B.Sc. in Computer Science and Engineering Thesis

qPMS Sigma - An Efficient and Exact Parallel Algorithm for the Planted (l, d) Motif Search Problem

Submitted by

Saurav Dhar
201105008

Amlan Saha
201105054

Dhiman Goswami
201105057

Supervised by

Dr. Md. Abul Kashem Mia

Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology

Dhaka, Bangladesh

February 2017

CANDIDATES' DECLARATION

This is to certify that the work presented in this thesis, titled, "qPMS Sigma - An Efficient and Exact Parallel Algorithm for the Planted (l, d) Motif Search Problem", is the outcome of the investigation and research carried out by us under the supervision of Dr. Md. Abul Kashem Mia.

It is also declared that neither this thesis nor any part thereof has been submitted anywhere else for the award of any degree, diploma or other qualifications.

Saurav Dhar
201105008

Amlan Saha
201105054

Dhiman Goswami
201105057

CERTIFICATION

This thesis titled, **"qPMS Sigma - An Efficient and Exact Parallel Algorithm for the Planted (l, d) Motif Search Problem"**, submitted by the group as mentioned below has been accepted as satisfactory in partial fulfillment of the requirements for the degree B.Sc. in Computer Science and Engineering in February 2017.

Group Members:

Saurav Dhar

Amlan Saha

Dhiman Goswami

Supervisor:

Dr. Md. Abul Kashem Mia
Professor
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology

ACKNOWLEDGEMENT

First and foremost we would like to offer our deepest gratitude to our supervisor, Professor Dr. Md. Abul Kashem Mia for introducing us to the fascinating world of bioinformatics, supporting us throughout the full session of our thesis with his patience and knowledge. He shared his wisdom with us in analyzing subject matters, valued our thinking approach to synthesize those topics and encouraged us at the time of disappointment. His invaluable guidance, deep insights and suggestions drove us towards better ways of thinking about the problem. One simply could not wish for a better or friendlier supervisor.

We would like to sincerely thank Md. Arifuzzaman and Md. Shazibul Islam Shamim, who have previously worked in this particular field and shared the primitive study materials and gave effective guidelines whenever we needed. The level of patience and time given by them is really praiseworthy.

Finally, we want to dedicate the essence of our purest respect to our parents who have made our existences possible. They always believed in us even at the moment when we were losing our believes in ourselves. It is not really we who reached here but of course it is our kind parents who gave us hope and inspiration.

Dhaka
February 2017

Saurav Dhar

Amlan Saha

Dhiman Goswami

Contents

List of Figures

List of Tables

List of Algorithms

ABSTRACT

Motif finding is an important step for the detection of rare events occurring in a set of DNA or protein sequences. Extraction of information about these rare events can lead to new biological discoveries. Motifs are some important patterns that have numerous applications including the identification of transcription factors and their binding sites, composite regulatory patterns, similarity between families of proteins, etc. Although several flavors of motif searching algorithms have been studied in the literature, we study the version known as (l, d)-motif search or Planted Motif Search (PMS). In PMS, given two integers l, d and n input sequences we try to find all the patterns of length l that appear in each of the n input sequences with at most d mismatches. We also discuss the quorum version of PMS in our work that finds motifs that are not planted in all the input sequences but at least in q of the sequences. Our algorithm is mainly based on the algorithms qPMSPrune, qPMS7, TraverStringRef and PMS8. We introduce some techniques to compress the input strings and make faster comparison between strings with bitwise operations. Our algorithm performs a little better than the existing exact algorithms to solve the qPMS problem in DNA sequence. We have also proposed an idea for parallel implementation of our algorithm.

Chapter 1

Introduction

In this chapter we briefly discuss about motifs,what are they and why are we interested in them. Then we briefly describe the motif finding problem and various versions of it. We also give the formal description of the planted motif search for (l, d)- motifs. Then we describe different approaches of motif searching algorithms and why we chose planted version of motif search. At the end of the chapter we give a short review of the mostly known approximate and exact PMS algorithm.

1.1 What are Motifs

Motif means a pattern. A DNA motif is defined as a nucleic acid sequence that has some biological significance. By biological significance we mean it can be DNA binding sites for a regulatory protein that is a transcription factor. A DNA motif may be a small segment of nucleotide chain of A,T,C or G only 5 to 25 base pair long.

1.1.1 Importance of Studying Motifs

The discovery of rare event in DNA or protein sequence may lead new biological discoveries [1]. The presence of motifs is one kind of such rare events. Various biological processes such as gene expression may be controlled by motifs. Motifs are contained in regulatory regions of a genome such as promoters, enhancers, locus control regions etc [2]. Generally, proteins known as transcription factors regulate the expression of a gene by binding to locations of motifs in regulatory regions. For example transcription factors such as TFIID, TFIIA and TFIIB usually bind to sequence 5'-TATAAA-3' in the promoter region of a gene in order to initiate its transcription. Such motifs and their locations in regulatory regions like binding sites are important and helpful to uncover the regulatory mechanism of gene expression which is very

sophisticated. Motif search also has many applications in solving some crucial biological problems. For example, finding motifs in DNA sequences is very important for the determination of open reading frames, identification of gene promoter elements, location of RNA degradation signals and the identification of alternative splicing sites [3]. In protein sequences, patterns have led to domain identification, location of protease cleavage sites, identification of signal peptides, protein interactions, determination of protein degradation elements, identification of protein trafficking elements etc. So we can agree to the fact that, motif identification plays an important role in biological studies.

1.1.2 Regulatory Regions and Transcription Factor Binding Sites

DNA regions involved in transcription and transcriptional regulation are called regulatory regions (RR). Every gene contains a RR typically stretching 100-1000 bp upstream of the transcriptional start site. A transcription factor (TF) is a protein that can bind to DNA and regulate gene expression. TFs influence gene expression by binding to a specific location in the respective genes regulatory region which is known as TFBSs or motifs. TFBSs are a part of either the promoter or enhancer region of a gene.

1.2 The Motif Finding Problem

Our target is to identify transcription factor binding sites or motif from a DNA sequence. We are given a set of DNA sequences of specified length and the length of the motif. We choose a position from every sequence which will be the starting sequence for the motif candidates Figure 1.1. Taking these sequences we make alignment matrix, profile matrix and finally consensus string. Then we evaluate the consensus string with the help of a scoring function. Now, we have to find the starting position for which the score of the consensus string will be maximum. It is the informal description of the motif searching problem.

1.2.1 Classification of Motif Searching Algorithms

For the importance of motifs, various motif finding algorithms have been researched and applied for the last two decades. We can classify these algorithms in to three categories. All motif searching algorithms follow one of the three approaches:

- **Combinatorial Approach :** Tries to explore exhaustively all the ways that a molecular process could happen thus leads to hard combinatorial problems for which efficient algorithms are required.

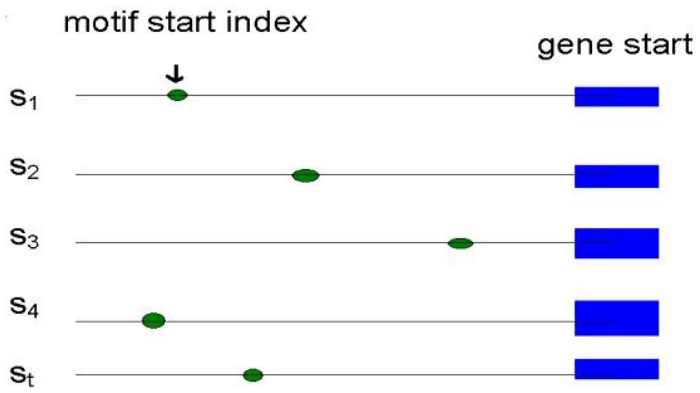

Figure 1.1: Starting index of motifs in a DNA sequence .

- **Probabilistic Approach :** Makes certain decisions randomly by extending the classical model of deterministic algorithms. They are often faster, simpler and more elegant than their combinatorial counterparts.
- **Phylogenetic Footprinting Approach :** Discovers regulatory elements in a set of orthologous regulatory regions from multiple species by identifying the best conserved motifs in those orthologous regions.

1.2.2 Combinatorial Approach

Among the various approaches combinatorial approach that has proven to be more accurate than the others. In our approach we will use combinatorial algorithms for the discovery of motifs. The are several versions of this approach. Mosty used three versions are:

- Planted Motif Search (PMS)
- Simple Motif Search (SMS)
- Edited-distance-based Motif Search (EMS) [4]

1.2.3 Planted Motif Search (PMS)

Among different versions of combinatorial approaches the PMS problem is more popular due to its closeness to motif reality. Motifs typically occur with mutations at binding sites. The binding sites are referred to as instances of a motif. A motif in PMS is referred to as a (l, d)-motif where l is its length and d is the maximum number of mutations allowed for its instances. Given a set of n sequences, the PMS Algorithm tries to find all the (l, d)-motifs in them. The PMS problem is essentially the same as the closest substring problem.

1.2.4 Quorum Planted Motif Search (qPMS)

A generalized version of the PMS Problem namely Quorum Planted Motif Search (qPMS) problem. The qPMS problem is to find all the motifs that have motif instances present in q out of the n input sequences. This version captures the nature of motifs more precisely than the PMS problem because in practice some motifs may not have motif instances in all of the input sequences. qPMS algorithms can be used to find DNA motifs and protein motifs as well as transcription factor binding sites.

1.2.5 Formal Definitions of qPMS

Definition 1 A string $x = x[1] \dots x[l]$ of length l is called an l-mer.

Definition 2 Given two strings $x = x[1] \dots x[l]$ and $s = s[1] \dots s[m]$ with $l < m$ we say $x \epsilon_l s$ if there exists $1 \leq i \leq l - m + 1$ such that $x[j] = s[l - m + 1]$ for every $1 \leq j \leq l$. We also say that x is an l-mer in s.

Definition 3 Given two strings $x = x[1] \dots x[l]$ and $y = y[1] \dots y[l]$ of equal length, the Hamming distance between x and y denoted by $d_H(x, y)$ is the number of mismatches between them. In other words, $d_H(x, y) = \Sigma_{1 \leq i \leq l} I_i$, where I_i is the indicator at position i. $I_i = 1$ if $x[i] \neq y[i]$ and $I_i = 0$ otherwise.

Definition 4 Given two strings x and s with $|x| < |s|$, the Hamming distance between x and s, denoted by $d_H(x, s)$ is $\min_{y \epsilon_{|x|} s} d_H(x, y)$.

Definition 5 Given a set of n strings $s_1, \dots, s_n$ of length m each, a string M of length l is called an (l, d, q)-motif of the strings if there are at least q out of the n strings such that the Hamming distance between each one of them and M is no more than d.

Definition qPMS Given n input strings $s_1, \dots, s_n$ of length m each, three integer parameters l, d and q, find all the (l, d, q)-motifs of the input strings. The PMS problem is a special case of the qPMS problem when $q = n$.

1.3 Literature Review

During the last two decades various types of PMS algorithm has been proposed in the literature. There are mainly two types of algorithms for PMS problem. They are *exact* and *approximate* algorithms. An exact algorithm always finds all the (l, d)-motifs present in the input sequences. While, an approximate algorithm may not find all the motifs.

1.3.1 Exact PMS Algorithms

An exact algorithm can find all the motifs in input sequences. In our research paper, we only consider exact algorithms. The exact variant of the PMS problem has been shown to be NP-hard [5]. It means that there is no PMS algorithm that takes only polynomial time to iterate. As a result, all known exact algorithms have an exponential worst case runtime. Due to NP-hardness approximate algorithm arises for PMS. An exact PMS algorithm can be built using two approaches:

Sample Driven Approach

For all $(m - l + 1)^n$ possible combinations of l-mers coming from different strings generate the common neighborhood. Unlike pattern driven approach in *Sample Driven (SD)* approaches all possible motifs generated from the l-mers in input strings are of interest which could be found in polynomial time. *Sample Driven* algorithms try to find comparative patterns by comparing the given length strings and looking for local similarities between them. They are based on constructing a local multiple alignment of the given non-coding DNA sequences and then extracting the comparative patterns from the alignment by combining the segments which is common to most of the non-coding DNA sequences.

Pattern Driven Approach

For all $|\Sigma|^l$ possible l-mers check which are motifs. Trying all possible motif candidates take exponential space. *Pattern Driven (PD)* algorithms are based on enumerating candidate patterns in a given length string and inputting substrings with high fitness. Compared with *SD* algorithms, *PD* algorithms can be performed intelligently so that patterns are not present in the data that are not generated.

All existing exact algorithms solve PMS problem in exponential time in some of its parameters. The most recent exact algorithms that have been proposed in the literature are Algorithm *qPMS9* due to [6], Algorithm *PMS8* due to [7], Algorithm *Pampa* due to [8], Algorithm *qPMS7* due to [1], Algorithm *PMSPrune* due to [9], Algorithm *PMS6* due to [10], Algorithm *Voting* due to [11], and Algorithm *RISSOTO* due to [12].

1.3.2 Approximate PMS Algorithms

Approximate PMS algorithms usually tend to be faster than exact PMS algorithms. Typically, approximate PMS algorithms employ heuristics such as local search, Gibbs sampling, expectation optimization etc. Some examples of approximate algorithms are Algorithm *MEME* due

to [13], Algorithm *PROJECTION* due to [14], Algorithm *Gibbs DNA* due to [15], Algorithm *WINNOWER* due to [16], and Algorithm *Random Projection* due to [17]. Some other approximate PMS algorithms are Algorithm *MULTIPROFILER* due to [18], Algorithm *PatternBranching* due to [19] and Algorithm *CONSENSUS* due to [20].

Algorithm MEME

The MEME algorithm extends the EM algorithm for identifying motifs in unaligned sequences. While a drawback of EM is that the maximum it finds is only local, MEME can either favor motifs that appear exactly once or appear zero or once in each sequence in a training set, or give no preference to a number of occurrences. In 2005 *Hall et al* acquired a set of correlated genes from genomic, transcriptomic and proteomic analyses. They applied MEME to scan 1000 bp of the 3' end of stop codon, where a 47 bp motif was found in six of the analyzed sequences. Then it was used to search the entire genome and 20 additional genes were identified to have the same motif. This motif was known to be bound to Puf protein, implying that Puf protein may control the transcription of the analyzed genes.

Algorithm Projection

This algorithm ameliorates the limitations of existing algorithms by using random projections of input. It extends previous projection-based searching techniques to solve a multiple alignment problem that is not effectively addressed by pairwise alignments. It is designed to efficiently solve the problems from the planted (l, d)-motif model and can do more reliably and substantially difficult instances than previous algorithms. For $t = 20$ and $n = 600$ this algorithm achieves performance close to the best possible, being limited primarily by statistical considerations.

Algorithm Winnower, SP-STAR, and cWinnower

Winnower first represents motif instances as vertices then it tries to delete spurious edges and recover motifs with the remaining vertices. SP-STAR is a local sum of pairwise score improvement algorithm which considers only the subsequences present in dataset and iteratively updates scores of the motifs. cWinnower improves its running time by a stronger constraint function.

Chapter 2

Preliminaries

In this chapter we have briefly described the necessary topics we need to know before we go into Motifs. We have started with the most fundamental part of life cells. Then from the cells, we have gone deeper into the genetic materials the genes, DNA, RNA and Proteins. We have discussed about the flow of information from the DNA to the Protein in detailed steps. We have illustrated the motif finding problem and classify different types of motifs. Later, we have shown building the consensus string from the alignment matrix and profile matrix. We have also discussed a consensus string evaluating function. We have concluded this chapter by giving a very simple brute-force motif finding algorithm.

2.1 What is Life Made of

In 1665 Robert Hooke discovered that every organisms in living bodies are composed of individual compartments known as cells. With this huge discovery in the field of Biology the study of life became the study of cells.

2.1.1 Cells

A cell is the smallest structural unit of an organism that is capable of independent functioning. Moreover, a cell as a complex mechanical system with many moving parts which not only stores all the necessary information to make a complete replica of itself but also contains all the machinery required to collect and manufacture its components, carry out the copying process, and start its new offspring. So, cells are the fundamental working units of every living system.

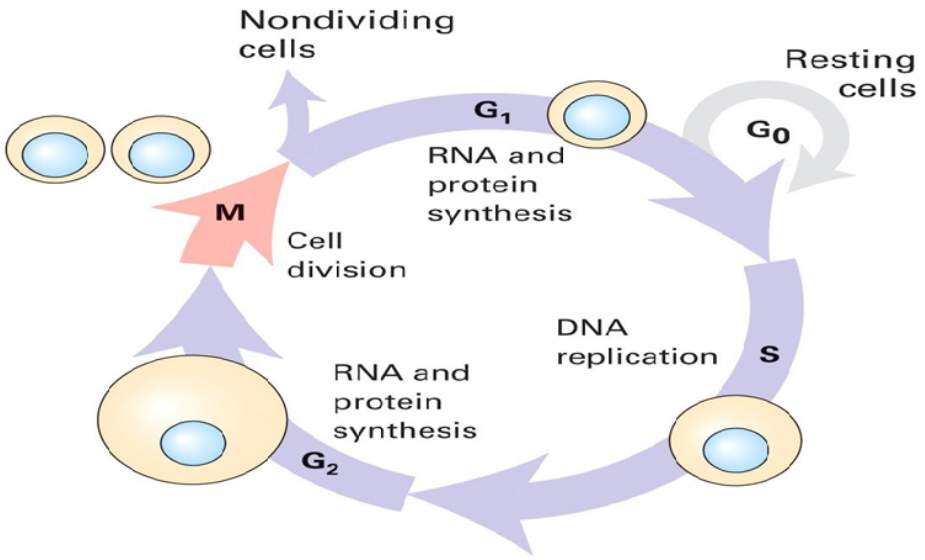

Figure 2.1: Life cycle of a cell.

2.1.2 Life Cycle of a Cell

A great diversity of cells exist in nature, but they all have some common features. All cells have a life cycle: they are born, eat, replicate, and die illustrated in Figure 2.1

2.1.3 Types of Cells

We can differentiate cells into two types based on their structures. They are called *Prokaryotic* and *Eukaryotic* cells. Prokaryotic cells do not contain a nucleus or any other membrane-bound organelle. Only contain one piece of circular DNA and no mRNA post transactional modification. Bacteria are an example of prokaryotes. Eukaryotic cells contain membrane-bound organelles, including a nucleus. They contain multiple chromosomes. Eukaryotes can be single-celled or multi-celled such as humans, plants and fungi.

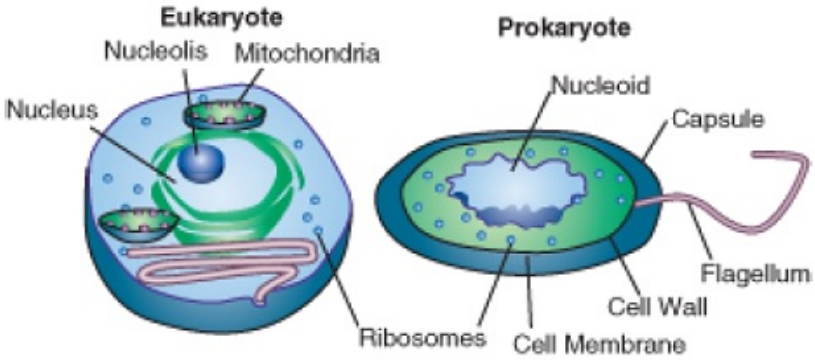

Figure 2.2: Prokaryotic and Eukaryotic cells.

2.2 Genetic Material of Life

All the genetic materials of a organism is called Genome. The genome is an organisms complete set of DNA, including all of its genes. Each genome contains all of the information needed to build and maintain that organism. They includes both the genes and non-coding sequences of the DNA. A bacteria genome contains about 600,000 DNA base pairs while human and mouse genomes have some 3 billion. Different organisms have different numbers of chromosomes, suggesting that they might carry information specific for each species. Human genome has 46 (23 pairs) distinct chromosomes. Each chromosome contains many genes.

2.2.1 Genes

Genes are discrete units of hereditary information located on the chromosomes and consisting of DNA. Gregor Mendel's experiments with garden peas suggested the existence of genes that were responsible for inheritance. Genes are small sections of DNA within the genome that code for proteins. They contain the instructions for our individual characteristics- like eye and hair color. The human genome contains approximately 25000 protein-coding genes. So, we can say that the main job of genes-

- Storing information about the characteristics.
- They express the genotype and phenotypes.
- The main task of genes is to produce proteins.
- Each gene contains the information required to build specific proteins needed in an organism.

2.2.2 DNA

DNA or deoxyribonucleic acid, is the hereditary material in humans and almost all other organisms. Nearly every cell in a persons body has the same DNA. DNA was discovered by Johann Friedrich Miescher by isolating a substance he called *nuclein* from the nuclei of white blood cells. The information in DNA is stored as a code made up of four chemical bases: *Adenine* (A), *Guanine* (G), *Cytosine* (C), and *Thymine* (T). Human DNA consists of about 3 billion bases, and more than 99 percent of those bases are the same in all people.

DNA bases pair up with each other, A with T and C with G, to form units called base pairs. Each base is also attached to a sugar molecule and a phosphate molecule. Together, a base, sugar, and phosphate are called a nucleotide. Nucleotides are arranged in two long strands that

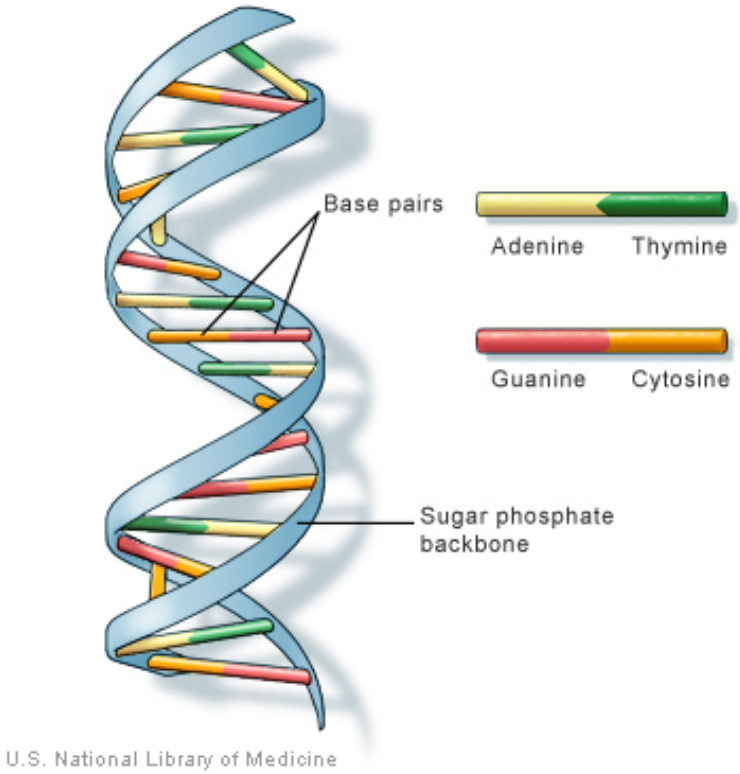

Figure 2.3: DNA double helix formed by base pairs attached to a sugar-phosphate backbone.

form a spiral called a double helix. The structure of the double helix is somewhat like a ladder, with the base pairs forming the ladder's rungs and the sugar and phosphate molecules forming the vertical sidepieces of the ladder.

Although,DNA has a double helix structure, it is not symmetric. It has a "forward" and "backward" direction. The ends are labeled 5' and 3' after the Carbon atoms in the sugar component like 5' AATCGCAAT 3'. DNA always reads 5' to 3' for transcription replication. DNA is a polymer of Sugar-Phosphate-Base. Bases held together by *Hydrogen* bonding to the opposite strand. Base pairs of G and C contain three hydrogen bonds and base pairs of A and T contain two hydrogen bonds. As a reason, G-C base-pairs are more stable than A-T base pairs.

2.2.3 RNA

RNA or ribonucleic acid, is similar to DNA chemically. It usually has only a single strand. RNA contains ribose while DNA contains deoxyribose. Deoxyribose lacks one oxygen atom. RNA contains the bases *Adenine* (A), *Uracil* (U) instead of *Thymine* in DNA, *Cytosine* (C) and *Guanine* (G). Unlike DNA, RNA comes in a variety of shapes and types. While DNA contains double helix, RNA may be of more than one type. RNA is usually single-stranded, while DNA is usually double-stranded. Some forms of RNA can form secondary structures by pairing up

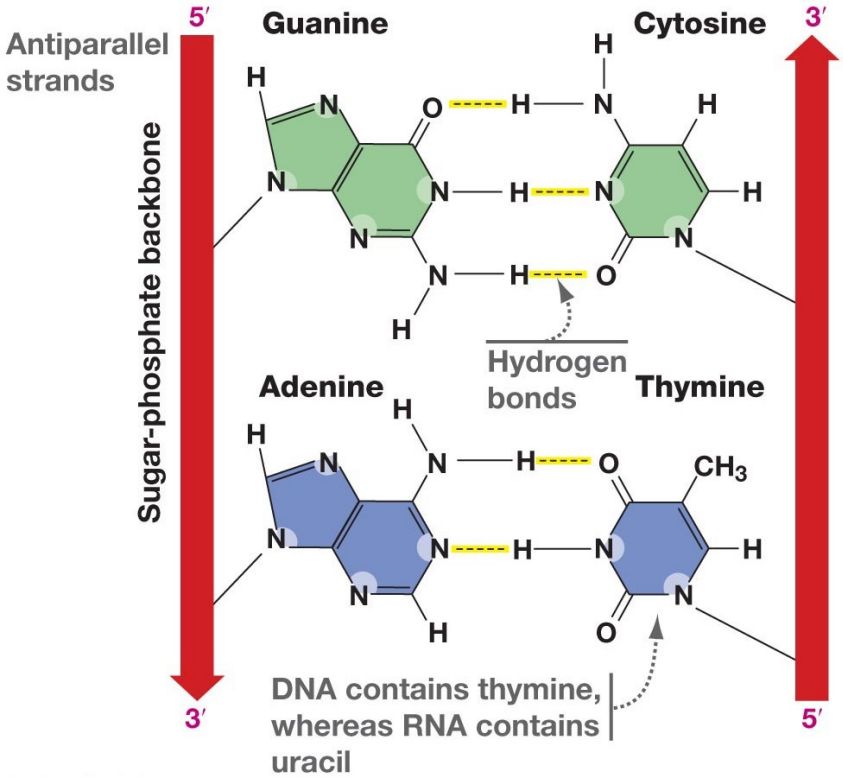

Figure 2.4: Hydrogen bonds between A-T and C-G.

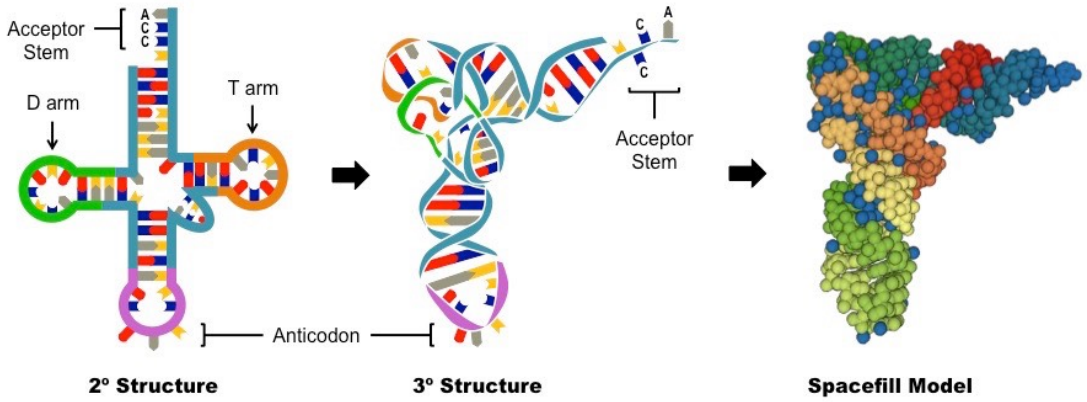

Figure 2.5: tRNA linear and comples structure.

with itself. This can have change it's properties. DNA and RNA can also pair with each other. RNA molecules are involved in protein synthesis and sometimes in the transmission of genetic information.

Several types of RNA exists. They can be classified by various functions like:

- **mRNA - Messenger RNA :** Encodes amino acid sequence of a polypeptide. When a Bioinformatician says "RNA" it usually means mRNA.
- **tRNA - Transfer RNA :** Brings amino acids to ribosomes during translation.

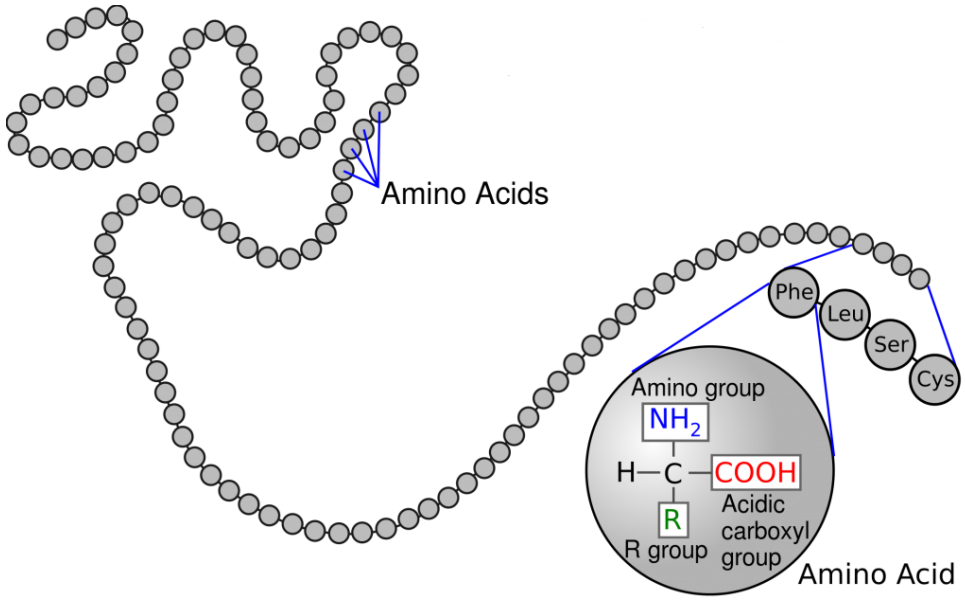

Figure 2.6: Primary protein structure - chain of amino acids.

- **rRNA - Ribosomal RNA :** With ribosomal proteins, makes up the ribosomes, the organelles that translate the mRNA.

2.2.4 Protein

Protein is a highly complex substance that is present in all living organisms. They are responsible for most of the complex functions that make life possible. They are polymer of smaller subunits called amino acids. Also called “poly-peptides”. There are twenty amino acids, each coded by three-base-sequences in DNA called “codons” whish are degenerate. Different chemical properties cause the protein chains to fold up into specific three-dimensional structures that define their particular functions in the cell.

Proteins do all essential work for the cell like:

- Build cellular structures.
- Digest nutrients.
- Execute metabolic functions.
- Mediate information flow within a cell and among cellular communities.

2.2.5 Mutation

Mutation is the permanent alteration of the nucleotide sequence of the genome of an organism or DNA or other genetic elements. Mutations result from errors during DNA replication or

other types of damage to DNA, which then may undergo error-prone repair or cause an error during other forms of repair or else may cause an error during replication. Mutations can serve the organism in three ways:

- The Good : A mutation can cause a trait that enhances the organisms function. For example mutation in the sickle cell gene provides resistance to malaria.
- The Bad : A mutation can cause a trait that is harmful, sometimes fatal to the organism. For example Huntingtons disease, a symptom of a gene mutation is a degenerative disease of the nervous system.
- The Silent : A mutation can simply cause no difference in the function of the organism.

2.3 What Carries Information Between DNA to Proteins

The information for making proteins is stored in DNA. There is a process (transcription and translation) by which DNA is converted to protein. By understanding this process and how it is regulated we can make predictions and models of cells.

2.3.1 Central Dogma of Molecular Biology

The paradigm that DNA directs its transcription to RNA, which is then translated into a protein is called central dogma of molecular biology. It was first proposed by Francis Crick. The central dogma of molecular biology explains the flow of genetic information, from DNA to RNA, to make a functional product, a protein.

The central dogma states that the pattern of information that occurs most frequently in our cells is:

- From existing DNA to make new DNA (DNA replication).
- From DNA to make new RNA (transcription).
- From RNA to make new proteins (translation).

2.3.2 DNA $\rightarrow$ RNA: Transcription

Transcription is the process by which DNA is copied to mRNA, which carries the information needed for protein synthesis. Transcription takes place in two broad steps. First, pre-messenger

Figure 2.7: Central dogma of molecular biology.

RNA is formed, with the involvement of RNA polymerase enzymes. The process relies on Watson-Crick base pairing, and the resultant single strand of RNA is the reverse-complement of the original DNA sequence. The pre-messenger RNA is then edited to produce the desired mRNA molecule in a process called RNA splicing.

Formation of pre-messenger RNA

The mechanism of transcription has parallels in that of DNA replication. As with DNA replication, partial unwinding of the double helix must occur before transcription can take place, and it is the RNA polymerase enzymes that catalyze this process. Unlike DNA replication, in which both strands are copied, only one strand is transcribed. The strand that contains the gene is called the sense strand, while the complementary strand is the antisense strand. The mRNA produced in transcription is a copy of the sense strand, but it is the antisense strand that is transcribed.

Ribonucleotide triphosphates (NTPs) align along the antisense DNA strand, with Watson-Crick base pairing (A pairs with U). RNA polymerase joins the ribonucleotides together to form a pre-messenger RNA molecule that is complementary to a region of the antisense DNA strand. Transcription ends when the RNA polymerase enzyme reaches a triplet of bases that is read as a “stop” signal. The DNA molecule re-winds to re-form the double helix. The Figure 2.8 illustrates the process.

Splicing

In Eukaryotic cells, RNA is processed between transcription and translation. Unprocessed RNA is composed of *Introns* and *Extrons*. The pre-mRNA is chopped up to remove the introns and create mRNA in a process called RNA splicing. Sometimes alternate RNA processing can lead to an alternate protein as a result.

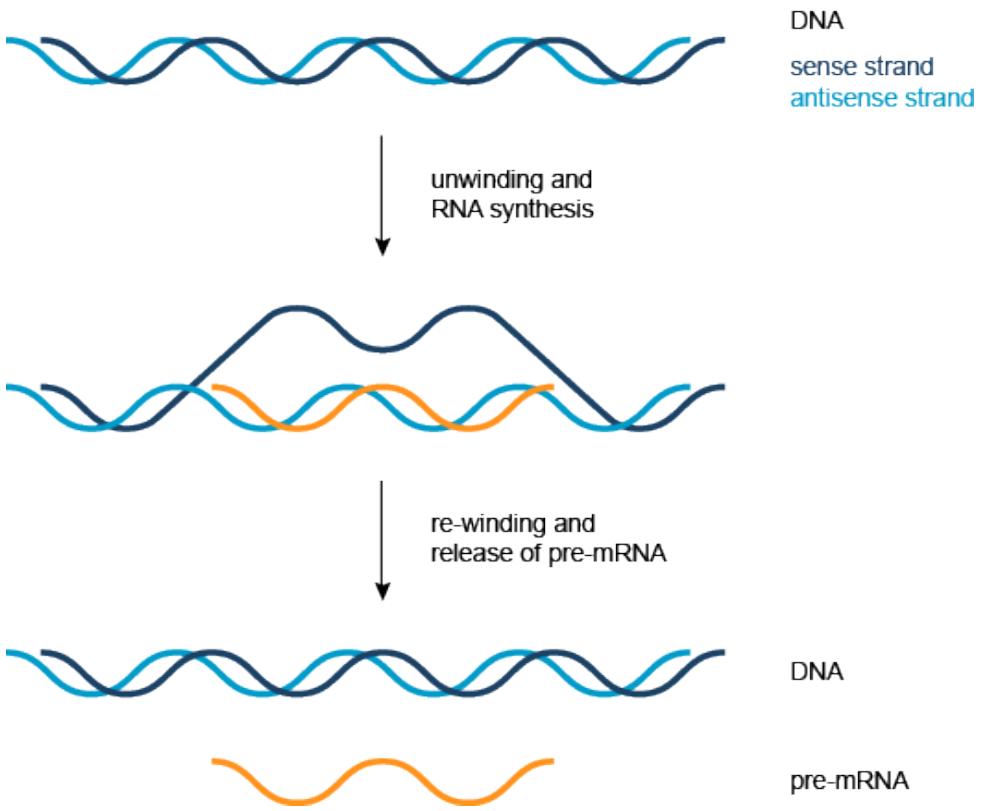

Figure 2.8: Formation of pre-messenger RNA.

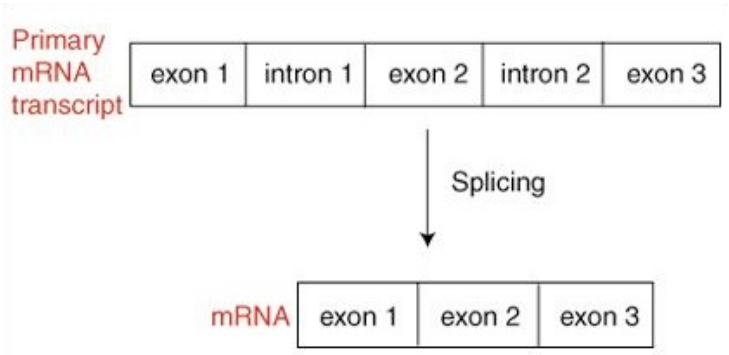

Figure 2.9: RNA splicing: Introns are spliced from the pre-mRNA to give mRNA.

2.3.3 RNA → Protein: Translation

There are twenty types of tRNAs, and twenty types of amino acids. Each type of amino acid binds to a different tRNA, and the tRNA molecules have a three-base segment called an anti-codon that is complementary to the "codon" in the mRNA. As in DNA base-pairing, the anti-codon on the tRNA sticks to the codon on the RNA, which makes the amino acid available to the ribosome to add to the polypeptide chain. When one amino acid has been added, the ribosome shifts one codon to the right, and the process repeats. The process of turning an mRNA into a protein is called translation, since it translates information from the RNA into the protein.

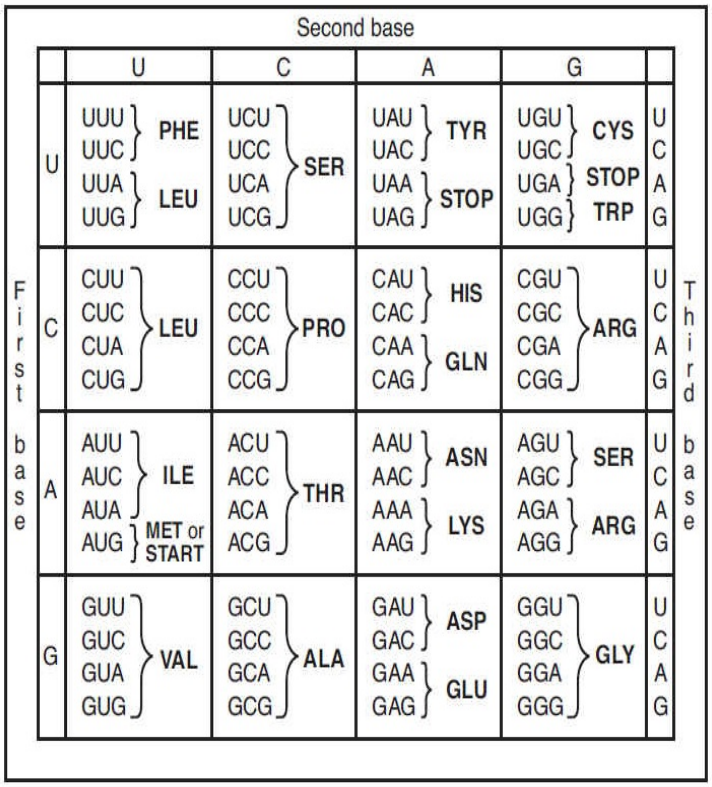

Figure 2.10: The genetic code, from the perspective of mRNA.

2.4 Motifs in Details

Motif usually means a pattern. For any sequence of objects to be called a pattern, it has to have at least more than one instance. To define more precisely, a motif should have significantly higher occurrence in a given array or list compared to what you would obtain in a randomized array of same components. Here array means an arranged set such as a genome or a protein structure. Genome is an array of nucleotides and protein structure is an array of amino acids.

2.4.1 Sequence Motifs

By sequence motif we mean a pattern of nucleotide or amino acid that has specific biological significance. In other words, they are pattern of a DNA array or protein sequence. They may be also called regulatory sequence motifs. This kind of motifs are commonly studied in bioinformatics. They are becoming increasingly important in the analysis of gene regulation. In our thesis we will only discuss about this type of motifs.

2.4.2 Structural Motifs

Structural motifs are a super-secondary structure in a chain like biological molecules such as protein. It is formed by three dimensional alignment of amino acids which may not be adjacent.

2.4.3 Regulatory Motifs in DNA

Regulatory motifs are some short nucleotide sequence that regulates the expression of genes such as controlling the situations under which the genes will be turned on or off. For example: [21] Fruit flies have a small set of *immunity genes* that are dormant in their genome. But when its organisms get infected somehow the genes got switched on. It turns out that many immunity genes in the fruit fly genome have strings that are reminiscent of TCGGGGATTTCC, located upstream of the genes start. This string are called $_{NF-\kappa}$B binding sites which are important examples of regulatory motifs. Proteins known as *transcription factors* bind to these motifs, encouraging RNA polymerase to transcribe the downstream genes.

2.4.4 Regulatory Regions

As discussed in [22] regulation of gene expression is an essential part of every organism. Certain regions, called cis-regulatory elements, on the DNA are footprints for the trans-acting proteins involved in transcription. DNA regions involved in transcription and transcriptional regulation are called regulatory regions (RR). Every gene contains a RR typically stretching 100-1000 bp upstream of the transcriptional start site. In Figure 2.11 we see the structure of a eukaryotic protein coding gene. Regulatory sequence controls when and where expression occurs for the protein coding region (red). Promoter and enhancer regions (yellow) regulate the transcription of the gene into a pre-mRNA which is modified to remove introns (light grey) and add a 5' cap and ploy-A tail (dark grey). The mRNA 5' and 3' untranslated regions (blue) regulate translation into the final protein product.

2.4.5 Transcription Factor Binding Sites

A transcription factor (TF) is a protein that can bind to DNA and regulate gene expression. The region of the gene to which TF binds is called a transcription factor binding site (TFBS). TFs influence gene expression by binding to a specific location in the respective genes regulatory region which is known as TFBSs or motifs. [23] TFBS can be located anywhere within the RR. TFBS may vary slightly across different regulatory regions since non-essential bases could mutate. TFBSs are a part of either the promoter or enhancer region of a gene. A promoter sits upstream and contains three important regions- the regulatory protein binding site, the tran-

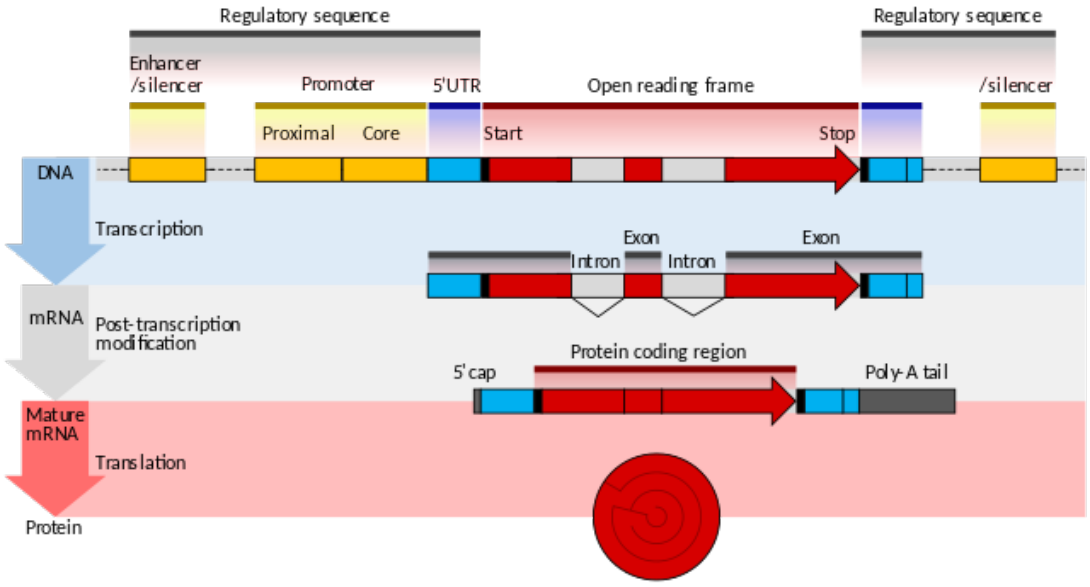

Figure 2.11: The structure of a eukaryotic protein-coding gene.

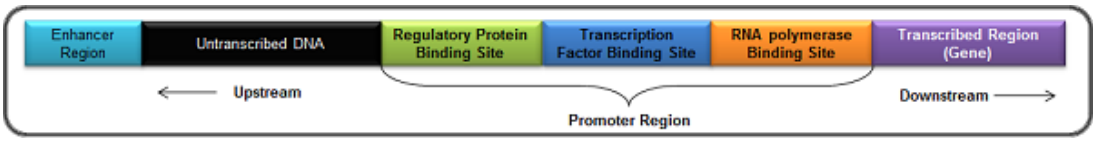

Figure 2.12: Transcription Factor Binding Sites.

scription factor binding site, and the RNA polymerase binding site. In Figure 2.12 we see a picture of promoter and enhancer. An enhancer is usually far upstream of a gene.

2.5 To The Motif Finding Problem

Our target is to identify a motif from a DNA sequence. In our study we only deal with transcription factor binding sites. We will give the formal definition of the motif finding problem later. Assume that we are given some DNA sequence of nucleotides which are generated randomly. Now, we want to find a secret pattern that occurs at least one time in every given DNA sequence without any prior knowledge about how it looks like. This pattern is our motif.

Some complications encountered while finding motifs:

- We do not know the motif sequence.
- Hence we donot know what to search for in the DNA sequence.
- We do not know where the motifs are located relative to the starting index of the sequence.
- Motifs can differ in one or more positions in their sequence which is called mutations.

CGGGGCTATGCAACTGGGTCGTCACATTCCCCTTTCGATA
TTTGAGGGTGCCCAATAAATGCAACTCCAAAGCGGACAAA
GGATGCAACTGATGCCGTTTGACGACCTAAATCAACGGCC
AAGGATGCAACTCCAGGAGCGCCTTTGCTGGTTCTACCTG
AATTTTCTAAAAAGATTATAATGTCGGTCCATGCAACTTC
CTGCTGTACAACTGAGATCATGCTGCATGCAACTTTCAAC
TACATGATCTTTTGATGCAACTTGGATGAGGGAATGATGC

Figure 2.14: The same DNA sequences of Figure 2.13 with the implanted pattern ATGCAACT.

CGGGGCTATcCAgCTGGGTCGTCACATTCCCCTTTCGATA
TTTGAGGGTGCCCAATAAggGCAACTCCAAAGCGGACAAA
GGATGgAtCTGATGCCGTTTGACGACCTAAATCAACGGCC
AAGGAaGCAACcCCAGGAGCGCCTTTGCTGGTTCTACCTG
AATTTTCTAAAAAGATTATAATGTCGGTCCtTGgAACTTC
CTGCTGTACAACTGAGATCATGCTGCATGCcAtTTTCAAC
TACATGATCTTTTGATGgcACTTGGATGAGGGAATGATGC

Figure 2.15: Same as Figure 2.14 with the implanted pattern ATGCAACT randomly mutated in two positions.

- How to discern functional motifs from random ones?

The idea is illustrated in the Figure 2.13, Figure 2.14 and Figure 2.14.

CGGGGCTGGGTCGTCACATTCCCCTTTCGATA
TTTGAGGGTGCCCAATAACCAAAGCGGACAAA
GGGATGCCGTTTGACGACCTAAATCAACGGCC
AAGGCCAGGAGCGCCTTTGCTGGTTCTACCTG
AATTTTCTAAAAAGATTATAATGTCGGTCCTC
CTGCTGTACAACTGAGATCATGCTGCTTCAAC
TACATGATCTTTTGTGGATGAGGGAATGATGC

Figure 2.13: Seven random sequences.

2.5.1 Alignment Matrix

To formulate the motif finding problem we first need to define what we mean by motif. As we saw in Figure 2.15 there may be mismatch in some positions of the motifs. Now consider a set of t DNA sequences $S_1, S_2, \ldots, S_t$. Each of which has n nucleotides. We select one position in each of these t sequences, thus forming an array $s = (s_1, s_2, \ldots, s_t)$, where $1 \leq s_i \leq n - l + 1$.

```
                                CGGGGCTATcCAgCTGGGTCGTCACATTCCCCTT...
                        TTTGAGGGTGCCCAATAAggGCAACTCCAAAGCGGACAAA
                                    GGATGgAtCTGATGCCGTTTGACGACCTA...
                                AAGGAaGCAACcCCAGGAGCGCCTTTGCTGG...
AATTTTCTAAAAAGATTATAATGTCGGTCCtTGgAACTTC
    CTGCTGTACAACTGAGATCATGCTGCATGCcAtTTTCAAC
                        TACATGATCTTTTGATGgcACTTGGATGAGGGAATGATGC
```

Figure 2.16: Superposition of the seven highlighted 8-mers from Figure 2.14.

Alignment		A	T	C	C	A	G	C	T
		G	G	G	C	A	A	C	T
		A	T	G	G	A	T	C	T
		A	A	G	C	A	A	C	C
		T	T	G	G	A	A	C	T
		A	T	G	C	C	A	T	T
		A	T	G	G	C	A	C	T
Profile	**A**	5	1	0	0	5	5	0	0
	T	1	5	0	0	0	1	1	6
	G	1	1	6	3	0	1	0	0
	C	0	0	1	4	2	0	6	1
Consensus		A	T	G	C	A	A	C	T

Figure 2.17: The alignment matrix, profile matrix and consensus string formed from the 8-mers starting at positions $s = (8, 19, 3, 5, 31, 27, 15)$ in Figure 2.14.

Here l is the length of the pattern. The superposition of the l-mers of Figure 2.15 is shown in Figure 2.16. Now taking the l-mers starting at these positions we can compile a *alignment matrix* of $t \times l$. The (i, j)th element of the *alignment matrix* is the nucleotide at the $s_i + j - 1$th position in the ith sequence. Illustrations are shown in Figure 2.17.

2.5.2 Profile Matrix

The *profile matrix* or profile, illustrates the variability of nucleotide composition at each position for a particular group of l-mers. Let the alphabet Σ in our motif search. Then the size of the *profile matrix* will be $|\Sigma| \times l$. For DNA $|\Sigma| = 4$. Based on the *alignment matrix*, we can compute the $4 \times l$ *profile matrix* whose (i, j)th element holds the number of times nucleotide i appears in column j of the *alignment matrix*, where i varies from 1 to 4. In Figure 2.17 we see that positions 3, 7, and 8 are highly conserved, while position 4 is not.

2.5.3 Consensus String

To further summarize the profile matrix, we can form a consensus string from the most popular element in each column of the alignment matrix. It is the nucleotide with the largest entry in the profile matrix. Figure 2.17 shows the alignment matrix for s = (8,19,3,5,31,27,15), the corresponding profile matrix, and the resulting consensus string ATGCAACT.

2.5.4 Evaluating Motifs

By varying the starting positions in s, we can construct a large number of different profile matrices from a given sample. Some profiles represent high conservation of a pattern while others represent no conservation at all. An imprecise formulation of the Motif Finding problem is to find the starting positions s corresponding to the most conserved profile.

Scoring Function

If $P(s)$ denotes the profile matrix corresponding to starting positions s, then we will use $M_{P(s)}(j)$ to denote the largest count in column j of $P(s)$. For the profile P(s) in Figure 2.17, $M_{P(s)}(1) = 5, M_{P(s)}(2) = 5$, and $M_{P(s)}(8) = 6$. Given starting positions s, the consensus score is defined to be $Score(s, DNA) = \sum_{j=1}^{l} M_{P(s)}(j)$. In our case, $Score(s, DNA) = 5 + 5 + 6 + 4 + 5 + 5 + 6 + 6 = 42$. $Score(s, DNA)$ can be used to measure the strength of a profile corresponding to the starting positions s.

2.5.5 Defining The Motif Finding Problem

the Motif Finding problem can be formulated as selecting starting positions s from the sample that maximize $Score(s, DNA)$.

Motif Finding Problem:
Given a set of DNA sequences, find a set of l-mers, one from each sequence, that maximizes the consensus score.
Input: A $t \times n$ matrix of DNA, and l, the length of the pattern to find.
Output: An array of t starting positions $s = (s_1, s_2, \ldots, s_t)$ maximizing *Score(s, DNA)*.

Chapter 3

Our Approach

In this chapter we have discussed our approach to solve the generalized Quorum Planted Motif Search problem. We have proposed some increment technique on some existing algorithms for solving qPMS problem more efficiently. We have also proposed the idea to implement parallelism for solving this problem.

3.1 Notations and Definitions

We have used some common notations to describe all the algorithms described and reviewed. The notations and definitions have been summarized in table table 3.1. The first five notations (Σ, s_i, l, d, q) are the input data of the problem. Other notations have been used in several steps the algorithms.

3.2 Previous Works

There are several exact algorithms to solve the planted motif search problem. For the sake of completeness we will describe the previous works which have motivated us to develop our approach. In this section we will review qPMSPruneI [9], qPMS7 [1], Traver String Ref [24] and PMS8 [7].

3.2.1 qPMSPrune

Algorithm qPMSPrune for the qPMS problem was proposed by [9]. Algorithm qPMSPrune introduces a tree structure to find the possible motif candidates. Then it uses branch-and-bound technique to reduce the search space and find the expected (l, d) motifs for the given set of

Table 3.1: Notations and Definitions

Notations	Definitions
Σ:	The alphabet
s_i:	The input string of length m over Σ ($1 \leq i \leq n$)
l:	The motif length
d:	The maximum number of mismatches allowed in each occurrences
q:	The minimum number of input strings that should have at least one occurrence.
$\|a\|$:	the length of a string a.
$a[j]$:	The jth letter of a string a.
$a \circ b$:	The string generated by concatenating two strings a and b
$d_H(a, b)$:	The Hamming distance between two strings a and b of the same length. It is given by the number of positions j such that $a[j] \neq b[j]$
s_{ij}^l:	The l-length substring of an input string s_i that starts from the jth position.
S_i:	The set of all the l-length substrings of an input string s_i. $S_i = \{s_{i1}^l, ..., s_{i,ml+1}^l\}$.
S_i^d:	The set of all the $(l+1)$-length substrings of an input string s_i defined by S_i^d= $\{s_{l+1}^{i0}, ..., s_{i,ml+1}^{l+1}\}$. Here, $s_{i0}^{l+1}[1] = s_{i,m-l+1}^{l+1}[l+1] = \emptyset$, and $d_H(\emptyset, \alpha) = \infty$ is assumed for any $\alpha \epsilon \Sigma$.
$\mathcal{B}(a, R)$:	The set of strings in the sphere of radius R centered at a. $\mathcal{B}(a, R)$ = $\{b \mid \|b\| = \|a\|, d_H(a, b) \leq R\}$
$n_{\mathcal{B}}(l, d)$:	$\|\mathcal{B}(a, d)\|$ for an l-length string a.
$x\|_P$:	The substring of x composed by sequencing the letters of x at the positions in a vector P. For example, $x\|_P$ = GCA for x = ACCGAT and $P = (4, 2, 5)$.
$P(j)$:	The jth element of a vector P.
$P_+(j)$:	The j-dimensional vecor composed of the first j elements of a vector P.
$P_-(j)$:	The $(\|P\| - j)$-dimensional vector composed of the last $\|P\| - j$ elements of a vector P.
$R_1(a, b, c)$:	The set of indices j satisfying $a[j] = b[j] = c[j]$.
$R_2(a, b, c)$:	The set of indices j satisfying $a[j] = b[j] \neq c[j]$.
$R_3(a, b, c)$:	The set of indices j satisfying $c[j] = a[j] \neq b[j]$.
$R_4(a, b, c)$:	The set of indices j satisfying $b[j] = c[j] \neq a[j]$.
$R_5(a, b, c)$:	The set of indices j satisfying $a[j] \neq b[j], b[j] \neq c[j]$ and $c[j] \neq a[j]$.

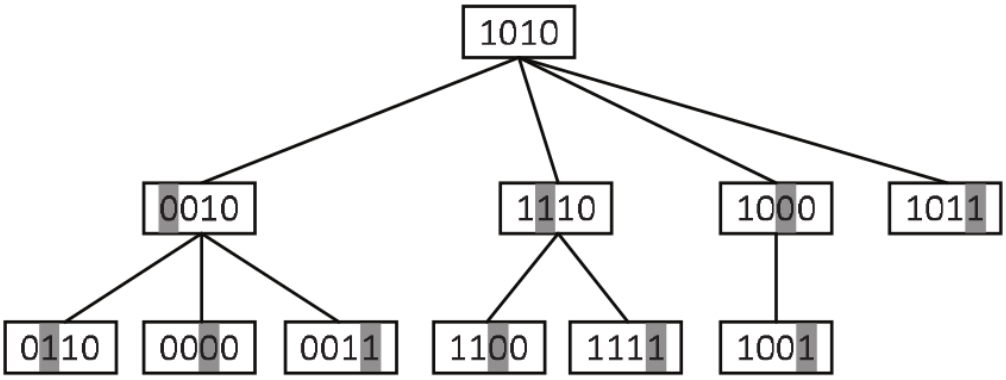

Figure 3.1: $\mathcal{T}_2(1010)$ with alphabet $\Sigma = \{0, 1\}$

inputs. Algorithm qPMSPrune uses the d-neighborhood concept to build the tree structure. The key points of the algorithm is described as follows.

Key Steps of qPMSPruneI

Most of the motif search algorithms combine a sample diven approach with a pattern driven approach. Here in the sample driven part the d-neighborhood ($\mathcal{B}(x_0, d)$) of a given input string x_0 is generated as a tree. Every l-mer in the neighborhood is a candidate motif.

The qPMSPrune algorithm is based on the following observation.

Observation 3.1 *Let M be any (l, d, q)-motif of the input strings $s_1, \ldots, s_n$. Then there exists an i (with $1 \leq i \leq n-q+1$) and a l-mer $x \epsilon s_i^l$ such that M is in $\mathcal{B}(\S, \lceil)$ and M is a $(l, d, q-1)$-motif of the input strings excluding s_i.*

Based on the above observation, for any l-mer x, it represents $\mathcal{B}(x, d)$ as a tree $\mathcal{T}_d(x)$. It generates the d-neighborhood of every l-mer x in s_1 by generating the tree $\mathcal{T}_d(x)$. The height of $\mathcal{T}_d(x)$ is d. The root of this tree will have x. If a node t is parent of a node t' then the hamming distance between the strings is $d_H(t, t') = 1$. As a result, if a node is at level h, then its hamming distance from the root is h.
For example, the tree $\mathcal{T}_2(1010)$ with alphabet $\Sigma = \{0, 1\}$ is illustrated in Figure fig. 3.1.

The trees are generated and explored in depth first manner. In pattern driven part, each node is checked whether it is a valid motif or not. While traversing a node t in the tree $\mathcal{T}_d(x)$ in a depth-first manner, the hamming distance between the input strings are calculated incrementally. If q' is the number of input strings s_j such that $d_H(t, s_j) \leq d$. If $q' \geq q - 1$, output t as a motif.

Moreover the algorithm prunes the branch if $q" < q-1$ where $q"$ is the number of input strings s_j such that $d_H(t, s_j) \leq 2d - d_H(t, x)$.

The time and space complexities of algorithm qPMSPrune are given by $O((n-q+1)nm^2 n_{\mathcal{B}}(l,d))$ and $O(nm^2)$ respectively. The pseudocode of qPMSPrune is shown in algorithm algorithm 1.

Algorithm 1 qPMSPrune

```
1: M ← ∅
2: for i = 1 to n − q + 1 do
3:    for j = 1 to m − l + 1 do
4:       T ← {S_h | 1 ≤ h ≤ N, h ≠ i}
5:       qPMSPruneI_Tree(0, s^l_ij, 0, T)
6:    end for
7: end for
8: Output M
```

Algorithm 2 qPMSPrune_Tree($k, x_k, p_k, \mathcal{T}$)

```
1: T' ← ∅
2: for all Q ϵ T do
3:    Q' ← FeasibleOccurrences2(k, x_k, Q)
4:    if Q' ≠ ∅ then
5:       T ← T' ∪ {Q'}
6:    end if
7: end for
8: if |T'| < q − 1 then
9:    return
10: end if
11: if IsMotif(x_k, q − 1, T') = true then
12:    M ← M ∪ {x_k}
13: end if
14: if k = d then
15:    return
16: end if
17: for p_{k+1} = p_k + 1 to l do
18:    for all α ϵ Σ \ {x_k[p_{k+1}]} do
19:       x_{k+1} ← x_k
20:       x_{k+1}[p_{k+1}] ← α
21:       qPMSPruneI_Tree(k + 1, x_{k+1}, p_{k+1}, T')
22:    end for
23: end for
```

Algorithm 3 FeasibleOccurrences2($k, x_k, \mathcal{Q}$)

```
Q' ← ∅
for all y ϵ Q do
    if d_H(x_k, y) ≤ 2d − k then
        Q' ← Q' ∪ {y}
    end if
end for
return Q'
```

Algorithm 4 IsMotif($x, q', \mathcal{T}$)

```
matched ← 0
for all Q ϵ T do
    found ← false
    for all y ϵ Q do
        if d_H(x, y) ≤ d then
            found ← true
            break the inner loop
        end if
    end for
    if found = true then
        matched ← matched + 1
        if matched ≥ q' then
            return true
        end if
    end if
end for
return false
```

3.2.2 qPMS7

Algorithm qPMS7 was proposed by [1] which was based on qPMSPrune [9]. Some speedup techniques were introduced to improve the runtime of Algorithm qPMSPrune. qPMS7 also searches for motifs by traversing trees. The primary difference from qPMSPrune is that it utilizes $r\epsilon\mathcal{S}_2$ as well as $x_0\epsilon\mathcal{S}_1$ to traverse a tree. The algorithm is based on the following observations.

Observation 3.2 Let M be any (l, d, q)-motif of the input strings $s_1, \ldots, s_n$. Then there exist $1 \leq i \neq j \leq n$ and l-mer $x\epsilon s_i^l$ and l-mer $y\epsilon s_j^l$ such that M is in $\mathcal{B}(x, d) \cup \mathcal{B}(y, d)$ and M is a $(l, d, q-2)$-motif of the input strings excluding s_i and s_j.

Using an argument similar to the one in [9], we infer that it is enough to consider every pair of input strings s_i and s_j with $1 \leq i, j \leq (n - q + 2)$. As a result, the above observation gets strengthened as follows.

Observation 3.3 Let M be any (l, d, q)-motif of the input strings $s_1, \ldots, s_n$. Then there exist

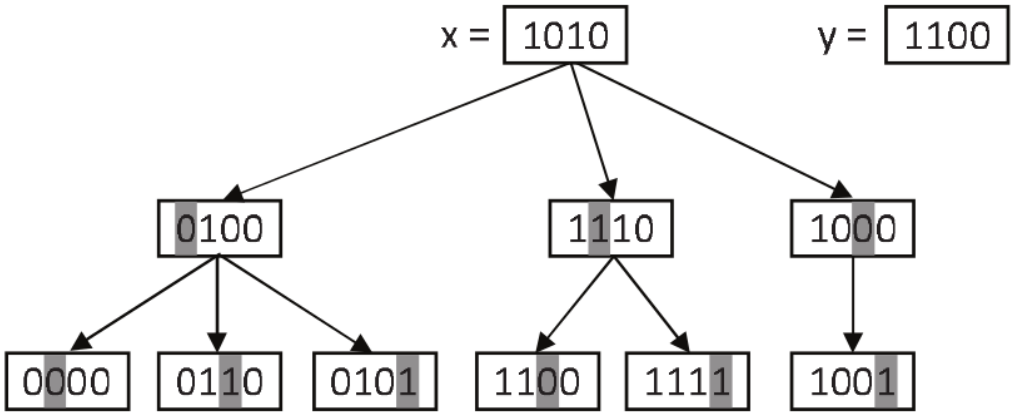

Figure 3.2: $\mathcal{G}_2(1010, 1100)$ with alphabet $\Sigma = \{0, 1\}$

$1 \leq i \neq j \leq n - q + 2$ and l-mer $x \epsilon s_i^l$ and l-mer $y \epsilon s_j^l$ such that M is in $\mathcal{B}(x, d) \cup \mathcal{B}(y, d)$ and M is a $(l, d, q - 2)$-motif of the input strings excluding s_i and s_j.

Algorithm qPMS7 uses a routine based on the above observations which finds all of the possible motifs. Algorithm qPMSPrune explores $\mathcal{B}(x, d)$ by traversing the tree $\mathcal{T}_d(x)$. In Algorithm qPMS7, $\mathcal{B}(x, d) \cup \mathcal{B}(y, d)$ is explored by traversing an acyclic graph, denoted as $\mathcal{G}_d(x, y)$ with similar constructing rule as $\mathcal{T}_d(x)$.

The time and space complexity of Algorithm qPMS7 are $O((n - q + 1)^2 n m^2 n_{\mathcal{B}}(x, d))$ and $O(nm^2)$ respectively. So in worst case scenario the time runtime of qPMS7 is worse than that of Algorithm qPMSPrune by a factor of $n - q + 1$. However, Algorithm qPMS7 is much faster than Algorithm qPMSPrune in practice.

Algorithm 5 qPMS7

```
1: M ← ∅
2: for i_1 = 1 to n − q + 1 do
3:    for j_1 = 1 to m − l + 1 do
4:       for i_2 = i_1 + 1 to n − q + 1 do
5:          for j_2 = 1 to m − l + 1 do
6:             T ← {S_h | 1 ≤ h ≤ N, h ≠ i_1, h ≠ i_2}
7:             qPMS7_Tree(0, s^l_{i_1,j_1}, s^l_{i_2,j_2}, s^l_{i_1,j_1}, 0, T)
8:          end for
9:       end for
10:   end for
11: end for
12: Output M
```

Algorithm 6 qPMS7_Tree($k, x_0, r, x_k, p_k, \mathcal{T}$)

$\mathcal{T}' \leftarrow \emptyset$
for all $Q \epsilon \mathcal{M}$ **do**
  $Q' \leftarrow$ FeasibleOccurrences3(x_0, r, x_k, p_k, Q)
  if $Q' \neq \emptyset$ **then**
    $\mathcal{T} \leftarrow \mathcal{T}' \cup \{Q'\}$
  end if
end for
if $|\mathcal{T}'| < q - 2$ **then**
  return
end if
if $d_H(x_k, x_0) \leq d$ and $d_H(x_k, r) \leq d$ and IsMotif($x_k, q-2, \mathcal{T}'$)=**true then**
  $\mathcal{M} \leftarrow \mathcal{M} \cup \{x_k\}$
end if
if $k = d$ **then**
  return
end if
for $p_{k+1} = p_k + 1$ **to** l **do**
  for all $\alpha \epsilon \Sigma \setminus \{x_k[p_{k+1}]\}$ **do**
    $z|_{I_+(p_{k+1}-1)} \leftarrow x_k|_{I_+(p_{k+1}-1)}$
    $z[p_{k+1}] \leftarrow \alpha$
    $x_{k+1}|_{I_+(p_{k+1})} \leftarrow z$
    if $d_H(z, x_0|_{I_+(p_{k+1})}) > d_H(z, r|_{I_+(p_{k+1})})$ **then**
      $x_{k+1}|_{I_-(p_{k+1})} \leftarrow x_k|_{I_-(p_(k+1))}$
    else
      $x_{k+1}|_{I_-(p_{k+1})} \leftarrow r|_{I_-(p_(k+1))}$
    end if
    qPMS7_Tree($k+1, x_0, r, x_{k+1}, p_{k+1}, \mathcal{T}'$)
  end for
end for

Algorithm 7 FeasibleOccurrences3($x_0, r, x_k, p_k, \mathcal{Q}$)

$Q' \leftarrow \emptyset$
$d_x \leftarrow d - d_H(x_k|_{I_+(p_k)}, x_0|_{I_+(p_k)})$
$d_r \leftarrow d - d_H(x_k|_{I_+(p_k)}, r|_{I_+(p_k)})$
for all $y \epsilon Q$ **do**
  $d_y \leftarrow d - d_H(x_k|_{I_+(p_k)}, y|_{I_+(p_k)})$
  if $B(x_0|_{I_-(p_k)}, d_x) \cap B(r|_{I_-(p_k)}, d_r) \cap B(y|_{I_-(p_k)}, d_y) \neq \emptyset$ **then**
    $Q' \leftarrow Q' \cup \{y\}$
  end if
end for
return Q'

3.2.3 TraverStringRef

TraverStringRef is an improved version of qPMS7. Four improvements were proposed for which will be explained below:

Feasibility Check without Precomputed Table

Like qPMS7 feasibility check is not performed by checking an occurrnce in the precomputed table. Here a theorem is followed: Three strings a, b, c of the same length satisfy

$$\mathcal{B}(a, d_a) \cap \mathcal{B}(b, d_b) \cap \mathcal{B}(c, d_c) \neq \emptyset$$

if and only if

$$\begin{gathered} d_a \geq 0, d_b \geq 0, d_c \geq 0, \\ d_a + d_b \geq d_H(a, b), \\ d_b + d_c \geq d_H(b, c), \\ d_c + d_a \geq d_H(c, a), \\ d_a + d_b + d_c \geq |R_2(a, b, c)| + |R_3(a, b, c)| + |R_4(a, b, c)| + |R_5(a, b, c)| \end{gathered}$$

Elimination of unnecessary Combinations

Unnecessary combinations are eliminated in qPMS7 and the procedure is quite same as qPMSPruneI. The unnecessary checks are suppressed when $q < n$ and $\mathcal{T} \leftarrow \{S_h | i_2 + 1 \leq h \leq n\}$.

String Reordering

This improvement is ensured by pruning the subtree rooted at current node as early as possible by checking the feasibility occurences in non decreasing order. Here the difference is when the input string from where the input reference occurences are taken is determined. Since the number of string in $\mathcal{B}(x_0, d) \cap \mathcal{B}(r, d)$ is a non decreasing function of $d_H(x_0, r)$ the reference r is taken from the input that maximizes minimum hammimg distance between x_0 and r to reduce size of the search tree.

Position Reordering

To ensure efficient pruning of subtrees the search tree's structure is investigated. Only the nodes satisfying the condition of not changing the hamming distance change the structure of the search

tree. To take the advantage of this procedure the position of the strings reordered so that the positions where the letters are different come earlier. For this an l-dimensional vector is computed for each pair at the root node of the search tree.

The pseudo-code of the algorithm TraverStringRef is given in Algorithm [algorithm 8].

Algorithm 8 TraverStringRef

$\mathcal{M} \leftarrow \emptyset$
for $i_1 \leftarrow 1$ **to** $n - q + 1$ **do**
  for $j_1 \leftarrow 1$ **to** $m - l + 1$ **do**
    $T \leftarrow \{S_h | i_1 + 1 \leq h \leq n\}$
    Sort the elements of $\mathcal{T}$ in the nonincreasing order
    while $|\mathcal{T}| \geq q - 2$ **do**
      $\mathcal{R} \leftarrow$ first element of $\mathcal{T}$
      $\mathcal{T} \leftarrow \mathcal{T} \backslash \mathcal{R}$
      for all $r \in \mathcal{R}$ **do**
        Initialize the vector J
        if $d_H(s^l_{i_1 j_1}, r) \leq 2d$ **then**
          TraverStringRef_Tree(0,$s^l_{i_1 j_1}, r, s^l_{i_1 j_1}, 0, \mathcal{T}, J$)
        end if
      end for
    end while
  end for
end for
Output $\mathcal{M}$

Algorithm 9 TraverStringRef_Tree($k, x_0, r, x_k, p_k, \mathcal{T}, J$)

$\mathcal{T}' \leftarrow \emptyset$
missed $\leftarrow 0$
for all $Q \epsilon T$ **do**
  $Q' \leftarrow$ FeasibleOccurrencesReordered3($k, x_0, r, x_k, p_k, Q, J$)
  if $Q' \neq \emptyset$ **then**
    $\mathcal{T} \leftarrow \mathcal{T}' \cup \{Q'\}$
  else
    missed $\leftarrow$ missed$+ 1$
    if missed $> |\mathcal{T}| - q + 2$ **then**
      return
    end if
  end if
end for
if IsMotifFast($x_k, q - 2, \mathcal{T}'$) $=$ **true then**
  $\mathcal{M} \leftarrow \mathcal{M} \cup \{x_k\}$
end if
if $k = d$ **then**
  return
end if
if $|\mathcal{T}'| < q - 2$ **then**
  return
end if
Sort the elements of $\mathcal{T}'$ in the nonincreasing order of $\min_{y \epsilon \mathcal{Q}} d_H(x_k, y)$(20) {The reference needs to be corrected}
for $p_{k+1} = p_k + 1$ **to** l **do**
  for all $\alpha \epsilon \Sigma \setminus \{x_k[J(p_{k+1})]\}$ **do**
    $x_{k+1} \leftarrow x_k$
    $x_{k+1}[J(p_{k+1})] \leftarrow \alpha$
    $d_0 \leftarrow d + d_H(x_0|_{J_-(p_{k+1})}, r|_{J_-(p_{k+1})})$
    if $d_H(x_{k+1}, r) \leq \min(d_0, 2d - k - 1)$ **then**
      TraverStringRef_Tree($k + 1, x_0, r, x_{k+1}, p_{k+1}, \mathcal{T}', J$)
    end if
  end for
end for

3.2.4 PMS8

The key concepts of increasing efficiency in PMS8 from qPMS7 and qPMSPrune are described below:

Sort rows by size

Sorted rows speed up the filtering step. As for sorted rows we need less tuples for lower stack size, it helps to reduce expensive filtering. It is because fewer l-mers remain to be filtered when the stack size increases.

Compress l-mers

To calculate the hamming distance between two l-mers at first we have to perform exclusive or of their compressed representation. One compressed l-mer requires $1 \times \lceil \log |\Sigma| \rceil$ bits of storage. So the table of compressed l-mers only requires O(n(m-l+1)) words of memory as we only need the first 16 bit of this representation.

Preprocess distances for pairs of l-mers

For every pair of l-mers we test if the distance is no more than 2d in advance.

Cache locality

As a certain row of an updated matrix is the subset of that row so we can store the updated row in the previous location of the row. so we have to keep track the number of elements belongs to the new row. This process can be repeated in every step of recursion and can be perform using by only a single stack where the subset elements are in contiguous position of memories and thus cache locality sustains.

Find motifs for a subset of strings

we will find the motif for some input strings and then test them against the remaining strings.

Memory and Runtime

since we store all the matrices in the place of a single matrix they only require $O(n(n-l+1))$ words of memory. $O(n^2)$ words of row size will be added for at most n matrices which share the same space. So total bits of l-mer pair takes $O((n(m-l+1)^2)/w)$ words where w is the number of bits in a machine word. So total memory used for this algorithm is $O(n(n-l+1) + (n(m-l+1)^2)/w)$.

Parallel implementation

If we think dividing the problem into $m-l+1$ subproblems then the number of subproblems will be embarrassingly greater for parallelization. So a fixed number of subproblems are assigned to each processor. Scheduler then spawns a separate worker thread to avoid use of a processor just for scheduling. The scheduler loops until all the subproblems are solved and after the completion of all threads the motifs are given to scheduler and it outputs the result.

3.3 Our Proposal qPMS-Sigma

In this section we propose an algorithm qPMS-Sigma with some techniques to optimize the space complexity as well as the runtime to solve the Quorum Planted Motif Search problem. Our work is mainly based on the Algorithm TraverStringRef.

3.3.1 String Compression

We can compress the input strings $s_1, \ldots s_n$ as a part of preprocessing of the algorithm. For our work we have compressed all the l-mers of the input strings into some group of unsigned 32-bit integers. Thus we make the motif matrix using the compressed l-mers. To be specific for an alphabet set Σ we compressed a l-mer into $\lceil \log_2(|\Sigma|) \rceil$ groups. Each of this group contains $\lceil l/32 \rceil$ 32-integers.

For example, for a DNA string of 32 characters, we need $\lceil \log_2(|\Sigma|) \rceil = 2$ groups of unsigned 32-bit integers, where each group contains $\lceil n/32 \rceil = 1$ integers.

We can calculate the hamming distance between two compressed l-mers in two steps. First for each group of integers find out the bitwise XOR values of the integers of both the l-mers. After that calculate the bitwise OR values among the XOR-ed values of each group. The number of set bits in the last set of integers is the hamming distance between the two strings.

An example of comparing two compressed l-mers is given in fig. 3.3.

String	After Ordering the Characters	Group 1	Group 0	XOR 1	XOR 0	OR	Number of Set Bits
AGGCTA	022130	011010	000110	110001	000011	110011	4
GAGCAT	202103	101011	000101				

Figure 3.3: l-mer Compression Example

A code snippet appendix A.1 is written in C++ describing the structure of compressed l-mer and the method of calculating hamming distance.

3.3.2 Motif Finding

After compressing the input sequences we need to find the motifs in them. For these we've modified the TraverStringRef algorithm by using the compressed l-mers. Then we have followed the steps of Algorithm TraverStringRef to find out the motifs.

3.3.3 Parallel Implementation of qPMS-Sigma

The Algorithm qPMS-Sigma can easily implement in parallel by traversing several search trees in parallel. Similar to the idea of parallelism mentioned in Algorithm TraverStringRef [24] we can divide the problem into $(m - l + 1)$ subproblem where each subproblem explores different search trees.

3.4 Space and Runtime Complexity

Here the space complexity of our algorithm is $O(nml(\log\lceil\Sigma\rceil)/w)$. Here w is the size of the integers that contain the compressed input sequences as well as the l-mers. For DNA sequences $\log|\Sigma| = 2$. So the space complexity becomes $O(nml/w)$.

The worst case time complexity is similar to TraverStringRef $O((n-q+1)^2 nm^2(m+\log n)n_{\mathcal{B}}(l,d))$. However, finding the hamming distance between two l-mers take about constant time for DNA sequence. Normally bitwise operations are a little faster than the other arithmetic operations. So, practically our algorithm runs a little faster than the former.

3.5 Experimental Results

We have compared the runtime of Algorithm qPMS-Sigma in the challenging states with the other well known algorithms. The proposed algorithm is implemented in C++ (using GNU C++ compiler). We have run the experiment on Ubuntu 14.04 (64 bit) operating system. The machine configuration is Intel®Core™i3-4005U CPU @ 1.70GHz 4, 4GB RAM.

The test dataset was generated in computational experiment of Algorithm TraverStringRef [24].

Table 3.2: Parameter Setting for Testing Data Set

Parameter	Setting
$\|\Sigma\|$	4 (DNA)
n	20
m	600
q	10,20
l	13, 15, 17,...

Table 3.3: Computational Results for DNA Sequences for $q = 20$

Algorithm	**(13,4)**	**(15,5)**	**(17,6)**	**(19,7)**	**(21,8)**	**(23,9)**
qPMS-Sigma	12s	50s	165s	755s	2909s	12153s
TraverStringRef	11s	47s	179s	745s	3098s	13027s
qPMS7	14s	133s	1046s	8465s	71749s	
PMS8	7s	48s	312s	1596s	5904s	19728s
qPMS9	6s	34s	162s	804s	2724s	8136s

The testing dataset is randomly generated according to the FM (fixed number of mutation) model [16]. We have used it in our experiment and showed the result in table 3.3 and table 3.4. Our algorithm shows a little better result than Algorithm TraverStringRef in some challenging states. Although it lags behind the qPMS9 in all cases. The comparisons are shown in table 3.3, table 3.4, fig. 3.4, fig. 3.5.

Table 3.4: Computational Results for DNA Sequences for $q = 10$

Algorithm	**(13,4)**	**(15,5)**	**(17,6)**	**(19,7)**	**(21,8)**	**(23,9)**
TraverStringRef	7	36	160	760.6	3643.3	17232.4
TraverStringRef	5.5	32	166.1	779.9	3700.6	17922.2
qPMS7	31	152	850.7		4865.0	29358
qPMSPrune	12.8	126.7	1116.5	10540.8		
PMS9						

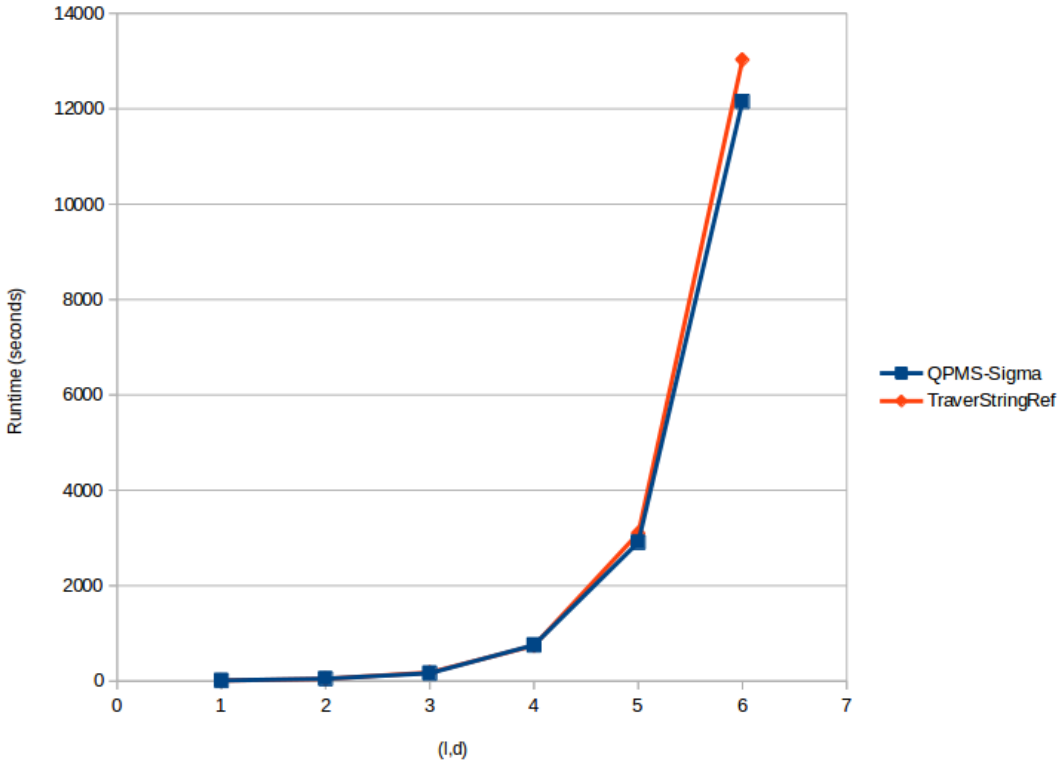

Figure 3.4: Run Time Comparison between qPMS-Sigma vs TraverStringRef in the challenging states for q=20

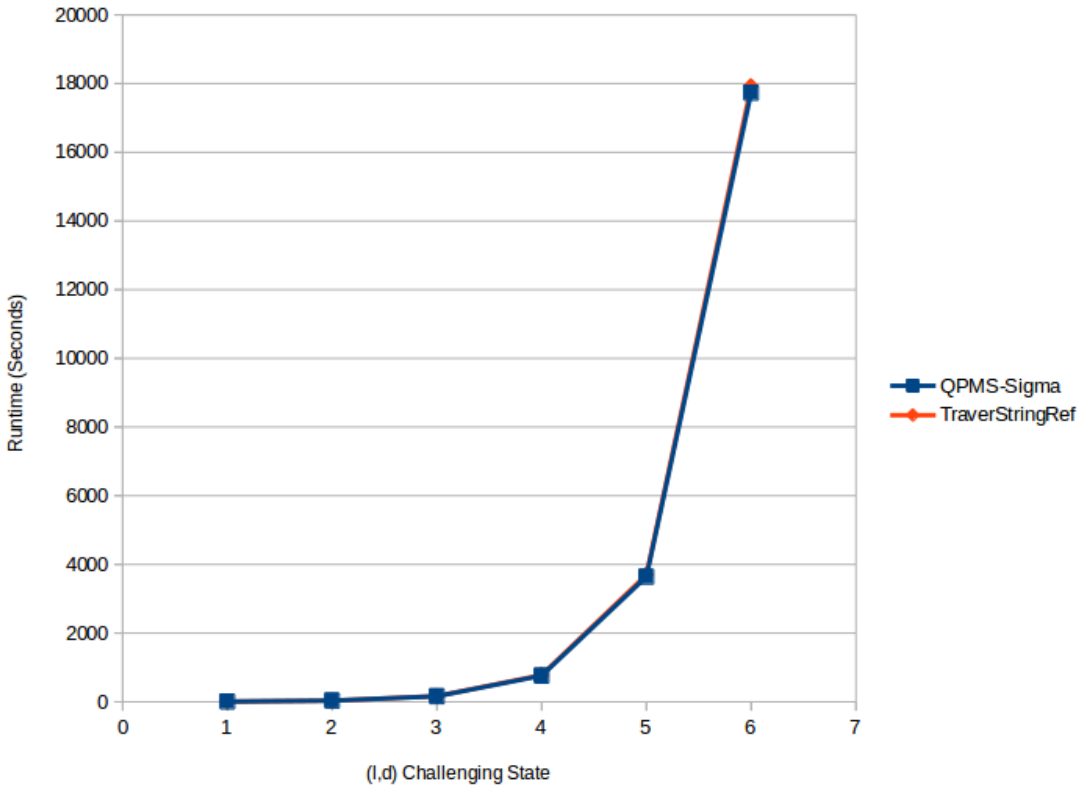

Figure 3.5: Run Time Comparison between qPMS-Sigma vs TraverStringRef in the challenging states for q=10

Chapter 4

Conclusion

4.1 Summary of Our Work

The main task of our thesis is to develop an effective algorithm for the Planted Motif Search problem. Our proposed algorithm qPMS-Sigma tries to find the (l, d)-motifs for n given sequences. It is an exact version of the PMS algorithm. qPMS-Sigma is based on the previous PMS algorithms qPMSPrune, qPMS7, TraverStringRef and PMS8. In our proposed algorithm we introduce clever techniques to compress the input sequences and thus space complexity is improved. We also include a faster comparison technique of the l-mers by adding bitwise comparison techniques. As we know the PMS is a NP-Hard problem, it is exponential in terms of time. So, parallel implementation techniques are proposed at the end of our work.

4.2 Future Prospects of Our Work

Though in our thesis we have given some ideas about parallel implementation of our algorithm, it is not implemented and tested on multiprocessor system. All comparison of our algorithm with the stated algorithms are done in single processor system. To understand the real speedup and slackness we need to experiment in a system involving a network of nodes having multiple processors. Our work is the extension of PMS8 codes which involves openMPI libraries. In future we want to continue our work using OpenMPI project which is a open source Message Passing Interface. Apart from the parallelism, more advanced pruning conditions can be used to reduce the search space. Different randomized approaches can also be included for improving the runtime, though it will make the exact version of our algorithm approximate.

References

[1] H. Dinh, S. Rajasekaran, and J. Davila, "qpms7: A fast algorithm for finding (, d)-motifs in dna and protein sequences," *PloS one*, vol. 7, no. 7, p. e41425, 2012.

[2] L. Duret and P. Bucher, "Searching for regulatory elements in human noncoding sequences," *Current opinion in structural biology*, vol. 7, no. 3, pp. 399–406, 1997.

[3] S. Pal, P. Xiao, and S. Rajasekaran, "Efficient sequential and parallel algorithms for finding edit distance based motifs," *BMC genomics*, vol. 17, no. 4, p. 465, 2016.

[4] S. Rajasekaran, "1. abstract 2. introduction 3. experimental techniques 4. computational techniques 4.1. statistics based techniques 4.2. discrete algorithmic techniques," *Frontiers in Bioscience*, vol. 14, pp. 5052–5065, 2009.

[5] M. Frances and A. Litman, "On covering problems of codes," *Theory of Computing Systems*, vol. 30, no. 2, pp. 113–119, 1997.

[6] M. Nicolae and S. Rajasekaran, "qpms9: an efficient algorithm for quorum planted motif search," *Scientific reports*, vol. 5, 2015.

[7] M. Nicolae and S. Rajasekaran, "Efficient sequential and parallel algorithms for planted motif search," *BMC bioinformatics*, vol. 15, no. 1, p. 34, 2014.

[8] J. Davila, S. Balla, and S. Rajasekaran, "Pampa: An improved branch and bound algorithm for planted (l, d) motif search," in *Tech. rep*, 2007.

[9] J. Davila, S. Balla, and S. Rajasekaran, "Fast and practical algorithms for planted (l, d) motif search," *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, vol. 4, no. 4, pp. 544–552, 2007.

[10] S. Bandyopadhyay, S. Sahni, and S. Rajasekaran, "Pms6mc: A multicore algorithm for motif discovery," *Algorithms*, vol. 6, no. 4, pp. 805–823, 2013.

[11] F. Y. Chin and H. C. Leung, "Voting algorithms for discovering long motifs.," in *APBC*, pp. 261–271, 2005.

[12] N. Pisanti, A. M. Carvalho, L. Marsan, and M.-F. Sagot, "Risotto: Fast extraction of motifs with mismatches," in *Latin American Symposium on Theoretical Informatics*, pp. 757–768, Springer, 2006.

[13] T. L. Bailey, C. Elkan, *et al.*, "Fitting a mixture model by expectation maximization to discover motifs in bipolymers," 1994.

[14] J. Buhler and M. Tompa, "Finding motifs using random projections," *Journal of computational biology*, vol. 9, no. 2, pp. 225–242, 2002.

[15] C. E. Lawrence, S. F. Altschul, M. S. Boguski, J. S. Liu, A. F. Neuwald, J. C. Wootton, *et al.*, "Detecting subtle sequence signals: a gibbs sampling strategy for multiple alignment," *SCIENCE-NEW YORK THEN WASHINGTON-*, vol. 262, pp. 208–208, 1993.

[16] P. A. Pevzner, S.-H. Sze, *et al.*, "Combinatorial approaches to finding subtle signals in dna sequences.," in *ISMB*, vol. 8, pp. 269–278, 2000.

[17] E. Rocke and M. Tompa, "An algorithm for finding novel gapped motifs in dna sequences," in *Proceedings of the second annual international conference on Computational molecular biology*, pp. 228–233, ACM, 1998.

[18] U. Keich and P. A. Pevzner, "Finding motifs in the twilight zone," in *Proceedings of the sixth annual international conference on Computational biology*, pp. 195–204, ACM, 2002.

[19] A. Price, S. Ramabhadran, and P. A. Pevzner, "Finding subtle motifs by branching from sample strings," *Bioinformatics*, vol. 19, no. suppl 2, pp. ii149–ii155, 2003.

[20] G. Z. Hertz and G. D. Stormo, "Identifying dna and protein patterns with statistically significant alignments of multiple sequences.," *Bioinformatics*, vol. 15, no. 7, pp. 563–577, 1999.

[21] N. C. Jones and P. Pevzner, "An introduction to bioinformatics algorithms," 2004.

[22] J.-J. M. Riethoven, *Regulatory Regions in DNA: Promoters, Enhancers, Silencers, and Insulators*, pp. 33–42. Totowa, NJ: Humana Press, 2010.

[23] W. Wei and X.-D. Yu, "Comparative analysis of regulatory motif discovery tools for transcription factor binding sites," *Genomics, proteomics & bioinformatics*, vol. 5, no. 2, pp. 131–142, 2007.

[24] S. Tanaka, "Improved exact enumerative algorithms for the planted (l, d)-motif search problem," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 11, no. 2, pp. 361–374, 2014.

Index

Appendix A

Codes

A.1 Compressed String

Here is a sample C++ structure for compressing a 1000 character long DNA string

```
1 //letterToInt[i] means a unique id for each character of the
     alphabet;
2 //Here,
3 //letterToInt['A']=0;
4 //letterToInt['B']=1;
5 //letterToInt['C']=2;
6 //letterToInt['D']=3;
7
8 struct CompressedlMer
9 {
10     unsigned int **cs;//[2][32];
11     int l;
12     CompressedlMer(char *string, int start, int l)    {//
          constructor
13         int group = 0;
14         int intPerGroup = ceil(l/sizeof(unsigned int));
15         int temp = l;
16         while(temp)    {
17             group++;
18             temp = temp/2;
19         }
20         cs = new unsigned int*[group];
21         for(int i = 0; i < group; i++)
```

```
22              cs[i] = new unsigned int[intPerGroup];
23
24          for(int i = 0; i < l; i++)    {
25              int j = letterToInt[string[start+i]];
26              int positionMask = 1<<(i%sizeof(unsigned int));
27              int intNo = i/sizeof(unsigned int);
28              int grp = 0;
29              while(j>0)    {
30                  if(j&1) cs[grp][intNo] = cs[grp][intNo] |
                       positionMask;
31                  j = j/2;
32                  grp++;
33              }
34          }
35      }
36
37      ~CompressedlMer()    {//destructor
38          for(int i = 0,j=1; j < l; i++,j*=2)    {
39              delete [] cs[i];
40          }
41          delete [] cs;
42      }
43  };
44
45  int hammingDist(const CompressedlMer &sa, const CompressedlMer
        &sb)    {
46      int intPerGroup = ceil(sa.l/sizeof(unsigned int));
47      int hamDist = 0;
48      for(int i = 0; i < intPerGroup; i++)    {
49          unsigned int flag = ~(0);    //all bits are set to 1
50          for(int group = 0, j = 1; j < sa.l; group++, j*=2)    {
51              flag = flag | (sa.cs[group][i]^sb.cs[group][i]);
52          }
53          hamDist = hamDist + __builtin_popcount(flag);
54      }
55      return hamDist;
56  }
```

Generated using Undegraduate Thesis LaTeX Template, Version 1.2. Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh.

This thesis was generated on Saturday 18th February, 2017 at 8:05pm.