulation of information. Binary representation, known as bits for short, is the primary method in which data of all different forms are stored, processed and represented. All letters for instance can be written in their ASCII form consisting of a 0 or a 1. For example, the ASCII form of the letter X is 01111000. Data therefore might consist of millions to billions of bits. However, during the processing, manipulation and transmission of information, errors could occur due to numerous factors. It is therefore of paramount importance to have a method to address and handle these errors that affect data or in the event of data loss, a method to correct them. Hence the need for coding theory and error correcting codes. Coding theory is the study of codes, their properties and the design of error correcting codes [51]. Coding theory is a sub-child of a much larger subject domain known as information theory.

In the late 1940’s, Richard Hamming came up with methods of correcting these errors. He developed the hamming codes and extended hamming codes to detect and correct errors [52] while working at the Bell laboratories. His invention was as a result of the computers back then being able to detect errors, but not being able to correct these detected errors [53]. The Hamming distance in information theory is a measurement of the number of character substitutions required to change one string input into another. It is a measure of the errors needed to convert one string to another. In formal terms, the hamming distance between a pair of given strings $a$ and $b$, which is denoted as $d_{Ham}(a, b)$ is the number of times or places where $a$ and $b$ are different. Where $a, b \in X^n$ [54].

For example, the hamming distance between 1011101 and 1111101 is one and the hamming distance between Bloom and Broom is also one because there is only one character difference between the both of them.
CHAPTER 3. APPLICATION OF FUNCTIONS TO FILE COMPARISON

In the above diagram, every two nodes (or words) connected by an edge has a hamming distance of 1 because it takes exactly one substitution to change from one to the other. For nodes separated by 2 bit positions (nodes that are joined by 2 edges), the hamming distance between them would be equal to 2.

Assume two strings $X = x_1, x_2, \ldots, x_n$ and $Y = y_1, y_2, \ldots, y_n$, the hamming distance $d_{Had}$ between them would therefore be the positions where they differ. This could be such that $d_{Had}(x, y)$ is the number of indices $i$ such that $x_i \neq y_i$ [55]. In the event that $X$ and $Y$ are exactly the same, the hamming distance between both of them would be equal to 0.

The basic requirement that has to be met before implementing the hamming distance algorithm is that the input strings have to be of equal length, this requirement is as a result of the ‘penalty’ incurred by the transposition of one string to another. Strings that do not have the same length value would have a much higher transposition value. This is because strings in this sense are sequences of characters generally used in programming languages and implemented as character arrays. When computing the hamming distance between any set of input strings, it is generally recommended to traverse each of the strings character by character marking the character differences. The number of differences would therefore be equal to the number of changes it would take to convert from one to another and can be used to denote their similarity.
In the above figure, the difference between the two strings occur at three positions and it would require three changes to convert one string to the other, therefore the hamming distance between the both if them is three.

The hamming distance is used in this project to determine the string similarity that exist among MSA files stored in the database. The hamming distance provides a method to match the individual characters in the MSA files, compute their similarity and display an output which is a measure of their similarity. In our implementation, the hamming distance algorithm, requires and reads equal length sequences, the first bits of each sequence to be matched is compared and scored by assigning null (0) values to similar characters and positive(1) values to mis-matches. This is done in succession for all the characters in the sequence and then calculated to get the difference between the sequence string.

3.4 Database Organization and Structure

3.4.1 SQL Overview

Structured query language or SQL as it commonly called is a query language containing a set of instruction used to communicate with a database management system. SQL commands are used to run and perform task on a database by translating human generated commands into SQL statements which the database can interpret. SQL is developed to support the transformation of input into output using certain relations. It is comprised of majorly three components.

- Data Manipulation Language: This is comprised mainly of SQL commands that are frequently utilized to perform data manipulation activities such as:
  1. Insert: This is used to insert data into the database.
  2. Update: This is used to update data on the database
  3. Delete: This is used to delete data from the database
  4. Select: This is used to query data in the database
• Data Definition Language: This is comprised of SQL statements that are less frequently used. They are mainly used to define the database structure. They include:

1. Create: Used to create a table on the database
2. Drop: Used to delete tables from the database

• Data Control Language: This is mostly used for granting and revoking access to a database.
Chapter 4

Working Design and Implementation strategy

This chapter describes the basic foundations of a prototype implementation of a system that stores, analyzes and compares MSA files based on the work described in the previous chapters. This therefore means that it highlights significant sections of the system. This implementation was done using a combination of some programming languages.

4.1 General

For the implementation of the previously discussed methodologies, different technologies are involved. All pieces are written in PHP because of its web based capabilities. On the back end, a MySQL database is chosen as the preferred database to store necessary data. Web pages are designed in HTML.

4.1.1 System and Development Tools

Development of any application which runs on the web is mostly structured in a client-server architecture, either in a two-tier or a three-tier layout. Several consideration should be made while choosing a desired architecture taking into consideration the complexity, flexibility and future maintenance of the application. The system developed for the comparison of MSA files in the thesis is a three-tier application which is composed of a web client, a web server and a DBMS Server. With the web browser as the first tier, a dynamic web content platform such as PHP as the second and lastly a database system as shown in figure 4.1 below.
CHAPTER 4. WORKING DESIGN AND IMPLEMENTATION STRATEGY

![Three Tier Architecture Diagram](image)

Figure 4.1: Three Tier Architecture

- **The Web Browser**: The web browser is the first layer in the three-tier structure architecture. It is needed for HTTP request calls, processing the HTTP responses and also processing and outputting HTML resources.

- **The Web Server**: The web server performs the task of connecting the two other tiers. It acts like a bridge linking the DBMS to the web browser, accepting input from the user, processing these as queries to the database and processing the content from the database to the user. It is composed of several components including a scripting language, a web server and an engine for the scripting language. In our application, the web server used is the Apache web server, which an efficient, fast and open-source web server. It is responsible for managing HTTP request and results to the database. PHP is used as the scripting language in this project.

- **The Database Server**: The last tier in the three-tier system is the database server which is responsible for data management. This is comprised of storage and retrieval activities, security as well as handing updates and access. MySQL server is used as the database server for this project.

### 4.1.1.1 PHP

Hypertext Preprocessor, commonly known as PHP is a high level, multi-platform, open-source scripting language used in the development of web applications. Mostly utilized in server-side applications, it can however be implemented in standalone applications or even run from the command line. PHP is supported on most popular operating systems having a simple language syntax which does not need variables to be declared before they are used. PHP codes begins with a `<?` and ends with a `?>` symbol. PHP codes can be imbedded into HTML pages and used to carry out different actions, thereby improving the capabilities of HTML. PHP scripts are used in this project to implement most of the necessary functions, strip data from MSA files and insert them into appropriate locations in the MySQL database, initialize hash algorithm and hamming distance methods.
4.1.1.2 MySQL database

In this thesis, the chosen back-end for data storage is the MySQL database. The MySQL database system is an open-source relational database management system that is designed for the fast and efficient storage, retrieval and manipulation of data which can be stored in different tables as against storing all the information in one large table. In this project implementation MySQL version 5.5 is used. Tables in MySQL may share characteristics and as such can be used to reference each other so as to allow for easy access and maintenance. The MySQL database system is a multi-thread database system which means that is capable of performing many operations at the same time while giving an impression that multiple instances of the application is being executed simultaneously.

For the storage and comparison of MSA files, a database table is needed which would hold and keep track of the various MSA file and their content. For this, it is necessary to create tables which can be loaded with desired information against specific columns. In this case columns would be created to save the file name, length, hash value and content. This is done by executing the create command which takes the form of:

```sql
CREATE TABLE MSA (id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
filename VARCHAR(256),
content TEXT(30),
hashvalue CHAR(32),
content_length INT(11));
```

This creates the necessary tables with the appropriate columns. These columns can then be populated with information from the MSA files by using the insert command, which then inserts the appropriate data into appropriate columns.

4.2 Overview

This section discusses the working methodology used in this project approach. The methodology used is a combination of the concepts discussed earlier in chapters two and three of this thesis project. The aim being to accurately compare MSA files stored in a local database, verify if a match of the file exists and if so accurately reference it. This is necessary so as to avoid re-computation of existing files and save both time, expensive I/O and database operations. As hash functionality is needed for bloom filter technology, we propose a system that utilizes a bloom filter to quickly check if a file has been pre-computed, running it in the main memory so as to avoid disk access time.

This is then combined with hash function technology to accurately match new files to pre-computed
files in our database. The flowchart in figure 4.2 gives a diagrammatic representation of the complete steps involved in this project.

Figure 4.2: Task flow sequence
4.2. OVERVIEW

4.2.1 Sequence Algorithm

These steps are further simplified and outlined by the proposed sequence algorithm below:

Let us assume $D$ to be the set of file which are already computed and needs to be stored in the database. $D_i$ represents the file $i$. Let $id(D_i)$ represent the identifier of the document $i$ which in this case, we use the file name as the identifier. Let $L$ represent the length of the file. $\pi$ represents the message digest of the first 10 characters of the first line of the file.

1. Database Initialization
   a. Scan $D$, and for each $D_i$ extract
      • $id(D_i)$, which is the identifier of the file.
      • $L_i$, length of the file $i$ in bits
      • $D_{i_1}$ where 1 represent the 10 characters of first line of the document $D_i$
   b. For $1 \leq i \leq |D|$
      • Calculate $\pi = \text{md5.hash}(D_{i_1})$ and store in the database.

2. Search
   a. Upload new file $S$ to be compared and match criteria e.g. Exact Match or Similar Match
   b. Extract
      • First characters of the first line, $S_1$ of the document $S$
      • Compute $\pi_s = \text{md5.hash}(S_1)$
   c. Query Sequence:
      • Check if uploaded file exists in bloom filter
      • If Exact Match(100%),
        For each file $i$ in database table,
        If $\pi_s = \pi_i$ then 100% match with file $i$
      • If Similar Match(90-100%), Use hamming distance
        Set $\text{distance} = 0$;
        For $1 \leq j \leq 70$
        If $\text{char}_j(S_1) \neq \text{char}_j(D_{i_1})$, $\text{distance}++$;
where \( \text{char}_j \) represent the character at position \( j \) in first lines of \( S \) and \( Di \)

If \( \text{distance} \geq 90\% \), files \( S \) and \( Di \) are 90\% and above similar

3. Output
   Matched files in the database similar to the file submitted for search.

### 4.3 Query Execution

In order to run a query on a new input file, compare with pre-existing files in the database and output files that match the needed criteria, several steps are required

1. Receive the required query file uploaded by the user and extract from the file the necessary information needed by the system for comparison.
2. Run the necessary tools needed to check if the file already exists in the database.
3. In the event that it does not, collect results and send output to the web service, otherwise run additional comparison steps.
4. Collect information provided at the end of the comparison output to get the final result.
5. Send the final results of this operation to the web service.

### 4.3.1 Receive Query and Extract Parts

The MySQL database functions as the backbone on which our proposed system runs. The database connection parameters and the database table is specified in the configuration script so as to ensure connection to the appropriate database table in MySQL.

File data is stored in the MySQL database and new queries are executed against these records in the database. Database tables are created with necessary columns by using create table commands to build the table and columns. In this prototype, we would create separate columns in the database table for filenames, its content, length and hash values. In order to run the query execution and accurately compare a new query file against a collection of stored records in our local directory, we would need to access, read and extract contents which are necessary for this comparison from files stored in our local directory as simplified in the below flow diagram.
4.3. QUERY EXECUTION

Once the directory has been located and files identified, columns are then populated with file data in order. Files names are stored, the first sequence lines of each file is also extracted and stored alongside their length.

The first sequence line that has been extracted then serves as the input sting of data which is then hashed by the md5 algorithm to produce a specific hash value for each of the files in the directory, which is then saved in the appropriate column.

This saved hash values then serves as a basis on which the 100% match check can be executed. This is done by executing the md5 module which accepts 10 characters of each MSA file already stored in the database as a set of arbitrary strings and produces a fixed-length hash value. The first 10 characters can be used to compute the hash value of an MSA sequence as sequences alignments are usually done using the progressive approach.

The progressive approach is a tree-based alignment strategy, which means that later sequences in a string are formed based on the previously aligned sequence. This guide-based topography strategy means that the bottom set of a sequence alignment follows the top set and as such if there are errors in the top set, there would be errors in the latter parts. This therefore infers that given two sequence files to be compared, if the top set of the both of them are a match, they should both follow the same order and can be said to be the same file. If the top sets are dissimilar, it can be guaranteed that they are not the same file. Therefore, there is no need to hash an entire MSA file. The hash value is generated by invoking the md5 module, and it iterates through the database, computes the hash value by converting this variable character length string into a 128-bit fixed message digest. This generated hash values are saved in another column with respect to their parent file information in the MySQL database. It should be remembered that similarity matching in this project is in two stages, a 100% match and a 90-100% match. For these two criteria, two different methods are utilized.

The hash value is used for the 100% matching and the hamming distance is used to identify files that fall into the 90-100% match. In the first stage of file comparison which involves identifying if the new query file matches any of the files in the database 100%, the hash value for all the files is very necessary as they provide a benchmark against which the hash value of the new file can be compared. For the 90-100% similarity matching, the hamming distance algorithm is utilized. This is a character to character matching of all the characters in the first line of the new query file to the
saved characters stored in the database.

Input: seq1, seq2
var: Integer i, integer distance
begin
    if (length(seq1) != length(seq2)) then
        return;
    else
        distance = 0;
        for i = 0 to length(seq1) - 1 do
            if seq1[i] != seq2[i] then
                distance = distance + 1;
            end if
        end for
    end if
end
return distance;

Listing 4.1: Hamming Distance

A score is assigned for each character to character mismatch between the input sequence and the saved sequences in the database. These scores are then summed up and the total computed as a percentage to get the level of similarity between strings. In the event that this percentage is 90 and above, it can be concluded that the query file has been pre-computed and file information for files that match this new file is displayed. If there is no match that is above 90%, the new file has not been previously computed.

4.3.1.1 Query Sequence

For query execution, new files are upload from the web interface and similarity criteria is specified before the operation begins. Once a new query file is received, the first step would be to run it against a bloom filter to determine if the file, by any chance does not exist in the database. This bloom filter is created by hashing the contents of the directory. To build this bloom filter, the number of items, hash function and a false positive rate are all needed. The bit array is computed based on the number of entries needed. In our case, 200 as our data set contains about 200 items and the false positive rate set at 10%. We therefore, calculate the length of the bit array and the number of hashes.

Computing this where:
4.3. QUERY EXECUTION

| n | =number of items in the filter(200) |
| p | =false positive rate(0.1) |
| m | =number of bits required for the filter |
| k | =number of hash functions required |

We calculate using the formula:

\[ m = -n * \ln(p) / (\ln(2)^2) \]
\[ k = m/n * \ln(2) \]

Therefore we get:

\[ m = -200 * \ln(0.1) / (\ln(2)^2) = 959 \]
\[ k = m/n * \ln(2) = 959/200 * 0.693147 = 3 \]

Therefore a value of M higher than this is ideal for the bloom filter, as such we use the value 1024 in this case for the length of the bit array.

To store elements in this bloom filter, we run the following steps on each of the files in the database.

1. Compute the MD5 hash value for the file. Eg: The MD5 hash value for file 1A0J_A.fas in our database is: E26C9897BB108D836CDA00FDB0E6E3EE
2. Calculate the decimal value for that hash value in this case E26C9897BB108D836CDA00FDB0E6E3EE = 300969390055718751295707608819115156462 which is used for the bit array
3. We then take 3 digits from the hash value, setting bit positions at 300,969,390,055,718,751,295,707,608,819,115,156,462. Therefore at these positions in the bit array, we turn on bit values
4. Repeat procedure for all files.

<table>
<thead>
<tr>
<th>4</th>
<th>5</th>
<th>055</th>
<th>115</th>
<th>156</th>
<th>295</th>
<th>300</th>
<th>969</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

![Figure 4.4: Bloom filter of sample file](image)

In the above diagram, it can be seen that bits are turn on at their positions, signifying that they are contained there. This bloom filter is necessary to save expensive computation time and minimize input/output operations. In the event it has been determined that the file may indeed exist in the database the next step would be to check for file similarity. Therefore, it is necessary to read and strip the file, store the necessary information from the file in respective column in the database to match the structure of the already existing structure. Once this has been done, its hash value is also computed and matched against the hash values of the other files in the database to see if
there is a 100% match between the new file and any of the existing files in the database. If there is a 100% match between the hash values in the database, it signifies that the two files are similar and as such the existing file information in the database is retrieved. Upon determination that there is no 100% matching among the query file and the existing files in the database, further percentage matching is then investigated. This is done by also using a script to invoke the hamming distance. The hamming distance in this case is used to determine the one to one position of the the individual characters in the file, calculating and summing them so as to determine their percentage matching. As mentioned earlier, as per requirement, a 90% percentage match is acceptable among files, Therefore the threshold value for the hamming distance is set at 90%.

4.3.2 Data Aggregation

All the necessary data extracted form the MSA files are stored in the MySQL database after a recursive extraction operation run on the files. The stored data in the database then serves as the base value against which all new queries are compared to.

4.3.3 Obtain Result and Answer Query

Once all the sub-query processes mentioned above have been completed, the final result of the query can be sent to the output, given that at this stage all the necessary information has been extracted, the needed comparison done and the result obtained.

4.4 Application and Results

The practical objective of this thesis project to develop and implement a system that stores and compares MSA files combining it with previously mentioned technologies. The application should provide support for uploading a new MSA file, check against the database if there is an existing match or a similar file and return an output to the user. It therefore must allow for uploads, allow for 100% matching and allow for a given threshold for similar search (in the case 90-100%). Screen-shots of the application are shown below and their functionality explained. To use the system, a user must visit the landing page using a browser. The home page presents an overview of the application and MSA files. It also contains links to the prediction and contact pages. The home page is shown in figure 4.3
The prediction page is the page that allows users upload the file to be compared. It provides comparison options for the user, either exact matches or similar files as shown in figure 4.4.
4.5 Query Example

For a working example of the sequence steps used in the application, we would consider a file 13PK_A.fas which is uploaded by the user for comparison. The following is the process performed on this file to verify it if matches exactly or has a 90% and above match.

- **Step 1** File 13PK_A.fas is stripped, its name, the file length, its first line are extracted.

```
Name: 13PK_A.fas,
Length 70,
First line: EKKSINECDLKGVVLIRVDHVPVNGKITEYRGLPRSLPTLAKVTE
GGSCVLMSHLGRPGIPMAQA
```

- **Step 2** From this first line, the first 10 characters from this first line is extracted and then used to compute the hash value.

```
First 10 characters: EKKSINECD,
Hash value : 358DD93AE66552A3CA82DCDA354E2408
```
4.6 PERFORMANCE

• **Step 3** This hash value is then checked against the hash values stored in the database to identify a 100% match. If there is a match, stored file information is retrieved as explained in section 4.3.

• **Step 4** In the event there is no 100% match, a similarity check is started to identify if there is a 90% and above match. This is done by using the complete first line of the query file and using the hamming distance algorithm to performing a comparison with all the stored first lines in the database to obtain a score. If this score has an above 90% match to any of the files in the database, the query file can be assumed to be similar to that particular file.

• **Step 5** The final output is display to the user

### 4.6 Performance

The performance of the system created was tested with acceptable results, but albeit not in detail. As such there exists the potential for improvements in the performance of the entire system. For instance, it was mentioned earlier that the efficiency of a bloom filter depends primarily on three important factors:

- The number of hash functions used in creating it.
- The quality of the hash functions used
- The number of bits in the array

Changing the number of hash functions would greatly influence the accuracy of the Bloom filter. The false positive rate of a bloom filter is directly influenced by the number of hash functions used to build it. The false positive rate of a bloom filter can be greatly reduced by increasing the number of hash functions. However there is a specific limit to this reduction [56]. Also, the number of hash functions used in a bloom filter affects the computation time as more iterations are needed. This is because to insert an element into a bloom filter, it would need to be passed through N hash functions, setting off the appropriate bits as it does. In simple terms, the higher the amount of hash functions used, the slower the bloom filter. It should however be noted that too few hash functions also lead to a high amount of false positives.

Another factor that could affect the efficiency of the bloom filter is the quality of the hash function used to create it. A hash function that has a uniform layout will make for a lower false positive rate thereby leading to a more effective bloom filter. The reverse is also the case when hash functions which are expensive to compute are used. The speed of the filter is impaired. Using high quality hash functions also help make for a uniformly distributed layout spreading over all possible values.
4.6.1 Experimental Testing

This system was tested on a Linux operating system running on Ubuntu 14.04, running on an Intel Core i5 processor with 2.5GHz and 1GB RAM. The database back-end for this is the MySQL 5.5 database. For this project, the dataset is 192 sequence files in fasta format.

4.6.2 Reliability and Functionality

The application has been tested intensively on our test system. There were no unexpected behaviour or system crashes after series of query execution. The system ran reliably with all functions proving effective.

4.6.3 Responsiveness and Time consumption

To determine the system and time consumption, query execution time was measured and the time taken to execute a simple query which involves uploading a file and comparing it against existing files in the database was recorded at 1.05 seconds. It should however, be noted that this value and performance is influenced by the size of the dataset and the time might vary with a much larger dataset as hash values and hamming distance of more records have to be compared. However with the results we got from the experimental testing, it is safe to say that queries are executed an average of 1.0 seconds for both similar and exact search criteria.
Chapter 5

Conclusion

This chapter summarizes the entire work done in this thesis which has touched many different aspects, technologies and systems. It has even cut across different fields of study and therefore presents opportunities for future extensions of this project work.

5.1 Summary

This thesis defines an approach that could be used for the comparison and storage of MSA files. This approach is composed of a sequence of steps which includes applying appropriate database technologies to determine how best the comparison should work, selecting a storage system for the project and then creating a PHP-web end to link and interact with the system. At first an examination of biological sequences including sequence algorithms and multiple sequence algorithms was made. This was necessary to first obtain an adequate background of the study and highlight the importance of Multiple sequence algorithms to life generally.

Sequence alignments has shown the importance of matching and scoring two or more sequence types so as to infer homology. This is very necessary so as to accurately infer characteristics between different life forms. A brief look at biological database was also made, including their usefulness and types. The latter parts of this thesis is concerned with the methodologies that could be accurately used to compare and store MSA files. For this, appropriate data structure technology was applied. By data structure technology, we simply refer to ways by which data can be utilized more efficiently in a computer system. This involves a set of procedures that handle the creation and manipulation of those structures. The different implementations that were considered include hash trees, database indexes and bloom filters. Out of these, Bloom filters were selected to be used based on its unique characteristics of being fast, efficient and space saving.

Chapter three dealt with a combination of the data structures, explained them a little and presented
an overview of how they are combined to function effectively. The results of this design is an implementation described in the latter chapters. Here we build a web-end to accept input files to be compared. The front end is necessary for user interaction and powers the underlying script necessary for computation. A MySQL database is also selected and used to handle the data back end. This is also essential as it hold information on the files to be compared and as such maintains a structured collection of the necessary data utilized in this project.

5.2 Implications for practice and discussions.

The major aim of this thesis is to develop a system that allows for the comparison of MSA files. This application should store the files locally with emphasis on its ability to identify pre-computed files, so as to avoid recomputation. The application should provide an easy to use web front-end that would allow for user iteration with the chosen database. Finally, appropriate data technologies should be integrated into the application so create a system that provides a fast and efficient means to identify exact or similar files.

The work described in this thesis is an important step in creating a local database system for the easy comparison of MSA files. The thesis has proven the ability to use database technologies to provide an effective means not just to store records in a database but also provide a method to quickly read the contents of a file by stripping a part of it, analyze this stripped part and compare it efficiently against stored database records. This system would be highly beneficial, firstly because of the need to have a local database system for MSA files and most importantly to save and reduce both the time and computation that needs to be performed each time a new MSA file is to be analyzed.

In this project we opted to use the MD5 cryptographic hash function because of its uniformly distributed outputs and also because it is extremely accurate at detecting the 100% match for new strings. However the MD5 algorithm is much slower than other hash functions because they were developed with security/cryptographic purposes in mind and as such have better collision resistant properties than speed. It should be noted that there are research papers that portray the MD5 algorithm as not being an optimal hash function to use on bloom filters as they claim that it the MD5 algorithm has a high false positive rate as compared to some other functions \cite{57} and that the speedup compared to some other functions such as the murmur hash is low \cite{58}. As such future work can be done on this to verify the performance variations between the different hashing methods.

The Database used in this project work is another important criteria that could influence the performance of the system. In this project, MySQL a relational database that is very fast, easy to use and reliable was chosen. Firstly because of its free, open source and LAMP( Linux, Apache, MySQL, Perl, PHP, Python) attributes and also because of its relational nature. Data in MySQL
are organized in a structured format, stored in tables consisting of rows and columns. MySQL offers certain advantages for the particular data-set used in this project when compared to other database systems such as NoSQL database programs for instance. Switching to an alternative database system would obviously have an effect on the overall performance of this project or the methodologies used maybe incompatible and as such the program may not run efficiently.

Lastly, further test should be conducted on the system using a much larger dataset. This would provide more accurate insights on the overall performance of the system and the impact a bloom filter running in main memory has on the overall system. In this current implementation, the overall speed achieved by implementing and using a bloom filter could not be accurately determined as the test set used in this project is small and cannot be effectively extrapolated to reflect the impact the bloom filter has. However, on a much larger dataset this effect would be quite obvious.
Bibliography


sharing protocol,” IEEE/ACM Transactions on Networking (TON), vol. 8, no. 3, pp. 281–293,
2000.


[27] R. Meersman, T. Dillon, and P. Herrero, On the Move to Meaningful Internet Systems:
OTM 2011 Workshops: Confederated International Workshops and Posters, EI2N+NSF
ICE, ICSP+INBAST, ISDE, ORM, OTMA, SWWS+MONET+SeDeS, and VADER 2011,
Hersonissos, Crete, Greece, October 17-21, 2011, Proceedings, ser. Information Systems
http://books.google.de/books?id=VXvFKi16O_QC


http://dx.doi.org/10.1002/spe.4380200607


1zWRhgQc3rYC


[38] D. Waltz, “Is indexing used for retrieval,” in Proc. of the 2 nd Workshop on Case-Based Reasoning,
1989, pp. 41–44.


Appendix A

Database tables

The screen-shots of the tables below present a graphic description of the MySQL database. In the first image, the database structure and necessary columns can be seen. The structure is created using information that would be useful in this project to identify and compare files.

<table>
<thead>
<tr>
<th>#</th>
<th>Column</th>
<th>Type</th>
<th>Collation</th>
<th>Attributes</th>
<th>Null</th>
<th>Default</th>
<th>Extra</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>id</td>
<td>int(11)</td>
<td></td>
<td>No</td>
<td>Yes</td>
<td>NULL</td>
<td>AUTO_INCREMENT</td>
<td>Change</td>
</tr>
<tr>
<td>2</td>
<td>filename</td>
<td>varchar(255)</td>
<td>latin1_swedish_ci</td>
<td>Yes</td>
<td>NULL</td>
<td></td>
<td></td>
<td>Change</td>
</tr>
<tr>
<td>3</td>
<td>content</td>
<td>text</td>
<td>latin1_swedish_ci</td>
<td>Yes</td>
<td>NULL</td>
<td></td>
<td></td>
<td>Change</td>
</tr>
<tr>
<td>4</td>
<td>hashvalue</td>
<td>char(32)</td>
<td>latin1_swedish_ci</td>
<td>Yes</td>
<td>NULL</td>
<td></td>
<td></td>
<td>Change</td>
</tr>
<tr>
<td>5</td>
<td>content_length</td>
<td>int(11)</td>
<td></td>
<td>Yes</td>
<td>NULL</td>
<td></td>
<td></td>
<td>Change</td>
</tr>
<tr>
<td>6</td>
<td>creation_time</td>
<td>timestamp</td>
<td></td>
<td>No</td>
<td>CURRENT_TIMESTAMP</td>
<td></td>
<td></td>
<td>Change</td>
</tr>
</tbody>
</table>

Figure 1: Database Structure

The screen-shot below shows the data table upon initialization of the system, when data has been added. It can be seen that the columns are populated with appropriate information necessary for the proper comparison.
Front End

The Front end for this project is a HTML web-page set that connects to the MySQL back-end and presents a graphical interface which allows for interaction with the system, file uploads and displays results to the end user.

Prediction/Send Query

From this page, a new query can be initialized and run. It provides a ‘browse’ button which serves to upload the file to the system and also a ‘submit’ button which starts the series of activities needed to correctly compare new query files against existing files in the database.
View Query

The view query page presents an overview of the results of previous queries run on the system. It presents an overview of results containing exact matches, in the event that there is an exact match to any of the existing files in the database and if there are no 100% matches but very similar files it also displays the list of files.
Implementation of the model

The following codes are use in the implementation of the system.

```php
<?php
require_once( __DIR__ . '/common.php' );
require_once( DIR_CLASSES . 'FastaFile.php' );
require_once( DIR_CLASSES . 'BloomFilter.php' );
require_once( DIR_CLASSES . 'HashFilter.php' );
require_once( DIR_CLASSES . 'HammingFilter.php' );
require_once( DIR_LIB . 'color.php' );
$config = parse_ini_file( "config.ini", true );
if ( !$config ) { die( "An error occurred while reading the config file.\n" ); }
$mysql_address = $config['mysql']['server'];
$mysql_user = $config['mysql']['user'];
$mysql_password = $config['mysql']['password'];
$mysql_database = $config['mysql']['database'];
$mysql_table = $config['mysql']['table'];
$bloom_file = $config['bloom']['file'];
$bloom_length = $config['bloom']['length'];
$bloom_false_positive_prob = $config['bloom']['false_positive_prob'];
$hamming_distance_threshold = $config['hamming']['similarity_threshold'];
$color = new Colors();
if ( $argc != 2 ) { die( "Usage: \$argv[0] FILE\n" ); }
if ( !file_exists( $argv[1] ) ) { die( "Error: Unable to open file: \" . $argv[1] . \"\n" ); }
if ( !file_exists( $bloom_file ) ) { die( "Error: Bloom filter file not found, please initialize the filter first.\n" ); }
$input = new FastaFile( $argv[1] );
// Bloom Filter search
$bloomFilter = BloomFilter::load( $bloom_file );
if( $bloomFilter->has( $input->getContent() ) ) { // Testing for 100% match... using length/hash comparison
    $hashFilter = new HashFilter( $mysql_address, $mysql_user, $mysql_password,
    $mysql_database, $mysql_table );
    $match = $hashFilter->has( $input );
    if( $match != false ) { echo $color->getColoredString("File found: $match\n", "green" );
        exit( 0 );
    }
} // Testing for >90% match... using Hamming distance algorithm
$hammingFilter = new HammingFilter( $mysql_address, $mysql_user,
```
```php
$mysql_password, $mysql_database, $mysql_table );
$matchs = $hammingFilter->has( $input, $hamming_distance_threshold );
if ( $matchs != false ) { foreach ( $matchs as $match ) {
    echo $color->getColoredString( "Similar file found: ${match }
", "yellow" );
}   exit( 0 ); } else {   echo $color->getColoredString( "File not found.
", "red" );   exit( 0 ); }
?>
```

Listing 1: Query
The below code is used to initialize the system

```php
<?php
require_once( __DIR__.'/common.php' );
require_once( DIR_CLASSES .'FastaFile.php' );
require_once( DIR_CLASSES .'ProgressBar.php' );
require_once( DIR_CLASSES .'BloomFilter.php' );
require_once( DIR_CLASSES .'HashFilter.php' );
$config = parse_ini_file( "config.ini", true );
if ( !$config ) { die( "An error occurred while reading the config file.\n" ); }

$mysql_address = $config['mysql']['server'];
$mysql_user = $config['mysql']['user'];
$mysql_password = $config['mysql']['password'];
$mysql_database = $config['mysql']['database'];
$mysql_table = $config['mysql']['table'];
bloom_file = $config['bloom']['file'];
bloom_length = $config['bloom']['length'];
bloom_false_positive_prob = $config['bloom']['false_positive_prob'];
$hamming_distance_threshold = $config['hamming']['similarity_threshold'];

$hashFilter = new HashFilter( $mysql_address, $mysql_user,
$mysql_password, $mysql_database, $mysql_table );
if ( $argc != 2 ) { die( "Usage: \"$argv[0]." SOURCE_DIR\n" ); }
$sourceDirectory = $argv[1];
$sourceDirectory = $sourceDirectory./"/
$files = glob( $sourceDirectory."*.fas", GLOB_BRACE );
if ( count( $files ) < 1 ) { die("No files found in directory.\n" ); }

$progressBar = new ProgressBar( count( $files ) );
if ( file_exists( $bloom_file ) ) { $bloomFilter = BloomFilter::load( $bloom_file ); } else {
$bloomFilter = BloomFilter::init( $bloom_length, $bloom_false_positive_prob ); }
foreach ( $files as $filename ) { try {
$file = new FastaFile( $filename );
$hashFilter->add( $file );
$bloomFilter->add( $file->getContent() );
} catch ( FastaFileException $e ) {
echo "File \'$filename\' has been skipped because an error occurred while trying to read it.\n\n";
} catch ( HashFilterException $e ) {
echo "File \'$filename\' has been skipped because an error occurred while writing the DB. Error Description: ". $e->getMessage() . "\n\n";
} $progressBar->update(); }
```
$bloomFilter->save( $bloom_file );
?>

Listing 2: System Initialization
The code below is also used for the hamming distance computation

```php
<?php

class HammingFilter {
    private $server;
    private $user;
    private $password;
    private $database;
    private $table;
    private $conn;

    public function __construct( $server, $user, $password, $database, $table ) {
        $this->table = $table;
        $this->conn = new mysqli( $server, $user, $password, $database );
        if ( $this->conn->connect_error ) {
            throw new FastaMySqlWriterException( "MySQL connection failed ({$this->conn->connect_errno}) {$this->conn->connect_error}." );
        }
    }

    public function has( $file, $similarity ) {
        $matches = array();
        $content = $file->getContentSummary();
        $possibleMatches = $this->getByLength( strlen( $content ) );
    }
```
foreach( $possiblMatchs as $possibleMatch )
{
    if ( $this->areSimilar( $content, $possibleMatch['content'], $similarity) )
    {
        array_push( $matchs, $possibleMatch['filename'] );
    }
}

if ( !empty( $matchs ) )
{
    return $matchs;
}

return false;

private function areSimilar( $string1, $string2, $similarity )
{
    if ( strlen( $string1 ) != strlen( $string2 ) )
    {
        return false;
    }

    $threshold = floor( ( 1 - $similarity ) * strlen( $string1 ) + 0.0000001 );

    $distance = 0;

    for( $i = 0; $i < strlen( $string1 ); $i++ )
    {
        if ( $string1[$i] != $string2[$i] )
        {
            $distance++;
        }
    }

    if ( $distance > $threshold )
    {
private function getByLength( $length )
{
    $stmt = $this->prepareStatement( "SELECT filename, content FROM
        {$this->table} WHERE content_length = ?" );

    $stmt->bind_param( "i", $length );

    $stmt->bind_result( $filename, $content );

    $stmt->execute();

    $results = array();

    while ( $stmt->fetch() )
    {
        array_push( $results, array( 'filename' => $filename, 'content' => $content ) );
    }

    return $results;
}

private function prepareStatement( $stmt )
{
    $prepared_stmt = $this->conn->prepare( $stmt );

    if ( (!$prepared_stmt )
    {
        throw new HammingFilterException( "Error preparing query ({$this->conn->errno}) {$this->conn->error}." );
    }

    return $prepared_stmt;
Listing 3: Hamming distance
The code below is also used for the Bloom filter

```php
<?php require_once( __DIR__ ."/.." . '/common.php' ); require_once( DIR_LIB . 'bloomfilter.php' );

class BloomFilter {
    public static function load( $filename )
    {
        $bf = BloomFilterImplementation::load( $filename );
        return new self( $bf );
    }

    public static function init( $len, $prob )
    {
        $bf = BloomFilterImplementation::createFromProbability( $len, $prob );
        return new self( $bf );
    }

    private $filter;
    private function __construct( $filter )
    {
        $this->filter = $filter;
    }

    public function add( $key )
    {
        $this->filter->add( $key );
    }

    public function has( $key )
    {
        return $this->filter->contains( $key );
    }

    public function save( $filename )
    {
        $this->filter->save( $filename );
    }
}
```

Listing 4: Bloom Filter
The code below is also used for the hash value computation

```php
<?php

require_once( 'FastaFile.php' );

class HashFilter
{
    private $server;
    private $user;
    private $password;
    private $database;
    private $table;
    private $conn;

    public function __construct( $server, $user, $password, $database, $table )
    {
        $this->table = $table;

        $this->conn = new mysqli( $server, $user, $password, $database );

        if ( $this->conn->connect_error )
        {
            throw new HashFilterException( "MySQL connection failed ({$this->conn->connect_errno}) {$this->conn->connect_error}.");
        }
    }

    public function __destruct()
    {
        $this->conn->close();
    }

    public function add( FastaFile $file )
    {
    }
}
```
{$content = $file->getContentSummary();

$basename = $file->getName();

$contentLength = strlen($content);

$hashValue = md5(substr($content, 0, 10));

$stmt = $this->prepareStatement("INSERT INTO {$this->table} (filename, content, content_length, hashvalue) VALUES (?, ?, ?, ?) ON DUPLICATE KEY UPDATE content = VALUES(content), content_length = VALUES(content_length), hashvalue = VALUES(hashvalue)");

$stmt->bind_param("ssis", $basename, $content, $contentLength, $hashValue);

$stmt->execute();

$stmt->close();

public function has(FastaFile $file)
{
    $content = $file->getContentSummary();

    $contentLength = strlen($content);

    $hashValue = md5(substr($content, 0, 10));

    $stmt = $this->prepareStatement("SELECT filename FROM {$this->table} WHERE content_length = ? AND hashvalue = ?");

    $stmt->bind_param("sis", $contentLength, $hashValue);

    $stmt->bind_result($result);

    $stmt->execute();

    $stmt->close();
}
$fetchOk = $stmt->fetch();

if ( $fetchOk == null || !$fetchOk ) {
    $result = false;
}

$stmt->close();
return $result;

public function getByLenght( $length )
{
    $stmt = $this->prepareStatement("SELECT filename, content FROM
($this->table) WHERE content_length = ?" );

    $stmt->bind_result( $filename, $content );
    $stmt->execute();

    return $stmt->fetch_all();
}

private function prepareStatement( $stmt )
{
    $prepared_stmt = $this->conn->prepare( $stmt );

    if ( !$prepared_stmt ) {
        throw new HashFilterException("Error preparing query ({$this->conn->errno}) {$this->conn->error}");
    }

    return $prepared_stmt;
}
class HashFilterException extends Exception
{
    public function __construct($message, $code = 0, Exception $previous = null)
    {
        parent::__construct( $message, $code, $previous );
    }

    public function __toString()
    {
        return __CLASS__ . ": [{$this->code}]: {$this->message}\n";
    }
}

Listing 5: Hash Value
The code below is also used to set the fasta file functionality

```php
<?php

class FastaFile
{
    private $filename;
    private $content;

    public function __construct( $filename )
    {
        $this->filename = $filename;
        $this->content = null;
    }

    public function getName()
    {
        return basename( $this->filename );
    }

    public function getContent()
    {
        return $this->loadContent();
    }

    public function getContentSummary()
    {
        return $this->getLine( 1 );
    }

    private function loadContent()
    {
        if ( $this->content == null )
        {
            $this->content = file_get_contents( $this->filename );
            if ( $this->content === false )
```
Listing 6: Fasta Format
Appendix B

Contents of the CD

The CD submitted alongside this printed copy of this thesis contains the following:

• All the source codes used in the execution of this project
• A PDF copy of this project
• A Latex copy of this project
• Figures and Screen-shots of the running system