A Parallel Graph-Based Approach for Protein Sequence Motif Discovery

by

Santan Challa

A dissertation submitted in partial satisfaction of the requirements for the degree of

Master of Science

Department of Computer Science
Faculty of Graduate Studies
University of Manitoba

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES
*****
COPYRIGHT PERMISSION

A PARALLEL GRAPH-BASED APPROACH FOR PROTEIN SEQUENCE MOTIF DISCOVERY

BY

Santan Challa

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of

Manitoba in partial fulfillment of the requirement of the degree

MASTER OF SCIENCE

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a MSc thesis entitled:

A Parallel Graph Based Approach for Protein Sequence Motif Discovery

submitted by:
Santan Challa

in partial fulfillment of the requirements for the degree of: *MSc*

Dr. Parimala Thulasiraman, Advisor

Dr. Michael Domaratzki
Computer Science

Dr. P. Shivkumar
Mathematics

Date of Oral Examination: *August 10, 2007*

The student has satisfactorily completed and passed the MSc Oral Examination.

Dr. Parimala Thulasiraman, Advisor

Dr. Michael Domaratzki
Computer Science

Dr. Dean Jin
Chair of MSc Oral

Dr. P. Shivkumar
Mathematics

(The signature of the Chair does not necessarily signify that the Chair has read the complete thesis.)

Abstract

This thesis provides a parallel, graph based approach to discover conserved regions such as motifs in protein sequences. The motif discovery problem has gained lot of significance in biological science over the past decade. Recently, various approaches have been used successfully to discover motifs. Some of them are based on probabilistic approach and the others on a combinatorial approach. This thesis follows a graph-based approach to solve this problem, in particular, using the idea of de Bruijn graphs. The de Bruijn graph has been successfully adopted in the past to solve problems such as local multiple alignment and DNA fragment assembly. The proposed algorithm harnesses the power of the de Bruijn graph to discover the conserved regions in a protein sequence. The sequential algorithm has 70% matches of the motifs with the MEME and 65% pattern matches with the Gibbs motif sampler. The algorithm is redesigned and parallelized on the high performance computers available on the Western Canada Research Grid (WestGrid). Performance analysis was made on a pure distributed memory machine using only message passing and on a hybrid machine using shared and distributed access space. Experiments showed that the hybrid implementation runs 3 times as fast as the pure distributed memory implementation.

Acknowledgements

I would like to express my sincere gratitude to all who have contributed to complete this thesis. I would like to thank my supervisor Dr. Parimala Thulasiraman for her continuous guidance and encouragement to develop this work.

We acknowledge the partial support from Natural Science and Engineering Research Council(NSERC) of Canada.

My sincere thanks to the management of Westgrid for allowing us to work on their systems.

Finally, a profound gratitude goes to all of my family members who provided continuous encouragement and emotional support to accomplish this endeavor.

Contents

List of Figures

Chapter 1

Introduction

A motif is a repeating pattern in a biological sequence that is conserved during the process of evolution. Motif discovery is a very important problem in biology. It finds applications in DNA or protein sequence analysis, comprehending disease susceptibility and disease cure. Numerous motif discovery algorithms have been proposed. Scientists have explored both combinatorial and probabilistic motif discovery approaches. Yet, the quest for efficient and faster algorithms continues.

Graph theory is plays an important role in computational biology [34]. Graph-based algorithms provide a simple and quick solution to computationally intensive problems such as DNA fragment assembly [22] and motif discovery. However, the amount of literature available on motif discovery using graph algorithms is not proportional to the potential of the graph-based algorithms.

Motif discovery/finding is computationally intensive. In the literature, very few attempts have been made in designing parallel algorithms for this problem. In this thesis, we develop a motif discovery algorithm based on de Bruijn graphs and parallelize this algorithm on a cluster. The algorithm has two phases: graph construction and graph traversal phases. In the graph construction phase, as the name implies, a de Bruijn graph is constructed. The graph traversal phase identifies the motifs. Both these phases are parallelized. Besides an efficient parallel algorithm, the architecture on which the algorithm is parallelized is crucial to the performance of the algorithm. The graph construction phase requires shared address space while the graph traversal phase requires distributed access space. Therefore, we have parallelized the algorithm on a hybrid architecture which harnesses the power of the shared and distributed access space. The hybrid architecture is a cluster of dual processor nodes available on the WestGrid consortium [27] (High performance computing infrastructure available across British Columbia and Alberta). We exploit fine grain parallelism within the nodes and coarse grain parallelism across nodes of the cluster.

We have found that our parallel algorithm was successful in mining signals for a larger number of sequences and at a faster rate when compared to some popular motif searching tools such as MEME [2]. The results of the sequential algorithm are published in the Bioinformatics and Life Science Computing symposium [8]. The next section explains few terms necessary for understanding the thesis. It is a short journey from cell architecture to protein manufacturing.

1.1 Cell to Protein

Life begins in a cell. A single cell contains a copy of the genetic information. This genetic information governs the growth, behaviour and reproduction of an organism. A cell contains a nucleus (only present in eukaryotic cells such as human cells) which is the building block of the cell. Nucleus consists of nucleoplasm, nucleoli and cajal bodies. The nucleoplasm primarily consists of chromatin. Chromatin is the base nuclear material which is responsible for the manufacture of chromosomes. Each nucleus is made up of 23 pairs of chromosomes. Chromosomes are long strands of

DNA (deoxyribonucleic acid) which contain the genetic information that is required for a cell to function. In 1953, Watson and Crick found the current structure of the DNA, Figure 1.1 [29], which is a double helix. Each DNA molecule consists of two strands (or chains) twisted like a corkscrew and linked together by a pair of bases. There are four bases in DNA molecule, namely, adenine (A), guanine (G), cytosine (C) and thymine (T). Adenine always forms a base pair with thymine and guanine forms a base pair with cytosine. Therefore, the two strands of the DNA are linked with either of the two base pairs, AT or GC. DNA is made up of genes (a human

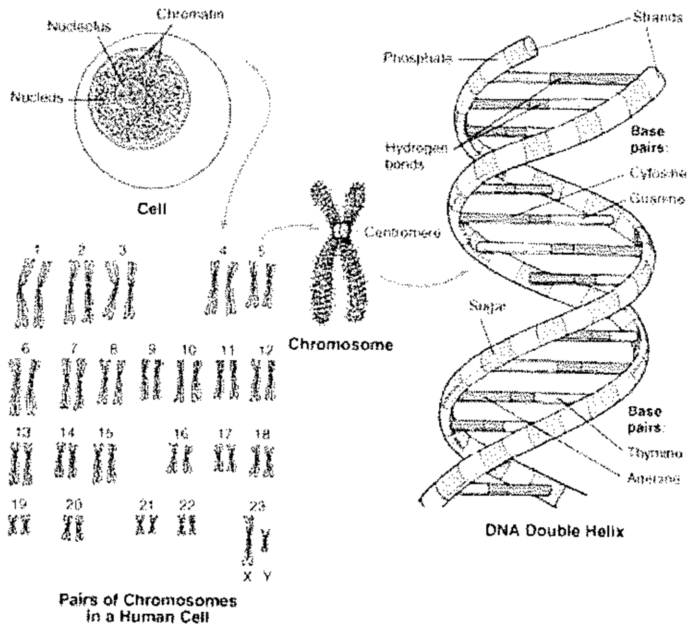


Figure 1.1: Structure of DNA [29]

genome contains more than 30000 genes) also called as blueprints. Each gene contains data for the synthesis of a unique protein (a complex 3D structure present in animal and vegetable tissue) for the construction of the body of an organism. Proteins are made up of 20 different kinds of amino acids. While DNA is a 4 letter alphabet (A, C, G and T) a protein is a 20 letter alphabet. The process in which DNA converts to proteins involves two stages: *transcription* and *translation*. In the transcription stage, an intermediate representation encoded in the Ribonucleic Acid (RNA) makes a replication of the protein coding information of the gene and produces a single stranded mRNA (stands for messenger ribonucleic acid). The mRNA also has four bases like the DNA with T replaced by U (base Uracil). Then the protein making machinery translates the mRNA into an amino acid sequence of a protein. This process of transforming a gene's information to proteins in called as *gene expression*. The transcription (DNA to RNA) and translation (RNA to proteins) is the *central dogma* of molecular biology [16, 30].

Motifs, also called as Transcription Factor Binding Sites (TFBS), play a vital role during the transcription stage in the regulation of the gene expression[1]. Every gene has a regulatory region (regulatory sequence). The regulatory region contains transcription factors (proteins). The regulatory sequences along with the transcription factors determine the state of the gene (either turn-on or turn-off). Figure 1.2 [7] presents regulatory regions in prokayote and eukaryote genes. If a cell gets infected in an organism, a specific immunity gene is turned-on, which produces proteins to cure the infected cell. Every gene contains certain short strings called binding sites in the regulatory region, which are responsible for turning on such immunity genes. The transcription factors unite with these binding sites at several locations. These sites which contain (bind) the transcription factors are called *motifs* [16]. All these binding

[1]How much of the gene's information should be converted to proteins is done by the cell, thereby regulating the gene expression. We can think of having a circuit in the organism's body where nodes in the circuit are genes connected by switches that regulate the genes. If a switch is changed in some way, the circuit also changes. Gene regulation is one of the well known research areas in molecular biology.

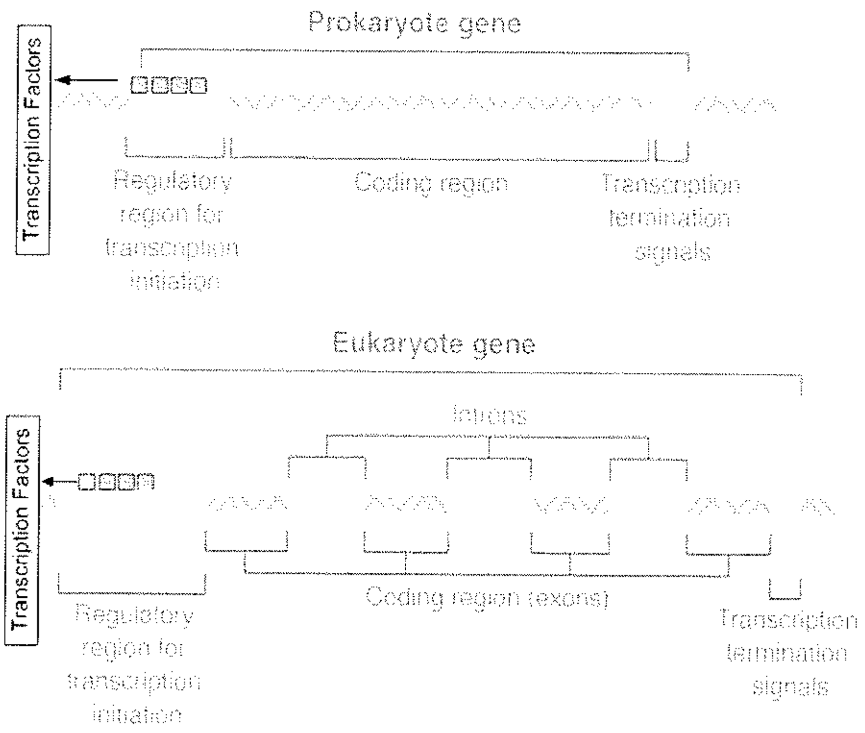

Prokaryote gene
Transcription Factors
Regulatory region for transcription initiation
Coding region
Transcription termination signals
Eukaryote gene
Introns
Transcription Factors
Coding region (exons)
Regulatory region for transcription initiation
Transcription termination signals

Figure 1.2: Transcripion Factors and Regulatory regions [7]

sites are about the same length. However, they are not so easy to locate because they are buried in a huge array of characters. The next chapter explains DNA and protein motifs, how they are represented and why it is difficult to find motifs.

The pioneering work of Pevzner et al. [22] introduces the application of graph algorithm in various biological problems such as DNA fragment assembly and local multiple sequence alignment. The next section discusses the motivation of this work.

1.2 Motivation

Pevzner et al. [22] effectively use a de Bruijn graph in their DNA fragment assembler EULER (assembles the entire genome sequence from fragments of DNA) to resolve *repeats*[2] in a genomic sequence (a sequence representing the entire genome). The authors construct a de Bruijn graph that represents repeats in the form of edges. An edge with multiplicity greater than one is treated as a repeat. The de Bruijn graph transforms all the edges with multiplicity greater than one into a single high-weight edge (proportional to its multiplicity), thus providing a clear way to resolve repeats. According to Pevzner et al. the graph-based approach for fragment assembly is quicker and more efficient than the traditional overlap-layout-consensus method [22]. DNA fragment assembly is not related to our work in any way. However, the idea to use a de Bruijn graph for motif finding originated from the research work of Pevzner et al. Though a motif and a repeat are entirely different entities, the method that is used for identifying repeats by Pevzner et al. can be successfully employed for motif discovery too. Here, instead of identifying repeats, more repetitive, conserved and short patterns are identified. More recently, Zhang and Waterman [41] followed a similar approach for local multiple alignment of DNA sequences. They have used the de Bruijn graph to build a consensus sequence that contains most of repeating and conserved patterns (not necessarily motifs). Furthermore, they perform a local pair wise alignment to find subsequences of each sequence that match with the consensus pattern. As mentioned in their paper, motif discovery is in fact, a subproblem of the multiple local alignment problem. Our work is profoundly influenced by the research efforts mentioned above. The only other researcher we are aware of who uses a de

[2]Repeats are replicas of a part or the entire DNA sequence, which are present along with the original sequence.

Bruijn graph for motif finding is Patwardhan [20]. Patwardhan has developed an application for motif discovery using messy de Bruijn graphs, which is available on the internet for testing purposes. However, Patwardhan's work does not study the efficiency of the graph based algorithm on a parallel platform.

Problem Statement: Motif Discovery is an NP-complete problem [1]. Owing to the huge amount of sequence data available, there is a need for efficient computational techniques to reduce the computational time for discovering motifs. The problem is, though serial algorithms are quite efficient in discovering motifs, they fail when we need to find motifs of smaller length in huge amount of data. In such cases, the serial algorithms either take a very long time and large amounts of memory or completely breakdown. Hence we have considered a parallel version which is capable of finding smaller and even less significant signals, and which is capable of handling the entire genomic sequences. The parallel version is designed to break the memory barrier and overcome the issues with data size.

In this thesis, our focus is on developing a graph based algorithm for motif discovery and parallelizing the algorithm for fast results on a data size larger than few thousand sequences. As mentioned earlier in this thesis, graph algorithms algorithms are fast and easy to visualize. The advantage of a de Bruijn graph is that it isolates domains of similarities. Hence it is quite easy to trace areas of similarities in the protein sequences. Further, the de Bruijn graph has an elegant way of representing the nodes and edges. Since every node represents a subsequence of a given sequence, there is no need to maintain a separate data structure to store the node positions, etc.

We have chosen to parallelize the given problem primarily because of the vast amount of protein sequence data available for analysis. Secondly, we identified regions in our algorithm which are highly parallelizable. The serial algorithm showed excellent performance until the data size was about 15MB. However, beyond that the algorithm failed to handle the huge amount of data, thereby yielding undesired results. It has succumbed to the overpowering computational intensity and insufficient memory. The parallel algorithm comes to rescue in such situations by overcoming these limitations. Hence we chose to port the serial algorithm to a parallel machine. The next section will explain the contributions of this thesis.

1.3 Contributions of this Thesis

This thesis aims to perform motif discovery on a larger scale. Instead of performing motif discovery on a few sequences or a few hundred sequences, we wish to take it to the genomic level. This requires high performance parallel computing.

Here are some contributions to this thesis:

- Developing an efficient graph based algorithm for finding motifs.
- Parallelizing the algorithm on high performance computers for fast results on large data sizes. Very little work exists in the literature on parallel algorithms for motif discovery.
- Harnessing the power of the computer architecture and finding a balance between shared and distributed memory machines. We have implemented the algorithm on a hybrid architecture, a cluster of dual processor machines.
- Studying the performance of the sequential and parallel algorithm.
- Comparing the correctness of the results obtained from our algorithm with that of popular software such as MEME.

The rest of thesis is organized as follows: Chapter 2 presents motifs and various discovery approaches to date. Chapter 3 discusses the state-of-the-art motif discovery approaches. Chapter 4 presents parallel computing architectures. Chapter 5 is a discussion on the previous research efforts in parallel motif discovery. Chapter 6 details both sequential and parallel algorithm proposed in this thesis. Implementation methodology and Results are discussed in Chapter 7. Finally, conclusions and future work are presented in Chapter 8.

Chapter 2

DNA and Protein Motifs

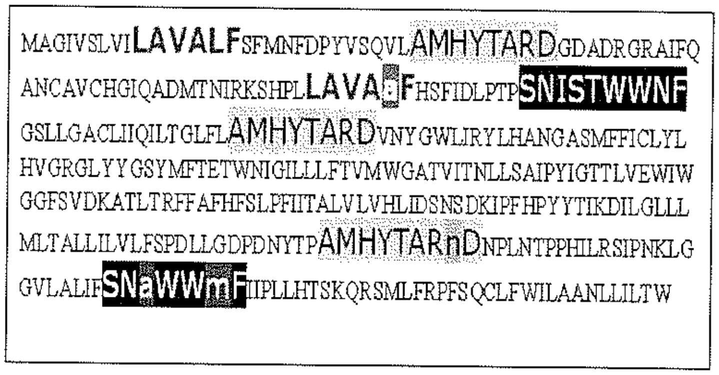
This chapter explains the importance of motifs in proteins. We will briefly discuss issues such as protein structures, the importance of sequence alignment and motif discovery in protein structure prediction.

2.1 DNA Motifs

In the previous chapter, we described the transcription process for a particular gene. The process starts by binding transcription factors (or proteins) in specific sites (called binding sites) in the promoter region of the gene. A promoter region is a region that helps to determine the next occurrence of the gene. The transcription factors may bind to several binding sites. This may happen because a transcription factor may control several genes in the DNA sequence. The transcription factor has a common DNA sequence pattern and is of the same length. These common sequence patterns are called *motifs*. The common sequence pattern for most of the transcription factors is unknown. Research in the laboratories for finding motifs is greatly underway in molecular biology. However, this is a time consuming task.

2.1.1 What is the motif finding problem?

Transcription factors control several genes in the DNA sequence. If one particular transcription factor controls five genes (gene1, gene2, gene3, gene4, gene5) as shown in Figure 2.1, then there are biding sites in the promoter of the genes, where the transcription factor binds. Suppose the promoter regions are also given. The problem now is to determine the binding sites of transcription factor and whether we will be able to find them without knowing their location in advance. If we find the binding sites then we should be able to find the transcription factors. Note that the binding sites are similar to each other but not identical. Therefore, the problem is to find a motif (common sequence pattern) that represents binding sites for an unknown transcription factor. The highlighted regions in Figure 2.1 illustrate motifs in a protein sequence.

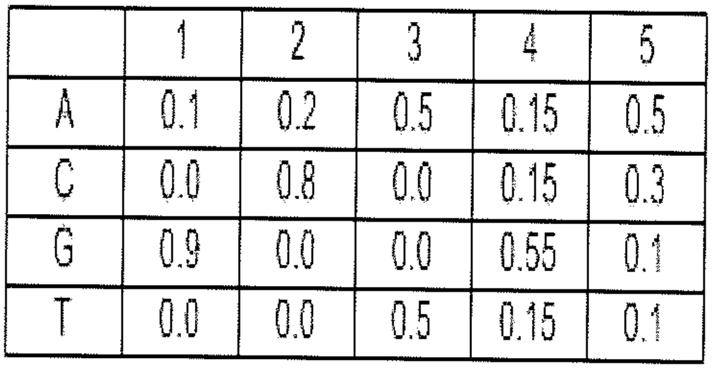


```
MAGIVSLVILAVALFSFMNFDPYVSQVLAMHYTARDGDADRGRAIFQ
ANCAVCHGIQADMTNIRKSHPLLAVA:FHSFIDLPTPSNISTWWNF
GSLLGACLIIQILTGLFLAMHYTARDVNYGWLIRYLHANGASMFFICLYL
HVGRGLYYGSYMFTETWNIGILLLFTVMWGATVITNLLSAIPYIGTTLVEWIW
GGFSVDKATLTRFFAFHFSLPFIITALVLVHLIDSNSDKIPFHPYYTIKDILGLLL
MLTALLILVLFSPDLLGDPDNYTPAMHYTARnDNPLNTPPHILRSIPNKLG
GVLALIFSNaWWmFIIPLLHTSKQRSMLFRPFSQCLFWILAANLLILTW
```

Figure 2.1: Motifs in a Protein Sequence

2.1.2 How is a motif represented?

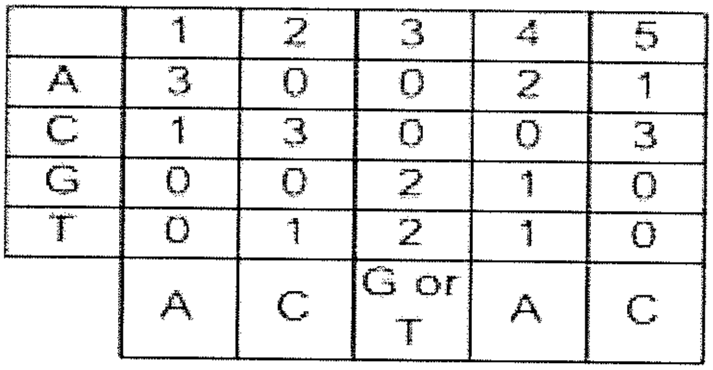
A motif can be represented as a Position Weight Matrix (PWM), Hidden Markov Model (HMM) [40] or as a consensus string. The PWM and the string based representations are the most common ways for expressing a motif. In a PWM, given the four bases A, C, G and T and string l, a matrix of size $l * 4$ is created where each slot in the matrix indicates the probability of each base being in that column. For example in Figure 2.2 [36], the probability of having base A as the first letter is 0.1. If $S = ACAAT$, then $P(S)$ (probability of S) is $0.1 * 0.8 * 0.5 * 0.15 * 0.1$. If this probability is greater than a certain threshold then it is called a *binding site*. In the



	1	2	3	4	5
A	0.1	0.2	0.5	0.15	0.5
C	0.0	0.8	0.0	0.15	0.3
G	0.9	0.0	0.0	0.55	0.1
T	0.0	0.0	0.5	0.15	0.1

Figure 2.2: Position Weight Matrix [36]

string based representation, a motif is represented as either a consensus sequence or a regular expression. For example, over the alphabet A,C,G,T, a fixed length $l = 4$ has 4^4 or 256 strings of "candidate" motifs. In this example, the entire search space is defined. As can be seen this is an exhaustive procedure. There are other flexibilities that can be introduced in the string based representation. For example, we can allow d mismatches, say $d = 1$, then the candidate motifs, "AAAT", "AATA", "ATAA", etc. are all matches to the motif, "AAAA". We can construct a matrix representing the entire candidate motifs. The matrix indicates the number of times each base (A,C,G and T) occurs at each position in the candidate motifs. The bases that have maximum frequency in each column form the consensus sequence (or motif). If the frequency of two bases is equal in any sequence, then the motif can have two representations. For example, the matrix in Figure 2.3 [36] represents two motifs namely "ACGAC" or "ACTAC" as indicated by the last row in the matrix (note that G and T have same frequency in column 3). The consensus string model is an enumerative algorithm and is guaranteed to produce the optimized model since every possible motif is considered. It is an exhaustive search process. It can handle lengths of less than 10. The running time of the model is, therefore, exponential. In this thesis, we will use the consensus string model.



	1	2	3	4	5
A	3	0	0	2	1
C	1	3	0	0	3
G	0	0	2	1	0
T	0	1	2	1	0
	A	C	G or T	A	C

Figure 2.3: Matrix for deriving Consensus Sequence [36]

2.1.3 Why is motif finding difficult?

Motif is like an encrypted message in a large array of characters. Motif searching would have been easier if all the motif patterns were exactly the same. However, as it can be seen from Figure 2.1, all motifs are not exact replicas of each other. They differ by at least two positions. The motifs patterns change because they undergo mutations. In addition to that there is no standard definition for a motif structure. A motif can be of any length (usually up to 20 bases long). The huge amount of data available for analysis makes motif finding even more difficult.

Till now we have discussed motif discovery in DNA sequences only. However, motif finding has tantamount importance in the three dimensional proteins. The following section will explain the structure of proteins, their importance in biology and the application of motif discovery in proteins.

2.2 Proteins

In a human body, proteins account for the tissues, muscles, the immune system and numerous functions within the cell. Not only in human beings, but proteins play a vital role in the life cycle of living organisms. Proteins are essentially manufactured by 20 amino acids. A human body makes few of those amino acids, while other amino acids come from protein diet. Amino acids are important because they have the ammunition to synthesize proteins. Amino acids are represented by all letters in the English alphabet except the following "B, O, U, J , X, Z". The arrangement of amino acid sequence in proteins define their structure and the functional properties.

2.2.1 Structure of Proteins

Proteins have 3 main structures namely: primary, secondary and the tertiary structures.

The primary structure is just a single dimensional, linear amino acid sequence [19]. Figure 2.4 [28] shows an example of a primary structure of protein called Lysozyme [28]. Lysozyme protects us from bacterial infection by destroying the cell walls of the bacteria. Lysozyme is part of the human body and it is present in human mucus and tears. Hydrogen bonds among the amino acids in the primary structure result in the secondary structure of proteins [19]. In this phase, the straight chains are twisted to

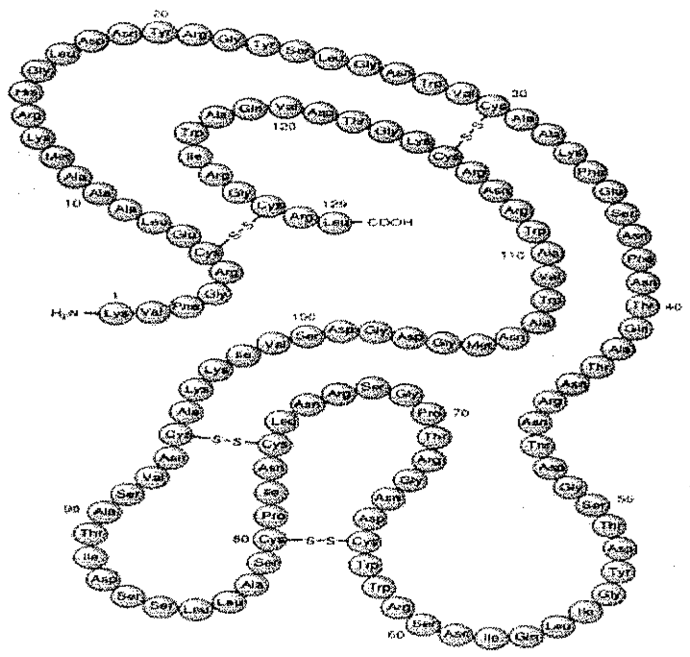


Figure 2.4: Primary Structure Protein [28]

form alpha-helix (coil or spiral), beta-sheets or strands and sometimes turns or loops. Alpha-helices and beta-sheets are shown in Figure 2.5 [14]. The secondary structure of proteins are more stable in comparison to the primary structure of proteins. The secondary structure gives more strength and flexibility to the proteins. Alpha-helices can be found in myosin (muscle protein), alpha-keratin (protein of hair). Beta-sheets can be found in beta-keratin or fibroin (protein found in silk). This gives us an idea of how strong the secondary structures of proteins are. There is an intermediate stage between the secondary structure and the tertiary structure, known as the super-secondary structure. These structures are often treated synonymous with structural motifs in proteins. We now know that secondary structure of proteins can have an alpha-helix. Super-secondary structures are a step ahead with more two or more than two alpha-helices, which may also include turn or loops. Tertiary structure of pro-

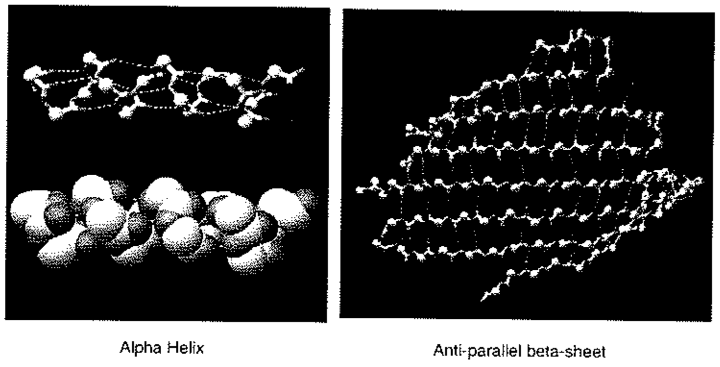


Figure 2.5: Secondary Protein Structure [14]

teins are complete three dimensional proteins formed as a result of attractions between alpha-helices and beta-sheets in secondary structures. When the secondary structures are folded into three dimensional structures, they become the tertiary structure of proteins. Figure 2.6 [15] is a 3D illustration of a protein called cytochrome with four helices. The tertiary structure determines the function of protein. Though we know that tertiary structures are 3D, it is really difficult to predict the actual structure of proteins. The linear chain of amino acids (primary structure of proteins) determine the tertiary structure of proteins. The coding information present in the DNA encodes information for the primary structure of proteins only. Hence, we can only predict the tertiary structure of proteins from their primary structures [19]. Protein structure prediction is one of the major research topics in Bioinformatics.

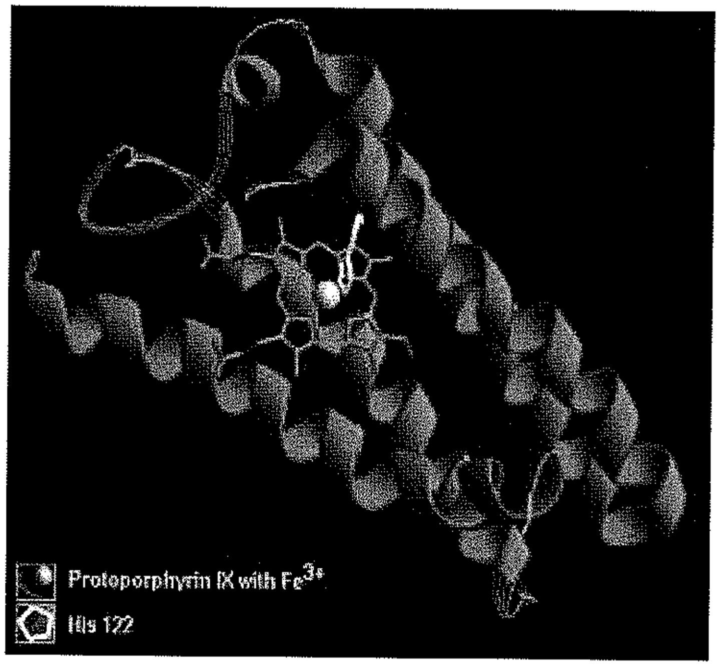


Figure 2.6: Tertiary Protein Structure [15]

2.3 Protein Structure Prediction

Proteins can be classified on the basis of similarities in their sequences and structure [19]. As discussed earlier, the 3D protein structure can be predicted from its amino acid sequence. However, similarity in sequences does not always reflect the similarity in the structures and vice-versa. Actually the number of sequences outnumber the total number of structures which are discovered to date, making the protein structure prediction even more difficult. Therefore, the protein structure prediction problem can be formulated as: Given a single-dimensional protein sequence, can we predict its exact three-dimensional structural element (tertiary structure)?

About 1000 protein families have been identified, whose amino acid sequences share similarities. One of the reliable ways to establish relationship between two protein families is to find similar sequences between them. Similar sequences most often imply same biochemical activity, same evolutionary origin and structural similarity. Sequence alignment is a popular approach for detecting similar regions in two or more sequences. There are two methods for performing sequence alignment, namely: pair-wise sequence alignment and multiple sequence alignment.

2.3.1 Sequence Alignment

Pair-wise alignment is essentially aligning two sequences in two rows and finding the characters or sets of matching characters. There are two kinds of pair-wise alignments, global alignment and local alignment. In global alignment similarities between two entire sequences are desired. Hence two big strings are compared at a stretch. For such comparison the two sequences should be almost similar. Hence global alignment is useful while comparing two protein sequences of the same family (most of the region is conserved in protein sequences). Local sequence alignment on the other hand, locates high density areas of similarity and ignores the regions that show less or no similarity. It is quite useful to find regions of similarity in sequences that belong to different families (or species) but still have few conserved regions in common. Figure 2.7 is an illustration of local alignment and global alignment.

Multiple Sequence Alignment (MSA) is aligning more than two sequences in such a way that most common regions in all the sequences are arranged in order. MSA is more reliable than the pairwise alignment because the pair-wise similarities do not exhibit regions of weak-similarities between sequences. Hence the similarities traced by pairwise alignments are statistically less significant [16]. Whereas in MSA, many comparisons are made simultaneously, thereby allowing to unearth very weakly similar

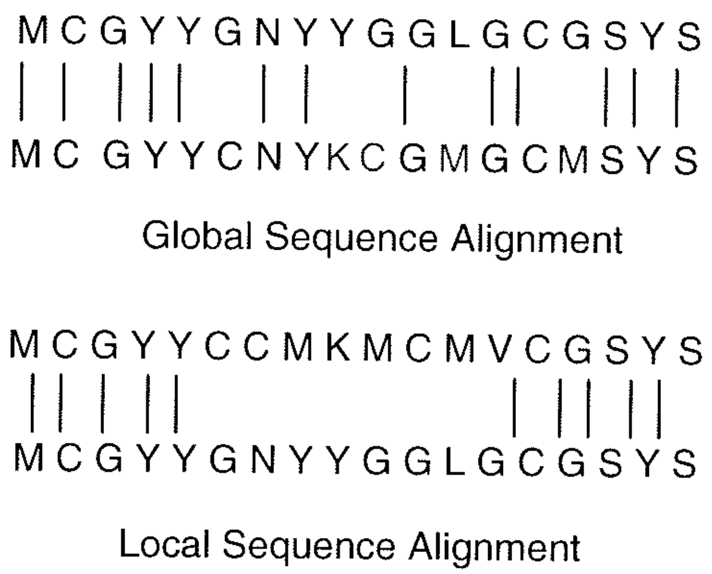

M C G Y Y G N Y Y G G L G C G S Y S
M C G Y Y C N Y K C G M G C M S Y S
Global Sequence Alignment
M C G Y Y C C M K M C M V C G S Y S
M C G Y Y G N Y Y G G L G C G S Y S
Local Sequence Alignment

Figure 2.7: Local and Global Sequence Alignment

regions.

All the alignment procedures discussed until now use "score" to determine the quality of alignment. Any alignment with best score is considered to be optimal alignment. Dynamic programming algorithms are considered to give mathematically most optimal solution. The technique is to reuse the previous solution and recursively define the optimal value for every sub-problem. Finally an optimal solution is derived from the partial solutions at each stage. Dynamic algorithms can be applied for global alignment, local alignment and local multiple alignment. The only drawback with these algorithms is, they are suitable for limited data size, and hence cannot be applied at a genomic level.

There are two approaches to perform multiple sequence alignment namely : global multiple sequence alignment and local multiple sequence alignment. As mentioned earlier, global MSA is an alignment that is performed on the entire sequences. Proteins that belong to same family and have large scale sequence similarities are suitable for global MSA. One of the well known approaches to perform global MSA is *progressive global alignment*. Progressive alignment adopts dynamic programming method for performing global MSA. Here, the program starts aligning most related sequences, and incrementally adds new and less related sequences to the existing alignment. CLUSTALW [38] is a popular global MSA program that adopts the progressive alignment technique.

Local multiple sequence alignment in protein sequences locates similar patterns in a set of sequences. These patterns include "Blocks" and "Motifs". In the context of proteins, blocks are conserved regions without gaps and which occur within a protein family. However, motifs are conserved regions among a set of proteins, which may not belong to the same family but can exhibit similar biochemical activity. As is evident, motif discovery/finding is a sub-problem of local multiple alignment.

In the context of proteins, motifs are either structural motifs or sequence motifs. Structural motifs are "folds" or three dimensional structural elements [19]. These three dimensional structures are formed as a result of the combination of secondary structures. As discussed earlier, structural motifs are treated synonymous with super-secondary structures. They are a combination of alpha-helices with loops, which can be arranged in a three-dimensional space. Structural motifs are one of those special folds that occur in many proteins and which enable us to determine the biological function of a protein. Finding structural motifs is a very difficult problem. X-ray crystallography is one of the reliable experimental approaches. It is very expensive and time consuming. On the other hand, researchers are trying to develop programs which can output an amino acid sequence for a structural motif and vice versa.

Recalling the protein structure prediction problem (can we predict the 3D protein structure from a given amino acid sequence?), sequence motifs also play a vital role in protein structure prediction. As mentioned earlier, sequence motifs are conserved amino acid patterns. They are different from structural motifs. Sequence motifs are important because they help us in determining the relationship among protein families [19]. The presence of similar sequence motifs in different protein may indicate that the two proteins share the same evolutionary origin, have similar biological function and may be structurally similar[1]. Finding the family relationships among the proteins is the key to making protein structural predictions [19].

2.4 Summary

In this chapter we discussed both DNA and protein motifs. Protein structure prediction is a vast topic. Though we covered few issues such a sequence alignment, and structure motif discovery, our prime focus is on deriving the relationship between sequence motif discovery and protein structure prediction. We have also explained how sequence motif discovery is a sub-problem of local multiple sequence alignment. In this thesis, we have developed a graph-based algorithm that traces sequence motifs in protein sequences. The following chapter will discuss various sequence motif finding/discovery techniques. It will also give a description of the de Bruijn graph construction.

[1]It is not necessary that a sequence motif always codes the same super-secondary structure

Chapter 3

Motif Discovery Approaches

This chapter attempts to explain some of the best-known motif finding algorithms. Motif finding algorithms can be classified into the following categories: probabilistic and combinatorial.

3.1 Probabilistic Approach

Gibbs sampling [10] and Expected Maximization (EM) [13] are the two statistical computing algorithms that use the probabilistic models. Software packages such as Gibbs motif sampler [39], AlignACE [25], PhyloGibbs [35] and BioProspector [18] use Gibbs sampling strategy to find the transcription factor binding sites. MEME [2] is a popular motif searching software that uses the EM algorithm.

The EM algorithm finds the maximum likelihood estimates of a probabilistic motif model. Gibbs sampling algorithm uses a randomized sampling (probabilistic sampling) technique to identify the motifs. The complete explanation of the EM and the Gibbs algorithms is beyond the scope of this thesis. Also note that algorithms that adopt the probabilistic approach (for example MEME and Gibbs Sampler) detect known motifs only (the starting point of at least one motif is known). Therefore, they are less successful in detecting planted motifs whose starting points are not known (subtle motifs). In this thesis, we intend to discover unknown motif patterns. Therefore, we do not adopt the probabilistic approach.

3.2 Combinatorial Approach

Another motif finding approach is the combinatorial approach. The underlying procedure for algorithms such as TEIRESIAS [24], the Random Projections [6], SP-STAR [21], MULTIPROFILER [17], and WINNOWER [21] is a combinatorial approach.

Combinatorial motif algorithms look for the patterns that are exact or nearly exact matches of the given motif (represented in a string pattern). The combinatorial approach can be explained as follows: Given a motif pattern and N sequences, each sequence subdivided into N^* subsequences, the combinatorial approach tries to find the Hamming distance between any subsequence in the given set of subsequences with the given motif pattern. The best match is the subsequence with the least Hamming distance from the given motif. In this section we will give a brief overview of the TEIRESIAS, SP-STAR, the Random Projections, MULTIPROFILER and the WINNOWER algorithms.

TEIRESIAS [5, 24] is a pattern enumeration algorithm. A pattern can be a set of bases, which may contain gaps or substitutions. TEIRESIAS is an exhaustive pattern search algorithm that enumerates longer patterns from a set of shorter patterns. Consider the following pattern defined by the user: "AGT.TCTC..GTC". Let 'S' denote the alphabet (A, C, G and T for a DNA sequence), every pattern should begin and end with any letter defined in the alphabet S. The '.' is used as a don't care character that can be substituted by any letter from S. There are two phases in the algorithm. The first phase is called *scanning*. The scanning phase identifies all the elementary patterns with the specified number of non-wildcards in this example. In this example, "T.TCTC" and "CTCA.G" are valid short patterns. However, "AG.GG" is not a valid pattern in the given pattern. The second phase is called *convolution*. The convolution phase combines the elementary patterns obtained from the first phase to get a longer valid pattern. In the above example the patterns "T.TCTC" and "CTCA.G" combine to form "T.TCTCA.G". In the similar fashion the algorithm tries to extend all valid elementary patterns on both sides of the current pattern to generate new long patterns. The process continues until more specific patterns that satisfy the user constraints are generated. Unlike the MEME and Gibbs sampling algorithms, which find approximate motifs, the TEIRESIAS algorithm guarantees to find the best and exact motif pattern. Since TEIRESIAS is an exhaustive search algorithm, in a worst-case scenario, it takes exponential time.

Pevzner and Sze [21] address a problem known as the *Motif Challenge Problem or a Planted motif problem.* The challenge problem attempts to find an unknown motif ($length = 15$) with at most 4 mismatches in a set of sequences each of length 600 bases. Such motifs are called subtle motifs which are intractable by popular algorithms such as MEME, Gibbs motif sampler and the TEIRESIAS. Whereas, the Random Projections algorithm by Buhler and Tompa [6], SP-STAR, MULTIPROFILER and the WINNOWER algorithms solve the challenge problem.

Pevzner and Sze [21] introduce the SP-STAR algorithm to solve the challenge problem. SP-STAR uses a scoring function called the *sum-of-pairs* function to choose an initial motif. After the initial motif is chosen, SP-STAR uses a local improvement heuristics to improve the initial motif. The local improvement strategy can employ either a consensus string strategy, or a Gibbs sampling strategy to find all the subsequences that maximize the scoring function. SP-STAR shows good performance (returns correct motifs) for sequences less or equal to 800. However, the performance degrades and SP-STAR generates random motifs instead of the required motifs once the length of the sequence is over 900.

Though SP-STAR successfully finds the (15,4) motifs (that is, motifs of length 15 with 4 mismatches), it was unsuccessful for more subtle patterns such as the (18, 6) patterns. Buhler and Tompa [6] provide a solution for tracking difficult planted motifs such as (16,5) and (18,6) in twenty sequences each of length 2000 using random projections. A projection is achieved by partitioning a sequence in a way such that all candidate motifs are hashed into n different buckets. The buckets that receive an unusually higher number of candidate motifs will have a higher probability to produce subtle motifs. A local searching technique such as the Gibbs sampling technique or the MEME can be used on each bucket to unearth the desired motif patterns. Similar to SP-STAR, the random projections algorithm provides a platform for the local searching algorithms to discover more subtle patterns. Hashing candidate motifs into buckets will give better starting points for the local searching algorithms, thus enabling them to hit upon remote motifs which are generally missed by other motif finding algorithms.

To further push the limits of subtle motif discovery, Keich and Pevzner [17] propose the MULTIPROFILER algorithm. MULTIPROFILER is based on an approach called the *Extended Sample Driven* (ESD) approach. ESD approach tests subsequences on just a *sample* sequence without testing it on the entire data range. First, ESD approach tries to find/generate all the *k-neighbours*[1] to a given substring in the given sample sequence. By generating all the k-neighbours[2] for a given substring m, it

[1]If two substrings differ from each other by at most k number of substitutions/mutations.

[2]MULTIPROFILER actually restricts the k-neighbourhood to just 2k neighbours for each sub-

is likely that we have the mutated form of that substring in all other sequences. Upon comparison of m with the k neighbours, it is easy to find which base is over-represented (occurring a lot of times) in each position, thereby drawing a consensus pattern. Thus, the ESD approach estimates the initial motif. MULTIPROFILER made two improvements to the ESD approach. First, MULTIPROFILER maintains a dictionary of the k-neighbours for every substring. Secondly, it uses multi-positional profiles. A single positional profile is just like a consensus matrix shown in Figure 2.3. A bi-positional profile is a matrix that contains the two letters (such as aa, ac, ag) in each cell of the first column and two positions (such as $(1,2)$ $(1,3)$) in each cell of the first row. The actual consensus patterns are deduced using the dictionary. Multi-positional profiles are employed to separate the actual pattern from the many random words or *noise* in the dictionary. MULTIPROFILER detects motifs that are usually intractable by the Random Projections. It has given 98% results for detecting (15,4) motifs from 20 sequences each of length 3000.

Pevzner and Sze [21] propose a graph based motif finding algorithm called WINNOWER [21, 5]. Since the thesis is based on a graph theoretical approach, we will explain the WINNOWER algorithm in detail. WINNOWER divides each sequence into subsequences. It builds a multipartite graph using the subsequences. Two "similar" subsequences in different sequences have an edge. The motif finding problem is reduced to finding cliques in the graph. The cliques represent motifs. Therefore, motif discovery is clique discovery in the multipartite graph. WINNOWER algorithm uses a technique that iteratively removes edges that are not part of any clique. The final graph contains only cliques that represent motifs. A cluster of cliques represents a cluster of motifs.

string to save time.

The WINNOWER algorithm takes a set of unaligned sequences as input each of length n. The user provides the length (l) of the motif required along with the allowable number of mismatches (d). Consider S is a set of sequences, S = $\{s_1, s_2, .., s_N\}$. Every sequence s_i is broken in a subsequence at every position j where $i = j < (n + l) - 1$. For example, if the sequence length is 6, then 3 subsequences each of length 4 can be generated. Therefore, the total number of subsequences of length l that can be generated from a given sequence of length n are $n + 1 - l$. Every subsequence becomes a vertex of a graph. Two vertices have an edge only when the hamming distance between them is at most twice the number of allowed mismatches. So if d is the allowed number of mismatches in a sequence, then the number of actual positions where the substrings in both vertices will differ is at most $2d$. Also, an edge exists between two vertices only if the vertices are in two different sequences. The edges represent the actual motif in question. Figure 3.1 [36] and Figure 3.2 [36] illustrate graph construction and edge filtering. Upon construction of the graph, the

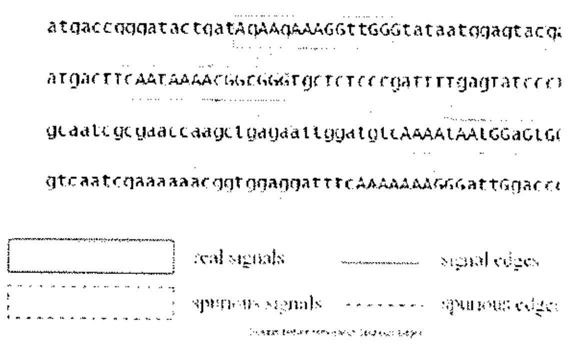

atgaccgggatactgatAgAAgAAAGGttGGGtataatggagtacg
atgacTTCAATAAAACGGCGGGTgCTCTcccgaTTTTgagTaTccc
gtaatcgcgaaccaagctgagaattggatgtcAAAAtAAtGGaCtGC
gtcaatcgaaaaaacggtggaggatttcAAAAAAAGGGattGgacc
real signals
signal edges
spurious signals
spurious edges

Figure 3.1: Clique formation [36]

algorithm tries to locate large cliques in the graph. According to Pevzner and Sze a dense multipartite graph with sequence length of 600 produces as many as 20,000

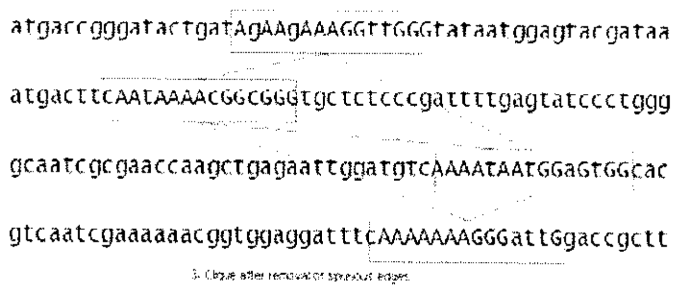

atgaccgggatactgatAgAAgAAAGGttGGGtataatggagtacgataa
atgacttcAAtAAAACGGCGGGtgctctcccgattttgagtatccctggg
gcaatcgcgaaccaagctgagaattggatgtCAAAtAAtGGaGtGGcac
gtcaatcgaaaaaacggtggaggatttcAAAAAAAGGGattGgaccgctt
3. Clique after removal of spurious edges

Figure 3.2: Cliques without edges [36]

"spurious" edges (edges that are not part of any clique, but still seem to be realistic signals to a motif). Locating a large clique in the graph is an NP-complete problem. Therefore, instead of locating cliques first, WINNOWER uses an iterative filtering strategy to prune all the spurious edges and unwanted vertices from the graph. In summary, WINNOWER works in two steps. First, construct a graph with edges between substrings (vertices) in different sequences and having a hamming distance of at most $2d$. Figure 3.1 [36] shows the graph construction with the sequences, the edges connecting the sequences and the cliques. The dotted lines represent spurious edges and sequences. Second, identify the edges that are not part of any clique and prune them. This step also involves locating and removing unconnected vertices (weak vertices). Perform the second step iteratively until all the spurious edges are removed and only cliques remain. Figure 3.2 [36] shows the cliques after edge filtering.

WINNOWER is tested for finding motifs of length 15 with 4 mismatches (15,4) in twenty sequences with 600 bases [21]. While other motif finding software tools such as MEME and Gibbs Sampler failed to recognize significant motifs in such long sequences, WINNOWER gave a performance of over 80% even for sequences of length 700 bases. However, WINNOWER suffers from certain disadvantages. Pevzner and

Sze [21] state that WINNOWER is a computationally intensive algorithm and requires substantial amount of time and memory. It requires lot of memory to store the values of the edges. Also a huge amount of time is spent on detecting the spurious edges. Pevzner and Sze also indicate that WINNOWER slows down for larger samples of sequence data.

The algorithms detailed so far are the most state-of-the-art and powerful motif discovery algorithms designed to find subtle motifs. Our algorithm has close resemblances with the WINNOWER strategy, which is also graph based. The similarity between the graph used in the WINNOWER strategy and the de Bruijn graph is that both break the sequences into overlapping sub-sequences. Both consider subsequences (character strings) as vertices. However, in the de Bruijn graph, unlike the WINNOWER strategy, an edge exists between any two overlapping subsequences in the same sequence. In a de Bruijn graph since every l-tuple is represented in the form of an edge, every repeating subsequence (in this case a motif) also becomes an edge. Therefore, the edges form the potential sites for conserved regions. The following section explains the construction of the de Bruijn graph.

3.3 Construction of a *de Bruijn* graph

A de Bruijn graph is a graph whose vertices are subsequences of a given sequence and whose edges indicate the overlapping subsequences. Consider the following sequence ACCGTCT. The sequence can be resolved into the following fragments of length 4: ACCG, CCGT, CGTC, and GTCT. Each fragment is called an l-tuple. An $l-1$ tuple is obtained by further fragmenting each l-tuple. For example, ACC and CCG are $l-1$ tuples of $ACCG$. The $l-1$ tuples form the nodes of the de Bruijn graph. An edge exits between any two $l-1$ tuples and the edges represent the l-tuples. Figure 3.3

illustrates the 3-tuple de Bruijn graph construction for the sequence ACCGTCT. The

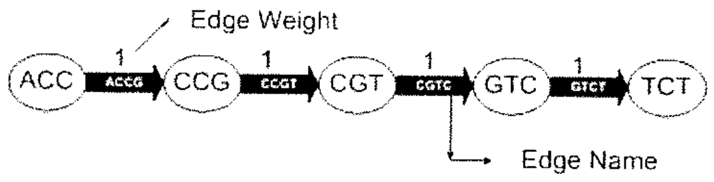


Figure 3.3: Construction of a de Brujn Graph

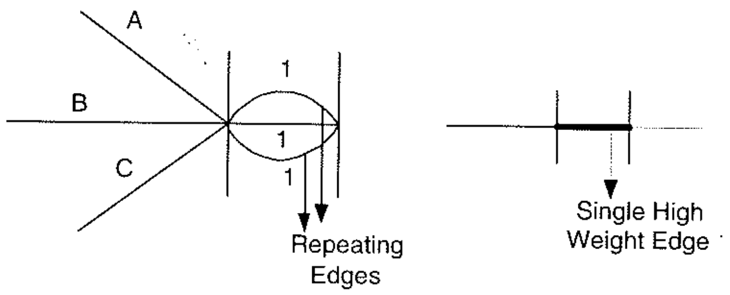


Figure 3.4: Gluing of Repeating Edges

multiplicity of each edge is represented by an edge with a weight. Initially every edge gets a weight 1. In case two consecutive vertices (vertices with the specified overlap) repeat, then the edge multiplicity is increased (in this case an increment in weight of the edge by 1). Similar to the strategy employed by Pevzner et al. [22], we glue the repeating edges thereby increasing the multiplicity of one single edge as shown in Figure 3.4. The conserved regions are most likely to reside on the most repeated edges, i.e., edges with greater multiplicity (which have greater weight attached to them).

3.4 Summary

In this chapter, we provide the background work on probabilistic and combinatorial approaches for the motif discovery. We explain the construction of the de Bruijn graph upon which this thesis is based. Implementing the de Bruijn graph on a parallel platform is difficult because it is basically a recursive graph. Therefore, there are a lot of dependencies. So it is important to identify regions which are fit for parallelization. The following chapter will provide an introduction to parallel computing.

Chapter 4

Parallel Computing Platforms

A sequential computer may take hours or sometimes days to solve a computationally intensive problem. Parallel computing comes to the rescue of such problems. Parallel processing allows multiple processors to work concurrently to solve a time consuming problem. It offers additional computing resources such as CPU cycles and memory.

In 1966 Michael J. Flynn [9] proposed a classification for parallel computing architectures popularly known as Flynn's taxonomy. Flynn categorized parallel computers into Single Instruction Single Data (SISD), Single Instruction Multiple Data (SIMD), Multiple Instruction Single Data (MISD) and Multiple Instruction Multiple Data (MIMD) computers.

SISD systems are essentially serial computers. Only one instruction is processed on a single data stream. Therefore the least amount of concurrency is exhibited by SISD computers. Classical PCs and workstations with single CPU fall under this category.

MISD is a set of processors acting on the sub-problems of a single problem. It is basically asynchronous in nature and has very little importance in the present scenario of parallel computers.

In SIMD systems(see Figure 4.1 [11]), a single instruction is executed on several processing units (CPUs) [11]. However, unlike SISD and MISD systems, the same instruction is executed over multiple data units. Every processing unit is independent of each other. The system is synchronous in nature and all the processing units are controlled by one global control unit. SIMD is suitable to work on data structures

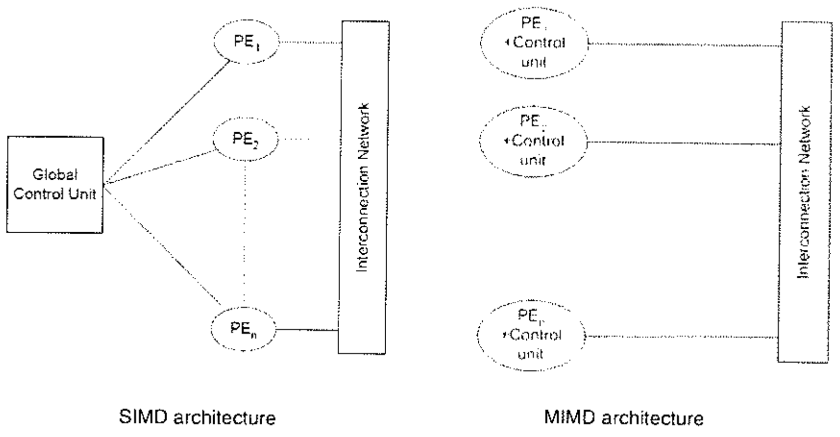


Figure 4.1: Control Structure of Parallel Platforms [11]

such as arrays and vectors. The data can be equally shared among all the processors. Consider the following addition application $c[i] = a[i] + b[i]$. In this operation processor 1 can take elements from $i = 1$ to 10. Processor 2 from $i = 10$ to 20, etc. SIMD systems do not show good performance when conditional statements (such as if, else) are used and also in cases where data cannot be distributed equally among all the processors [11]. Consider a scenario where there is a conditional statement such as "if.. else". In the SIMD systems a single instruction should be executed on all the processors. In this case, if certain condition is true one set of processors run while others are idle and the same in case of an else condition.

SIMD systems are increasingly becoming obsolete because of the following factors:

Extensive design which leads to greater production times, poor resource utilization (all the processors might not be utilized at the same time when data size is not uniform) and performance highly depends on the nature of the application.

MIMD is the modern day parallel computer. These systems are elegant both architecturally and functionally. Several processing elements work on different instruction sets and work on different sets of data simultaneously. There is no centralized control for all the processing elements. Figure 4.1 [11] illustrates the difference between SIMD and MIMD systems. Execution of the tasks is not necessarily synchronous, thus making the system very flexible.

Workstation clusters, Symmetric Multiprocessor (SMP) processors, grid computers and most present day super computers fall under this category. Most of the MIMD machines fall into the following categories: Shared memory machines and distributed memory machines.

4.1 Shared Memory Machines

As the name suggests, shared memory machines have a common address space which is accessible to a set of processors. All the processors work independently while sharing the same memory location. Figure 4.2 illustrates the share memory architecture. Shared memory is classified into two models: Uniform Memory Access (UMA) and Non-Uniform Memory Access (NUMA)

UMA machines have a set of processors with identical processing speed and hardware configuration. All processors have equal access to memory. Sometimes UMA machines are also known as Symmetric Multiprocessors (SMPs). Explicit communication among the processors via message passing at the hardware level is not required in the case of SMPs. Programming in SMPs is also quite simple and owing to the

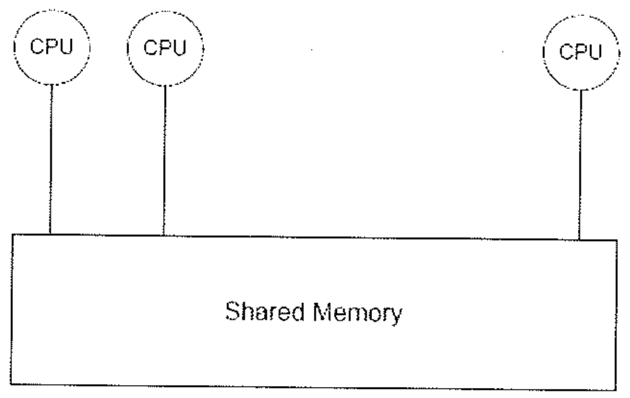


Figure 4.2: Shared Memory Architecture [11]

global address space, data sharing is very easy too.

A network of SMPs form a NUMA machine, also known as the "distributed shared memory machines". In these machines, memory access is not uniform. Since the SMPs are connected, one SMP can access the memory of another SMP. However, the access time is not uniform as in the case of UMA machines.

The major disadvantage of shared memory machines is their inability to scale for large number of processors. Synchronization among the tasks is the responsibility of the programmer. Hence it is quite important to use locks and critical sections in shared memory programming.

4.1.1 Parallel Programming using OpenMP

OpenMP is becoming a standard programming language for shared memory machines. It is a directive based Application Programming Interface (API) which allows programming in C, C++, Fortran and Java. Apart from compiler directives, the OpenMP API consists of runtime routines and environment variables [4]. Using these components of the API, OpenMP can convert a serial code written in C, C++, etc., into a parallel code. One of the goals of OpenMP is to provide a standard for programming on shared memory architectures.

All the OpenMP directives are written along with the normal C++ or Fortran code. The program runs normally in the serial format until an OpenMP directive is traced by the compiler. Every OpenMP directive follows certain format. A typical OpenMP directive is as follows: "*pragma omp [directive name][clause[clause]..]*". Every OpenMP directive is preceded by *pragma omp*. If we want to create a parallel region where all the processors participate, then the appropriate directive is "parallel". The clause can be an "IF" clause. If certain statement evaluates true, then the set of OpenMP threads are created in the parallel region. This is followed by the variable list. Shared variables are equally accessible to all the processors. Private variables are accessible only to the individual processors accessing them. Here is an example of an OpenMP directive for parallelizing a "for-loop": "*pragma omp parallel for default(shared) private(i,j,myvariables)*".

Programming using OpenMP is easy because by adding few parallel directives to the serial code, one can convert a serial program to a parallel program. No explicit communication among the processors is required. Figure 4.3 [4] illustrates a typical fork-join model that is adopted in an OpenMP program. The foundation of OpenMP is on the concept of multi-threading. Every OpenMP program begins with a "master thread" [4]. The program is essentially serial until a parallel OpenMP compiler directive is encountered. Once a parallel compiler directive is seen, the OpenMP program forks into a set of threads. This region is also known as the parallel region. Each thread is independently executed by a separate processor or shared among the processors. Once the threads are executed, they join into the master thread and the program is again serial from that point.

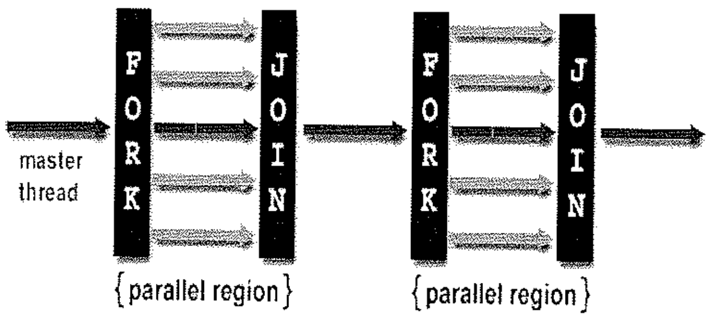


Figure 4.3: Fork-Join model [4]

4.2 Distributed Memory Machines

A cluster of interconnected workstations which do not have a common memory, but have a local memory, can be called as Distributed Memory machines. In such an architecture, explicit message passing among the processors is required for communication. Therefore, the network bandwidth and the network topology influence the performance of the parallel algorithm to a large extent. There are various topologies suggested in the literature such as bus, tree, star, mesh, hypercube etc. It is very important to choose the right topology based on the interconnection among the processing elements. Since distributed memory machines rely on message passing for communication, network bandwidth plays a vital role in the time taken for the arrival of the messages. Figure 4.4 [11] gives a picture of the distributed memory network. Scalability is the biggest advantage in distributed memory systems. Every processor has its own local memory, there is no need to explicitly synchronize the tasks. Parallel Virtual Machine (PVM) and Message Passing Interface are two main libraries for distributed memory machines. Since we have employed MPI in our thesis, we would like to explain few features of MPI. MPI is the current standard for message passing on homogeneous cluster of machines. MPI is portable as it can be

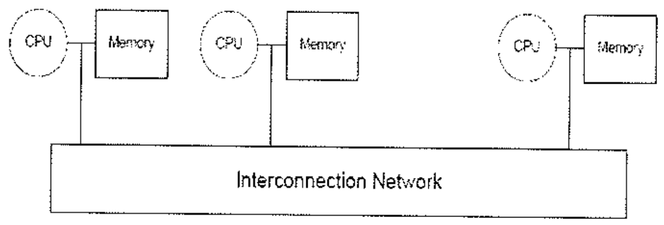


Figure 4.4: Distributed Memory Model [11]

run of various platforms (which have an implementation of MPI such as Lam-MPI) without changing the source code [4]. The parallelism in MPI is completely specified by the user. Similar to OpenMP, MPI creates its own parallel environment within a program. Within the environment MPI recognizes the total number of processors to be used and assigns a rank for each processor starting from zero. The communication among the processors from this point will be via their rank.

4.3 Hybrid Model

The hybrid shared-distributed-memory model is cluster of machines which can support both shared-memory and distributed memory mechanisms. Figure 4.5 [4] illustrates a typical hybrid model. There is no particular API which enables us to write hybrid programs. However the programs should include directives from API's which support shared and distributed programming. A cluster of SMPs is a good example for a hybrid model. As discussed earlier, SMP's are primarily shared memory machines. However, where there is an interconnection among a set of SMP's they behave just like a cluster of work stations with distributed memory. Hence a shared memory can be used for smaller tasks within the node and for large scale parallelism, the nodes across the cluster can be utilized. The hybrid model is the current and future

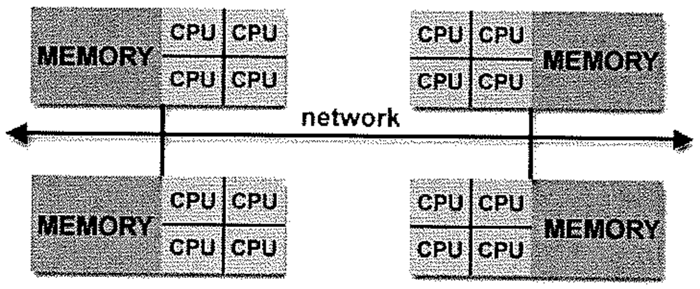


Figure 4.5: Hybrid Model [4]

trend in parallel computing. It has the advantages of distributed and shared memory machines. They are currently used in all High Performance Computing (HPC) operations.

4.4 Summary

In this chapter we have discussed various parallel computing trends. We have explained the early classification of parallel computers. All the modern parallel computers belong to the MIMD class. We have discussed the advantages and disadvantages of shared and distributed memory models. In this thesis we have adopted a Hybrid model. We have used the high performance computing facilities available on the Westgrid [27] consortium for our experiments. We will explain more about application of the hybrid model on the motif discovery problem in Chapter 6. The next chapter will discuss previous research efforts to parallelize the motif discovery problem.

Chapter 5

Related Work: Parallel Approaches to Motif Discovery

In this chapter, we will describe the literature and previous research efforts that adopt parallel computing techniques for the motif finding problem. Software tools have been developed that apply parallelization to the graph-based approach. We will describe parallel motif discovery tools such as ParaMEME [12]. Specifically, we will elaborate on two recent parallel motif discovery programs namely "Motif discovery toolkit" by Baldwin et al. [3], and ParSeq by Qin et al. [23].

ParaMEME is a parallel program for the MEME software. As we have mentioned earlier, MEME is based on a probabilistic approach that uses the Expected Maximization (EM) algorithm. Ever since MEME has become a popular motif searching software, the number of users accessing the system simultaneously has increased. Therefore, MEME was ported over to the 400-node Intel Paragon XP/S platform using a parallel programming language called MP++. The ParaMEME is scalable for up to 64 processors. The original MEME has 4 *for loops*. ParaMEME parallelizes two out of the 4 *for loops*. Every job submitted by the user is assigned to 8 nodes of the cluster. Grundy et al. [12] state that ParaMEME shows an efficiency of over 70% even for large sequence datasets.

Sutou et al. [37] propose a parallelization of a motif discovery method called the Modified PrefixSpan method. PrefixSpan is a data mining technique used to extract frequent patterns from a database. Modified PrefixSpan is a variation of the original method designed for faster pattern extraction. To reduce the computational time taken, the Modified PrefixSpan method is parallelized on 8 processors. The parallelization method adopts a dynamic load balancing scheme, where the master processor generates a job pool and each slave processor gets data from the job pool. Every slave processor extracts most frequently repeated patterns. Once a slave processor completes its job, the master job generates more jobs and inserts them into the job pool. Sutou et al. state that the parallel version of the Modified PrefixSpan is 6 times faster in comparison to the serial version.

Baldwin et al. [3], develop a motif discovery tool called *Motif Discovery Toolkit*. The toolkit uses graph algorithms to predict the gene regulation sites. Owing to the speed and efficiency of the graph algorithms when compared to other motif finding algorithms, the developers of the toolkit choose to make extensive use of techniques applied in state-of-the art graph algorithms. To discover the binding sites (motifs) in a larger sample of data (the complete genomic sequence, not just fixed length sequences), the toolkit implements the graph algorithm on a cluster of homogeneous nodes.

There is striking similarity between the working models of the WINNOWER algorithm and the motif discovery toolkit. The toolkit resolves the unaligned and arbitrary sequences into overlapping subsequences, each with a fixed length. It constructs a graph using the subsequences as vertices. To identify the complex interconnected regions in the graph (vertex cover) the toolkit employs the graph algorithms such as WINNOWER. The motif discovery tool not only identifies clusters of motifs, but also builds motif models (PWMs representing the potential motifs) from the cluster information.

During the graph construction phase, the motif discovery toolkit constantly builds a graph of lesser size and lower density using a vertex reduction technique called *kernelization* [31]. The final optimal graph is called the kernel. The parallelization process begins after the kernel is built. The second phase called *branching* uses a tree structure to probe the search space of the kernel. The master processor contains a scheduler that resolves the entire kernel into independent subtrees and distributes the subtrees to the slave processors on a manager-worker basis (jobs are assigned to the slave processors on demand dynamically).

Qin et al. [23] develop a parallel motif searching tool based on a serial motif finding tool called *ParSeq* [17]. ParSeq is not a graph-based motif finding algorithm. ParSeq uses a query (which is in the form of a regular expression such as $(AG|TC)^*$), which incorporates biochemical and structural properties of the motifs while making the motif search.

ParSeq [33] uses an approximate search algorithm that returns the best character strings based on the regular expression. The client provides the regular expression. For example the expression (taken from ParSeq documentation) "$(ACT)@X10/hdp_kd(>0)$" stands for a definite string ACT followed by a character string with 10 arbitrary characters. The character string should posses hydrophobicity score bigger than 0 according to Kyte/Doolitle score [31]. Every regular expression generates a certain number of hits. The query can be changed and be made more specific to obtain more accurate motifs. ParSeq works as a filter that will further narrow down the search.

Qin et al. [23] parallelize the serial ParSeq [33]. The search is parallelized on client-server based architecture. The parallel search is performed on the server side, while the client places the request using a query. All the processors on the server side access the sequence database to search for the motifs. The search space in the database is equally divided into n chunks for n processors. Each processor gets its own chunk of sequences against which they search for the motif patterns. The parallel version of ParSeq is tested on a homogeneous network of 32 processors. Qin et al. [33] propose to implement ParSeq on a heterogeneous network in future.

ParSeq is an advanced motif searching technique that is designed to search for more variable motifs that are associated with certain biochemical properties. ParSeq expects the client to read the documentation for building the regular expressions because all those details are only specific to ParSeq. The number of hits purely depends on the description of the motif in the query. If the description is inaccurate, then the query will return too many matches, which is apparently a loophole in the software.

5.1 Summary

Our work involves parallelization of a graph algorithm. We use protein sequences with variable length as test data. Both the WINNOWER algorithm and the motif discovery toolkit developed by Baldwin et al. [3] do not seem to consider a weighted graph. Also, they consider a clique or a vertex cover to represent motifs. We used a directed weighted de Bruijn graph instead, and in our algorithm, the edges represent potential motifs, not the complex interconnected structures. The following chapter will provide an outline of our parallel algorithm.

Chapter 6

Sequential and Parallel Algorithm

In this chapter, our graph based algorithm is explained. The algorithm works in two stages. In the first stage the de Bruijn graph is constructed as explained in the section 3.3 of chapter 3. The second stage is the graph traversal stage.

6.1 Graph Construction

A de Bruijn graph is a graph whose vertices are subsequences of a given sequence and whose edges indicate the overlapping subsequences. Initially a protein sequence is broken into a set of l-tuples. Each l-tuple is again broken into a set of $l-1$- tuples. An edge joins two overlapping $l-1$-tuples as shown in Figure 3.3. The multiplicity of the corresponding directed edge increases when successive $l-1$-tuples within the same sequence or in a different sequence repeat. This corresponds to an edge gaining a higher weight. This procedure is known as gluing of similar edges. The advantage with such an arrangement as stated by Pevzner et al. [22] is that it is quite easy to identify different regions of similarity among a set of sequences. These high weight edges form potential sites for conserved regions. Note that all the sequences are not part of one graph but made of many subgraphs. Every sequence has it own sub-graph. However, whenever there are similar regions in two different sequences, the sub-graphs overlap. Thus, the de Bruijn graph is a set of overlapping sub-graphs as shown in Figure 6.1. Every line in Figure 6.1 represents a separate sequence.

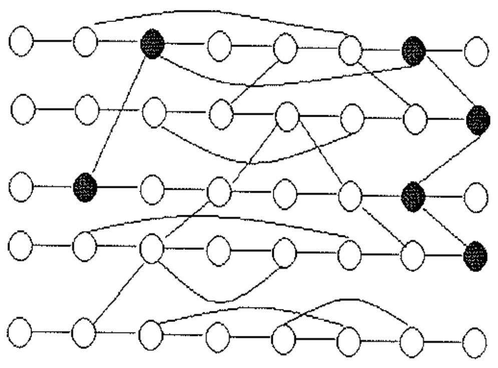


Figure 6.1: Sub-graph Overlap in Sequences

6.2 Parallelization of Graph Construction

Initially we developed a hierarchial scheme where individual nodes will construct only a portion of the graph and the entire assembly takes place on a single master node. Though the method looks feasible, we found that this method is highly cumbersome. While a few nodes do the graph construction, others remain idle. In fact we ended up performing more loops than the normal serial program. Even though there is sufficient room for further parallelization, we were not able to exploit it using distributed memory machines. In order to extract maximum parallelism while not keeping any nodes idle, we shifted to the hybrid model.

In the hybrid model every node constructs its own graph. The construction is parallelized using multiple threads. The adjacency list used to store the graph node information is used as a shared data structure. Each thread updates the shared data structure individually. In such an arrangement, the time taken to construct the whole adjacency list is equally shared among the shared memory threads, thereby exhibiting fine grain parallelism. Once the adjacency list is constructed, the graph traversal is performed on individual nodes using message passing.

6.3 Graph Traversal

The next stage is graph traversal. A hash table contains all the nodes as its keys. Corresponding to every node is its adjacent node and the edge connecting them, maintained in the form of an adjacency list. The set of graphs obtained from the graph construction procedure followed above are highly connected with numerous cycles and thus messy. The aim is to identify the dense regions of the graph where the edges have a high weight. Since there is no single path that covers all the nodes or edges, the best way is to individually traverse each sub-graph. For every sequence whose first node has an indegree zero becomes the starting point of the traversal. Hence we maintain a queue of nodes that have an indegree zero. We use a recursive Depth First Search (DFS) for performing the traversal. In order to avoid falling into an infinite loop when cycles are encountered, the algorithm visits each node just once. The corresponding nodes connected to a given node and the weights of the edges connecting the nodes are extracted from the hash table. After obtaining the high weight edges, the algorithm first tries to locate the entire repeated region called as the "active region". An active region is a high-weight region where a lot of sequences coincide upon a particular consecutive edge set. In other words, an active region is a region that contains consecutive edges that have repeated themselves. All the high weight edges need not represent motifs because there is a fair chance for them being repeats. Hence the search is for more prominent edges. Therefore we tried to identify edges that have more weight in an active region in comparison to the other edges. Figure 6.2 represents the active regions with some edges having higher weights than other. From the figure one can observe that BC and CD have a greater

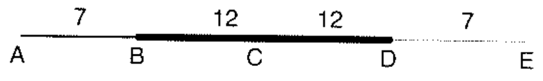


Figure 6.2: Active Region

edge weight in comparison to edges AB and AE in the active region ABCDE. This concept has actually worked in mining desirable motifs from the sequences. However our algorithm failed to recognize motifs that do not exactly match with each other, for example MNPPPQF and MCPPPQF (with the second position differing). Hence we have sought to make changes in the way the comparison of $l-1$ tuples is made in the initial stages. Our initial strategy was to look for straight $l-1$-tuple matches. Later, in order to find motif with slight mismatches, we introduce a parameter called distance, d. Two $l-1$-tuples with hamming distance 1 will be considered as one single $l-1$-tuple. The nodes that follow will connect to the given $l-1$-tuple depending on the hamming distance from the already connected nodes. This way we were able to unearth subtle or weak motifs. The hamming distance depends on the amount of overlap of the $l-1$-tuples which in turn depends on the initial motif length chosen. Therefore, a longer $l-1$-tuple (> 7) will have the distance parameter set to 2 so that the hamming distance between any two $l-1$-tuples is 2. Hence two nodes with a hamming distance $<= 2$ will be considered as similar nodes.

6.4 Parallelization of Graph Traversal

As explained above, graph traversal mainly involves tracing edges with higher than certain threshold weight. This is a completely independent process which does not have any dependencies. Therefore, the whole traversal operation can be performed individually on different processors. The graph is extremely large owing to the huge amount of test data. The graph traversal consumes a significant amount of time when performed on a single processor. In this scenario a distributed memory machine will be of greater help in comparison to a shared memory machine. This is because, during graph traversal edges may be shared by processors requiring constant synchronization which is costly. In the distributed memory machine, the data is distributed among the processors and each processor performs the traversal independently and through message passing for sharing of information. In the graph traversal, coarse-grained parallelism is exploited. Therefore, we chose to parallelize the entire traversal process on a distributed memory machine.

We adopt a dynamic "manager-worker" framework to perform the graph traversal. Every node builds its own graph using the shared memory (the process explained above). Once the graph is constructed, the "manager" builds the zero indegree queue. The size of the indegree queue depends on the number of sequences. Since the indegree queue contains the starting points for the traversal, they form the work units for the worker processors. Therefore, whenever the worker needs work, the manager sends a chunk of zero indegree nodes. The worker processor performs the depth first search, traces the high weight edges and identifies the potential sites. Once the process is complete, the worker sends the results back to the manager and requests for more work. The whole process is an on-demand process, thus making it entirely dynamic. The manager filters the repeated potential sites and maintains a final vector containing all the potential motifs. The manager worker communication continues as long as the zero indegree queue is not empty. When there is not enough work, the manager sends a termination signal to all the worker processors.

6.5 Summary

Identifying smaller motifs (length $<= 8$)and less significant motifs (motifs which differ by more than 2 positions) in a huge amount of data is a herculean task. There are two reasons, fist more number of comparisons to be made for motifs less significant motifs. The problem becomes worse when the length of the motifs that we are looking are small. If the sequence length is 100, the number of l-tuples of length 8 will be 93 $(100 - 8 + 1)$ and the number of $l - 1$-tuples is 186. If a data file contains more than 10,000 sequences, the number of $l - 1$-tuples will be 19,986. Hence the number of comparisons to be made while looking for edges greater than a certain weight (in the traversal stage) will increase exponentially. Therefore, it is important to use an appropriate data structure (such as hash table) to store the node information. The next chapter describes the implementation in more detail.

Chapter 7

Implementation and Result Analysis

This chapter discusses the implementation and results of the algorithm. The algorithm is also tested against the popular MEME for correctness.

7.1 Implementation Platform for Parallelization

The parallel algorithm is implemented on a cluster which belongs to the Westgrid [27]. Westgrid is a resource provider which adopts a grid-enabled system to facilitate High Performance Computing (HPC). Currently Westgrid is the biggest resource provider in Canada encompassing about 14 educational institutions in 4 provinces.

Clusters of computers located in geographically different locations when connected to form one massively powerful computer, which is called as a "grid". Drug design, weather forecast patterns, protein folding, earthquake simulation and particle physics experiments are some of the numerous applications of the grid.

The Westgrid consists of various clusters which vary in their design and the nature of problems they are designed to handle. For example, the nexus and the cortex group at the University of Alberta are shared memory clusters. The glacier is a 1680-processor Beowulf cluster hosted by the University of British Columbia. Hydra is a visualization facility hosted by the Simon Fraser University. Further information can be found on the official Westgrid website [27].

We have chosen to implement our algorithm on the glacier cluster. Glacier consists of 3 head nodes (available for the users) and 840 IBM eServer BladeCenter HS20 machines with dual 3.0 GHz Xeon processors. Every node has about 2 to 4 Gb of memory. On glacier, C++ is supported by Portland Group 7.0-2 (pgCC) Intel 9.1 (icc) GNU 3.2.3 (g++) compilers. Both Intel and Portland group of compilers are available for compiling MPI programs. Intel compilers are available for the OpenMP support.

We have employed the Intel compilers to compile our hybrid program. We were unable to use the Portland group of compilers because no explicit support for OpenMP was provided. Though other clusters, such as nexus, did provide compilers for hybrid programs, we did face few technical issues which are not resolved to date. An important point to notice here is that, though clusters may provide support for distributed and shared memory computing "individually", when we mix both paradigms, many technical issues arise. For example, a shared memory cluster such as "Cortex" does not have C++ support for OpenMP. Since we adopted advanced C++ data structures such as hashtables, vectors and Queue we had to explicitly provide the location of the STL (standard template library) to compile the program. On glacier, we were able to compile and run the program without any problems.

7.2 Sequential Results

The sequential algorithm is implemented using Java and C++ running on a Linux platform. Initially the algorithm was tested on the DNA sequences that belong to the prokaryotic family. The advantage with these sequences is that they have very low or negligible number of repeats. In addition to that, the common repeating patterns in the prokaryotic sequences (the "TATA" patterns) are already known. Upon successful initial testing we tested the algorithm on the protein sequences. We chose to start off with the protein responsible for redox (oxidation and reduction) reactions called "Cytochrome" [26, 32]. The initial testing began on just twenty sequences, with the longest sequence being 577 nucleotides. However, we ended up testing the algorithm successfully on 15000 sequences. In this section, we will make a comparison of our algorithm with popular motif searching tools namely "MEME" and the "Gibbs sampler". We could not compare our algorithm with Patwardhan et al.'s [20] software because the software did not provide the approximate or exact runtime for a job. Though many tools for searching motifs were available, we have chosen MEME and Gibbs sampler because they provide for testing both DNA and protein sequences. The only parameter that we ask the user to enter apart from the sequence set is the expected length of each motif. Figure 7.1 shows the comparison among the three algorithms. Both MEME and Gibbs sampler have a character limit. Therefore, we could not test them for more than 30 protein sequences each averaging 200 nucleotides. Clearly our algorithm has a speed advantage over the other motif discovery tools. This comparison just gives an approximate picture of the speed levels. The next step is testing the accuracy of the results. The graph algorithm had matched 70% of the motifs with the MEME and 65% of the patters with the Gibbs motif sampler. Out of the first 10 motifs (length = 8) returned by MEME, it had

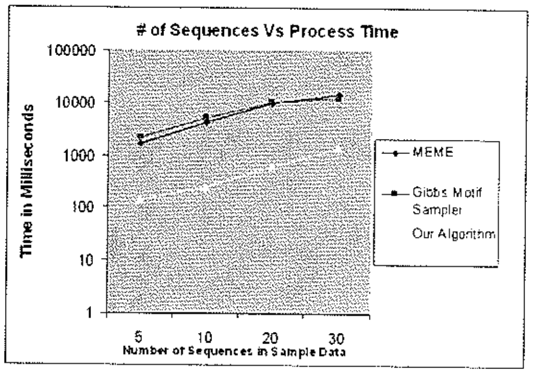


Figure 7.1: Sequences Vs Process Time

7 straight matches and two partial matches (differing by one character). Analysis shows that motif length and allowed Hamming distance are the primary factors that affect the performance of my algorithm. A small change in the expected length of a tuple (motif) will bring drastic change in the total time consumed. This is due to the overlap of the subsequences in the graph. Hence, changing the length of the motif will affect the length of the $l-1$-tuple. Therefore, smaller l-tuples result in a greater number of nodes in the graph. Second, the Hamming distance also has a great impact on the performance because it is just an additional level of comparison which takes up a lot of processing time. However, the correct selection of Hamming distance based on the overlap of subsequences will ensure the retrieval of weaker motifs which are otherwise neglected.

7.3 Performance analysis of the Hybrid model

The sequential version stands as a benchmark for all further tests on the parallel machine. In comparison to the sequential algorithm, the parallel version can handle more data. In this section we will make a comparison of the sequential, the plain MPI and the hybrid model. We develop both speedup and the efficiency curves for the hybrid model. Proteins such as Cytochrome, Keratin, Basigin, and Neurothelin were taken as test data. However, this time we tried to extract sequences at the genomic level. The whole cytochrome is about 110 Mega Bytes. In the hybrid model we provide a provision to read multiple protein sequence files. Each of these files is converted into a binary file which excludes the sequence headers and contains just the sequences. Hence our program can read as much data as possible and the user just has to specify the file names.

Figure 7.2 is a comparison of the time taken by Hybrid and plain MPI methods on varying number of processors with constant data size (40MB). As it is evident from the graph, the hybrid program runs almost 3 times as fast as the plain MPI. Figure 7.3 is a comparison of the serial, MPI and the Hybrid programs. From the

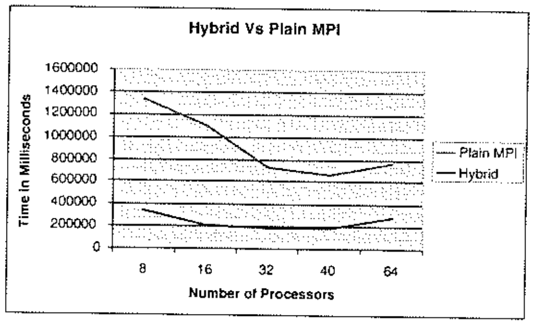


Figure 7.2: Comparison of Hybrid and Plain MPI programs

graph it can be observed that the hybrid program did not perform well when the data size was small. In fact MPI and the serial program showed better performance until the data size was about 15MB. The hybrid program consumes considerable amount of time in the graph construction part where synchronization among the threads is

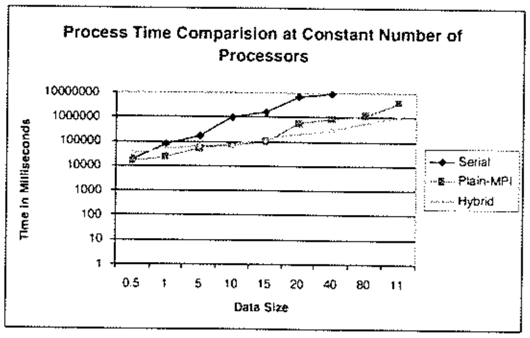


Figure 7.3: Process times of Serial, Hybrid and Plain MPI

required. However this deficiency is compensated as the data size is increases. Though the synchronization time remains constant, more data is shared among the threads, thereby reducing the processing time. The speedup and efficiency curves for both

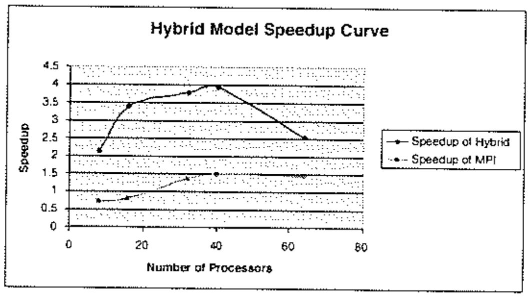


Figure 7.4: Speedup of Hybrid

hybrid and MPI programs are illustrated in Figures 7.4 and 7.5. Its quite clear that the hybrid program has shown better performance than plain MPI. Motif length and Hamming distance still have the same effect on the runtime of the program as described above. In addition, the performance of the parallel program depends on the following factors: size of the Indegree Vector, the number of worker processors, and the size of the input file. The program is scalable for 32 processors. As seen

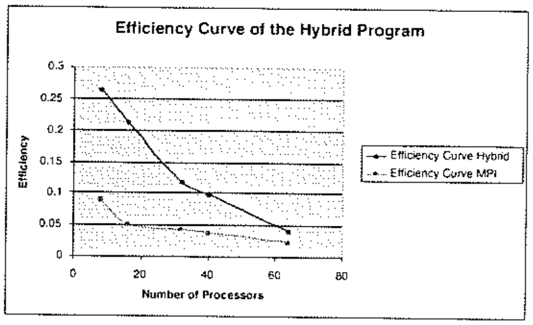


Figure 7.5: Efficiency Curves

in Figure 7.4 the hybrid program shows a steep drop in speedup when 64 processors were used.

Apparently, increasing the number of slave processors brings about more communication overhead. Repeated tests have shown that better speedup can be achieved when the program is run on 16 nodes (32 processors) using optimum chunk size (number of nodes with indegree zero) for each slave processor.

7.4 Evaluation and Summary

In this section we will review our evaluation procedures and the challenges that we have faced during the evaluation phase.

For evaluation purposes, we need to choose parameters which are common for both serial and parallel versions of the algorithm. Therefore we chose parameters such as data size (the number of sequences), the correctness of the results and the total runtime. We evaluated serial algorithm for all the three parameters. In the case of parallel algorithm, the accuracy of the motifs returned could only be compared with our serial results. We were unable to compare accuracy against other parallel versions because, none of the parallel motif discovery tools that we mentioned earlier in the

Chapter 5 were freely available on the web for testing purposes. ParaMEME [12] is the only motif discovery tool which has a web application for testing purposes, but unfortunately due to some technical reasons, their official website is currently not available.

Different motif discovery tools have different purposes. For example, few motif discovery tools are designed to find subtle motifs in DNA sequences, while other tools are designed to find functional motifs to classify the protein families. Therefore comparing the runtime of different tools will have variations depending on the kind of motifs being searched. The results shown in Figure 7.1 are only approximate speed comparisons.

To summarize our results, the factors which have a profound influence on the performance of our parallel algorithm are as follows: the desired length of the motif, the total chunksize chosen by the master processor, the Hamming distance (required to unearth subtle motifs), the number of slave processors and the size of the input file.

The next chapter will conclude this thesis with a few directions on the future work.

Chapter 8

Conclusions and Future Work

In this thesis we show that the de Bruijn graph can accelerate the rate of discovering motifs by more than 10 times. The graph based approach is so powerful that even minute repetitions in sequences are traced. Upon optimization, we were able to scale the algorithm to 20 nodes and handle about 1GB of input data with ease. The algorithm based on de Bruijn graphs had 70% matches of the motifs with the MEME and 65% pattern matches with the Gibbs motif sampler. The sequential algorithm showed that motif length and allowed Hamming distance are the primary factors that affect the performance of the algorithm. The algorithm was redesigned to be implemented on a cluster of parallel machines. Performance analysis were made on a pure distributed memory machine using only message passing and on a hybrid machine using shared and distributed access space. Experiments showed that the hybrid implementation runs 3 times as fast as the pure distributed memory implementation on a data size of 40MB. The architecture on the glacier cluster restricted us to only two threads. In future, we intend to run the algorithm on a cluster where each node has more processors per node. We would like to see if increasing the number of threads will have any positive effect of the performance of the algorithm. Though our algorithm has the speed advantage, its accuracy level remains the same. Parallelization did not have any effect on the quality of motifs returned by the algorithm. Therefore, future work consists of employing some good statistical methods to find weaker planted motifs. Our algorithm is not designed to solve the motif challenge problem, but few improvements during the node comparison stage (which includes generating weaker patterns and comparing with existing nodes), will certainly brighten the chances of solving the planted motif problem.

Bibliography

[1] Tatsuya Akutsu, Hiroki Arimura, and Shinichi Shimozono. On approximation algorithms for local multiple alignment. In *RECOMB 20000: Proceedings of the fourth annual international conference on Computational molecular biology*, pages 1–7, New York, NY, USA, 2000. ACM Press.

[2] Timothy L. Bailey and Charles Elkan. Fitting a mixture model by expectation maximization to discover motifs in biopolymers. In *The Second International Conference on Intelligent Systems for Molecular Biology*, pages 28–36, Stanford, CA, USA, 1994. AAAI Press.

[3] Nicole E. Baldwin, Rebecca L. Collins, Michael A. Langston, Christopher T. Symons, Michael R. Leuze, and Brynn H. Voy. High performance computational tools for motif discovery. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium*, Eldorado Hotel, Santa Fe, NM, USA, April 2004. IEEE.

[4] Blaise Barney. Introduction to parallel computing. `http://www.llnl.gov/computing/tutorials/openMP/`, July 2007.

[5] Brona Brejova, Chrysanne DiMarco, Tomas Vinar, Sandra Romero Hidalgo, Gina Holguin, and Cheryl Patten. Finding patterns in biological sequences.

Technical Report CS-2000-22, University of Waterloo, December 2000.

[6] Jeremy Buhler and Martin Tompa. Finding motifs using random projections. In *RECOMB 2001: Proceedings of the fifth annual international conference on Computational biology*, pages 69–76, New York, NY, USA, 2001. ACM Press.

[7] Dr. Steven M. Carr. DNA: The Molecule Of Life. `http://www.mun.ca/biology/scarr/4241F2_DNA.html`.

[8] Santan Challa and Parimala Thulasiraman. A graph based approach to discover conserved regions in DNA and protein sequences. In *AINA Workshops (1)*, pages 672–677, 2007.

[9] Michael J. Flynn and Kevin W. Rudd. Parallel architectures. *ACM Comput. Surv.*, 28(1):67–70, 1996.

[10] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *Readings in uncertain reasoning*, pages 452–472, 1990.

[11] Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar. *Introduction to Parallel Computing.* Addison Wesley, USA, second edition, 2003.

[12] William N. Grundy, Timothy L. Bailey, and Charles P. Elkan. ParaMEME: a parallel implementation and a web interface for a DNA and protein motif discovery tool. *Bioinformatics*, 12(4):303–310, June 1996.

[13] H. O. Hartley. Maximum likelihood estimation from incomplete data. *Biometrics*, 14(2):174–194, June 1958.

[14] J. Walshaw J. Cooper and Alan Mills. Alpha-helix and beta-sheets. `http://swissmodel.expasy.org/course/text/chapter1.htm`. Secondary structure and backbone conformation.

[15] J. Walshaw J. Cooper and Alan Mills. Tertiary protein structure and folds. `http://swissmodel.expasy.org/course/text/chapter4.htm`. Cytochrome with 4 Helices.

[16] Neil C. Jones and Pavel A. Pevzner. *An Introduction to Bioinformatics Algorithms*. The MIT Press, USA, second edition, August 2004.

[17] Uri Keich and Pavel A. Pevzner. Finding motifs in the twilight zone. *Bioinformatics*, 18(10):1374–1381, April 2002.

[18] Xiaole Liu, Jun S. Liu, and Douglas L. Brutlag. BioProspector: discovering conserved DNA motifs in upstream regulatory regions of co-expressed genes. *Pacific Symposium on Biocomputing*, pages 127–138, 2001.

[19] David W. Mount. *Bioinformatics: Sequence and Genome Analysis*. Cold Spring Harbor Laboratory Press, New York, USA, first edition, 2001.

[20] Rupali Patwardhan, Haixu Tang, Sun Kim, and Mehmet M. Dalkilic. An approximate de Bruijn graph approach to multiple local alignment and motif discovery in protein sequences. *VLDB 2006 Workshop on Data Mining in Bioinformatics: Lecture Notes in Bioinformatics*, 4316:158–169, September 2006.

[21] Pavel A. Pevzner and Sing-Hoi Sze. Combinatorial approaches to finding subtle signals in DNA sequences. In *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, pages 269–278, La Jolla, CA, USA, 2000. AAAI Press.

[22] Pavel A. Pevzner, Haixu Tang, and Michael S. Waterman. An Eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences of the United States*, 98(17):9748–9753, August 2001.

[23] Jun Qin, Simon Pinkenburg, and Wolfgang Rosenstiel. Parallel Motif Search Using ParSeq. In *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN 2005)*, pages 601–607, Innsbruck, Austria, February 2005. ATCA Press.

[24] Isidore Rigoutsos and Aris Floratos. Combinatorial pattern discovery in biological sequences: The TEIRESIAS algorithm. *Bioinformatics*, 14(1):56–57, February 1998.

[25] Frederick P. Roth, Jason D. Hughes, Preston W. Estep, and George M. Church. Finding DNA regulatory motifs within unaligned noncoding sequences clustered by whole-genome mRNA quantitation. *Nature Biotechnology*, 16(10):939–945, 1998.

[26] Cytochrome P450 cysteine heme-iron ligand signature. `http://www.expasy.org/cgi-bin/nicedoc.pl?PDOC00081`.

[27] Westgrid Canada Research Grid. `http://www.westgrid.ca/home.html`.

[28] Lysozyme. `http://users.rcn.com/jkimball.ma.ultranet/BiologyPages/L/Lysozyme.html`, December 2001. Primary Structure Protein.

[29] Structure of DNA. `http://www.merck.com/mmhe/sec01/ch002/ch002a.html`, February 2003.

[30] Molecular biology. http://www.bioalgorithms.info/slides.htm, September 2005.

[31] ParSeq: A Software Tool for Searching Motifs with Structural and Biochemical Properties in Biological Sequences. http://www-pr.informatik.uni-tuebingen.de/parseq/, September 2005.

[32] Cytochrome. `http://en.wikipedia.org/wiki/Cytochrome`, June 2006. Cytochrome is a protein family.

[33] Martin Schmollinger, Igor Fischer, Christiane Nerz, Simon Pinkenburg, Friedrich Gotz, Michael Kaufmann, Klaus-Jorn Lange, Rolf Reuter, Wolfgang Rosenstiel, and Andreas Zell. ParSeq: searching motifs with structural and biochemical properties. *Bioinformatics*, 20(9):1459–1461, February 2004.

[34] Joao Carlos Setubal and Joao Meidanis. *Introduction to Computational Molecular Biology.* PWS Publishing Company, Boston, USA, 1997.

[35] Rahul Siddharthan, Eric D. Siggia, and Erik van Nimwegen. Phylogibbs: A gibbs sampler incorporating phylogenetic information. *PLoS Computational Biology*, 1(7):534–556, December 2005.

[36] Wing-Kin Sung, Tan Yee Fan, Neo Shi Yong, and Wang Gang. Combinatorial methods in bioinformatics, Lecture 10: Motif Finding. `http://www.comp.nus.edu.sg/~ksung/cs5238/2004Sem1/note/note_taking_list.htm`, 2004.

[37] Toshihide Sutou, Keiichi Tamura, Yasuma Mori, and Hajime Kitakami. Design and implementation of parallel modified prefixspan method. In *International Symposium on High Performance Computing*, volume 2858, pages 412–422, 2003.

[38] Julie D. Thompson, Desmond G. Higgins, and Toby J. Gibson. CLUSTALW: improving the sensitivity of progressive multiple sequence alignment through

sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22(22):4673–4680, September 1994.

[39] William Thompson, Eric C. Rouchka, and Charles E. Lawrence. Gibbs recursive sampler: finding transcription factor binding sites. *Nucleic Acids Research*, 31(13):3580–3585, 2003.

[40] Tetsushi Yada, Yasushi Totoki, Masato Ishikawa, Kiyoshi Asai, and Kenta Nakai. Automatic extraction of motifs represented in the hidden markov model from a number of DNA sequences. *Bioinformatics*, 14(4):317–325, 1998.

[41] Yu Zhang and Michael S. Waterman. An Eulerian path approach to local multiple alignment for DNA sequences. *Proceedings of the National Academy of Sciences of the United States of America*, 102(5):1285–1290, February 2005.