# Development of an Efficient Algorithm for DNA Sequence Alignment Based on Cosine Similarity


## Authors

T.M. Ariq Ahsan — Student Id: 084406
MD Saleh Faize — Student Id: 084444

## Supervisor

**Prof. Dr. M. A. Mottalib**
Head of the Department
Department of Computer Science & Engineering (CSE)
Islamic University of Technology (IUT)

## Co-Supervisor

**Abid Hasan**
Lecturer
Department of Computer Science & Engineering (CSE)
Islamic University of Technology (IUT)


**A Thesis submitted to the Department of Computer Science & Engineering (CSE)
in Partial Fulfillment of the requirements for the degree of
Bachelor of Science in Computer Science & Engineering (CSE)**



Department of Computer Science & Engineering (CSE)
Islamic University of Technology (IUT)
Organization of the Islamic Cooperation (OIC)
Gazipur, Bangladesh



September, 2012

# ABSTRACT

In our thesis we wanted to work with approximate gene matching with the help of the cosine similarity factor. Though several other gene matching algorithms has been invented since the post Sanger method period but quite a little advancement has been done in this field. We have chalked out a new formula for gene sequence matching and implemented gap algorithm in it and then evaluated it with some of the well established algorithm (The Dot-Matrix method, The Dynamic Programming and The Word Method.). We sacrificed efficiency for accuracy but we think our acumen of time was not bad either. We have our sight set upon further developing it and more assessment of it in near future.

# TABLE OFCONTENTS:

# Chapter 1. Introduction

## 1.1 Overview

Bioinformatics is the collection of biological data which are derived from statistical and structural analysis. There is information stored in our genetic code. That information is the prior concern in the fields of bioinformatics. But the genetic code's information processing is not the only main task of bioinformatics; here information is also collected from experimental results from various sources like- patient statistics and scientific literature, for processing. Research in bioinformatics includes method development for storage, retrieval and analysis of the data. We can see now that bioinformatics is a rapidly developing branch of biology and is highly interdisciplinary, using techniques and concepts from informatics, statistics, mathematics, chemistry, biochemistry, physics, and linguistics. It has many practical applications in different areas of biology and medicine.

We find the introduction of computing in bioinformatics in 1920s when the scientists realized the establishment of biological laws from data analysis by induction. It is also known as the traditional method which is found in history. However the development of powerful computers and the availability of experimental data launched bioinformatics as an independent field where we can treat data in a faster manner than previous age. For example, we can say the development of dimension in image viewing and with the help of that now it is possible of viewing three-dimensional structure of DNA or amino acid. Today, practical applications of bioinformatics are readily available through the World Wide Web, and are widely used in biological and medical research. As the field is rapidly evolving, the very definition of bioinformatics is still the matter of some debate.

For several reasons we find a natural relationship in computer science and biology. The first reason is the phenomenal rate of biological data being produced provides challenges like massive amounts of data have to be stored, analyzed and made

accessible. The second reason is the nature of the data if often such that a statistical method, and hence computation, is necessary. This applies in particular to the information on the building plans of protein and of the temporal and spatial organization of their expression in the cell encoded by the DNA. The third reason is there is a strong analogy between the DNA sequence and a computer program. For example we can say that the DNA represents a Turing Machine.

Analyses in bioinformatics focus on three types of datasets: genome sequences, macromolecular structures, and functional genomics experiments (e.g. expression data, yeast two–hybrid screens). But bioinformatics analysis is also applied to various other data, e.g. taxonomy trees, relationship data from metabolic pathways, the text of scientific papers, and patient statistics. A large range of techniques are used, including primary sequence alignment, protein 3D structure alignment, phylogenetic tree construction, prediction and classification of protein structure, prediction of RNA structure, prediction of protein function, and expression data clustering. Algorithmic development is an important part of bioinformatics, and techniques and algorithms were specifically developed for the analysis of biological data (e.g., the dynamic programming algorithm for sequence alignment).

Bioinformatics has a large impact on biological research. Giant research projects such as the human genome project would be meaningless without the bioinformatics component. The goal of sequencing projects, for example, is not to corroborate or refute a hypothesis, but to provide raw data for later analysis. Once the raw data are available, hypotheses may be formulated and tested in silicon. In this manner, computer experiments may answer biological questions which cannot be tackled by traditional approaches. This has led to the founding of dedicated bioinformatics research groups as well as to a different work practice in the average bioscience laboratory where the computer has become an essential research tool.

## 1.2 Problem Statement:

The main problem of high dimensional data is the inclusion of noisy and irrelevant data in the information set. As the datasets become large the number of noisy,

redundant and uninformative gene also increases resulting in space-time complexity. So we work with on an efficient algorithm for sequence alignment based on cosine similarity.

## 1.3 Research Challenges:

The high dimensionality results in an immense feature space and thus execution of a brute force exhaustive search should not be encouraged. Therefore, to achieve an accurate and efficient evaluation of samples an optimal method needs to be devised. The desired outcome of the method is minimizing the number of features and increasing the predictive power of the classifiers. To add more intensity to the problem domain this field of bioinformatics produces inadequate testing and training samples. Along with the removal of noisy, irrelevant and redundant information the proposed method must be able to handle the correlation factor existing between the features and thus utilize the combined predictive power. This study encompasses all these factors and theoretically expects to bring about better results.

## 1.4 Motivation:

Our motivation was for finding a new approach for sequence alignment techniques. And as we said we try to give one. We try to give an efficient algorithm on sequence alignment technique based on cosine similarity.

## 1.5 Scopes:

This study aims at giving a new approach for better sequence alignment. As we said some of the methods do not work well as those take all the characters of the sequence for comparison. But in our approach we try to solve that by using chunks. But still lot of work is needed for improving our algorithm. For that purpose the door for implementation is open for all interested ones. There can be work done like- for reducing the time complexity, giving scores to the alignments, for overcoming the generalization problem and others.

## 1.6 Thesis Outline:

In Chapter 1 we have talked about the introduction of our study in a précised manner. Chapter 2 deals with the basic feature selection method and some highlighted evolutionary approaches with a brief discussion about PSO method. Chapter 3 will be discussed about our proposed algorithm and some elaborate discussion. Chapter 4 will consist of the experimental analysis and result comparisons.

# Chapter 2.Literature Review

## 2.1Sequence Alignment:

The method of arranging DNA, RNA or protein sequences for similarity region identification is called a sequence alignment in the language of bioinformatics. A sequence alignment can be a consequence of functional, structural or evolutionary relationships between the sequences. In sequence alignment we not only look for exact matching but also relative matching by using gaps. Gaps are used by insertion, extension and deletion. Mismatches can be interpreted as point mutation and gaps as indels where two sequences share a common ancestor. In sequence alignment of proteins, the degree of similarity between amino acid occupying a particular position in the sequence can be interpreted as a rough measure of how conserved a particular region or sequence motif is among lineages. The absence of substitution, or the presence of only very conservative substitutions in a particular region of the sequence suggest that this region has structural or functional importance. But in DNA and RNA nucleotide bases are more similar to each other than amino acids, the conservation of base pairs can indicate a similar functional or structural role.

## 2.2 Sequence Alignment Types:

### 2.2.1 Global and Local Alignments:

We can align the sequence with our hands where it is very short. But in real life we have to work with extremely numerous sequences to align which is beyond question if we use only our hands. For this reason human knowledge is applied to construct algorithms to produce high-quality sequence alignments. In the computational approaches we have two types of alignments. They are – Global Alignment and Local Alignment.

In global alignment calculation, it uses a form of global optimization that forces the alignment to span the entire length of all query sequence. Whether local alignments identify similarity regions within long sequences that are often widely diverge overall. Local alignments are often preferable, but can be more difficult to calculate because of

the additional challenge of identifying the regions of similarity. A variety of computational algorithms have been applied to the sequence alignment problem, including slow but formally correct methods like dynamic programming, and efficient, heuristic algorithms or probabilistic methods that do not guarantee to find best matches designed for large-scale database search.

In global alignment, a kind of attempt is found to align every residue in almost every sequence. It is extremely useful when the sequences in the query set are similar and of roughly equal size. For example Needlman-Wunsch algorithm used a general global alignment technique based on dynamic programming. But for dissimilar sequences local alignments are more useful which are suspected to contain regions of similarity or similar sequence motifs within their larger sequence context. For example we found a general local alignment method by Smith-Waterman algorithm which also used dynamic programming. At last it can be said that if there exists sufficient similarity then there will be no difference between local and global alignments.

We will face a problem in aligning the sequences if there exists an overlap between the downstream part of one sequence with the upstream part of another sequence. In this situation neither global nor local alignment is appropriate. Here the fact is that a global alignment would attempt to force the alignment to extend beyond the overlapped region. On the other hand a local alignment might not fully cover the overlapped region. So here the introduction of a hybrid method is made which is known as semiglobal or glocal methods. The name glocal comes from the first part of global and last part of local. These methods try to find the best possible alignment including the start and end of one or the other sequence.

## 2.2.2Macromolecular Alignments:

## DNA:

DNA, the full form goes like this-‘Deoxyribonucleic acid.’ It is a nucleic acid that contains genetic instructions which are used in the development and functioning of all known living organisms. It is known that the genetic information which is carried by the DNA segments is called ‘GENE.’ Besides RNA and proteins, DNA is one of the major macromolecules which are essential for all known forms of life.

DNA consists of two long polymers of simple units called nucleotides, with backbones made of sugars and phosphate groups joined by ester bonds. These two strands run in opposite directions to each other and are therefore anti-parallel. Attached to each sugar is one of four types of molecules called nucleobases. The nucleobases are classified into two types: the purines, A and G, being fused five- and six-membered heterocyclic compounds, and the pyrimidines, the six-membered rings C and T. It is the sequence of these four nucleobases along the backbone that encodes information. Within cells DNA is organized into long structures called chromosomes. During cell division these chromosomes are duplicated in the process of DNA replication, providing each cell its own complete set of chromosomes. In living organisms DNA does not usually exist as a single molecule, but instead as a pair of molecules that are held tightly together. These two long strands entwine like vines, in the shape of a double helix. The nucleotide repeats contain both the segment of the backbone of the molecule, which holds the chain together, and a nucleobase, which interacts with the other DNA strand in the helix.

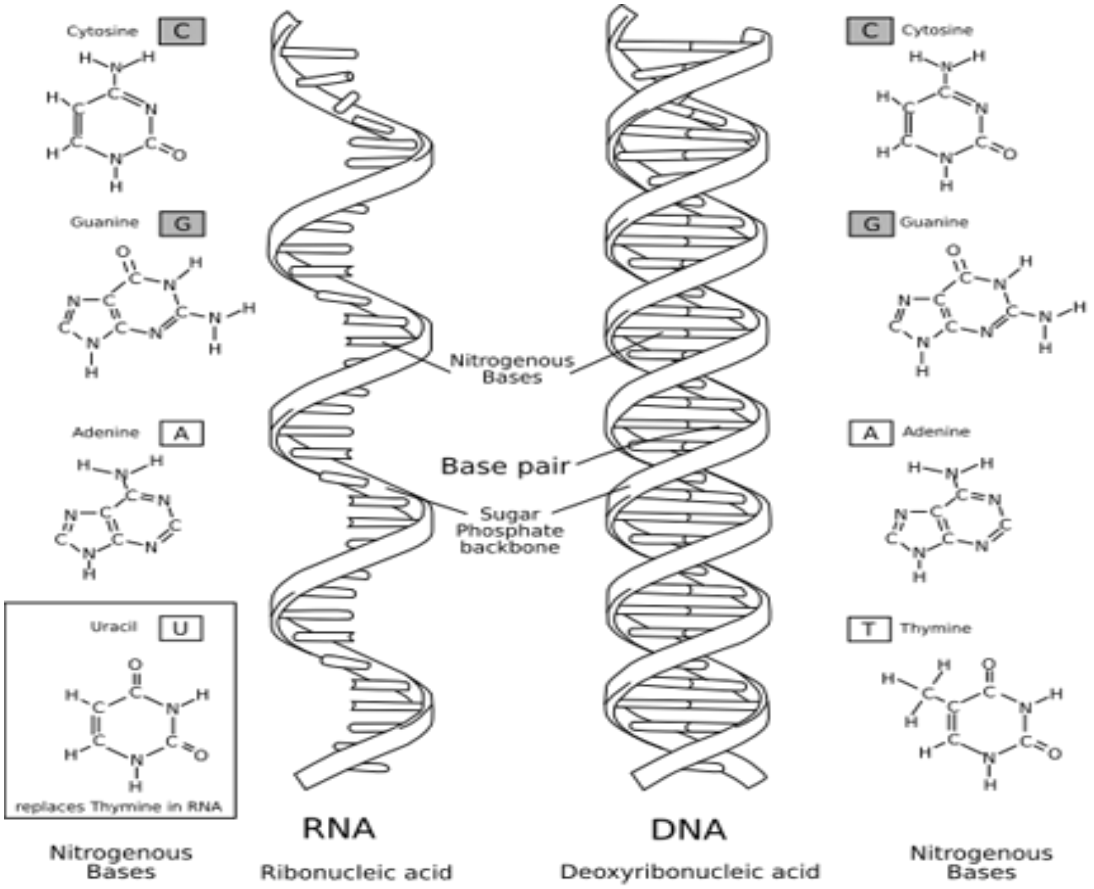



**Figure 1: Single Helix RNA and Double Helix DNA.**

## RNA:

Ribonucleic acid, or RNA, is part of a group of molecules known as the nucleic acids, which are one of the four major macromolecules (along with lipids, carbohydrates and proteins) essential for all known forms of life. Like DNA, RNA is made up of a long chain of components called nucleotides. Each nucleotide consists of a nucleobase, a ribose sugar, and a phosphate group. The sequence of nucleotides allows RNA to encode genetic information. All cellular organisms use messenger RNA (mRNA) to

carry the genetic information that directs the synthesis of proteins. In addition, many viruses use RNA instead of DNA as their genetic material. Some RNA molecules play an active role in cells by catalyzing biological reactions, controlling gene expression, or sensing and communicating responses to cellular signals. One of these active processes is protein synthesis, a universal function whereby mRNA molecules direct the assembly of proteins on ribosomes. This process uses transfer RNA (tRNA) molecules to deliver amino acids to the ribosome, where ribosomal RNA (rRNA) links amino acids together to form proteins. The chemical structure of RNA is very similar to that of DNA, with two differences: (a) RNA contains the sugar ribose, while DNA contains the slightly different sugar deoxyribose (a type of ribose that lacks one oxygen atom), and (b) RNA has the nucleobaseuracil while DNA contains thymine. Unlike DNA, most RNA molecules are single-stranded and can adopt very complex three-dimensional structures.

## Protein:

Proteins are biochemicalcompounds consisting of one or more polypeptides typically folded into a globular or fibrous form, facilitating a biological function.

A polypeptide is a single linear polymer chain of amino acids bonded together by peptide bonds between the carboxyl and amino groups of adjacent amino acid residues. The sequence of amino acids in a protein is defined by the sequence of a gene, which is encoded in the genetic code. In general, the genetic code specifies 20 standard amino acids; however, in certain organisms the genetic code can include seleno-cysteine and in certain archaeapyrolysine. Shortly after or even during synthesis, the residues in a protein are often chemically modified by posttranslational modification, which alters the physical and chemical properties, folding, stability, activity, and ultimately, the function of the proteins. Sometimes proteins have non-peptide groups attached, which can be called prosthetic groups or cofactors. Proteins can also work together to achieve a particular function, and they often associate to form stable proteincomplexes. Like other biological macromolecules such as

polysaccharides and nucleic acids, proteins are essential parts of organisms and participate in virtually every process within cells. Many proteins are enzymes that catalyze biochemical reactions and are vital to metabolism. Proteins also have structural or mechanical functions, such as actin and myosin in muscle and the proteins in the cytoskeleton, which form a system of scaffolding that maintains cell shape. Other proteins are important in cell signaling, immune responses, cell adhesion, and the cell cycle. Proteins are also necessary in animals' diets, since animals cannot synthesize all the amino acids they need and must obtain essential amino acids from food. Through the process of digestion, animals break down ingested protein into free amino acids that are then used in metabolism.

We have said earlier about the macromolecules which are important for every living organism. Now we will discuss about their sequence alignment shortly in the following:

The sequence alignments of the macromolecules are almost same but the exception exists in their representation of respective characters. For DNA the representative characters are-A, T, C, G; for RNA-A, U, C, G and for proteins the representative characters are various amino acids' characters. The algorithmic approach is same for the three macromolecules. We will discuss about the sequence techniques both the existing ones and ours in the later parts.

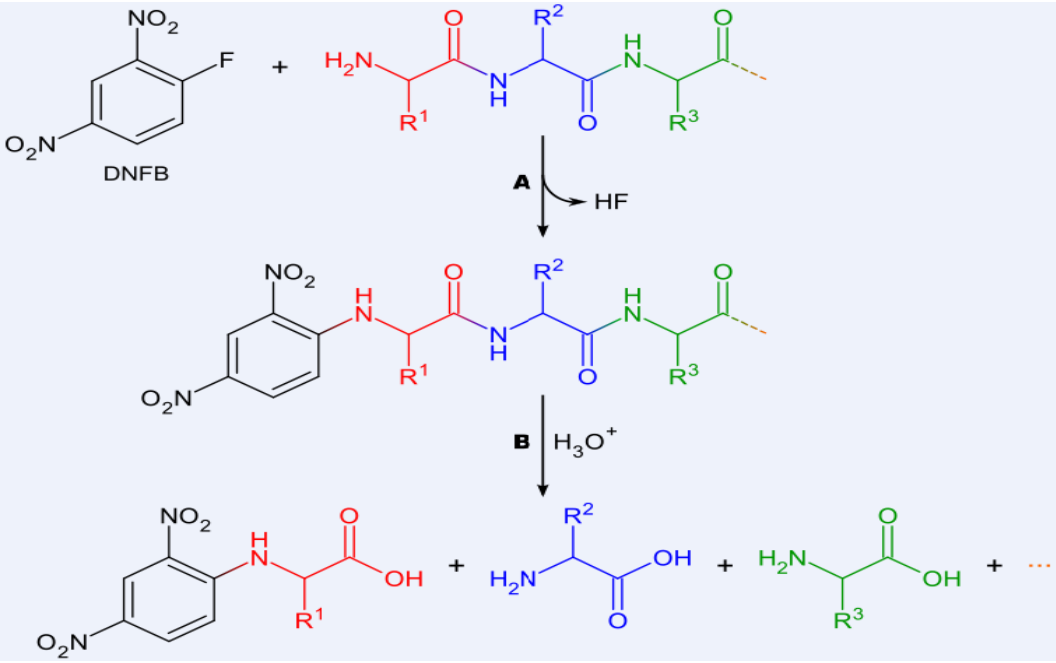



**Figure 2: Protein Structure.**

## 2.3 Existing Sequence Alignment Techniques:

There are two types of sequence alignment techniques. They are pairwise and multiple sequence alignment. In pairwise sequence alignment technique we try to find the best possible matching as local or global alignments of two query sequences. On the other hand multiple sequence alignment is an extension of pairwise sequence alignment where we also try to find the best possible matching from more than two sequences. As we work with pairwise sequence alignment in our thesis so we will try to discuss some of the pairwise alignment techniques in the following:

### 2.3.1 Pairwise Sequence Alignment Techniques:

#### 2.3.1.1 Dot-matrix Method:

The dot-matrix approach produces a group of alignments for individual sequence regions. It is conceptually simple. It is easy to visually pick certain sequence features (such as-insertions, deletions, repeats, or invert repeats) in the absence of noise. In order to construct the plot of dot-matrix we use a two-dimensional matrix where two

sequences are written along the top row and leftmost column. The recurrence plot is selected by putting a dot at the place where the characters match appropriately. Very closely related sequences appear as a single line along the matrix's main diagonal as plotted dots.

The main advantage of this techniques is repetitiveness in a single sequence. Here we can plot a sequence against itself and if they share significant similarities will appear as lines off the main diagonal. If a protein has multiple similar structural domains then this situation can occur.

This method incorporates some problems which we cannot neglect. We use dot plots for displaying the information. For this reason the technique includes- noise, lack of clarity, non-intuitiveness, difficulty in extracting match positions between two sequences. The main advantage is the wastage of space where the match data is inherently duplicated across the diagonal and most of the actual plotted area is covered either by empty space or noise. So there can be loss of information too and it is not desired.

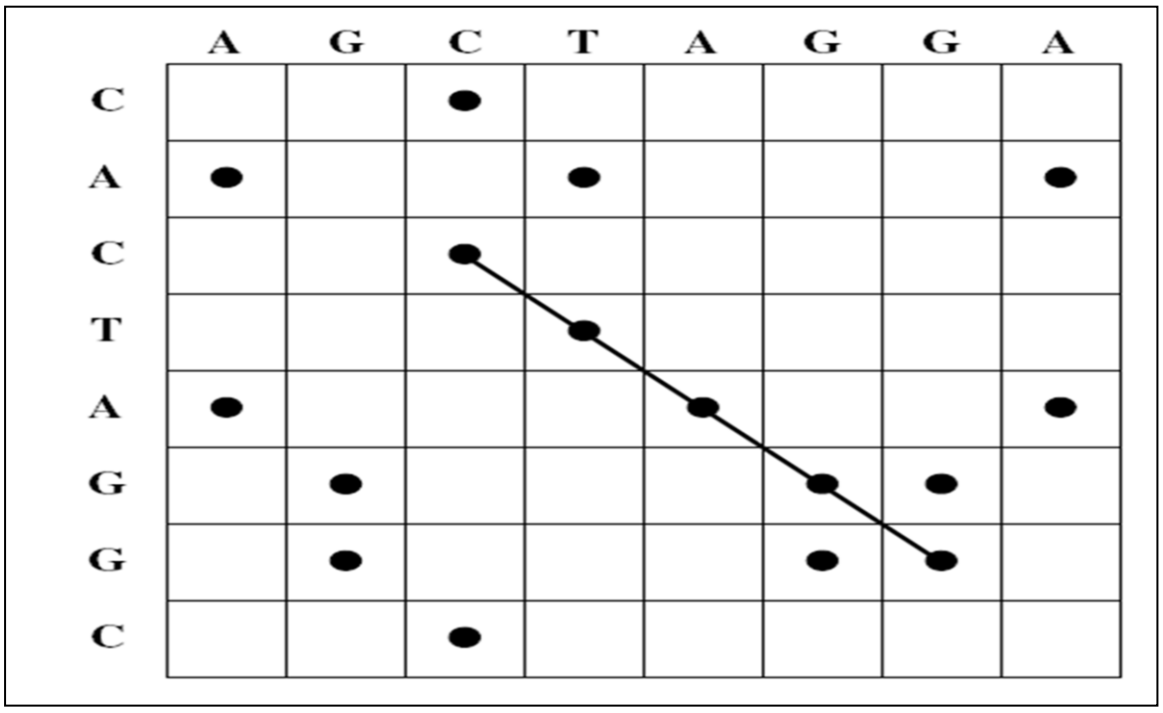



**Figure 3: Dot-Matrix Plot**

### 2.3.1.2 Dynamic Programming:

For the production of global and local alignments dynamic programming technique is applied. Needleman-Wunsch used dynamic programming for producing global alignments where Smith-Waterman used this technique for producing local alignments. Here protein alignments use a substitution matrix for matching and DNA and RNA may use scoring matrix. In this technique we can use gap penalties for DNA, RNA and proteins. In standard dynamic programming, the score of each amino acid position is independent of the identity of its neighbors and here base stacking effects are not taken into account but it is possible to take account by modifying the algorithm efficiently. A common extension to standard linear gap costs is the usage of two different gap penalties for opening a gap and for extending a gap. Typically the former is much larger than the latter, e.g. -10 for gap open and -2 for gap extension. Thus, the number of gaps in an alignment is usually reduced and residues and gaps are kept together, which typically makes more biological sense. The Gotoh algorithm implements affine gap costs by using three matrices.

Dynamic programming can be useful in aligning nucleotide to protein sequences, a task complicated by the need to take into account frameshift mutations (usually insertions or deletions). The framesearch method produces a series of global or local pairwise alignments between a query nucleotide sequence and a search set of protein sequences, or vice versa. Its ability to evaluate frameshifts offset by an arbitrary number of nucleotides makes the method useful for sequences containing large numbers of indels, which can be very difficult to align with more efficient heuristic methods. In practice, the method requires large amounts of computing power or a system whose architecture is specialized for dynamic programming. The dynamic programming technique guarantees in finding an optimal alignment from a given particular scoring function. But making a good scoring function is an empirical rather than a theoretical matter most of the time.

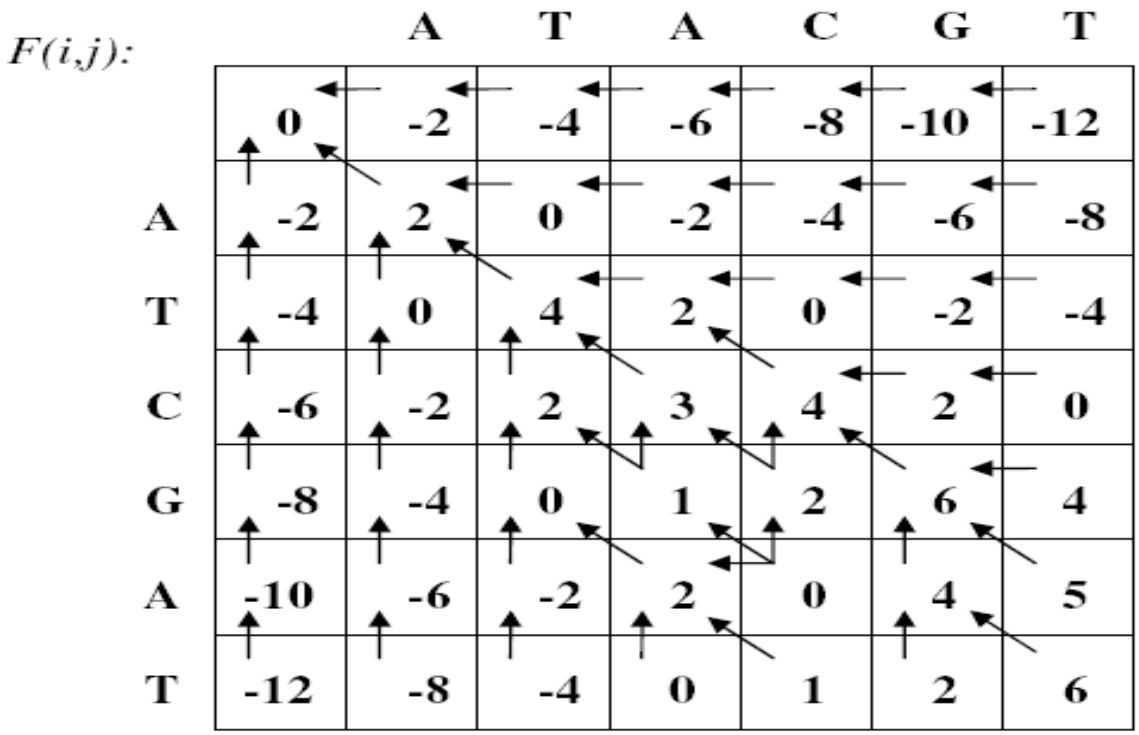



**Figure 4: Dynamic Programming Plot.**

## 2.3.1.3 Word Methods:

The k-tuple method is also known as world methods. It uses heuristic methods which cannot guaranty of an optimal alignment solution but is significantly more efficient than dynamic programming. These methods are used in large databases where it is assumed that there will be no significant match between two sequences. World methods identify a series of short, nonoverlapping subsequences in the query sequence that is then matched to candidate database sequences. The relative positions of the word in the two sequences being compared are subtracted to obtain an offset; this will indicate a region of alignment if multiple distinct words produce the same offset. Only if this region is detected do these methods apply more sensitive alignment criteria; thus, many unnecessary comparisons with sequences of no appreciable similarity are eliminated.

In the FASTA method, the user defines a value k to use as the word length with which to search the database. The method is slower but more sensitive at lower values of k, which are also preferred for searches involving a very short query sequence. The BLAST family of search methods provides a number of algorithms optimized for particular types of queries, such as searching for distantly related sequence matches. BLAST was developed to provide a faster alternative to FASTA without sacrificing much accuracy; like FASTA, BLAST uses a word search of length k, but evaluates only the most significant word matches, rather than every word match as does FASTA. Most BLAST implementations use a fixed default word length that is optimized for the query and database type, and that is changed only under special circumstances, such as when searching with repetitive or very short query sequences.

## 2.3.2 Multiple Sequence Alignment Techniques:

Though we do not work with multiple sequence alignment but our proposed technique has the capability of that and for that reason the technique can be modified in future. That's why we are going to give a brief discussion on existing multiple sequence alignment techniques in the following:

### 2.3.2.1 Dynamic Programming:

The technique of dynamic programming is theoretically applicable to any number of sequences; however, because it is computationally expensive in both time and memory, it is rarely used for more than three or four sequences in its most basic form. This method requires constructing the n-dimensional equivalent of the sequence matrix formed from two sequences, where n is the number of sequences in the query. Standard dynamic programming is first used on all pairs of query sequences and then the "alignment space" is filled in by considering possible matches or gaps at intermediate positions, eventually constructing an alignment essentially between each two-sequence alignment. Although this technique is computationally expensive, its guarantee of a global optimum solution is useful in cases where only a few sequences need to be aligned accurately. One method for reducing the computational demands of

dynamic programming, which relies on the "sum of pairs" objective function, has been implemented in the MSA software package.

### 2.3.2.2 Progressive Methods:

Progressive, hierarchical, or tree methods generate a multiple sequence alignment by first aligning the most similar sequences and then adding successively less related sequences or groups to the alignment until the entire query set has been incorporated into the solution. The initial tree describing the sequence relatedness is based on pairwise comparisons that may include heuristic pairwise alignment methods similar to FASTA. Progressive alignment results are dependent on the choice of "most related" sequences and thus can be sensitive to inaccuracies in the initial pairwise alignments. Most progressive multiple sequence alignment methods additionally weight the sequences in the query set according to their relatedness, which reduces the likelihood of making a poor choice of initial sequences and thus improves alignment accuracy. Many variations of the Cluster progressive implementationare used for multiple sequence alignment, phylogenetic tree construction, and as input for protein structure prediction. A slower but more accurate variant of the progressive method is known as T-Coffee.

### 2.3.2.3 Iterative Methods:

Iterative methods attempt to improve on the heavy dependence on the accuracy of the initial pairwise alignments, which is the weak point of the progressive methods. Iterative methods optimize an objective function based on a selected alignment scoring method by assigning an initial global alignment and then realigning sequence subsets. The realigned subsets are then themselves aligned to produce the next iteration's multiple sequence alignment. Various ways of selecting the sequence subgroups and objective function are reviewed in.

## 2.3.2.4 Motif Finding:

Motif finding, also known as profile analysis, constructs global multiple sequence alignments that attempt to align short conserved sequence motifs among the sequences in the query set. This is usually done by first constructing a general global multiple sequence alignment, after which the highly conserved regions are isolated and used to construct a set of profile matrices. The profile matrix for each conserved region is arranged like a scoring matrix but its frequency counts for each amino acid or nucleotide at each position are derived from the conserved region's character distribution rather than from a more general empirical distribution. The profile matrices are then used to search other sequences for occurrences of the motif they characterize. In cases where the original data set contained a small number of sequences, or only highly related sequences, pseudocounts are added to normalize the character distributions represented in the motif.

## 2.3.2.5 Techniques Inspired by Computer Science:

A variety of general optimization algorithms commonly used in computer science have also been applied to the multiple sequence alignment problem. Hidden Markov models have been used to produce probability scores for a family of possible multiple sequence alignments for a given query set; although early HMM-based methods produced underwhelming performance, later applications have found them especially effective in detecting remotely related sequences because they are less susceptible to noise created by conservative or semiconservative substitutions.Genetic algorithms and simulate annealing have also been used in optimizing multiple sequence alignment scores as judged by a scoring function like the sum-of-pairs method. More complete details and software packages can be found in the main article multiple sequence alignment.

# Chapter 3.Proposed Method:

## 3.1 Overall Concept:

Sequences are sufficiently flexible to be able to express the same meaning through different alignment. At the same time, inconsistency of surface expressions has persisted as a serious problem in natural sequencing processing. For example, in the biomedical domain, cardiovascular disorder can be described using various expressions: cardiovascular diseases, cardiovascular system disorder, and disorder of the cardiovascular system. It is a nontrivial task to find the entry from these surface expressions appearing in sequence. This study addresses approximate sequence matching, which consists of finding all the possible matching in a sequence collection V such that they have similarity that is no smaller than a threshold _ with a query string x. This task has a broad range of applications, including sequencing correction, relative sequence look-up, record linkage, and duplicate sequence.

Formally, the task obtains a subset$Y_{x,\alpha} \subseteq$ V

$Y_{x,\alpha}$ = { y∈ V | sim(x,y) ≥ $\alpha$ },……(1)

Wherein (x, y) present the similarity between x and y. A naive solution to this task is to compute similarity values |V | times, i.e., between x and every string $y \epsilon V$. However, this solution is impractical when the number of strings |V | is huge (e.g., more than one million).

## 3.2 Proposed Method and Algorithm:

We assume that the characters of a sequence are represented arbitrarily by a set. Although it is important to design a sequence representation for an accurate similarity measure, we do not address this problem: our emphasis is not on designing a better representation for sequence matching but on establishing an efficient algorithm.

Our representation is given by n-grams: all subsequences of size n in a string. We use trigrams throughout this study as an example of sequence representation. For example the sequence “ATCGTAGTATTAGTTACCT” is expressed in 21 elements of letter

trigrams(1), {‘$$A’, ‘$AT’, ‘ATC’, ‘TCG’, ‘CGT’, ‘GTA’, ‘TAG’, ‘AGT’, ‘GTA’, ‘TAT’, ‘ATT’, ‘TTA’, ‘TAG’, ‘AGT’, ‘GTT’, ‘TTA’, ‘TAC’, ‘ACC’, ‘CCT’, ‘CT$’, ‘T$$’}. We used two ‘$$’ signs to indicate the starting and ending of the sequence. But we discard two characters (‘$$A’ and ‘T$$’) from comparison in the sequence alignment technique for experiencing a better efficiency. In general, a string x consisting of |X| characters yields (|x| + n − 1) elements of n grams. We call |x| and |X| the length and size, respectively, of the string x.

Let X and Y denote the feature sets of the strings x and y, respectively. The cosine similarity between the two strings x and y is,

$$\text{Cosine}(X, Y) = \frac{|X \cap Y|}{\overline{X\ |Y|}} \quad \text{.....(2)}$$

By integrating this definition with Equation 1, we obtain the necessary and sufficient condition for

$$[\alpha \quad \overline{X\ |Y|}] \leq |X \cap Y| \leq \min\{|X|, |Y|\} \quad \text{.....(3)}$$

This inequality states that two strings x and y must have at least $\tau = [\alpha \quad \overline{X\ |Y|}]$ features in common. When ignoring $|X \cap Y|$ in the inequality, we have an inequality about |X| and |Y |,

$$[\alpha^2 |X|] \leq |Y| \leq [\frac{|X|}{\alpha^2}] \quad \text{.....(4)}$$

This inequality presents the search range for retrieving similar strings; that is, we can ignore strings whose feature size is out of this range. Other derivations are also applicable to similarity measures, including Dice, Jaccard, and overlap coefficients.

We explain one usage of these conditions. Let query string x = “ATCGTAGTATTAGTTACCT” and threshold for approximate dictionary matching $\alpha$ = 0.7 with cosine similarity. Representing the strings with letter trigrams, we have the size of x, |X| = 19. The inequality (4) gives the search range of |Y| of the retrieved strings, $10 \leq |Y| \leq 39$. Presuming that we are searching for strings of |Y | = 15,we obtain the necessary and sufficient condition for the approximate sequence matching from the inequality (3), $\tau = 15 \leq |X \cap Y|$. Now if we take sequence of 22 characters

and we run our program with trigrams with the previous threshold value then we will not get any desired values. For that the cosine(X,Y) value will be 0.1467 which is not anywhere near to our threshold value. So we will not always take trigrams for comparison between the sequences for alignment. For this reason we try to make our program to work in an efficient environment by giving user the feasibility of selecting the number of characters of a chunk.

In the above discussion we said about the exact matching between two sequences. But in real life we rarely have the probability of exact matching. That's why we incorporate gaps for relative matching. In that purpose we also use the same equations and procedures. Like there will be threshold value according which we will try to find our desired sequence alignment and for comparison we also take out two characters ('$$A' and 'T$$') like the previous one. For finding the relative sequence alignment we do not negotiate with accuracy.

Our proposed algorithm is given below:

**Input: V : collection of sequences**

**Input: x: query sequence**

**Input: $\alpha$: threshold for the similarity**

**Output: Y: list of sequences similar to the query**

**1.X string to character(x);**

**2.Y [];**

**3.forl min y(|X|, _) to max y(|X|, $\alpha$) do**

**4.$\tau$ =min overlap(|X|, l, $\alpha$);**

**5. R = overlapjoin(X, Y, V, l);**

**6. for each r $\epsilon$2 R do append r to Y;**

**7. end**

**8. return Y;**

## 4.Experimental Analysis:

We report the experimental results of approximate dictionary matching on large-scale datasets with person names, biomedical names, and general English words. We implemented various systems of approximate dictionary matching.

## 4.1 Datasets

We used three large datasets with person names (IMDB actors), general English words (Google Web1T), and biomedical names (UMLS).

• IMDB(International Medical Database): This dataset comprises disease names extracted from the IMDB database6. We used all virus names (1,098,022 strings; 18 MB) from the file diease.rr.ogf. The average number of letter trigrams in the strings is 17.2. The total number of trigrams is 42,180. The system generated index files of 83 MB in 56.6 s.

• GoogleWeb1T unigrams: This dataset consists of English word unigrams included in the Google Web1T corpus (LDC2006T13). We used all word unigrams (13,588,391 strings; 121 MB) in the corpus after removing the frequency information. The average number of letter trigrams in the strings is 10.3. The total number of trigrams is 301,459. The system generated index files of 601 MB in 551.7 s.

• UMLS: This dataset consists of English names and descriptions of biomedical concepts included in the Unified Medical Language System (UMLS). We extracted all English concept names (5,216,323 strings; 212 MB) from MRCONSO.RRF.aa.gz and MRCONSO.RRF.ab.gz in UMLS Release 2009AA. The average number of letter trigrams in the strings is 43.6. The total number of trigrams is 171,596. The system generated indexes of 1.1 GB in 1216.8 s. For each dataset, we prepared 1,000 query strings by sampling strings randomly from the dataset. To simulate the situation where query strings are not only identical but also similar to dictionary entries, we introduced random noise to the strings. In this experiment, one-third of the query strings are unchanged from the original (sampled) strings, one-third of the query strings have one letter changed, and one-third of the query strings have two letters changed. When changing a letter, we randomly chose a letter position from a uniform distribution, and replaced the letter at the position with an ASCII letter randomly chosen from a uniform distribution.

## 4.2 Results:

To examine the scalability of each system, we controlled the number of strings to be indexed from 10%–100%, and issued 1,000 queries. Figure 1 portrays the average response time for retrieving strings whose cosine similarity values are no smaller than 0.7. Although LSH (B=16) seems to be the fastest in the graph, this system missed

many true positives7; the recall scores of approximate dictionary matching were 15.4% (IMDB), 13.7% (Web1T), and 1.5% (UMLS). Increasing the parameter B improves the recall at the expense of the response time. LSH (B=64) 8. It not only ran slower than the proposed method, but also suffered from low recall scores, 25.8% (IMDB), 18.7% (Web1T), and 7.1% (UMLS). LSH was useful only when we required a quick response much more than recall. The other systems were guaranteed to find the exact solution (100% recall). The proposed algorithm was the fastest of all exact systems on all datasets: the response times per query (100% index size) were 1.07 ms (IMDB), 1.10 ms (Web1T), and 20.37 ms (UMLS). The response times of the Naıve algorithm were too slow, 32.8 s (IMDB), 236.5 s (Web1T), and 416.3 s (UMLS). The proposed algorithm achieved substantial improvements over the AllScan algorithm: the proposed method was 65.3 times (IMDB), 227.5 times (Web1T), and 13.7 times (UMLS) faster than the Naıve algorithm. We observed that the Signature algorithm, which is Algorithm 3 with- out lines 17–18, did not perform well: The Sig- nature algorithm was 1.8 times slower (IMDB), 2.1 times faster (Web1T), and 135.0 times slower (UMLS) than the AllScan algorithm. These results indicate that it is imperative to minimize the number of candidates to reduce the number of binary-search operations. The proposed algorithm was 11.1–13.4 times faster than DivideSkip. Figure 2 presents the average response time of the proposed algorithm for different similarity measures and threshold values. When the similar- ity threshold is lowered, the algorithm runs slower because the number of retrieved strings |Y| in- creases exponentially. The Dice coefficient and cosine similarity produced similar curves. Table 2 summarizes the run-time statistics of the proposed method for each dataset (with cosine similarity and threshold 0.7). Using the IMDB dataset, the proposed method searched for strings whose size was between 8.74 and 34.06; it retrieved 4.63 strings per query string. The proposed algorithm scanned 279.7 strings in 4.6 inverted lists to obtain 232.5 candidate strings. The algorithm performed a binary search on 4.3 inverted lists containing 7,561.8 strings in all. In contrast, the AllScan algorithm had to scan 16,155.1 strings in 17.7 inverted lists and considered 9,788.7 candidate strings, and found only 4.63 similar strings. This table clearly demonstrates three key contributions of the proposed algorithm for efficient approximate dictionary matching.

First, the pro- posed algorithm scanned far fewer strings than did the AllScan algorithm. For example, to obtain candidate strings in the IMDB dataset, the pro- posed algorithm scanned 279.7 strings, whereas the AllScan algorithm scanned 16,155.1 strings. Therefore, the algorithm examined only 1.1%– 3.5% of the strings in the entire inverted lists in the three datasets. Second, the proposed algorithm considered far fewer candidates than did the AllScan algorithm: the number of candidate strings considered by the algorithm was 1.2%– 6.6% of those considered by the AllScan algorithm. Finally, the proposed algorithm read fewer inverted lists than did the AllScan algorithm. The proposed algorithm actually read 8.9 (IMDB), 6.0 (Web1T), and 31.7 (UMLS) inverted lists during the experiments9. These values indicate that the proposed algorithm can solve $\tau$ -overlap join problems by checking only 50.3% (IMDB), 53.6% (Web1T), and 51.9% of the total inverted lists retrieved for queries.

Table 1: Run-time statistics of the proposed algorithm for each dataset

| Averaged item | UMLS | WEB1T | IMJT | UCN |
|---|---|---|---|---|
| Min\|y\| | 4.6 | 5.35 | 21.87 | minimum size of trigrams of target strings |
| Max\|y\| | 279.7 | 20.46 | 88.48 | maximum size of trigrams of target strings |
| T | 232.5 | 9.09 | 47.77 | minimum number of overlaps required/sufficient per query |
| \|y\| | 4.6 | 3.22 | 111.79 | number of retrieved strings per query |
| **Total Match:** | N/A | N/A | N/A | — averaged for each query and target size: |
| Inverted list | 17.7 | 11.2 | 61.1 | number of inverted lists retrieved for a query |
| Strings | 16155.1 | 52557.6 | 49561.4 | number of strings in the inverted list |
| Unique strings | 9788.7 | 44834.6 | 17457.5 | number of unique strings in the inverted list |
| **Gapped match:** | N/A | N/A | N/A | — averaged for each query and target size: |
| Inverted list | 4.6 | 3.1 | 14.3 | number of inverted lists scanned for generating candidates |
| Strings | 273.7 | 552.7 | 1756.3 | number of strings scanned for generating candidates |
| candidates | 232.5 | 523.7 | 1149.7 | number of candidates generated for a query |

## 4.3 Parameter Generation:

Positional features Position-specific k-mers are the most common features used for finding signals in the DNA stream data. These features capture the correlations between different nucleotides and their relative positions. The nucleotides bordering the splice site are of primary importance as they may capture binding information. The simplest features of this type are position-specific 1-mers, which describe the occurrence of a specific nucleotide in a particular location in the sequence. These features also define the consensus sequence. We consider sequences of length 160, so there are 4 × 160 or 640 possible position-specific 1-mers. We use this basic feature set to construct position-specific k-mer features. Position-specific k-mers capture the correlations between k-adjacent nucleotides. At each position i in the sequence, these features represent the substrings appearing at positions i, i+1, .., i + k − 1. This feature type is useful for discovering species-specific functional signals, as well as

evolutionary conserved functional signals. For each position-specific k-mer we record the presence or absence of that feature in the neighborhood of the splice site. This results in a set of $(n - k + 1)\times 4k$ potential features for each value of k and sequence of length n.

Construction Method. This construction method starts with an initial set of position-specific k-mer features and extends them to a set of position-specific (k+1)-mers by appending the letters of the alphabet to each position-specific k-mer feature. As an example, suppose an initial set of 2-mers F initial = {$ac_2$, $cg_5$}, where the subscript denotes the starting position. F constructed = {$aca_2$, $acc_2$, $acg_2$, $act_2$, $cga_5$, $cgc_5$, $cgg_5$, $cgt_5$} is the extended set of position specific 3-mers. Incrementally, in this manner, we can construct level k + 1from level k.To capture the correlations between different nucleotides in non-consecutive positions in the sequence, we describe conjunctive position-specific features. We construct these complex features from conjunctions of basic position-specific features. This feature type is useful for discovering interacting functional signals in the sequence. The dimensionality of this kind of feature is inherently high. For each conjunctive positional feature, we record the presence or absence of that feature in the neighborhood of the splice site. For each iteration, if the number of conjuncts is k, we have a total of _nk_ × 4k such features for a sequence of length n.

**Construction Method:** We construct conjunctions of basic features by starting with an initial conjunction of basic features and adding another conjunct basic feature in an unconstrained position. Let our basic set be F basic = {$a_1$, $c_1$, . . . , $g_n$, $t_n$}, where a1 denotes nucleotide a at the first sequence position, and so on. If our initial set is F initial = {a1, g2}, we can extend it to the level 2 set of position-specific base combination.

So in this case we find that, we have these types of parameter:

Cosine(X,Y)= The cosine function for similarity.

(X *Y*)= The correlated set that has common number of element.

τ= The amount of least common elements.

α= Accuracy Threshold.

## 4.4 Comparing Existing Method:

Our previous chapter gives a few insight about result generation according to our proposal. Here we represent some of our results graphically to show the comparison between our method and the existing.

Table: Runtime on different data sample

| DNA Sample | Sequence length (thousands) | Compressed Data(%Byte) | Time Consumed(hrs) |
|---|---|---|---|
| Human(Homo Sapiens) | 3453 | +/- 8 | 2.5 |
| E. Coli | 52 | +/- 4 | 1 |
| Snake(Naja naja) | 896 | +/- 12 | 1.45 |
| Paddy (Oryza Sativa) | 256 | +/- 40 | 2(apprx) |
| Cow(Bos Indicus) | 2568 | +/- 21 | 2.5 |
| Moss(Asprizella F.) | 24 | +/- 1 | .57 |

We can also graphically represent the existing methods with our methods for different sirt of datasets.

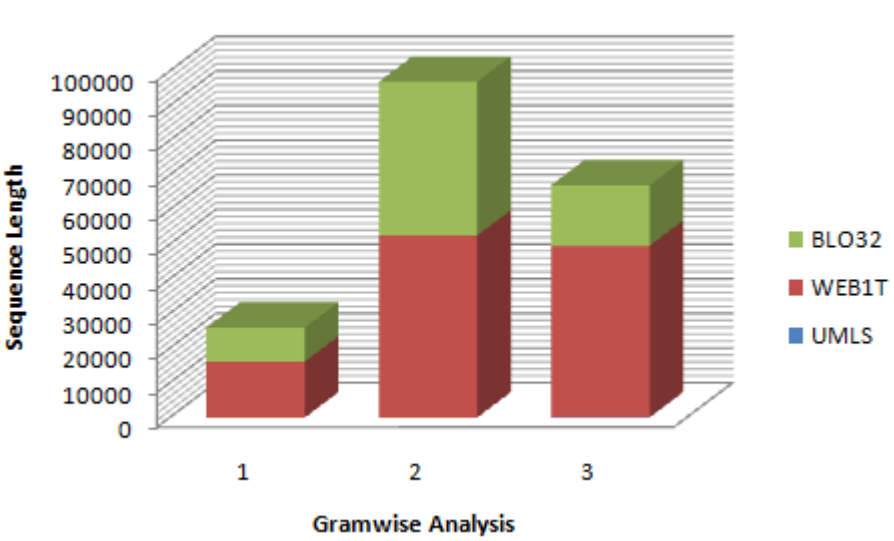
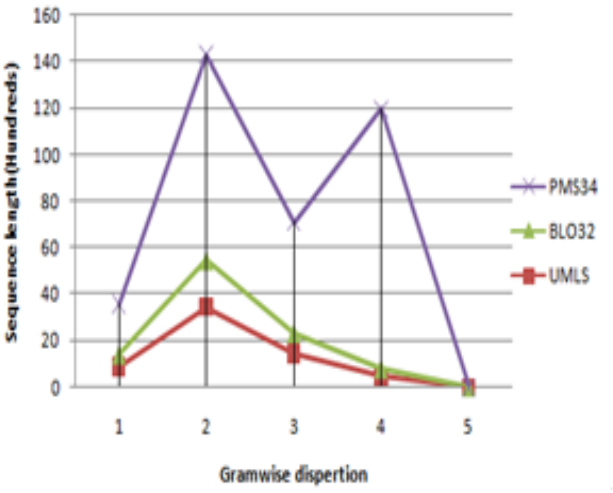



**Figure 5: Comparison between sequence length and gram-wise analysis.**

Here we are showing the gram wise analysis for the sequences. As we are taking a much bigger n-gram with every evaluation of the reference sequence in length more grams taken gives a good feedback whereas less gram analysis gives a result that is hazier. Below Run time analysis shows another situation where system derivatives and gram wise representation varies.



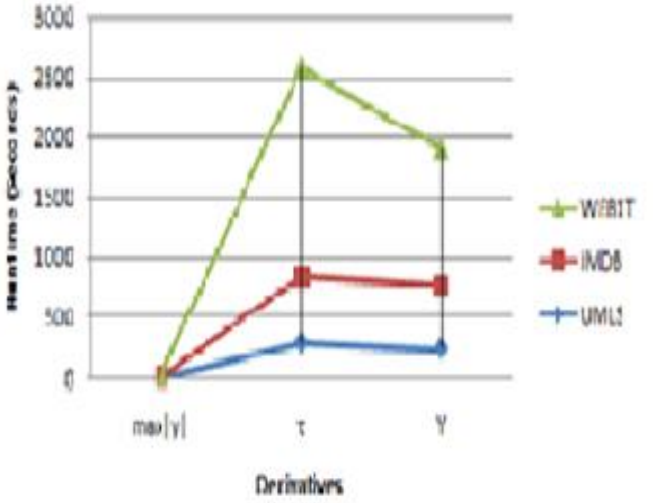




**Figure 6: Comparative runtime analysis for  Runtime for different methods.**

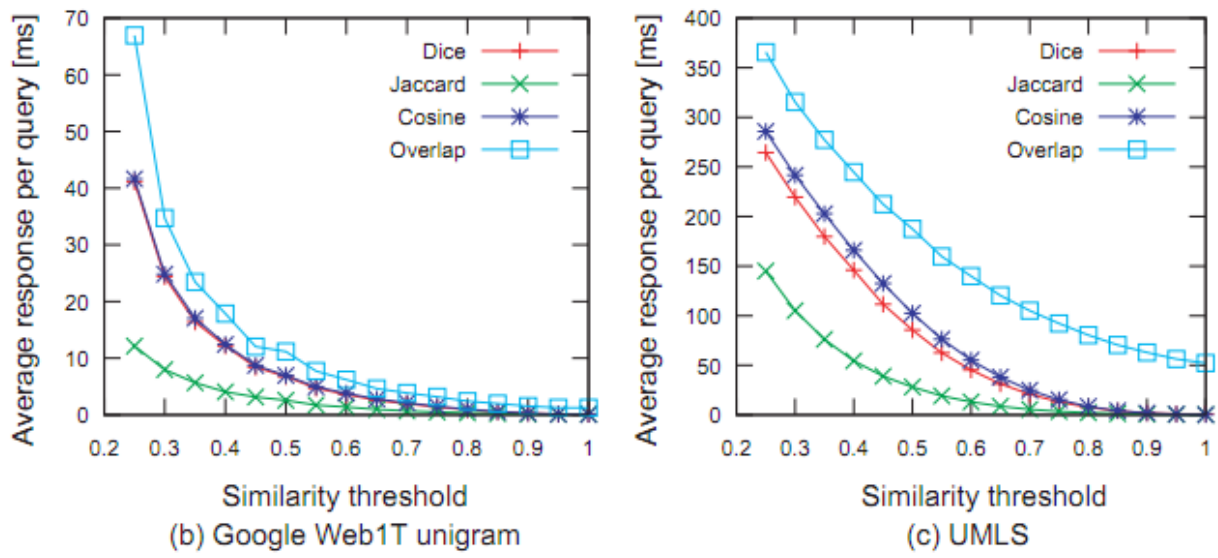



**Figure 7: Runtime analysis for some of our benchmark datasets:1.UMLS 2.WEB1T**

## Chapter 5. Conclusion:

We present a simple and efficient algorithm for approximate DNA sequence matching with the cosine measures. We conducted experiments of approximate DNA sequence matching on large-scale datasets with different DNA sequences for biolife and diseases. Even though the algorithm is very simple, our experimental results showed that the proposed algorithm executed very quickly. We also confirmed that the proposed method drastically reduced the number of candidate strings considered during approximate dictionary matching. We believe that this study will advance practical NLP applications for which the execution time of approximate dictionary matching is critical.

Our performance can be more fast if we can somehow bound our n-gram derivative to necessary level according to our τ overlap number and accuracy threshold α. In future we hope to pursue over this notions.

## References:


1. Andoni, Alexandr and PiotrIndyk. 2008. Near- optimal hashing algorithms for approximate nearest neighbor in high dimensions. Communications of the ACM, 51(1):117–122.
2. Arasu, Arvind, VenkateshGanti, and RaghavKaushik. 2006. Efficient exact set-similarity joins. In VLDB ’06: Proceedings of the 32nd International Conference on Very Large Data Bases, pages 918–929.
3. Behm, Alexander, ShengyueJi, Chen Li, and Jiaheng Lu. 2009. Space-constrained gram-based indexing for efficient approximate string search. In ICDE ’09: Proceedings of the 2009 IEEE International Conference on Data Engineering, pages 604–615.
4. Bergsma, Shane and GrzegorzKondrak. 2007. Alignment-based discriminative string similarity. In ACL ’07: Proceedings of the 45th Annual Meet- ing of the Association of Computational Linguistics, pages 656–663.
5. Bocek, Thomas, Ela Hunt, and Burkhard Stiller. 2007. Fast similarity search in large dictionaries. Technical Report ifi-2007.02, Department of Informatics (IFI), University of Zurich. 858Chandel, Amit, P. C. Nagesh, and SunitaSarawagi. 2006. Efficient batch top-k search for dictionary based entity recognition. In ICDE ’06: Proceed- ings of the 22nd International Conference on Data Engineering.
6. Charikar, Moses S. 2002. Similarity estimation tech- niques from rounding algorithms. In STOC ’02: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing, pages 380–388.
7. Chaudhuri, Surajit, VenkateshGanti, and RaghavKaushik. 2006. A primitive operator for similarity joins in data cleaning. In ICDE ’06: Proceedings of the 22nd International Conference on Data Engineering.
8. Cohen, William W., PradeepRavikumar, and Stephen E. Fienberg. 2003. A comparison of string distance metrics for name-matching tasks. In Proceedings of the IJCAI-2003 Workshop on Information Integration on the Web (IIWeb-03), pages 73–78.

9. Davis, Jason V., Brian Kulis, Prateek Jain, SuvritSra,and Inderjit S. Dhillon. 2007. Information-theoretic metric learning. In ICML ’07: Proceedings of the 24th International Conference on Machine Learning, pages 209–216.
10. Gravano, Luis, Panagiotis G. Ipeirotis, H. V. Jagadish,Nick Koudas, S.Muthukrishnan, and DiveshSrivastava. 2001. Approximate string joins in a database (almost) for free. In VLDB ’01: Proceedings of the27th International Conference on Very Large DataBases, pages 491–500.
11. Henzinger, Monika. 2006. Finding near-duplicate web pages: a large-scale evaluation of algorithms. In SIGIR ’06: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 284– 291.
12. Huynh, Trinh N. D., Wing-Kai Hon, Tak-Wah Lam,and Wing-Kin Sung. 2006. Approximate string matching using compressed suffix arrays. Theoreti-cal Computer Science, 352(1-3):240–249. Kim, Min-Soo, Kyu-Young Whang, Jae-Gil Lee, and Min-Jae Lee. 2005. n-Gram/2L: a space and time efficient two-level n-gram inverted index structure. In VLDB ’05: Proceedings of the 31st International Conference on Very Large Data Bases, pages 325– 336.
13. Lee, Hongrae, Raymond T. Ng, and Kyuseok Shim.2007. Extending q-grams to estimate selectivity of string matching with low edit distance. In VLDB ’07: Proceedings of the 33rd International Conference on Very Large Data Bases, pages 195–206.
14. Li, Chen, Bin Wang, and Xiaochun Yang. 2007. Vgram: improving performance of approximate queries on string collections using variable-length grams. In VLDB ’07: Proceedings of the 33rd International Conference on Very Large Data Bases, pages 303–314.
15. Li, Chen, Jiaheng Lu, and Yiming Lu. 2008. Effi-cient merging and filtering algorithms for approximate string searches. In ICDE ’08: Proceedings of the 2008 IEEE 24th International Conference onData Engineering, pages 257–266.
16. Liu, Xuhui, Guoliang Li, JianhuaFeng, and LizhuZhou. 2008. Effective indices for efficient approximate string search and similarity join. InWAIM’08:

Proceedings of the 2008 The Ninth International Conference on Web-Age Information Management, pages 127–134.

17. Manku, Gurmeet Singh, Arvind Jain, and AnishDas Sarma. 2007. Detecting near-duplicates for web crawling. In WWW ’07: Proceedings of the 16th International Conference on World Wide Web, pages 141–150.
18. Navarro, Gonzalo and Ricardo Baeza-Yates. 1998. Apractical q-gram index for text retrieval allowing errors. CLEI Electronic Journal, 1(2).Ravichandran, Deepak, Patrick Pantel, and Eduard Hovy. 2005. Randomized algorithms and nlp: using locality sensitive hash function for high speed noun clustering. In ACL ’05: Proceedings of the 43rd Annual Meeting on Association for Computa- tional Linguistics, pages 622–629.
19. Sarawagi, Sunitaand AlokKirpal. 2004. Efficient set joins on similarity predicates. In SIGMOD ’04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data, pages743–754.
20. Wang, Wei, Chuan Xiao, Xuemin Lin, and ChengqiZhang. 2009. Efficient approximate entity extraction with edit distance constraints. In SIGMOD ’09: Proceedings of the 35th SIGMOD International Conference on Management of Data, pages 759–770.
21. Winkler, William E. 1999. The state of record link-age and current research problems.Technical Report R99/04, Statistics of Income Division, Internal Revenue Service Publication.
22. Xiao, Chuan, Wei Wang, and Xuemin Lin. 2008. Ed-Join: an efficient algorithm for similarity joins with edit distance constraints. In VLDB ’08: Proceedings of the 34th International Conference on Very Large Data Bases, pages 933–944.