I, Sushma Koneru, hereby submit this original work as part of the requirements for the degree of:

Master of Science

in Electrical Engineering

It is entitled:

Hardware Implementation of the Expectation Maximization Algorithm for Motif Finding

Student Signature: Sushma Koneru

This work and its defense approved by:

Committee Chair: Carla Purdy, C, PhD

Carla Purdy, C, PhD

Harold Carter, PhD

Harold Carter, PhD

Wen Ben Jone, PhD

Wen Ben Jone, PhD
Hardware Implementation of the Expectation Maximization

Algorithm for Motif Finding

A Thesis submitted to the

Graduate School at the

University of Cincinnati

In partial fulfillment of the

requirements for the degree of

MASTER OF SCIENCE

In the Department of Electrical and Computer Engineering

of the College of Engineering

2009

By

Sushma Koneru

B.E.(Hons) in Electronics and Instrumentation Engineering, 2006

Birla Institute of Technology and Science, Pilani, India

Committee Chair: Dr. Carla Purdy
Abstract

Transcription factors are proteins which bind to deoxyribonucleic acid (DNA) and regulate gene expression, i.e. synthesis of a gene product. Understanding the mechanisms that regulate the expression of genes has been a major challenge in molecular biology. The identification of binding sites to DNA for transcription factors is a very important task in this challenge. These binding sites are known as motifs, and they are short DNA segments. The use of computational methods for prediction of these motifs is very effective. Several computational methods have been developed over the past few years for motif finding in DNA sequences. These techniques have also been extended to find motifs in proteins. For proteins, a motif is an amino-acid recurring pattern. But one of the major drawbacks of currently available methods is that the execution time is very high for large DNA or protein data sets. A hardware implementation of these computational methods would be a good alternative to reduce execution time.

In this thesis, we have implemented the Expectation Maximization (EM) algorithm in hardware. EM is one of the widely used algorithms for motif finding. The entire hardware design has been realized using Verilog HDL modules. These modules can also be synthesized to generate gate-level netlists and be ported onto a field programmable gate array (FPGA). The functionality and performance of the design is tested on multiple data sets of varying sequence lengths. The performance of the design has also been compared to one of the popular software approaches, MEME. The tests show that the hardware approach can achieve speedups by a factor of 100 or greater.
Acknowledgements

I would like to thank my advisor Dr. Carla Purdy for her constant support and guidance throughout my thesis work. Her valuable suggestions have been very useful and this work wouldn't have been possible without her constant support and motivation. I would also like to thank Dr. Harold Carter and Dr. Wen-Ben Jone for spending their invaluable time and serving on my thesis committee.

I am thankful to my family for their encouragement and moral support. They are completely responsible for whatever I am today. They have been my support system all throughout my life.

I would also like to thank all my dear friends at University of Cincinnati for making my stay unforgettable and making this journey truly memorable. Lastly, I would like to thank almighty for all his blessings.
TABLE OF CONTENTS

LIST OF FIGURES ........................................................................................................................................ VII

LIST OF TABLES ........................................................................................................................................ IX

1 INTRODUCTION ................................................................................................................................. 1

1.1 Motivation ........................................................................................................................................... 1

1.2 Aim of the thesis ............................................................................................................................... 1

1.3 Introduction to Verilog HDL ........................................................................................................... 2

1.4 Introduction to EM algorithm .......................................................................................................... 3

1.5 Organization of the thesis ................................................................................................................. 4

2 BACKGROUND AND PREVIOUS WORK ON MOTIF FINDING ......................................................... 5

2.1 DNA ................................................................................................................................................... 5

2.2 RNA ................................................................................................................................................ 7

2.3 Transcription, transcription factors, motifs ..................................................................................... 9

2.4 Overview of motif finding algorithms ............................................................................................. 12

2.4.1 Previous work in motif finding algorithms ................................................................................. 13

2.4.1.1 Word-based algorithms ......................................................................................................... 14

2.4.1.2 Probabilistic algorithms ........................................................................................................ 15

2.4.1.3 Hardware approaches ........................................................................................................... 18

2.4.1.4 Other approaches ................................................................................................................ 20

2.4.2 Previous work on comparison of motif finding algorithms ..................................................... 23

2.4.2.1 Statistics used for comparison .............................................................................................. 23

2.4.2.2 Datasets ................................................................................................................................ 27

2.4.2.3 Results of comparison studies ............................................................................................. 29
3 EXPECTATION MAXIMIZATION ALGORITHM (EM) ......................... 32

3.1 Algorithm ............................................................................................................................ 32
3.2 Application of EM to the motif finding problem ................................................................. 34
  3.2.1 Expectation step............................................................................................................ 37
  3.2.2 Maximization step ........................................................................................................ 38
3.3 An example showing application of EM algorithm to motif finding......................... 39

4 HARDWARE IMPLEMENTATION OF EXPECTATION MAXIMIZATION ALGORITHM ................................................................. 45

4.1 Architecture ......................................................................................................................... 45
4.2 Details of design.................................................................................................................. 48
4.3 Modules ............................................................................................................................... 50
  4.3.1 Random number implementation ................................................................................. 50
  4.3.2 Formation of motif and non-motif matrix .................................................................... 52
  4.3.3 Formation of probability matrix ................................................................................ 55
  4.3.4 Expectation ................................................................................................................... 58
  4.3.5 Maximization ................................................................................................................ 59
  4.3.6 Comparison .................................................................................................................. 61

5 PERFORMANCE ANALYSIS ................................................................. 62

5.1 Experimental setup.............................................................................................................. 62
5.2 Functional verification ........................................................................................................ 62
5.3 Data sets used for performance analysis ............................................................................. 63
5.4 Performance analysis........................................................................................................... 67
  5.4.1 Prediction accuracy and reliability ............................................................................... 69
  5.4.2 Speedup ........................................................................................................................ 79
6 CONCLUSIONS AND FUTURE WORK ............................................... 82

6.1 Conclusions ......................................................................................................................... 82
6.2 Future work ......................................................................................................................... 82

BIBLIOGRAPHY........................................................................................................... 84

APPENDIX A: STATISTICAL PARAMETERS FOR THE SOFTWARE TOOL

MEME .............................................................................................................................. 90
LIST OF FIGURES

Figure 2.1 Structure of part of DNA [7] 5
Figure 2.2 Flattened version of DNA [9] 6
Figure 2.3 Comparison of DNA and RNA structures [11] 8
Figure 2.4 Gene transcription [13] 9
Figure 2.5 Transcription process [14] 10
Figure 2.6 Transcription factors [14] 11
Figure 2.7 Venn diagram showing nTP, nTN, nFP and nFN. 24
Figure 2.8 Statistics showing comparison of 13 tools [18] 30
Figure 3.1 Algorithm starts with a single site and later alternates between assigning 35
sites to a motif (left side) and then updates the motif model (right side) [1]
Figure 3.2 Flowchart of EM algorithm 36
Figure 4.1 Block diagram of the architecture developed 46
Figure 4.2 Sequences RAM 48
Figure 4.3 5-bit LFSR 51
Figure 4.4 State diagram for formation of motif and non-motif matrix 55
Figure 4.5 State machine for formation of probability matrix 57
Figure 4.6 State machine for expectation step 59
Figure 4.7 State machine for maximization step 61
Figure 5.1 Distribution of data sets 64
Figure 5.2 Distribution of sequence length in data sets 65
Figure 5.3 Distribution of data sets based on number of sequences in a data set and 67
sequence lengths

Figure 5.4 Graph showing minimum, average and maximum values of statistical parameters

Figure 5.5 Graph showing the speedup over MEME for 6 data sets.
LIST OF TABLES

Table 2.1 Comparison of execution speeds of MEME, ParaMEME and MAMA 19
Table 2.2 List of algorithms and tools, the operating technique used, the web address, reference and year 22
Table 2.3 Selected web-based resources for regulatory motifs or TFBSs [29] 28
Table 3.1 Dataset of sequences 39
Table 3.2 Motif matrix 40
Table 3.3 Non-motif matrix 40
Table 3.4 Frequency of bases 41
Table 3.5 Probability of bases in motif 41
Table 3.6 Probability matrix 42
Table 3.7 Probability of motif occurring at a particular position 43
Table 4.1 Dataset of sequences 53
Table 4.2 Motif and non-motif matrix 53
Table 4.3 Motif and non-motif matrices 55
Table 4.4 Probability matrix 56
Table 5.1 Table showing data sets with their type, number of sequences, sequence lengths 66
Table 5.2 Results of nucleotide level sensitivity on the data sets 70
Table 5.3 Results of nucleotide level positive predictive value on the data sets 71
Table 5.4 Results of nucleotide level performance coefficient on the data sets 72
Table 5.5 Results of nucleotide level correlation coefficient value on the data sets 73
Table 5.6 Results of binding site level sensitivity on the data sets 74
Table 5.7 Results of binding site level positive predictive value on the data sets 75
Table 5.8 Results of binding site level average site performance on the data sets 76
Table 5.9 Difference in the statistical parameters for hardware design and MEME 78
Table 5.10 Number of devices available in Stratix II, Starix II GX and Stratix III 79
Table 5.11 Area report on Stratix II and Stratix III devices 81
Table 5.12 Number of devices available in Virtex 5, Virtex 6, Sparta –3A, Spartan6 81
1 INTRODUCTION

1.1 Motivation

The gene is the basic unit in deoxyribonucleic acid (DNA) which contains the required information to determine heredity. The expression of genes is a process in which a functional gene product is synthesized from DNA using gene information. In gene expression, transcription factors play a very important role. Transcription factors are proteins which bind to DNA and control the synthesis of a gene product. Identification of transcription factors is very crucial for gene expression and it is a challenging task, due to the complexity of biological sequences. DNA sequences contain a few recurring patterns which can be potential binding sites for transcription factor. These recurring patterns are known as motifs [1]. For proteins, a motif is an amino-acid recurring pattern. Discovery of these motifs has gained a lot of attention since it is very useful for gene expression [2]. This task of discovering motifs could be considered as equivalent to pattern matching in a set of sequences. But this would be a very complex task in the case of DNA sequences as they are not simple, and can have insertions, deletions or mutations of nucleotides.

Several software approaches have been developed for motif matching over the past few years. The computation time for these software approaches has been very high because of the complexity of biological sequences and the probabilistic approaches used in the software. A hardware alternative would help in decreasing the computation time by employing the techniques of concurrent simulation, pipelining and parallelization.

1.2 Aim of the thesis

Motif matching is very important since it helps in identifying binding sites which are critical in understanding the biological content of genome sequences. Several software
approaches have been proposed for motif matching and many of them are based on the Expectation Maximization (EM) algorithm, as it has been proved to be one of the efficient algorithms for motif matching. Though the algorithm is very efficient, it has high execution times for large DNA or protein data sets. A few approaches have been developed based on the principle of parallelization using multiple processors. But this requires many processors and hence increases the cost incurred. A direct hardware implementation of the algorithm would be a very viable alternative.

This thesis deals with the implementation of the EM algorithm on hardware, where the whole algorithm is divided into different steps and implemented using submodules. The modules are described in Verilog HDL [3] and to ensure that the modules are synthesizable, they are synthesized using Altera Quartus II.

1.3 Introduction to Verilog HDL

Verilog HDL is a Hardware Description Language (HDL). Gateway Design System Corporation, which is now a part of Cadence Design Systems, introduced Verilog 1985 [3]. The Verilog language enables the designer to describe digital system architecture at various levels of abstraction such as behavioral, gate-level and RTL. Most of the computer-aided tools for circuit design have the capability of processing Verilog codes and hence this language provides a gateway to use computer-aided tools in the design flow.

A design in Verilog contains many inter-connected modules which form a hierarchy. Modules are the basic elements of Verilog design. They are used to define a particular functionality which is later instantiated at different places. Modules have input, bidirectional and
output ports which are used for communication with other modules [4]. Each module can be either an element or a collection of a few submodules.

Verilog has various advantages over other languages or design practices. Some of them are [5]:

- It offers different levels of abstraction in the same model. So we can use gates, switches, RTL or behavioral code to describe a hardware module.
- All synthesis tools support Verilog HDL. Hence the modules can later be ported onto FPGA’s or ASIC’s.
- Verilog HDL libraries for post-synthesis simulation are provided by all the fabrication vendors. Hence we can choose from a wide range of vendors.
- Designing with Verilog makes it easier to develop and debug very complex circuits where gate-level schematics become incomprehensible.

1.4 Introduction to EM algorithm

EM algorithm is one of the widely used approaches for motif finding. Most of the currently available software tools use the EM algorithm, as it has been proved to be one of the most efficient methods. So we have decided to implement the EM algorithm on hardware. EM uses a probabilistic approach to find motifs in sequences. This algorithm was introduced by Lawrence and Reilly [6] for solving the motif finding problem. The input to the algorithm is a set of unaligned sequences and motif length (W), and it returns a probabilistic model of the shared motif. The shared motif is the common subsequence present in all the input sequences. The starting point of the shared motif is unknown and differs among sequences. Finding the starting point of the motif turns out to be the major challenge.
Each motif can be considered as a sequence of independent discrete random variables. Hence, the observed frequencies of the letters in the columns are the maximum likelihood estimates of the distribution of the random variables. But since the sequences are unaligned, the offsets should also be estimated. Given the input data and initial guess of motif, EM initially estimates the probability \( (z_{ij}) \) that the shared motif starts at position ‘j’ in dataset sequence ‘i’. Later \( z_{ij} \) is used to re-estimate the probability of letter ‘l’ in column ‘c’ of the motif, \( \rho_{lc} \), for each letter in the alphabet and \( 1 \leq c \leq W \), where \( W \) is the width of the motif. The algorithm alternatively estimates \( z \) and \( \rho \) until the change in \( \rho \) from iteration to iteration is less than \( \epsilon \) (tolerance value). A detailed description of the algorithm is presented in Chapter 3.

1.5 Organization of the thesis

The thesis is organized as follows. Chapter 2 presents the basics of molecular biology, an overview of the motif matching algorithms available and a comparison of them. It also discusses the statistical parameters and datasets used for comparative analysis. Chapter 3 describes the Expectation Maximization (EM) algorithm and the various steps involved in it. Chapter 4 deals with the hardware implementation of the Expectation Maximization algorithm. The block-level architecture of the design is also presented in this chapter. Chapter 5 analyses the simulation results and compares them with their software counterparts. Chapter 6 concludes by presenting a summary of the work and discusses possible avenues that could be explored in the future.
2 BACKGROUND AND PREVIOUS WORK ON MOTIF FINDING

2.1 DNA

Deoxyribonucleic acid (DNA) contains the genetic information which makes each organism unique. Each cell in an organism has the same DNA [7]. A strand of DNA is formed with nucleotides linked into chains and phosphate and sugar groups alternating [8]. DNA forms a double helix as shown in Figure 2.1.

![Figure 2.1 Structure of part of DNA [7]](image)

Each nucleotide has a sugar group, a phosphate group and one of the base pairs [7]. Base pairs are formed using the chemical bases adenine (A), guanine (G), cytosine (C), and thymine (T) [7]. ‘A’ always pairs up with ‘T’, and ‘C’ with ‘G’ to form base pairs [7]. Figure 2.2 shows an unwound version of DNA. In the figure, ‘S’ represents the five carbon sugar called
deoxyribose. ‘P’ refers to the phosphate group. The structure of DNA can be considered as a ladder where the sides of the ladder are formed by alternating sugar and phosphate units, and steps are formed by nucleotide base pairs. These base pairs are held together by hydrogen bonds. As we can see in Figure 2.2, the two ends of a DNA strand are in different directions, that is, they run anti-parallel [8]. The left side of the strand begins with a phosphate group at the top and ends with a sugar unit. But the right side of the strand begins with a sugar molecule at the top and ends with a phosphate group at the bottom. The sugar end of a DNA strand is called the 3' end, and the phosphate end is called the 5' end.

Figure 2.2 Flattened version of DNA molecule [9]
The two strands of DNA are complementary to each other. For example, if one strand is AGTACGC, the other strand would be TCATGCG. This property of complementary strands is very important for storage and propagation of gene information. The order of bases determines the biological instructions contained in a strand of DNA [8]. The DNA sequence of human contains approximately three million bases and at least 99 percent of the human DNA sequences is common in all human beings [7].

2.2 RNA

Ribonucleic acid (RNA) is a polymer which consists of a long chain of nucleotides. Nucleotides consist of a sugar group, a phosphate group and a base. RNA is similar to DNA but has three main differences [10].

1. RNA contains a ribose sugar group while DNA contains a deoxyribose sugar group. The difference between the sugar groups is that ribose has a hydroxyl group (OH) on the 2' carbon atom but deoxyribose doesn’t have it.

2. RNA is mostly single stranded while DNA is double-stranded.

3. The bases in RNA are Adenine (A), Guanine (G), Cytosine (C), and Uracil (U). RNA does not have Thymine (T) which is present in DNA. In RNA, the complementary base to Adenine is Uracil and not Thymine.

Figure 2.3 clearly shows these differences.
There are three types of RNA, namely mRNA, tRNA and rRNA [12]. mRNA is known as messenger RNA and it contains a copy of the gene information. It contains information which is complementary to one strand of DNA and identical to the other strand, except for substitutions of ‘U’ for ‘T’. tRNA is known as transfer RNA and it can bind to an amino acid at one end and mRNA at the other end. rRNA is called ribosomal RNA and it is a key component in a cell’s ribosome. A detailed description of tRNA and rRNA has not been provided as a part of this thesis as it is out of the scope of the current work.
2.3 Transcription, transcription factors, motifs

Transcription

Transcription is a process by which RNA is transcribed from DNA with the help of an enzyme called RNA polymerase. For the transcription process to start, RNA polymerase has to bind to a promoter region [13]. A promoter region contains base sequences which indicate the beginning of the gene region. When RNA polymerase recognizes a promoter region and tries to bind to it, the DNA is two stranded as shown in Figure 2.4. This is called a closed complex. In the vicinity of the initiation site, the hydrogen bonds between bases in DNA are broken, and the DNA is unwound.

![Figure 2.4 Gene transcription](image)

In transcription, nucleotides are added in the 5' to 3' direction. Only one of the DNA strands is transcribed to form mRNA. Each nucleotide base pairs up with the complementary base to form mRNA. For the formation of mRNA, Adenine pairs with Uracil and not with Thymine. Thymine pairs up with Adenine, Guanine pairs with Cytosine and Cytosine pairs with Guanine. Figure 2.5
shows an example of the transcription process. In this figure, the DNA sequence GATCAT is transcribed into CUAGUA.

Figure 2.5 Transcription process [14]

**Transcription factors**

To help RNA polymerase bind to DNA, we need protein complexes known as transcription factors [15]. Transcription factors can be described as proteins which bind to DNA sequences and control the transcription from DNA to RNA. These transcription factors contain DNA binding domains (DBD), which are attached to either enhancer or promoter regions of DNA sequences. Enhancer and promoter regions are short portions of DNA which can bind to proteins called activators, which in turn help to initiate transcription. These regions are adjacent to the genes that they regulate [15]. Activators help in initiating the transcription process and transcription factors control the transcription. Figure 2.6 shows the transcription factor binding to a DNA segment at a DNA binding site. TF in the figure refers to transcription factor. The transcription factor controls the transcription process by helping RNA polymerase to bind to DNA segments.
The number of transcription factors within an organism is directly proportional to genome size. Genome refers to the hereditary information encoded in the DNA. Organisms with large genome size have more transcription factors [15]. In the human genome, there are approximately 2600 proteins that contain DNA binding domains and most of them function as transcription factors [15]. This is approximately 10% of the entire human genetic code; hence transcription factors are the single largest family of human proteins [15].

**Motifs**

We have many databases available online which contain data sets of DNA sequences. Each data set contains a group of DNA sequences from a particular organism such as human, mouse or yeast. The DNA sequences in a particular data set have a few recurring patterns which can be potential DNA binding sites for a transcription factor. These recurring patterns are known as motifs [16]. Motifs are short, typically 5 to 20 base pairs long [16]. A base pair consists of two
nucleotides connected via hydrogen bonds. The same DNA sequence can have zero, one or multiple copies of the same motif.

**Types of motifs**

A motif is considered to be conserved if the motif is expressed with the same abundance in all the sequences being studied. A motif is considered to be over-represented if the frequency of occurrence of the motif is significantly higher than its frequency in the background model. The background model refers to the all subsequences which are present in the DNA dataset except the motifs. More details on the background model are provided in Chapter 3 and 4. Constrained motifs do not allow mutations, insertions or deletions. There are two other types of motifs, namely palindromic motifs and spaced-dyad motifs [16]. If a motif is the same as its own reverse complement, it is known as a palindromic motif. An example of a palindromic motif is CACGTG. If a motif consists of two conserved sites which are separated by a gap, it is known as a spaced-dyad motif [16]. The gap is normally fixed but can vary in a few cases.

**2.4 Overview of motif finding algorithms**

The identification of motifs is cumbersome due to the complexity of DNA sequences and insufficient knowledge of gene regulation. A number of algorithms using different techniques have been developed for motif finding over the years. The problem of motif finding can be defined as detecting motifs from a set of DNA sequences which can be good candidates for binding sites [16]. This can be compared to pattern matching where we try to identify an unknown pattern which occurs frequently in a set of strings. This section presents a brief description of motif finding algorithms developed so far. Motif finding algorithms are compared
in the later sections and a description of statistical parameters and datasets used for comparison is also presented.

2.4.1 Previous work in motif finding algorithms

A number of algorithms have been developed for motif finding. Most of the algorithms can be categorized into two groups, namely word-based algorithms and probabilistic algorithms [16]. Word-based algorithms use exhaustive enumeration. Nucleotide frequencies are counted and compared, and hence global optimality is ensured. This approach is suitable for short motifs which are normally present in eukaryotes [16]. Eukaryotes are the organisms whose cells have a nucleus. Their cells contain complex structures. All large organisms fall in the category of eukaryotes. By using good data structures like suffix trees [16], the word-based approach can be made computationally fast and is very useful for finding constrained motifs, where there are no mutations, insertions or deletions. But in real data sets, motifs are not constrained and the word-based approach has the drawback of finding spurious motifs [16]. In probabilistic algorithms, parameters are estimated using maximum likelihood or Bayesian inference [16]. Motif models use techniques like Expectation Maximization, Gibbs sampling and other greedy approaches for finding solutions. These techniques are explained in detail in the later sections. The probabilistic approach requires few input parameters but it relies on the probabilistic models of gene regulator regions [16]. Most of these algorithms are useful for finding long motifs and hence this approach is mostly used for motif finding in prokaryotes. Prokaryotes are the organisms whose cells lack a cell nucleus. They are mostly unicellular.

In this section, we present previous work on motif finding algorithms by different researchers over the last few years. We have categorized them into word-based algorithms, probabilistic algorithms, hardware approaches and other approaches.
2.4.1.1 Word-based algorithms

The motif finding algorithm Oligo-analysis was developed by van Helden *et al.* [17] using the word-based approach. This algorithm compares the occurrences of words with expected values and detects statistically significant motifs [18]. Though the algorithm is simple, it worked efficiently in extracting motifs for yeast [16]. But this algorithm is rigorous and exhaustive and can be used only for short conserved motifs. van Helden *et al.* [19] later extended the algorithm to find spaced dyad motifs. The important parameter in this algorithm is the choice of probabilistic model for estimating occurrence significance [18]. One of the drawbacks of this method is that it does not allow variations within a nucleotide [16].

Tompa [20] developed an algorithm around the same time for finding short motifs in DNA sequences using the exact word-based approach [16]. This algorithm was primarily applied to ribosome binding sites. Yeast motif finder (YMF) was developed by Tompa and Sinha [21] using a similar approach. The model was derived from the study of known transcription factors in the yeast *Saccharomyces cerevisiae* [21]. The algorithm takes as input the set of sequences, number of non-spaced motifs to be recapitulated, and a transition matrix constructed from the yeast model, and produces motifs with z-scores [16]. This algorithm ensures global optimality. It was tested on 23 well known regulons of yeast. Regulons are defined as a group of genes under regulation by the same regulatory protein. YMF was able to generate the principal transcription factor for 18 of the regulons [16].

Another algorithm, introduced by Sagot [22], uses a suffix tree for representation of sequences. Vanet *et al.* [23] developed another algorithm to search for single motifs in bacteria [16]. This method also uses suffix trees for representation. The use of suffix trees for representation helped in reducing the computation time of word-based approach algorithms
though it was still exhaustive. Later, Marton and Sagot [24] extended the algorithm developed by Vanet et al. for searching combination of motifs [16].

Weeder [25] developed by Pavesi et al. also uses a suffix tree but the input pattern is not constant. It automatically searches the whole space with different motif lengths and compares the top-scoring motifs of each run using a clustering method to identify the best candidates which are likely to be transcription factor binding sites [18]. It is a consensus method and all oligos are enumerated up to maximum length and later their occurrences in input sequences are computed. (Oligos are defined as short nucleic acid polymers which typically contain twenty or fewer bases.)

Eskin and Pevzner [26] developed the Mismatch Tree Algorithm (MITRA) which uses a suffix tree and is capable of handling insertions and deletions along with mismatches in selected sequences. This algorithm was able to discover longer motifs, i.e., those with combined length greater than 30 bp [26].

Pevzner and Sze [27] developed an algorithm named WINNOWER which uses graph theoretical methods along with the word-based approach. It represents motif instances as vertices and later deletes spurious edges and finally recovers the remaining vertices. CWINNOWER [28] is an extension of this algorithm with a stronger constraint function which helps in decreasing the running time [29].

2.4.1.2 Probabilistic algorithms

Probabilistic algorithms use techniques such as Expectation maximization (EM) and Gibbs sampling and its extensions for motif finding.
Lawrence and Reily [6] developed EM for motif finding in 1990. It is an extension of greedy algorithms proposed by Hertz et al. [30] which finds a common motif occurring once in every sequence. The main advantage of EM is the computational speed, but it needs proper starting points to yield good results. A detailed description of the algorithm is presented in Chapter 3. Multiple EM for Motif Elicitation (MEME) was developed by Bailey and Elkan [31], and is an extension of the EM algorithm. MEME can be used for motifs in a set of sequences and it operates on starting points which actually occur in the sequences. This helps increase the probability of finding globally optimal motifs. MEME is a very computationally demanding algorithm but has been proven to give good results [31]. Gaudet et al. [32] proposed an algorithm named Improbizer which uses EM to find motifs which occur with low frequency. It compares the normal occurrence of a nucleotide by chance to the actual frequency of occurrence of the nucleotide by constructing a Gaussian model of motif placement [29].

Lawrence et al. [33] developed the Gibbs sampler technique for motif finding. Gibbs sampler is based on a Markov Chain Monte Carlo (MCMC) approach [16]. As the results of the current step depend only on that of the previous step, it is a Markov chain. The method to select the next step is based on random sampling and hence it is also called Monte Carlo. Gibbs Sampler works on the assumption that there is at least one motif in every sequence.

Many algorithms have been developed as extensions to the Gibbs sampling approach. One of the algorithms is AlignACE developed by Roth et al. [34], which returns overrepresented motifs in the input sequences as weight matrices [18]. To measure the degree of overrepresentation, AlignACE uses a maximum a priori likelihood (MAP) score [16]. Later, Hughes et al. [35] used this algorithm to find motifs in yeast related organisms, but applied group specificity to score motifs instead of the MAP score. Group specificity takes into
consideration motifs that are preferably associated with the genes, unlike the MAP score where a few ubiquitously occurring motifs are given a high score [16].

BioProspector is another variant of the Gibbs sampling algorithm for motif finding. Liu et al. [36] tested this algorithm on RAP1 protein in yeast, CRP protein in Escherichia coli and TATA-box motif in Bacillus subtilis and were successful in finding motifs [16]. A Markov model is added to estimate promoter sequences in genes, which help in modeling adjacent nucleotide dependency [18]. This algorithm can also be used for modeling gapped motifs with palindrome patterns that are common in prokaryotes [29].

Thijs et al. [37] developed MotifSampler which is a modification of the Gibbs sampling algorithm. It uses a higher order Markov model for the background model [38]. This algorithm was tested on several datasets including sequences from plants containing the G-box motif, bacterial genes which are regulated by the FNR protein [16].

GLAM was developed by Frith et al. [39] based on the Gibbs sampling algorithm. This algorithm automatically optimizes the motif width and computes the statistical significance of the generated output [18]. This algorithm cannot find multiple motifs in a set of sequences. Favorov et al. [40] developed SesiMCMC based on the Gibbs sampling approach for finding multiple motifs in an input set of DNA sequences [29].

Gibbs-ssT was developed by Shinda [41]. It combines simulated annealing with Gibbs sampling to overcome the problem of local optima. Simulated tempering helps in improving search methods in the solution space [16]. Shinda [41] tested Gibbs-ssT on synthetic data and sequences extracted from yeast and observed good resistance to local optima [41].
Hertz and Stormo [42] developed an algorithm using the greedy approach for motif finding. It is an extension of earlier work by Hertz et al. [30]. This algorithm searches a pair of sequences for the motif with greatest information content and then tries to find a third sequence to be added, one with the greatest information content [18]. ANN-Spec developed by Workman and Stormo [43], finds patterns to distinguish a motif from background. This algorithm uses a weight matrix to represent the motif model. It finds motifs with higher correlation coefficients and higher specificity compared to the Gibbs sampler [33].

QuickScore, proposed by Regneir and Denise [44], uses an exhaustive search to predict probabilities of frequent or rare patterns in a set of sequences. This algorithm uses mathematical computations for effectively calculating z-scores and probability values [18]. Liu et al. [45] developed MDScan, which is an enumerative deterministic greedy algorithm [38]. It uses a MAP score to evaluate best candidates for motifs. The algorithm constructs motif models using several top motif candidates and then uses a greedy approach to improvise [38].

2.4.1.3 Hardware approaches

Sandve et al. [2] implemented a variant of MEME on specialized hardware to accelerate motif matching. MEME works on the basic principle of the EM algorithm. It has been observed that MEME spends the maximum amount of time on calculation of initial starting positions. It tries to match the guessed motif against sequences in this step, which involves a lot of motif matching computations. This is the major bottleneck in MEME. Sandve et al.[2] have proposed a method called ‘Massively parallel Acceleration of the MEME Algorithm’ (MAMA), which would accelerate the first step in MEME.
The acceleration is performed by using existing pattern matching hardware known as PMC (Pattern Matching Chip). PMC was developed by Interagon for finding patterns [46]. This chip performs matching with massive parallelization [2]. As the chip was not exclusively developed for motif matching, the authors had to incorporate a few modifications to use PMC in their design. MEME uses dynamic programming concepts in the first iteration to accelerate it. This step in MEME was replaced with modified PMC hardware in MAMA, and the rest of the algorithm was left unchanged.

Sandve et al.[2] have compared the execution speed of MAMA with MEME and paraMEME. ParaMEME uses the concept of parallelization of MEME. It uses a cluster of computers for computation and executes different steps in parallel. MAMA was tested on 5 data sets of human promoter regions. The data sets were in the range of 100 to 1600 sequences each, of 5000 bp length. It was also tested on the largest data set provided with MEME (mini-drosoph). MEME and MAMA were run on 2.8 GHz Pentium4 processor with 1 GB of memory [2]. MAMA has a specialized hardware attached to processor. ParaMEME was run on cluster of twelve 3.8 GHz Pentium4 PC’s, each with 1 GB of memory. The results obtained are shown in the table 2.1.

<table>
<thead>
<tr>
<th>Data set</th>
<th>Size (Mbp)</th>
<th>MEME (hrs)</th>
<th>ParaMEME (hrs)</th>
<th>MAMA(hrs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mini-drosoph</td>
<td>0.5</td>
<td>2.6</td>
<td>0.19</td>
<td>0.27</td>
</tr>
<tr>
<td>Hs_100</td>
<td>0.5</td>
<td>2.7</td>
<td>0.20</td>
<td>0.23</td>
</tr>
<tr>
<td>Hs_200</td>
<td>1</td>
<td>11</td>
<td>0.87</td>
<td>0.50</td>
</tr>
<tr>
<td>Hs_400</td>
<td>2</td>
<td>104</td>
<td>3.6</td>
<td>1.7</td>
</tr>
<tr>
<td>Hs_800</td>
<td>4</td>
<td>Not tested</td>
<td>15</td>
<td>6.4</td>
</tr>
<tr>
<td>Hs_1600</td>
<td>8</td>
<td>Not tested</td>
<td>64</td>
<td>13</td>
</tr>
</tbody>
</table>

Table 2.1 Comparison of execution speeds of MEME, ParaMEME and MAMA [2]
It can be observed that the performance of MAMA is better than MEME, and is comparable to paraMEME. As the data set size increases, it can be seen that performance of MAMA improves and achieves more speedup compared to paraMEME. On average, MAMA is 10 times faster than MEME.

### 2.4.1.4 Other approaches

There are various other approaches for motif finding. The FMGA algorithm was developed by Lit et al. [47] based on genetic algorithms. Position weight matrices are used for mutation in genetic algorithms [16]. A self organizing neural network structure is also used for motif finding by Liu et al.[48]. Kaplan et al. [49] developed an algorithm which uses a structure based approach for motif finding without any prior knowledge of transcription factor binding sites [16]. A deterministic motif finding algorithm was developed by Jain and Hon [50] for the human genome. An ensemble algorithm was developed by Hu et al. [38]. This combines multiple predictions from various runs using two different algorithms. The main aim of this algorithm is to improve prediction accuracy [16]. This technique did improve performance significantly.

A number of algorithms have been developed using the technique of phylogenetic footprinting. We have skipped discussion of these algorithms since they are out of the scope of this work. In the next section, we discuss the statistics and datasets to be considered for comparison of motif finding algorithms. The results of previous comparison studies are also discussed in Section 2.4.

Table 2.2 presents a list of the motif matching algorithms and tools which have been discussed so far.
<table>
<thead>
<tr>
<th>S. No.</th>
<th>Tool/Algorithm</th>
<th>Operating technique</th>
<th>URL (Accessed on 10/13/09)</th>
<th>Reference</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>EM</td>
<td>Probabilistic</td>
<td>-</td>
<td>6</td>
<td>1990</td>
</tr>
<tr>
<td>2</td>
<td>Gibbs Sampler</td>
<td>Probabilistic</td>
<td>-</td>
<td>33</td>
<td>1993</td>
</tr>
<tr>
<td>3</td>
<td>MEME</td>
<td>Probabilistic</td>
<td><a href="http://meme.nbcr.net/meme4_3_0/intro.html">http://meme.nbcr.net/meme4_3_0/intro.html</a></td>
<td>31</td>
<td>1995</td>
</tr>
<tr>
<td>4</td>
<td>AlignACE</td>
<td>Probabilistic</td>
<td><a href="http://atlas.med.harvard.edu/cgi-bin/alignace.pl">http://atlas.med.harvard.edu/cgi-bin/alignace.pl</a></td>
<td>34</td>
<td>1998</td>
</tr>
<tr>
<td>6</td>
<td>Consensus</td>
<td>Probabilistic</td>
<td><a href="http://bifrost.wustl.edu/consensus/">http://bifrost.wustl.edu/consensus/</a></td>
<td>30</td>
<td>1999</td>
</tr>
<tr>
<td>8</td>
<td>SMILE</td>
<td>Word</td>
<td>-</td>
<td>24</td>
<td>2000</td>
</tr>
<tr>
<td>9</td>
<td>WINNOWER</td>
<td>Word</td>
<td>-</td>
<td>27</td>
<td>2000</td>
</tr>
<tr>
<td></td>
<td>Algorithm</td>
<td>Technique</td>
<td>Web Address</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>-----------------</td>
<td>--------------------</td>
<td>------------------------------------------------------------</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>14</td>
<td>cWINNOWER</td>
<td>Word</td>
<td>-</td>
<td>28</td>
<td>2003</td>
</tr>
<tr>
<td>15</td>
<td>Improbizer</td>
<td>Probabilistic</td>
<td><a href="http://users.soc.ucsc.edu/~kent/improbizer/improbizer.html">http://users.soc.ucsc.edu/~kent/improbizer/improbizer.html</a></td>
<td>32</td>
<td>2004</td>
</tr>
<tr>
<td>18</td>
<td>QuickScore</td>
<td>Probabilistic</td>
<td><a href="http://algo.inria.fr/dolley/QuickScore/">http://algo.inria.fr/dolley/QuickScore/</a></td>
<td>44</td>
<td>2004</td>
</tr>
<tr>
<td>19</td>
<td>FMGA</td>
<td>Genetic algorithm</td>
<td>-</td>
<td>47</td>
<td>2004</td>
</tr>
<tr>
<td>20</td>
<td>SeSiMCMC</td>
<td>Probabilistic</td>
<td><a href="http://favorov.imb.ac.ru/SeSiMCMC/info.html">http://favorov.imb.ac.ru/SeSiMCMC/info.html</a></td>
<td>40</td>
<td>2004</td>
</tr>
<tr>
<td>21</td>
<td>Ensemble</td>
<td>Combination of algorithms</td>
<td>-</td>
<td>38</td>
<td>2005</td>
</tr>
<tr>
<td>22</td>
<td>Gibbs-ssT</td>
<td>Probabilistic</td>
<td>-</td>
<td>41</td>
<td>2006</td>
</tr>
<tr>
<td>23</td>
<td>MAMA</td>
<td>Hardware</td>
<td>-</td>
<td>2</td>
<td>2006</td>
</tr>
</tbody>
</table>

Table 2.2 List of algorithms and tools, the operating technique used, the web address, reference and year
2.4.2 Previous work on comparison of motif finding algorithms

To give a clear idea to users about the limitations and potentials of available motif matching algorithms, several comparison studies have been done by researchers. These studies also help in providing a benchmark for newly developed algorithms. Accurate analysis of these algorithms is very difficult, since motifs are short sequences occurring in a large amount of data and there can also be mutations, insertions or deletions. Moreover, tools are developed based on different approaches and motif models. So, one algorithm might compute good results on a particular dataset but might fail on a different dataset. And complete information of regulatory mechanisms is not available which makes evaluation of algorithms tougher [16]. In this section, we discuss the common statistical parameters used to assess algorithm performance, datasets used by researchers and results of the comparison studies.

2.4.2.1 Statistics used for comparison

To analyze the performance of different tools, we need to have statistical parameters for the basis of comparison. Based on the various studies conducted by researchers, we have tried to summarize the statistical parameters commonly used for comparative evaluation of tools in terms of prediction accuracy and reliability. We can group the performance criteria into three types, Nucleotide level, Transcription factor binding site level and Sequence level accuracy [38]. The range of values for all these statistics is from 0 to 1 except for the correlation coefficient. For correlation coefficient, the value ranges from -1 to +1. If the statistical parameters are close to 1, it indicates good performance of the algorithm.
**Nucleotide level prediction accuracy**

At the nucleotide level, for each predicted binding site which overlaps with the target binding site, we can have the following parameters for prediction of nucleotide accuracy [18, 38].

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$nTP$-</td>
<td>True positives, the number of common nucleotide positions in predicted and known binding sites</td>
</tr>
<tr>
<td>$nTN$-</td>
<td>True negatives, the number of nucleotide positions neither in predicted nor known binding sites</td>
</tr>
<tr>
<td>$nFP$-</td>
<td>False positives, the number of nucleotide positions only in predicted binding sites and not in known binding sites</td>
</tr>
<tr>
<td>$nFP$-</td>
<td>False positives, the number of nucleotide positions only in predicted binding sites and not in known binding sites</td>
</tr>
</tbody>
</table>

We try to represent these parameters in the form of a Venn diagram as shown in Figure 2.7. Let Set A contain known binding sites and set B contain predicted binding sites.

**Figure 2.7 Venn diagram showing $nTP$, $nTN$, $nFP$ and $nFN$**

The sensitivity at the nucleotide level can be defined as follows [18, 38]:

\[
\text{Sensitivity} = \frac{nTP}{nTP + nFP + nFN}
\]
The positive predictive value can be defined as follows [18]:

\[ nPPV = \frac{nTP}{nTP + nFP} \]

The specificity can be defined as [18]:

\[ nSp = \frac{nTN}{nTP + nFP} \]

The performance coefficient helps in comparing specificity and positive predictive value in a single measurement. It can be defined as [18, 38]:

\[ nPC = \frac{nTP}{nTP + nFN + nFP} \]

The value of \( nPC \) ranges between 0 and 1, where 1 is the value for perfect prediction.

Nucleotide correlation coefficient is defined by Burset and Guigo [51] as:

\[ nCC = \frac{nTP.nTN − nFN.nFP}{\sqrt{(nTP + nFN)(nTN + nFP)(nTP + nFP)(nTN + nFN)}} \]

The value of \( nCC \) ranges from -1 to +1, where -1 indicates perfect anti-correlation and +1 indicates perfect correlation. So for accurate prediction the value of \( nCC \) is +1 and for random prediction of motifs independently, the value of \( nCC \) is 0 indicating no correlation [18]. At the nucleotide level, the average score is calculated over all binding sites in a sequence, followed by the average score of all the sequences in a dataset. Then the average score over all the datasets is calculated [38].
**Binding site level accuracy**

Binding site level accuracy shows whether the predicted binding sites overlap with the known binding sites by at least a specified cutoff. The cutoff can be set depending on the accuracy we require for the analysis. We can define the following parameters for prediction of binding site level accuracy [18, 38].

| $sTP$- | True positives, the number of predicted binding sites overlapping with known binding sites. |
| $sFP$- | False positives, the number of predicted binding sites not overlapping with known binding sites |
| $sFN$- | False negatives, the number of known binding sites not overlapping with predicted binding sites |

As we defined nucleotide sensitivity, positive predictive value and performance coefficient, we can also define the accuracy metrics for each sequence at the binding site level.

Sensitivity at the binding site level can be defined as [18, 38]:

$$sSn = \frac{sTP}{sTP + sFN}$$

Positive predictive value at the binding site level can be defined as [18]:

$$sPPV = \frac{sTP}{sTP + sFP}$$

Performance coefficient can be defined as [38]:

26
\[ s_{PC} = \frac{s_{TP}}{s_{TP} + s_{FN} + s_{FP}} \]

The average performance at the binding site level can be computed as [18]:

\[ s_{ASP} = \frac{s_{Sn} + s_{PPV}}{2} \]

At the binding site level, the average score of binding site level accuracy over all the sequences in a dataset is calculated, followed by the average over all the datasets in a benchmark.

**Sequence level accuracy**

To evaluate sequence level accuracy, we calculate sequence level success rates, which compute the number of sequences with at least one correct predicted motif over the total number of sequences. Let \( N \) be the total number of sequences and \( N_s \) be the number of sequences with at least one correct predicted motif. Sequence success rate can be defined as [38]:

\[ s_{Sr} = \frac{N_s}{N} \]

For an algorithm, the sequence level accuracy is the average of \( s_{Sr} \) over all sequences for an input dataset.

**2.4.2.2 Datasets**

There are many web-based resources available for regulatory motifs datasets. Researchers or users can download the datasets and modify them according to their analysis perspective. Wei and Yu [29] provided information of all the web-based resources available for regulatory motif datasets. The data has been tabulated in Table 2.3.
Table 2.3 Selected web-based resources for regulatory motifs or TFBSs [29]

<table>
<thead>
<tr>
<th>Database</th>
<th>Explanation</th>
<th>URL (Accessed on 10/14/09)</th>
</tr>
</thead>
<tbody>
<tr>
<td>JASPAR</td>
<td>A collection of transcription factor DNA-binding preferences</td>
<td><a href="http://mordor.cgb.ki.se/cgi-bin/jaspar2005/jaspar_db.pl">http://mordor.cgb.ki.se/cgi-bin/jaspar2005/jaspar_db.pl</a></td>
</tr>
<tr>
<td>TRANSFAC</td>
<td>Database on eukaryotic transcription factors, their genomic binding sites and DNA-binding profiles</td>
<td><a href="http://www.gene-regulation.com/pub/databases.html#transfac">http://www.gene-regulation.com/pub/databases.html#transfac</a></td>
</tr>
<tr>
<td>TRRD</td>
<td>Transcription regulatory regions database</td>
<td><a href="http://wwwmgs.bionet.nsc.ru/mgs/grnw/">http://wwwmgs.bionet.nsc.ru/mgs/grnw/</a></td>
</tr>
<tr>
<td>RegulonDB</td>
<td>A computational model of mechanisms of transcriptional regulation</td>
<td><a href="http://regulondb.ccg.unam.mx/html/What_is_RegulonDB.jsp">http://regulondb.ccg.unam.mx/html/What_is_RegulonDB.jsp</a></td>
</tr>
<tr>
<td>TFD</td>
<td>Transcription factor databases</td>
<td><a href="http://www.ifti.org/">http://www.ifti.org/</a></td>
</tr>
</tbody>
</table>

Creation of good datasets is very difficult since tools have been developed using different algorithms and have been targeted for different motif models. So creating a dataset which is unbiased to any of the algorithms or motif models is very challenging. Tompa et al. [18] created good datasets from the TRANSFAC database for comparison of algorithms. They created 52 datasets, 26 from human, 12 from mouse, 6 from fly and 8 from yeast [18]. The datasets were of three different types: real, generic and Markov. More details of the data sets has been provided in Chapter 5.

Hu et al. [38] created large datasets from *Escherichia Coli* K-12 in the RegulonDB database. These datasets were used for evaluation on prokaryotes whereas the earlier datasets generated by Tompa et al. [18] were used on eukaryotes. The datasets generated by Hu et al. [38], ECRDB70, ECRDB62A and ECRDB70B-X are available at http://dragon.bio.purdue.edu/pmotif/.
2.4.2.3 Results of comparison studies

A comparative analysis of 13 algorithms, AlignACE, GLAM, QuickScore, SeSiMCMC, Improbizer, MEME, MITRA, MotifSampler, Oligo/dyad analysis, ANN-Spec, Consensus, Weeder and YMF was conducted by Tompa et al. [18] on eukaryotic datasets present in the TRANSFAC database. Based on the study, it has been found that the absolute measure of correctness of these algorithms was very low. The maximum value for binding site correlation coefficient was 0.20 and the maximum value for sensitivity was 0.22 [18]. The authors warned that the predictions might not be completely accurate since datasets used might not be perfect, as the knowledge of underlying regulatory mechanisms is still incomplete.

Figure 2.8 shows the results of the 13 tools. Figure 2.8a summarizes the seven statistics, nSn, nPPV, nPC, nCC, sPPV and sASP over all the data sets. Figure 2.8b shows the combined nCC value based on the species. Figure 2.8c shows the combined nCC values based on the data set type. Figure 2.8d shows the seven statistics calculated over all the tools after removing the real data sets. Figure 2.8e shows the combined specificity of all the tools on the negative data sets. The statistics explained in the previous section have been calculated for each tool. In general, Weeder was observed to produce good results compared to other algorithms and the authors stated its success might be attributed to its ability to predict no motifs in negative datasets. SeSiMCMC did better than other algorithms on fly datasets while MEME and YMF did better on mouse datasets. All the algorithms performed well on yeast datasets. The correlation coefficient increased by 39% when real datasets were removed. The real datasets were removed as they are very pessimistic and degrade the performance of the algorithm. MotifSampler performed better on real datasets. The authors suggested that using a combination
of tools would yield better results than a single one. This strategy was used in one set of experiments by Hu et al. [38] and did improve performance.

Hu et al. [38] conducted another comparison study of motif finding algorithms. They compared AlignACE, MotifSampler, MEME, BioProspector and MDScan on prokaryotic
datasets in the RegulonDB [29] database. The authors allowed minimal parameter tuning during evaluation to avoid bias towards any particular algorithm. They found that the nucleotide level accuracy is very low for all the algorithms. The scores are better on eukaryotic datasets [18] because the sequence length is shorter in prokaryotic datasets. MEME was found to have the best sensitivity and sequence success rate. BioProspector had highest the performance coefficient [38]. The prediction accuracy was found to be better at the binding site level than at the nucleotide level. The performance coefficient score of AlignACE was lowest at the binding site level. MEME was found to handle diverse sequences very well [38]. With increase in sequence length, the performance of all the algorithms decreased. Also the Gibbs sampling strategy became very inefficient with longer sequence lengths. The authors had developed an ensemble algorithm [16] which combines the predictions of multiple algorithms with multiple motifs. The ensemble algorithm was found to perform better than other algorithms by 52%. So combining the predictions of various runs using different tools was found to yield better results [38]. This technique has been suggested in Section 6.2 as one of the ways to extend the current work.
3 EXPECTATION MAXIMIZATION ALGORITHM (EM)

Expectation maximization (EM) was first introduced by Dempster et al. [52] in 1977. The EM algorithm is widely used in biological applications that use probabilistic models. One of the major applications of EM is to the motif finding problem. EM facilitates parameter estimation in models which have incomplete data [53]. In the motif finding problem, we have a dataset of unaligned DNA sequences and we need to find a common motif which is present in all the sequences. There can be some mutations in the motif. The information available to us is the set of unaligned sequences and length of the motif W. Hence the problem boils down to finding the starting positions of the motif in each sequence. The parameters of the probabilistic model which might have generated the dataset are estimated by the EM algorithm. It is a two component model. The first component is the set of motifs which have fixed width and the second component is all the other non-motif subsequences in the dataset and is called background [54].

This chapter explains the EM algorithm. An example of EM has also been presented to provide a clear picture of the algorithm.

3.1 Algorithm

Do et al.[53] provided a good analogy to explain the EM algorithm using a coin flipping experiment. In the coin flipping experiment, we suppose we have two coins, A and B. Let $\rho_A$ be the probability of coin A to land on heads on any given toss and $\rho_B$ be the probability of coin B to land on heads on any given toss. The main goal of the procedure is to estimate $\rho_A$, $\rho_B$ by performing the following experiment:
Step 1: We randomly choose one of the two coins, which we have chosen is not known.

Step 2: Perform ten tosses with the selected coin.

Step 3: Repeat steps 1 and step 2 five times.

Hence, the complete experiment involves 50 coin tosses, i.e., randomly choose a coin 5 times and independently toss each selected coin 10 times. Let us call each repetition of step 1 and 2 a trial.

During the experiment, let us keep track of the following variables:

\[ z = (z_1, z_2, z_3, z_4, z_5) \] - value of the selected coin in step 1. \( z \in \{A, B\} \) for trial ‘i’, \( 1 \leq i \leq 5 \).

\[ x = (x_1, x_2, x_3, x_4, x_5) \] - is the number of heads observed in step 2. \( x_i \in \{0,1,2,3,4,5,6,7,8,9,10\} \) for trial ‘i’, \( 1 \leq i \leq 5 \).

If we know whether the selected coin was A or B, estimation of \( \rho_A, \rho_B \) would be straightforward using the following formulae:

\[
\rho_A = \frac{\text{No of heads with coin } A}{\text{Total number of flips with coin } A}
\]

\[
\rho_B = \frac{\text{No of heads with coin } B}{\text{Total number of flips with coin } B}
\]

However, if we do not know which coin was chosen, calculation of \( \rho_A, \rho_B \) becomes complex. The model is incomplete and the hidden parameters are \( (z_1, z_2, z_3, z_4, z_5) \). If we make an intuitive guess of these hidden parameters, then the model would be complete. And the problem would be reduced to estimation of the maximum likelihood on the complete data. We start with an initial guess of \( \rho_A^0, \rho_B^0 \) and determine if coin A or B could have more likely generated the observed x.
values. Later, we assume that the estimated ‘z’ values are correct, and apply the maximum likelihood procedure to estimate \( \rho_A^1, \rho_B^1 \). These steps are repeated until the values \( \rho_A^j, \rho_B^j \) converge. The speed of convergence depends heavily on the initial guess.

The EM algorithm is similar to the coin flipping experiment. In the coin flipping experiment, we estimate the information about the selected coin, whether it is A or B. Instead of picking a single most likely estimation of the hidden parameters (A or B), the EM algorithm computes the probabilities for each possible completion of the hidden parameters, using the known parameters [53]. The algorithm iterates over two steps. In the first step, known as the expectation step, we guess the probability distribution of hidden parameters using the available model parameters. These probabilities are used to calculate a weighted training set which has information about all the possible completions. In the next step, known as the maximization step, we re-estimate the new parameters of the model using these completions. Both these steps are repeated until the algorithm converges.

### 3.2 Application of EM to the motif finding problem

In motif finding, we are given a dataset of sequences \( S \). We need to find motifs which can be considered as subsequences occurring in these sequences. The sequences can be divided into two parts, namely, motif and background. The background model contains all the subsequences which are not present in the motif model.

The algorithm assumes that there is one motif in each sequence and it is referred to as the one-occurrence-per sequence model [31]. Let the length of the motif be \( W \). The location of the motif in a sequence is unknown and we need to estimate the offset location of the motif in each sequence. The algorithm starts with an initial guess of the offset locations of the motif in the
dataset. Assuming that the initial guess is correct, the algorithm proceeds to estimate the probability ‘z\textsubscript{ij}’ that the shared motif begins in location ‘j’ in sequence number ‘i’ [31]. Later in the maximization step, z\textsubscript{ij} is used to calculate the probability (\(\rho\textsubscript{lc}\)) of alphabet ‘l’ occurring in column ‘c’ of the shared motif. \(\rho\textsubscript{lc}\) is calculated for each letter in the entire length of the motif, \(1 \leq c \leq W\). EM later re-estimates \(z\textsubscript{ij}\) from \(\rho\textsubscript{lc}\) and the loop repeats until the change in \(\rho\) is sufficiently low in two consecutive iterations, as determined by a preset tolerance \(\epsilon\). The calculation of \(z\) and \(\rho\) is explained in detail in the next section. Figure 3.1 shows the steps in EM algorithm.

Figure 3.1 Algorithm starts with a single site and later alternates between assigning sites to a motif (left side) and then updates the motif model (right side) [1]
The flowchart in Figure 3.2 shows the various steps in the EM algorithm.

Figure 3.2 Flowchart of EM algorithm
3.2.1 Expectation step

To explain the different steps in the EM algorithm, we first need to define a few variables. Let $S$ be the dataset of sequences and $S_i$ be the $i^{th}$ sequence. Let the length of the motif be $W$ and the length of each sequence be $L$. We assume that all the sequences are of equal length. As defined earlier, let $z_{ij}^x$ be the estimated probability after ‘x’ iterations that the motif starts in position ‘j’ in sequence ‘i’. $\rho_{lc}^x$ be the probability after ‘x’ iterations that letter ‘l’ occurs in position ‘c’ of the motif [31]. We define another variable $X_{ij}$ which equals 1 if the motif occurs in position ‘j’ in sequence ‘i’. If the motif does not occur in the specified position, the value of $X_{ij}$ is 0.

To calculate $z_{ij}^x$, we first calculate the probability of sequence $S_i$, given $X_{ij} = 1$, with the current model $\rho^x$ using the following equation [31]:

$$P(S_i/(X_{ij} = 1), \rho^x) = \prod_{c=1}^{W} \rho_{lc,c}^x$$

For correct prediction of probability, we might also include the background probability in the above calculation. Bayes’ rule is used for the estimation of $z^x$ from $P(S_i/(X_{ij} = 1), \rho^x)$. According to Bayes’ rule [31],

$$P(A/B) = \frac{P(A)P(B/A)}{P(B)}$$

Applying Bayes’ rule to estimate $z_{ij}^x$, we get the following equation [31]:

$$z_{ij}^x = P\left(\frac{X_{ij}=1}{\rho^x}, S_i\right) = \frac{\rho^{0}(X_{ij}=1)p(S_i/X_{ij}=1, \rho^x)}{\sum_{c=1}^{L-W+1} \rho^{0}(X_{ic}=1, \rho^x)p^{0}(X_{ik}=1)}$$
P\(^0\)(X\(_{ij}\)=1) is the probability calculated initially that the shared motif begins in location ‘j’ in sequence ‘i’. P\(^0\)(X\(_{ij}\)=1) is assumed to be equal and uniform and is calculated using:

\[
P^0(X_{ij} = 1) = \frac{1}{L-W+1}
\]

After applying the value of P\(^0\), the equation for z\(_{ij}\) simplifies to the following [31]:

\[
z_{ij}^x = \frac{P(S_i/X_{ij} = 1, \rho^x)}{\sum_{c=1}^{L-W+1} P(S_i/X_{ic} = 1, \rho^x)}
\]

After estimating z\(_{ij}\)\(^x\) using the above equations, we normalize the values of z\(_{ij}\) such that

\[
\sum_{j=1}^{L-W+1} z_{ij} = 1
\]

3.2.2 Maximization step

In the maximization step, we calculate \(\rho_{lc}\), the probability of letter ‘l’ occurring in position ‘c’ in the motif. Let \(n_l\) be the total number times letter ‘l’ appears in the given dataset. \(n_{lc}\) be the number of letter ‘l’s in position ‘c’ of motif. We calculate the total number of letters in a particular location using the following formula [31]:

\[
n_{lc} = \sum_i \sum_{(j/(S_i,j+c-1)=l)} z_{ij} \quad \text{for } c > 0
\]

\[
n_{lc} = n_l - \sum_{j=1}^{w} n_{lk} \quad \text{for } c = 0
\]

The values for background are represented by \(c=0\). Using the values of \(n_{lc}\), we calculate the probability of letter ‘l’ occurring in position ‘c’ of motif, \(\rho_{lc}\) as shown in the equation below [31]:
\[ \rho_{ic}^x = \frac{n_{ic} + d_{ic}}{\sum_i (n_{ic} + d_{ic})} \]

We add pseudo counts \( d_{ic} \) and \( d_{ic} \) to prevent the probability from being zero.

3.3 An example showing application of EM algorithm to motif finding

In the previous section, we have explained the various steps involved in the EM algorithm. We present an example in this section to give a clear picture of the flow of the algorithm. Let us take a set of 4 DNA sequences and let the motif width be 3. The randomly chosen start locations for DNA sequences are shown below:

<table>
<thead>
<tr>
<th>Seq number</th>
<th>Sequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ACTATAGC</td>
</tr>
<tr>
<td>2</td>
<td>TGCGAAAT</td>
</tr>
<tr>
<td>3</td>
<td>AAAGGATC</td>
</tr>
<tr>
<td>4</td>
<td>GTCTGATG</td>
</tr>
</tbody>
</table>

Table 3.1 Dataset of sequences

Let the randomly chosen initial starting positions of motifs be:

For sequence 1: 4

For sequence 2: 2

For sequence 3: 6

For sequence 4: 3
The motif matrix formed using the above random starting positions is:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ATA</td>
<td>GCG</td>
<td>ATC</td>
<td>CTG</td>
</tr>
</tbody>
</table>

Table 3.2 Motif matrix

And the non-motif matrix formed is:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTGC</td>
<td>TAAAT</td>
<td>AAAGG</td>
<td>GTGTG</td>
</tr>
</tbody>
</table>

Table 3.3 Non-motif matrix

**Probability matrix**

The probability matrix is formed by counting the frequency of bases in each position for the motif matrix. For example, in the above case, the number of A’s occurring in the first position of the motif matrix is 2. Similarly we find the frequency of C, T, G in the motif matrix in different positions. The values are tabulated in Table 3.4:
To find the probabilities, we divide all the values by 4, since the bases can be either A, C, T or G.

We assume the occurrence of all the bases is equally likely.

The probability matrix formed is:

<table>
<thead>
<tr>
<th>Base</th>
<th>$\rho_1$</th>
<th>$\rho_2$</th>
<th>$\rho_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.5</td>
<td>0</td>
<td>0.25</td>
</tr>
<tr>
<td>C</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>0.5</td>
<td>0.25</td>
</tr>
<tr>
<td>G</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
</tr>
</tbody>
</table>

Table 3.5 Probability of bases in motif

We also find the background probabilities by calculating the number of A, C, T and G in the non-motif matrix and dividing it by the total number of elements in the non-motif matrix. The background probability is represented as $\rho_0$. After calculating the background probabilities, the modified probability matrix is shown in Table 3.6.
Table 3.6 Probability matrix

<table>
<thead>
<tr>
<th>Base</th>
<th>$\rho_0$</th>
<th>$\rho_1$</th>
<th>$\rho_2$</th>
<th>$\rho_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.35</td>
<td>0.5</td>
<td>0</td>
<td>0.25</td>
</tr>
<tr>
<td>C</td>
<td>0.10</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>T</td>
<td>0.25</td>
<td>0</td>
<td>0.5</td>
<td>0.25</td>
</tr>
<tr>
<td>G</td>
<td>0.30</td>
<td>0.25</td>
<td>0.25</td>
<td>0.25</td>
</tr>
</tbody>
</table>

Expectation step

After computing the probability matrix, EM initially estimates the probability $z_{ij}$, that the shared motif starts at position ‘j’ in dataset sequence ‘i’. $z_{ij}$ is calculated using the probability matrix formed. To find $z_{11}$ (probability of motif occurring in position 1 in sequence 1), the probability of the occurrence of the base in the motif and in the background are multiplied [31].

$$z_{11} = \rho_A \times \rho_C \times \rho_T \times \rho_A \times \rho_T \times \rho_A \times \rho_0 \times \rho_0$$

$$= 0.5 \times 0.25 \times 0 \times 0.35 \times 0.25 \times 0.35 \times 0.30 \times 0.10$$

$$z_{12} = \rho_A \times \rho_C \times \rho_T \times \rho_A \times \rho_T \times \rho_A \times \rho_0 \times \rho_0$$

$$= 0.35 \times 0.25 \times 0.75 \times 0.25 \times 0.25 \times 0.35 \times 0.30 \times 0.10$$

$$z_{13} = \rho_A \times \rho_C \times \rho_T \times \rho_A \times \rho_T \times \rho_A \times \rho_0 \times \rho_0$$

$$= 0.35 \times 0.10 \times 0 \times 0 \times 0 \times 0.35 \times 0.30 \times 0.10$$

$$z_{14} = \rho_A \times \rho_C \times \rho_T \times \rho_A \times \rho_T \times \rho_A \times \rho_0 \times \rho_0$$

$$= 0.35 \times 0.10 \times 0.25 \times 0.50 \times 0.75 \times 0.25 \times 0.30 \times 0.10$$
\[ z_{15} = \rho_{A0} \times \rho_{C0} \times \rho_{T0} \times \rho_{A0} \times \rho_{T1} \times \rho_{A2} \times \rho_{G3} \times \rho_{C0} \]
\[ = 0.35 \times 0.10 \times 0.25 \times 0.35 \times 0 \times 0 \times 0.50 \times 0.10 \]

\[ z_{16} = \rho_{A0} \times \rho_{C0} \times \rho_{T0} \times \rho_{A0} \times \rho_{T0} \times \rho_{A1} \times \rho_{G2} \times \rho_{C3} \]
\[ = 0.35 \times 0.10 \times 0.25 \times 0.35 \times 0.25 \times 0.50 \times 0 \times 0.25 \]

Similarly, we find the probability of a motif beginning at a particular position for the other sequences. These are shown in Table 3.7.

<table>
<thead>
<tr>
<th>Sequences</th>
<th>Position 1</th>
<th>Position 2</th>
<th>Position 3</th>
<th>Position 4</th>
<th>Position 5</th>
<th>Position 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>z_{11}</td>
<td>z_{12}</td>
<td>z_{13}</td>
<td>z_{14}</td>
<td>z_{15}</td>
<td>z_{16}</td>
</tr>
<tr>
<td>S2</td>
<td>z_{21}</td>
<td>z_{22}</td>
<td>z_{23}</td>
<td>z_{24}</td>
<td>z_{25}</td>
<td>z_{26}</td>
</tr>
<tr>
<td>S3</td>
<td>z_{31}</td>
<td>z_{32}</td>
<td>z_{33}</td>
<td>z_{34}</td>
<td>z_{35}</td>
<td>z_{36}</td>
</tr>
<tr>
<td>S4</td>
<td>z_{41}</td>
<td>z_{42}</td>
<td>z_{43}</td>
<td>z_{44}</td>
<td>z_{45}</td>
<td>z_{46}</td>
</tr>
</tbody>
</table>

**Table 3.7 Probability of motif occurring at a particular position**

After finding ‘z’ values for all the sequences, we normalize z values such that each row adds up to 1 (assuming one motif per sequence).

Once all the z values are calculated, we proceed to the maximization step.

**Maximization step**

In the maximization step, \( z_{ij} \) is used to re-estimate the probability of letter ‘l’ in column ‘c’ of the motif, \( \rho_{lc} \), for each letter in the alphabet and \( 1 \leq c \leq W \). ( \( W \) is the width of the motif).

In the above example, to calculate the probability that ‘A’ occurs in the first column of the motif, we look at all the possible locations where a motif starts with ‘A’. In sequence 1, ACTATAGC,
the motif can start with ‘A’ at locations 1, 4 and 6. Similarly for other sequences, we look at the locations where the motif can start with A, and add all the ‘z’ values. The sum of the ‘z’ values, where ‘A’ is located in first column of motif is divided by the total sum of all the ‘z’ values as shown below to get the probability of occurrence of ‘A’ in location ‘1’.

$$\rho_{A1} = \frac{z_{11} + z_{14} + z_{16} + z_{25} + z_{26} + z_{31} + z_{32} + z_{33} + z_{36}}{\text{Sum of all z values}}$$

Similarly, we find $\rho_{A2}$, $\rho_{A3}$, $\rho_{A0}$, $\rho_{C1}$, $\rho_{C2}$, $\rho_{C3}$, $\rho_{C0}$, $\rho_{T1}$, $\rho_{T2}$, $\rho_{T3}$, $\rho_{T0}$, $\rho_{G1}$, $\rho_{G2}$, $\rho_{G3}$, $\rho_{G0}$ also. The pseudo counts are added to make the probability nonzero. This prevents the algorithm being stuck at local maxima. We add 1 in the numerator and 4 in the denominator of the above equation. Four is added in the denominator because the number of bases is 4. The new equation is:

$$\rho_{A1} = \frac{z_{11} + z_{14} + z_{16} + z_{25} + z_{26} + z_{31} + z_{32} + z_{33} + z_{36} + 1}{\text{Sum of all z values} + 4}$$

The new values of $\rho$ are compared with the previous values in the probability matrix. If differences are less than the tolerance $\epsilon$, the algorithm stops. If any difference is greater than $\epsilon$, the algorithm repeats again until it converges.
This chapter describes the hardware implementation of the Expectation Maximization algorithm. The various modules included in the hardware implementation have been explained here. All the modules have been implemented in Verilog HDL. This chapter also includes details of the architecture and explains the benefits of using it.

4.1 Architecture

This section explains the architecture of the hardware implementation. The connection between different modules is shown in Figure 4.1.
Figure 4.1 Block diagram of the architecture developed
The architecture consists of six main modules namely, LFSR module, Matrix formation module, Probability matrix module, Expectation module, Maximization module and Comparison module. All the modules have been implemented in Verilog HDL. LFSR module is used for random number generation. The random numbers generated are used as start positions of the motif. If there are ‘N’ sequences, ‘N’ random numbers are generated and each number corresponds to a start location of the motif. The random numbers generated are stored in RAM. Depending on the start locations of the motif and the width of the motif, two matrices, motif and non-motif matrix are formed in the Matrix formation module. The matrices formed are stored in two different RAM’s.

The probability matrix module forms the probability matrix using the Motif and Non-motif matrices. The probability of occurrence of each base at any location is calculated in this module. After formation of the probability matrix, the Expectation module calculates ‘z’ values. Depending on the ‘z’ values calculated in the Expectation module, a new Probability matrix is formed in the Maximization module. The values of the Probability matrix at the end of the iteration are compared with the values at beginning of the iteration in the Comparison module. If the differences are small than the chosen tolerance €, the algorithm stops execution, and the motif start locations are stored in a RAM. If any difference is greater than €, the algorithm repeats again.

The whole design is divided into six different modules to facilitate pipelining. The LFSR module generates random numbers for the next iteration after formation of the Probability matrix since the random numbers are no longer necessary. The whole design can be divided into a five-stage pipeline but it would increase the hardware resources needed. Hence, we have decided to pipeline only random number generation with the remaining modules.
4.2 Details of design

The input to the algorithm is the dataset of sequences and the width of the motif W. The dataset of sequences is stored in a RAM module named ‘sequences’. As shown in the Figure 4.2, the inputs to the RAM are din_seq, raddr_seq, waddr_seq, we_seq, re_seq, clk_seq, rst_seq. The output from the RAM is dout_seq.

![Figure 4.2 Sequences RAM](image)

The Verilog code snippet for ‘Sequences’ is shown below:

```verilog
module sequences(clk,re_seq,we_seq,raddr_seq,waddr_seq, rst_seq din_seq,dout_seq);
input [ADDR-1:0]raddr_seq; // contains address of location to be read
input [ADDR-1:0]waddr_seq; // contains address to location to be written to
input clk; //clock signal
input rst_seq; // reset signal
input re_seq; // asserted high for read operation
```

DNA base pairs are formed from the chemical bases adenine (A), guanine (G), cytosine (C), and thymine (T). Hence the dataset of sequences consists strings of the four bases A, G, C and T. To represent four bases in binary, two digits are required. The bases are represented in binary as follows:

A – 00
C – 01
T – 10
G – 11

For example, a sequence ATGCAA is represented in RAM as 001011010000. To convert all the sequences in A, C, T and G to binary format, a perl script has been developed.

The key to implementation of the Expectation Maximization algorithm is dividing the whole algorithm into different steps and realizing different states in each step. The whole design is divided into hardware submodules and each module is realized using a finite state machine. A finite state machine is a design which contains a finite number of states, with transitions between those states and actions to perform in a particular state. The next section explains various sub-modules involved in the design and the state machines developed for each module.
4.3 Modules

The hardware implementation of the EM algorithm consists of six main submodules: random number generation, formation of motif and non-motif matrix, formation of probability matrix, expectation step, maximization step and comparison between iterations. The algorithm begins with the generation of random starting positions. A Linear Feedback Shift Register has been used to generate random numbers. Using the random numbers generated, two matrices known as motif and non-motif are created. The motif matrix contains the subsequences likely to be motifs and the non-motif matrix contains the background sequences. After formation of motif and non-motif matrices, the probabilities for each of the bases at each location in the motif are calculated. The background probabilities are also calculated. Then, expectation and maximizations steps are executed until the difference in values of probabilities is sufficiently low for two successive iterations.

4.3.1 Random number implementation

This module is used for random number generation. The random numbers generated are used as start locations of motif in sequences. For generation of random numbers in hardware, a Linear Feedback Shift Register (LFSR) has been used. A LFSR is a series of shift registers, for which input is a linear function of the previous state [55]. The linear function is constructed using XOR of a few bits as input to the LFSR. These bits represent the nonzero coefficient of a primitive polynomial over a finite field of characteristic 2. The length of the LFSR determines the size of the field. Figure 4.3 shows a 5-bit LFSR.
The 5-bit LFSR is constructed using five D-flip-flops, and an XOR gate with inputs as tap1 and tap4. Tap 1 is output from the second flip-flop from the left and Tap 4 is output from the last flip-flop. The values fed to the XOR gate are referred to as taps. The output of the XOR gate is fed into the least significant bit. The seed to LFSR is its initial value. The values can be predicted since the current state depends on the previous state and the LFSR is a finite state machine [55]. But, we can create a very long sequence of bits using a good feedback function. For an n-stage LFSR, the maximum number of random numbers generated is $2^n - 1$. It generates all n-bit binary numbers except zero. A LFSR does not generate random numbers if the initial value of LFSR is all zeroes. So, we need to make sure that the seed is not 0. A lot of research has been done over the years to know the taps which form the maximum length. The tables available online [55] have been used to decide on the taps for our design.

A few other implementations of random number generators on hardware are multiple LFSR, cellular automata, and multiple cellular automata [57]. A multiple LFSR uses $n$ LFSR’s of length $m$ and one bit from each LFSR is used in a particular step. This helps in improving the performance of the LFSR. Another popular random number generator on hardware is cellular

---

**Figure 4.3 5-bit LFSR [56]**
Automata. A 1-dimensional cellular automaton consists of a string of cells. The value of each cell depends on the values of the neighboring cells on its left and right. In a particular step, the state of a cell depends on its present state and/or the states of its neighboring cells based on a set of rules. To improve the performance of a cellular automaton random number generation, several cellular automata can be combined, and the value from each cellular automaton can be used to generate the random number. This type of implementation is known as multiple cellular automata. Based on the experiments performed by researchers, it has been observed that multiple LFSR performs the best in generation of random numbers [57]. But in terms of operating frequency, it has been observed that LFSR implementation has the maximum frequency. If random number implementation is not in the critical path of our design, then it would be better to use multiple LFSR to increase performance.

4.3.2 Formation of motif and non-motif matrix

After generation of random numbers, sequences are divided into two different sets based on the random numbers. The random numbers generated are used for guessing the starting locations of the motif in each sequence. Suppose there are 50 sequences in a dataset, we generate 50 random numbers to guess the start location of the motif in each sequence. Depending on the length of the motif, two matrices, motif and non-motif are formed. Motif matrix has the set of motifs. Non-motif matrix has the background sequences. Let us consider an example to get a clear picture. The table below shows the dataset of sequences and Motif width is 4.
Let the random numbers generated be 3, 1, 3, 2, 4. The motif and non-motif matrix formed from the random numbers are shown in Table 4.2.

<table>
<thead>
<tr>
<th>Motif matrix</th>
<th>Non-motif matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>TGGG</td>
<td>ACTC</td>
</tr>
<tr>
<td>GTCG</td>
<td>ACTA</td>
</tr>
<tr>
<td>CTAT</td>
<td>GAAT</td>
</tr>
<tr>
<td>TTCC</td>
<td>TCAT</td>
</tr>
<tr>
<td>AGCT</td>
<td>CTCA</td>
</tr>
</tbody>
</table>

Table 4.2 Motif and non-motif matrix

A finite state machine has been developed for formation of motif and non-motif matrices. For formation of these matrices, a ‘for’ loop has to be created to iterate through all the sequences. But, using a ‘for’ loop in Verilog would create redundant logic in hardware after synthesis. To avoid this, a state machine is created. The state machine has four states, S0, S1, S2, and S3. Two flags are created, f1 and f2. Flag f1 determines whether the base belongs to motif or non-motif matrix. Flag f2 is used to repeat the loop through all the sequences in the dataset.
These flags are inputs to the state machine. In the details of the hardware implementation, bases are referred as bits as they are stored in binary format.

Flag f1 = 0 (if bit is not in motif, implies the bit has to be stored in non-motif RAM)

= 1 (if bit has to be stored in motif RAM)

Flag f2 = 0 (if all sequences are iterated)

= 1 (if there are more sequences to be iterated)

Depending on the flags, the state machine goes from one state to another. S0 is the initial state. All the flags and registers are reset in this state. S1 is used for storing bits in the motif matrix. S2 state is used for storing bits in the non-motif matrix. S3 is the final state used to indicate all the sequences have been covered and motif and non-motif matrices are formed. The state diagram developed for this module is shown in Figure 4.4. It shows the various transitions between states based on the values of flags.
4.3.3 Formation of probability matrix

This module is used for calculation of the probability of occurrence of each base in every location of the motif. It also calculates the background probability. Let us consider the following motif and non-motif matrices.

<table>
<thead>
<tr>
<th>Motif matrix</th>
<th>Non-motif matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>TGGG</td>
<td>ACTC</td>
</tr>
<tr>
<td>GTCG</td>
<td>ACTA</td>
</tr>
<tr>
<td>CTAT</td>
<td>GAAT</td>
</tr>
<tr>
<td>TTCC</td>
<td>TCAT</td>
</tr>
<tr>
<td>AGCT</td>
<td>CTCA</td>
</tr>
</tbody>
</table>

Table 4.3 Motif and non-motif matrices
The probability of occurrence of A, C, T and G at position 1, 2, 3, 4 are calculated in this module. The background probability is also calculated. The probability of the occurrence of ‘A’ in position 1 is 0.2, as A occurs only once in 5 locations. Similarly, we calculate the probability of C, T and G in position 1 as 0.2, 0.4, and 0.2. After calculating the probability for position 1, we calculate the probabilities for positions 2, 3 and 4 in a similar fashion. Table 4.4 shows the probability matrix formed for the dataset. \( \rho_0 \) represents the background probability, \( \rho_1 \) to \( \rho_4 \) represent motif probabilities.

<table>
<thead>
<tr>
<th>Base/Probability</th>
<th>( \rho_0 )</th>
<th>( \rho_1 )</th>
<th>( \rho_2 )</th>
<th>( \rho_3 )</th>
<th>( \rho_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.35</td>
<td>0.2</td>
<td>0</td>
<td>0.2</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0.3</td>
<td>0.2</td>
<td>0</td>
<td>0.6</td>
<td>0.2</td>
</tr>
<tr>
<td>T</td>
<td>0.3</td>
<td>0.4</td>
<td>0.6</td>
<td>0</td>
<td>0.4</td>
</tr>
<tr>
<td>G</td>
<td>0.05</td>
<td>0.2</td>
<td>0.4</td>
<td>0.2</td>
<td>0.4</td>
</tr>
</tbody>
</table>

Table 4.4 Probability matrix

A state machine has been developed to realize the logic explained above. Three flags f1, f2 and f3 are used to decide transitions between states. Flag f1 is used to ensure that all the sequences in the data set are iterated. Flag f2 makes sure that all the locations in the motif are iterated for each sequence. Flag f3 is used to iterate through non-motif locations in each sequence.

Let \( N \) be the number of sequences, \( L \) be the length of each sequence and \( M \) be the length of the motif. Let \( i, j \) and \( k \) be three different counters.

Flag \( f1 = 1 \) if \( i < N \) (if there are more sequences to be iterated)
f1 = 0 if i >= N (if all sequences are iterated)

Flag f2 = 1 if j < M (if there are more motif locations to be iterated)

f2 = 0 if j >= M (if all locations in motif are iterated)

Flag f3 = 1 if k < L-M (if we have more non-motif locations to be iterated)

f3 = 0 if k >= L-M (if all non-motif locations are iterated)

The purpose of the flags is to ensure that the state machine stays in a required state until the current flag changes its value. It essentially facilitates execution of a ‘for’ loop. The state machine built for this module has 7 states. S0 is the initial state. In this state all reset operations are carried out. S1 to S3 states are used for calculating probability values at motif locations. S4 to S6 states are used for calculation of background probability values. The state machine in Figure 4.5 shows the transition between different states depending on the flag values.

Figure 4.5 State machine for formation of probability matrix
4.3.4 Expectation

Expectation step is used for calculation of an expected log likelihood based on the current estimated probability model. After formation of the probability matrix, the probability of the occurrence of a shared motif at a particular location, $z_{ij}$ is calculated, where ‘i’ is the current sequence and ‘j’ is the location in the sequence. The range of ‘i’ is from 1 to N, where N is the number of sequences. The range of ‘j’ is from 1 to L-M, where L is the length of each sequence and M is the length of the shared motif. The calculation of probability $z_{ij}$ has been explained in Chapter 3. After calculation of $z_{ij}$, the values of $z_{ij}$ for a particular sequence are normalized to 1, since the number of motifs per sequence is assumed to be one.

For implementation of the expectation step in Verilog HDL, a state machine has been developed with five states. The inputs to the state machine are three flags, f1, f2 and f3, as shown in Figure 4.6. Let N be the number of sequences, L be the length of each sequence and M be length of motif. Let i1, j1 and k1 be three different counters.

Flag f1 = 1 if i1 < N (if there are more sequences to be iterated)

\[ f1 = 0 \text{ if } i1 \geq N \text{ (if all sequences are iterated)} \]

Flag f2 = 1 if j1 < L-M (if there are more valid start locations of motif to be iterated)

\[ f2 = 0 \text{ if } j1 \geq L-M \text{ (if all valid start locations of motif are iterated)} \]

Flag f3 = 1 if k1 < L (if we need to iterate through more bits in a sequence)

\[ f3 = 0 \text{ if } k1 \geq L \text{ (if the entire sequence is iterated)} \]
S0 is the initial state and all flags and variables are reset in this state. S1, S2 and S3 are used for calculation of $z_{ij}$. The flags are used to ensure that the probability values are calculated for all sequences at every location. Flag $f_1$ is used to iterate in the same state until all the sequences are covered. Flag $f_2$ is used to ensure that the computations are done for every valid location in the sequence. Flag $f_3$ is used for staying in state S1 until the entire length of the sequence is covered. The state machine reaches S4 after all the probability values are calculated and normalization is done in this step.

![State machine for expectation step](image)

**Figure 4.6 State machine for expectation step**

4.3.5 Maximization

In the maximization step, we compute the parameters which maximize log likelihood calculated in the expectation step. Here the probability of occurrence of a base at a particular location in the motif is calculated using the values of $z_{ij}$ computed in the expectation step. Basically, the probability matrix is calculated again using the expected parameters. The calculation of the probability $\rho_{lc}$, (where ‘l’ is the location in motif and ‘c’ is the current base) is explained in Chapter 3. The background probability is also calculated in this step.
To do these computations, a state machine has been developed and is implemented in Verilog HDL. The state machine developed for the maximization step is similar to the state machine developed for the expectation step. But the conditions for the flags vary and operations performed at each state are different. This state machine has five states and the inputs to the state machine are three flags, f1, f2 and f3. Let N be the number of sequences, L be the length of each sequence and M be the length of the motif. Let i2, j2 and k2 be three different counters.

Flag f1 = 1 if i2 < N (if there are more sequences in data set to be iterated)

    f1 = 0 if i2 >= N (if all sequences in data set are iterated)

Flag f2 = 1 if j2 < M (if we have to iterate through more locations in motif)

    f2 = 0 if j2 >= M (if all locations in motif are iterated)

Flag f3 = 1 if k2 < L-M (if there are more valid start locations of motif to be iterated)

    f3 = 0 if k2 >= L-M (if all valid start locations of motif are iterated)

S0 is the initial state; all the flags and registers are reset in this state. The state machine stays in state S1 for L-M iterations. In this state, the base at each location is matched and the corresponding counter is incremented. After matching all the bases for the current sequence, the state machine transitions to state S2, and stays in this state for M iterations. This entire loop is repeated for all the sequences. Counter i1 is incremented once the computations are performed for a particular sequence. In state S3, the values of counters j2 and k2 are reset, and the operations at states S1 and S2 are repeated for the next sequence. This is continued till i2 counter reaches N. After i2 reaches N, the state machine transitions to the final state, and the background
probability is calculated here. The state machine for the maximization step is shown in Figure 4.7.

![Figure 4.7 State machine for maximization step](image)

### 4.3.6 Comparison

In this module, the values of $\rho_{lc}$ after the maximization step are compared with the initial probability values in probability matrix. If the difference in values is less than $\epsilon$, the algorithm stops. If the difference is greater than $\epsilon$, the required variables are reset and expectation and maximizations steps are repeated again. The comparison and resetting of variables to the required values is performed in this module. The values of the probability matrix are stored in two different RAM’s. The first RAM contains the values at the beginning of the iteration and second RAM contains the probability values at the end of the maximization step in the same iteration. If the algorithm ends, the starting locations of the motif are derived from the probability values and stored in a different RAM.
5 PERFORMANCE ANALYSIS

In this chapter, we will evaluate our design by testing it on various data sets. Information about the data sets used for evaluation and the reason for choosing these data sets are given. The design is tested on 12 different data sets. The later sections present the performance of the design on all the data sets, on the basis of statistical parameters explained in Chapter 2. We also compared the design with its software counterpart, MEME in terms of computation time, prediction reliability and accuracy.

5.1 Experimental setup

The experimental setup for our design consists of various modules: random number generation module, matrix formation module, probability matrix module, expectation module, maximization module, comparison module and RAM modules.

The ε value is set to 0.001. This value has been taken from the default parameters of the software tool, MEME [31]. The maximum number of iterations is set to 50. All the modules have been implemented in Verilog HDL. The simulator used was ModelSim PE 6.5 [58]. The simulations were carried on a Cygwin platform running on Windows workstation, with a 1.8 GHz Pentium Dual Core processor consisting of 2GB RAM. The software version of MEME was downloaded from http://meme.nbcr.net/meme4_3_0/meme-download.html. MEME is developed using C and C++.

5.2 Functional verification

Verilog HDL modeling serves as a way to design the logic which can later be synthesized. The advantage offered by the Verilog HDL behavioral description is utilized here to
emulate the algorithm description, without actually worrying about the hardware implementation, thus saving time and money. The functional verification of the design consists of two main steps. First, the functionality of all the modules was verified separately. Test-benches have been developed to verify the design. After all the modules were verified individually, the whole architecture was verified. To verify the whole architecture, a test-bench was developed to verify the basic functionality. Later, the design was tested on the 12 different data sets. The results obtained are presented in later sections. The simulations were repeated on each data set 10 times, with a different random seed for each run.

5.3 Data sets used for performance analysis

To evaluate our design, 12 data sets are selected from data sets created by Tompa et al. [18]. These data sets have been prepared exclusively by the authors for assessing the performance of motif matching computational tools. The data sets created were unbiased to any specific algorithm and hence provide a uniform ground for comparison. Moreover, the data sets are very challenging and prediction of potential transcription factors is very complex on them. The data sets were prepared from the TRANSFAC database [18]. Fifty two data sets were created by Tompa et al., of which 6 are from fly, 26 from human, 12 from mouse and 8 from yeast. The data sets are of three types: Real, Generic and Markov. Real datasets had real promoter sequences, generic datasets were randomly chosen from the same genome, and Markov datasets are generated using a Markov chain of order 3 [18]. Figure 5.1 shows the distribution of data sets based on the number of sequences in each data set.
As shown in the graph above, the data sets contain different numbers of sequences. The number of sequences in each data set varies from 1 to 35. Good data sets should have different numbers of sequences and varying sequence lengths, to cover all the corner cases [18]. Figure 5.2 shows the distribution of sequence lengths for each data set. All the sequences in a data set are of equal length. The figure below shows that the length of the sequences is distributed from 500 bp to 3000 bp. This ensures that the algorithms are tested on varied sequence lengths. Some algorithms perform better on smaller sequence lengths but do not perform well with large sequence lengths. Preparing a data set with varied sequence lengths helps in evaluating an algorithm for these cases.
The datasets are in the FASTA format [18]. FASTA is a text-based format of representing nucleotide sequences. Nucleotides are represented using letters in this format. As the nucleotides are represented in a text-based format, we can easily use scripting languages like Perl or Python to parse the data and get the necessary information.

Out of 52 data sets available, 12 data sets have been chosen for analysis of our design. Data sets of ‘Real’ type were not used, as all the software tools did not perform well on them. Tompa et al. have suggested that future evaluators not use data sets of type ‘Real’ for performance analysis [18]. They also suggested that negative data sets not to be used for performance analysis [18]. 4 negative data sets were present in the 52 data sets. The negative data sets did not have any motifs and hence they have been removed in our experiments. The data sets which did not have motifs in at least 80% of the sequences were also removed, as the Expectation Maximization algorithm assumes there is at least one motif in each sequence. Two of the data sets had only one sequence. These data sets were also removed from our analysis. After removing these categories of data sets, we were left with 20 data sets. Out of 20 data sets, we picked 12 data sets that are of varying lengths and cover all cases. In the 12 data sets picked
for the analysis, 6 data sets are of type ‘Generic’ and the other 6 are of type ‘Markov Chain’. There are 6 of each type of data set (Markov, Generic) used, and they contain 2 data sets from human, 2 from yeast and 2 from mouse. The data sets were chosen such that they are of varying sequence lengths and have different number of sequences. The table below shows the data sets, type of data sets, number of sequences and sequence lengths.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Type</th>
<th>No of sequences</th>
<th>Sequence length</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Generic, Human</td>
<td>18</td>
<td>2000</td>
</tr>
<tr>
<td>2</td>
<td>Generic, Human</td>
<td>9</td>
<td>1000</td>
</tr>
<tr>
<td>3</td>
<td>Markov, Human</td>
<td>11</td>
<td>500</td>
</tr>
<tr>
<td>4</td>
<td>Markov, Human</td>
<td>5</td>
<td>3000</td>
</tr>
<tr>
<td>5</td>
<td>Generic, Mouse</td>
<td>5</td>
<td>500</td>
</tr>
<tr>
<td>6</td>
<td>Generic, Mouse</td>
<td>7</td>
<td>1000</td>
</tr>
<tr>
<td>7</td>
<td>Markov, Mouse</td>
<td>4</td>
<td>500</td>
</tr>
<tr>
<td>8</td>
<td>Markov, Mouse</td>
<td>3</td>
<td>1500</td>
</tr>
<tr>
<td>9</td>
<td>Generic, Yeast</td>
<td>9</td>
<td>1000</td>
</tr>
<tr>
<td>10</td>
<td>Generic, Yeast</td>
<td>8</td>
<td>500</td>
</tr>
<tr>
<td>11</td>
<td>Markov, Yeast</td>
<td>3</td>
<td>500</td>
</tr>
<tr>
<td>12</td>
<td>Markov, Yeast</td>
<td>11</td>
<td>1000</td>
</tr>
</tbody>
</table>

Table 5.1 Table showing data sets with their type, number of sequences, sequence lengths
The graph in figure 5.3 shows the distribution of the data sets based on number of sequences in a data set and sequence length. The 12 data sets which are used in our analysis are marked in red. It can be observed from the graph that the selected data sets cover the range of sequence lengths and data set size.

![Graph showing distribution of data sets based on number of sequences in a data set and sequence lengths]

**Figure 5.3 Distribution of data sets based on number of sequences in a data set and sequence lengths**

### 5.4 Performance analysis

The following factors are taken into consideration for performance analysis:

- Predication accuracy and reliability
- Speedup

A good design should be accurate and reliable and should also produce quick results. Accuracy measures the closeness of the predicted value to the true value. Reliability helps in
estimating the extent to which the results are consistent over repeated tests. To make sure that our design is accurate and reliable, we tested it on the 12 data sets described above. On each of the data sets, the design was simulated for 10 times using different seeds. A total of 120 experiments were conducted on the design. All the simulations were carried out using ModelSim PE 6.5b on Cygwin platform running on Windows OS. The statistical parameters explained in Chapter 2 are calculated for each of these 120 runs. We picked these parameters based on the analysis by Tompa et al. [18] which compares various motif matching tools. The parameters calculated for performance analysis are:

- Nucleotide level sensitivity ($nSn$)
- Nucleotide level positive predictive value ($nPPV$)
- Nucleotide level performance coefficient ($nPC$)
- Nucleotide level correlation coefficient ($nCC$)
- Binding site level sensitivity ($sSn$)
- Binding site level positive predictive value ($sPPV$)
- Binding site level average site performance ($sASP$)

After calculation of the above stated parameters for each run, we also calculated mean, minimum value, maximum value, standard deviation and 95% confidence interval. The results have been tabulated in Tables 5.2 – 5.9. The experiments were also performed on the software tool MEME. These results are given in Appendix A.

We have also compared the computation time of our design with the MEME tool. We have summarized the speedup achieved by our current design compared to MEME in Section 5.4.2.
5.4.1 Prediction accuracy and reliability

In this section, the results of all the seven statistics are summarized. We have calculated statistics for each of the runs, and then calculated the parameter’s minimum value, maximum value, mean, standard deviation and 95% confidence interval for each data set. This analysis was performed on the hardware design and on MEME. For calculation of these parameters, a perl script has been developed. The perl script uses the perl modules Statistics::Point Estimation and Statistics::Descriptive for the calculation of mean, standard deviation and 95% confidence interval. The results for MEME are tabulated in Appendix A.

Nucleotide level sensitivity \((nSn)\)

The nucleotide level sensitivity is calculated for all data sets, and each data set is run for 10 iterations. The values are calculated for our design and MEME. We have calculated minimum value, maximum value, mean, standard deviation and 95% confidence interval for nSn. The values for the hardware design are presented below in Table 5.2.
<table>
<thead>
<tr>
<th>Data Set No.</th>
<th>Minimum value</th>
<th>Maximum value</th>
<th>Mean</th>
<th>Standard deviation</th>
<th>Confidence Interval (95%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.108</td>
<td>0.129</td>
<td>0.116</td>
<td>0.007</td>
<td>(0.111,0.120)</td>
</tr>
<tr>
<td>2</td>
<td>0.109</td>
<td>0.125</td>
<td>0.117</td>
<td>0.005</td>
<td>(0.113,0.120)</td>
</tr>
<tr>
<td>3</td>
<td>0.104</td>
<td>0.128</td>
<td>0.116</td>
<td>0.007</td>
<td>(0.111,0.121)</td>
</tr>
<tr>
<td>4</td>
<td>0.101</td>
<td>0.122</td>
<td>0.112</td>
<td>0.007</td>
<td>(0.106,0.117)</td>
</tr>
<tr>
<td>5</td>
<td>0.103</td>
<td>0.129</td>
<td>0.117</td>
<td>0.009</td>
<td>(0.110,0.124)</td>
</tr>
<tr>
<td>6</td>
<td>0.104</td>
<td>0.130</td>
<td>0.117</td>
<td>0.010</td>
<td>(0.110,0.125)</td>
</tr>
<tr>
<td>7</td>
<td>0.098</td>
<td>0.126</td>
<td>0.118</td>
<td>0.006</td>
<td>(0.113,0.122)</td>
</tr>
<tr>
<td>8</td>
<td>0.117</td>
<td>0.128</td>
<td>0.119</td>
<td>0.007</td>
<td>(0.118,0.121)</td>
</tr>
<tr>
<td>9</td>
<td>0.104</td>
<td>0.126</td>
<td>0.117</td>
<td>0.008</td>
<td>(0.111,0.123)</td>
</tr>
<tr>
<td>10</td>
<td>0.100</td>
<td>0.125</td>
<td>0.111</td>
<td>0.010</td>
<td>(0.104,0.118)</td>
</tr>
<tr>
<td>11</td>
<td>0.101</td>
<td>0.128</td>
<td>0.114</td>
<td>0.009</td>
<td>(0.108,0.121)</td>
</tr>
<tr>
<td>12</td>
<td>0.092</td>
<td>0.128</td>
<td>0.116</td>
<td>0.008</td>
<td>(0.095,0.122)</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td>0.115</td>
<td>0.007</td>
<td></td>
</tr>
</tbody>
</table>

**Table 5.2 Results of nucleotide level sensitivity on the data sets**

From the table, it can be inferred that nucleotide level sensitivity is approximately between 0.10 and 0.13. The average value of nSn for the hardware design is 0.11 and for MEME is 0.10. The average nSn for both the approaches differs by 1%. The nucleotide level sensitivity
is low as we have little knowledge on the underlying biology of regulatory mechanisms, and it is very tough to develop a universal tool with high nucleotide level sensitivity.

**Nucleotide level positive predictive value (nPPV)**

The nucleotide level positive predictive value is calculated for all data sets, and each data set is run for 10 iterations. We have calculated minimum value, maximum value, mean, standard deviation and 95% confidence interval for nPPV. The values for the hardware design are presented below in Table 5.3.

<table>
<thead>
<tr>
<th>Data Set No.</th>
<th>Minimum value</th>
<th>Maximum value</th>
<th>Mean</th>
<th>Standard deviation</th>
<th>Confidence Interval (95%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.180</td>
<td>0.234</td>
<td>0.204</td>
<td>0.020</td>
<td>(0.189,0.219)</td>
</tr>
<tr>
<td>2</td>
<td>0.195</td>
<td>0.244</td>
<td>0.223</td>
<td>0.016</td>
<td>(0.211,0.235)</td>
</tr>
<tr>
<td>3</td>
<td>0.187</td>
<td>0.240</td>
<td>0.216</td>
<td>0.016</td>
<td>(0.204,0.228)</td>
</tr>
<tr>
<td>4</td>
<td>0.181</td>
<td>0.235</td>
<td>0.200</td>
<td>0.021</td>
<td>(0.185,0.215)</td>
</tr>
<tr>
<td>5</td>
<td>0.210</td>
<td>0.251</td>
<td>0.231</td>
<td>0.023</td>
<td>(0.219,0.237)</td>
</tr>
<tr>
<td>6</td>
<td>0.171</td>
<td>0.190</td>
<td>0.185</td>
<td>0.009</td>
<td>(0.175,0.184)</td>
</tr>
<tr>
<td>7</td>
<td>0.192</td>
<td>0.244</td>
<td>0.223</td>
<td>0.019</td>
<td>(0.209,0.237)</td>
</tr>
<tr>
<td>8</td>
<td>0.183</td>
<td>0.236</td>
<td>0.209</td>
<td>0.019</td>
<td>(0.195,0.224)</td>
</tr>
<tr>
<td>9</td>
<td>0.180</td>
<td>0.236</td>
<td>0.214</td>
<td>0.017</td>
<td>(0.201,0.226)</td>
</tr>
<tr>
<td>10</td>
<td>0.200</td>
<td>0.247</td>
<td>0.222</td>
<td>0.016</td>
<td>(0.210,0.237)</td>
</tr>
<tr>
<td>11</td>
<td>0.181</td>
<td>0.246</td>
<td>0.206</td>
<td>0.022</td>
<td>(0.190,0.222)</td>
</tr>
<tr>
<td>12</td>
<td>0.187</td>
<td>0.254</td>
<td>0.215</td>
<td>0.028</td>
<td>(0.192,0.238)</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td>0.212</td>
<td>0.018</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.3 Results of nucleotide level positive predictive value on the data sets
The average value of nPPV for the hardware design is 0.21 and for MEME it is 0.18. This shows that the prediction of positive nucleotides by our design is better than MEME by 3%.

**Nucleotide level performance coefficient (nPC)**

The nucleotide level performance coefficient is calculated for all data sets, and each data set is run for 10 iterations. We have calculated minimum value, maximum value, mean, standard deviation and 95% confidence interval for nPC. The values for the hardware design are presented below in Table 5.4.

<table>
<thead>
<tr>
<th>Data Set No.</th>
<th>Minimum value</th>
<th>Maximum value</th>
<th>Mean</th>
<th>Standard deviation</th>
<th>Confidence Interval (95%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.043</td>
<td>0.069</td>
<td>0.056</td>
<td>0.009</td>
<td>(0.049,0.062)</td>
</tr>
<tr>
<td>2</td>
<td>0.042</td>
<td>0.049</td>
<td>0.045</td>
<td>0.002</td>
<td>(0.043,0.047)</td>
</tr>
<tr>
<td>3</td>
<td>0.040</td>
<td>0.063</td>
<td>0.053</td>
<td>0.007</td>
<td>(0.047,0.058)</td>
</tr>
<tr>
<td>4</td>
<td>0.041</td>
<td>0.053</td>
<td>0.046</td>
<td>0.003</td>
<td>(0.043,0.049)</td>
</tr>
<tr>
<td>5</td>
<td>0.040</td>
<td>0.069</td>
<td>0.052</td>
<td>0.010</td>
<td>(0.044,0.059)</td>
</tr>
<tr>
<td>6</td>
<td>0.040</td>
<td>0.049</td>
<td>0.045</td>
<td>0.003</td>
<td>(0.042,0.047)</td>
</tr>
<tr>
<td>7</td>
<td>0.043</td>
<td>0.067</td>
<td>0.053</td>
<td>0.009</td>
<td>(0.046,0.060)</td>
</tr>
<tr>
<td>8</td>
<td>0.061</td>
<td>0.069</td>
<td>0.064</td>
<td>0.002</td>
<td>(0.062,0.067)</td>
</tr>
<tr>
<td>9</td>
<td>0.040</td>
<td>0.049</td>
<td>0.043</td>
<td>0.0029</td>
<td>(0.041,0.046)</td>
</tr>
<tr>
<td>10</td>
<td>0.042</td>
<td>0.056</td>
<td>0.048</td>
<td>0.004</td>
<td>(0.045,0.051)</td>
</tr>
<tr>
<td>11</td>
<td>0.041</td>
<td>0.048</td>
<td>0.045</td>
<td>0.002</td>
<td>(0.043,0.046)</td>
</tr>
<tr>
<td>12</td>
<td>0.060</td>
<td>0.068</td>
<td>0.065</td>
<td>0.002</td>
<td>(0.063,0.066)</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td>0.051</td>
<td>0.004</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.4 Results of nucleotide level performance coefficient on the data sets
From Table 5.4, it can be inferred that the average value of nPC for our design is 0.051. The average value of nPC for MEME is 0.06. Hence, our design can be considered equivalent to MEME with an average difference of 0.9%.

**Nucleotide level correlation coefficient (nCC)**

The nucleotide level correlation coefficient is calculated for all data sets, and each data set is run for 10 iterations. The values for the hardware design are presented in Table 5.5.

<table>
<thead>
<tr>
<th>Data Set No.</th>
<th>Minimum value</th>
<th>Maximum value</th>
<th>Mean</th>
<th>Standard deviation</th>
<th>Confidence Interval (95%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.075</td>
<td>0.107</td>
<td>0.091</td>
<td>0.011</td>
<td>(0.083,0.100)</td>
</tr>
<tr>
<td>2</td>
<td>0.061</td>
<td>0.077</td>
<td>0.067</td>
<td>0.005</td>
<td>(0.063,0.071)</td>
</tr>
<tr>
<td>3</td>
<td>0.062</td>
<td>0.078</td>
<td>0.074</td>
<td>0.005</td>
<td>(0.070,0.078)</td>
</tr>
<tr>
<td>4</td>
<td>0.072</td>
<td>0.106</td>
<td>0.089</td>
<td>0.011</td>
<td>(0.080,0.097)</td>
</tr>
<tr>
<td>5</td>
<td>0.061</td>
<td>0.075</td>
<td>0.068</td>
<td>0.004</td>
<td>(0.064,0.071)</td>
</tr>
<tr>
<td>6</td>
<td>0.070</td>
<td>0.107</td>
<td>0.086</td>
<td>0.012</td>
<td>(0.076,0.095)</td>
</tr>
<tr>
<td>7</td>
<td>0.070</td>
<td>0.089</td>
<td>0.078</td>
<td>0.006</td>
<td>(0.073,0.083)</td>
</tr>
<tr>
<td>8</td>
<td>0.064</td>
<td>0.078</td>
<td>0.071</td>
<td>0.005</td>
<td>(0.067,0.074)</td>
</tr>
<tr>
<td>9</td>
<td>0.064</td>
<td>0.078</td>
<td>0.072</td>
<td>0.004</td>
<td>(0.069,0.075)</td>
</tr>
<tr>
<td>10</td>
<td>0.070</td>
<td>0.088</td>
<td>0.079</td>
<td>0.006</td>
<td>(0.074,0.084)</td>
</tr>
<tr>
<td>11</td>
<td>0.073</td>
<td>0.109</td>
<td>0.094</td>
<td>0.010</td>
<td>(0.087,0.102)</td>
</tr>
<tr>
<td>12</td>
<td>0.063</td>
<td>0.079</td>
<td>0.071</td>
<td>0.005</td>
<td>(0.067,0.075)</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td>0.078</td>
<td>0.007</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.5 Results of nucleotide level correlation coefficient value on the data sets

73
We can observe that average nCC is 0.078 for hardware design. For MEME, nCC is 0.08. The value of nCC can fall in the range of -1 to +1. Here, -1 indicates perfect anti-correlation and +1 indicates perfect correlation. As our values are just above 0, there is very little correlation. The same performance was observed for MEME with an average difference of 0.2%

**Binding site level sensitivity (sSn)**

The binding site level sensitivity value is calculated for all data sets, and each data set is run for 10 iterations. The values for hardware design are presented below in Table 5.6.

<table>
<thead>
<tr>
<th>Data Set No.</th>
<th>Minimum value</th>
<th>Maximum value</th>
<th>Mean</th>
<th>Standard deviation</th>
<th>Confidence Interval (95%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.100</td>
<td>0.127</td>
<td>0.109</td>
<td>0.010</td>
<td>(0.102,0.117)</td>
</tr>
<tr>
<td></td>
<td>0.083</td>
<td>0.099</td>
<td>0.092</td>
<td>0.006</td>
<td>(0.087,0.096)</td>
</tr>
<tr>
<td>3</td>
<td>0.090</td>
<td>0.109</td>
<td>0.101</td>
<td>0.007</td>
<td>(0.096,0.106)</td>
</tr>
<tr>
<td>4</td>
<td>0.106</td>
<td>0.128</td>
<td>0.118</td>
<td>0.008</td>
<td>(0.112,0.124)</td>
</tr>
<tr>
<td>5</td>
<td>0.080</td>
<td>0.089</td>
<td>0.085</td>
<td>0.003</td>
<td>(0.082,0.087)</td>
</tr>
<tr>
<td>6</td>
<td>0.092</td>
<td>0.107</td>
<td>0.101</td>
<td>0.005</td>
<td>(0.097,0.105)</td>
</tr>
<tr>
<td>7</td>
<td>0.082</td>
<td>0.087</td>
<td>0.084</td>
<td>0.002</td>
<td>(0.083,0.086)</td>
</tr>
<tr>
<td>8</td>
<td>0.100</td>
<td>0.129</td>
<td>0.115</td>
<td>0.010</td>
<td>(0.107,0.123)</td>
</tr>
<tr>
<td>9</td>
<td>0.110</td>
<td>0.117</td>
<td>0.114</td>
<td>0.002</td>
<td>(0.112,0.116)</td>
</tr>
<tr>
<td>10</td>
<td>0.082</td>
<td>0.099</td>
<td>0.090</td>
<td>0.005</td>
<td>(0.086,0.094)</td>
</tr>
<tr>
<td>11</td>
<td>0.110</td>
<td>0.119</td>
<td>0.114</td>
<td>0.003</td>
<td>(0.111,0.116)</td>
</tr>
<tr>
<td>12</td>
<td>0.101</td>
<td>0.128</td>
<td>0.115</td>
<td>0.009</td>
<td>(0.109,0.122)</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td>0.103</td>
<td>0.005</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.6 Results of binding site level sensitivity on the data sets
sSn helps in estimating the fraction of known binding sites that are predicted. From Table 5.6, it can be observed that site sensitivity varies from 0.08 to 0.13. The average value of sSn for hardware design is 0.103. For MEME, the average sSn is 0.105. The performance of both the designs is similar with an average difference of 0.02%. The low value of sSn is because of the incomplete knowledge of regulatory mechanisms and the complexity of DNA binding sites.

**Binding site level positive predictive value (sPPV)**

The binding site level positive predictive value is calculated for all data sets. The values for hardware design are presented below in Table 5.7.

<table>
<thead>
<tr>
<th>Data Set No.</th>
<th>Minimum value</th>
<th>Maximum value</th>
<th>Mean</th>
<th>Standard deviation</th>
<th>Confidence Interval (95%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.154</td>
<td>0.240</td>
<td>0.203</td>
<td>0.032</td>
<td>(0.179,0.227)</td>
</tr>
<tr>
<td>2</td>
<td>0.165</td>
<td>0.205</td>
<td>0.178</td>
<td>0.015</td>
<td>(0.167,0.189)</td>
</tr>
<tr>
<td>3</td>
<td>0.183</td>
<td>0.205</td>
<td>0.193</td>
<td>0.007</td>
<td>(0.187,0.199)</td>
</tr>
<tr>
<td>4</td>
<td>0.162</td>
<td>0.198</td>
<td>0.185</td>
<td>0.012</td>
<td>(0.176,0.194)</td>
</tr>
<tr>
<td>5</td>
<td>0.142</td>
<td>0.240</td>
<td>0.193</td>
<td>0.032</td>
<td>(0.170,0.217)</td>
</tr>
<tr>
<td>6</td>
<td>0.161</td>
<td>0.188</td>
<td>0.176</td>
<td>0.009</td>
<td>(0.169,0.182)</td>
</tr>
<tr>
<td>7</td>
<td>0.148</td>
<td>0.243</td>
<td>0.191</td>
<td>0.033</td>
<td>(0.167,0.215)</td>
</tr>
<tr>
<td>8</td>
<td>0.201</td>
<td>0.229</td>
<td>0.215</td>
<td>0.009</td>
<td>(0.208,0.222)</td>
</tr>
<tr>
<td>9</td>
<td>0.188</td>
<td>0.203</td>
<td>0.194</td>
<td>0.005</td>
<td>(0.190,0.198)</td>
</tr>
<tr>
<td>10</td>
<td>0.163</td>
<td>0.186</td>
<td>0.174</td>
<td>0.008</td>
<td>(0.168,0.180)</td>
</tr>
<tr>
<td>11</td>
<td>0.200</td>
<td>0.228</td>
<td>0.218</td>
<td>0.008</td>
<td>(0.212,0.224)</td>
</tr>
<tr>
<td>12</td>
<td>0.152</td>
<td>0.241</td>
<td>0.200</td>
<td>0.025</td>
<td>(0.182,0.218)</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td>0.193</td>
<td>0.016</td>
<td></td>
</tr>
</tbody>
</table>

*Table 5.7 Results of binding site level positive predictive value on the data sets*
From Table 5.7, we can observe that the average sPPV is 0.193. The average sPPV for MEME was observed to be 0.183. sPPV provides the fraction of predicted sites that are known. The hardware design predicts a good portion of known sites and differs from MEME by 1% on average.

**Binding site level average site performance (sASP)**

The binding site level average site performance was calculated for all data sets. Average site performance is the average of site level sensitivity (sSn) and site level positive predictive value (sPPV). To calculate minimum value of sASP, the average of minimum sSn and minimum sPPV was computed. Similarly, the maximum values of sASP were calculated. The values are presented below in Table 5.8.

<table>
<thead>
<tr>
<th>Data Set No.</th>
<th>Minimum value</th>
<th>Maximum value</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.127</td>
<td>0.1835</td>
<td>0.15525</td>
</tr>
<tr>
<td>2</td>
<td>0.124</td>
<td>0.152</td>
<td>0.138</td>
</tr>
<tr>
<td>3</td>
<td>0.136</td>
<td>0.157</td>
<td>0.1465</td>
</tr>
<tr>
<td>4</td>
<td>0.134</td>
<td>0.163</td>
<td>0.1485</td>
</tr>
<tr>
<td>5</td>
<td>0.111</td>
<td>0.1645</td>
<td>0.13775</td>
</tr>
<tr>
<td>6</td>
<td>0.126</td>
<td>0.1475</td>
<td>0.13675</td>
</tr>
<tr>
<td>7</td>
<td>0.115</td>
<td>0.165</td>
<td>0.14</td>
</tr>
<tr>
<td>8</td>
<td>0.150</td>
<td>0.179</td>
<td>0.1645</td>
</tr>
<tr>
<td>9</td>
<td>0.149</td>
<td>0.16</td>
<td>0.1545</td>
</tr>
<tr>
<td>10</td>
<td>0.122</td>
<td>0.1425</td>
<td>0.13225</td>
</tr>
<tr>
<td>11</td>
<td>0.155</td>
<td>0.1735</td>
<td>0.16425</td>
</tr>
<tr>
<td>12</td>
<td>0.126</td>
<td>0.1845</td>
<td>0.15525</td>
</tr>
</tbody>
</table>

Table 5.8 Results of binding site level average site performance on the data sets
From Table 5.8, we can observe that the average sASP for hardware design is 0.147. Average sASP for MEME is 0.133. The average site performance of hardware design is better than MEME by 2.4%.

The minimum, average and maximum values of all the statistical parameters for hardware design are represented in graph shown below in Figure 5.4.

![Graph showing minimum, average and maximum values of parameters](image)

**Figure 5.4 Graph showing minimum, average and maximum values of parameters**

Table 5.9 summarizes the percentage average difference observed in statistical parameters for the hardware design and MEME.
Statistical parameter | Average difference between MEME and hardware design (%)
--- | ---
nSn | 1
nPPV | 3
nPC | 0.9
nCC | 0.2
sSn | 0.02
sPPV | 1
sASP | 2.4

Table 5.9 Difference in the statistical parameters for hardware design and MEME

After calculation of statistical parameters, it can be observed that the performance of our design is comparable to the performance of MEME in terms of prediction accuracy and reliability. The average difference between both the approaches is at most 3%. We have used LFSR for generation of random numbers. The performance of the hardware design can be improved by using a multiple LFSR or multiple cellular automata implementation for random number generation. The values of statistical parameters are not very high for both the approaches due to the complexity of biological systems. Moreover, the knowledge on transcription factors is still incomplete. We rely on the information of true binding sites from the data available in TRANSFAC database. Tompa et al. [18] have stated that any database is prone to errors and hence cannot be considered as a standard for evaluation of motif matching tools. The average length of the binding sites stated in TRANSFAC database is long, while the real binding sites are normally short.
5.4.2 Speedup

Comparison with MEME

Speedup, with reference to our analysis, can be defined as the ratio of execution time of the software tool MEME to the execution time of our design. To test the execution time on hardware, the Verilog RTL code was synthesized and ported on to Stratix family. We have chosen Stratix as it has more memory elements than Cyclone. Altera Quartus II was used for synthesis, and for porting the design onto these devices. Quartus II is a software tool provided by Altera for analysis and synthesis of HDL designs. It enables the developer to compile their designs, perform timing analysis and examine RTL diagrams.

Table 5.10 shows the maximum available Arithmetic Look up Tables (ALUT), User Input / Output pins (I/O pins) and Memory bits in Altera devices.

<table>
<thead>
<tr>
<th>Devices</th>
<th>ALUT’s</th>
<th>User I/O Pins</th>
<th>Memory bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stratix II</td>
<td>12480</td>
<td>343</td>
<td>41938</td>
</tr>
<tr>
<td>Stratix II GX</td>
<td>27104</td>
<td>386</td>
<td>1369728</td>
</tr>
<tr>
<td>Stratix III</td>
<td>38000</td>
<td>296</td>
<td>5455872</td>
</tr>
</tbody>
</table>

Table 5.10 Number of devices available in Stratix II, Stratix II GX and Stratix III

Stratix II EP2S15F484C3 and Stratix III EP3SE50F484C2 devices have been used for testing the design. Using Stratix II, we have observed that the maximum length of sequence which can be handled by the tool is 50. The maximum length of sequence which can be handled by Stratix III is 75. Though Stratix III has more memory elements and ALUT’s, we could not use it for very big designs due to insufficient RAM memory of the processor. This problem has also been observed on a machine with 4 GB RAM. The minimum sequence length of the data sets
created by Tompa et al. [8] is 500. So, we needed to create 3 data sets of length 50 and 3 data sets of length 75 from their data sets for testing purposes.

The design was tested on these 6 data sets. The minimum frequency at which the design can operate without any setup and hold violations for a sequence of length 50 was observed to be 72.44 MHz (time period = 13.804 ns). On a sequence of length 75, the clock frequency was observed to be 58.6 MHz (17.064 ns). These values are obtained from the Timing Analysis report generated by the tool at the end of the simulation. We calculated the time taken for each simulation based on the maximum time period and the number of clock cycles. The same data sets were run on the MEME software tool. It has been observed that the average speedup is 122. The graph below shows the individual speedup for each data set.

![Graph showing the speedup over MEME for 6 data sets](image)

**Figure 5.5 Graph showing the speedup over MEME for 6 data sets**

From the graph, it can be observed that the hardware implementation of the EM algorithm is faster than the software tool MEME. Table 5.11 shows the estimated area on Stratix II and Stratix III in terms of LUTs and number of pins used.
Comparison with hardware approaches

As mentioned in Chapter 2, Sandve et al. [2] developed MAMA which uses specialized hardware PMC. The authors tested the performance of MAMA on 5 data sets of human promoter regions. We could not compare performance of our design with MAMA due to the inavailability of the data sets used by Sandve et al [2]. As our design is implemented completely on hardware, we can expect to have significant speedup than MAMA. Moreover, from the results in Figure 5.5, it can be observed that speedup do not vary with sequence length. So, the design can be extended to larger sequence lengths also. Table 5.12 shows the number of logic cells, user I/O pins and embedded block RAM in Xilinx devices. Xilinx platform can be used to port designs with longer sequence lengths. Additionally, running the simulations on a processor with larger RAM can also help in porting large sequences.
6 CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

The thesis presents the hardware design of the Expectation Maximization algorithm for motif finding. The entire design has been divided into submodules, and all the modules are developed using Verilog HDL. As the entire design has been implemented in Verilog HDL, it can be used with any of the HDL simulators or synthesis tools available in the market. The design can also be implemented in FPGA’s provided by companies such as Altera and Xilinx. It could also be implemented on custom hardware. An architecture containing all the modules has also been presented.

Each module was initially verified for proper functionality by subjecting it to a number of test cases. Later, the entire design was functionally verified. After functional verification, performance analysis was carried out on the design by testing it on 12 data sets. The data sets used for performance analysis have varying sequence lengths and different numbers of sequences. The performance of the design is measured in terms of its reliability, accuracy and speedup. The design was observed to perform well on all the data sets. The results obtained are almost equal to the results of commonly used software tools in terms of accuracy and reliability, with a maximum difference of 3%. Our design was observed to perform on average 122 times faster than the software tool MEME.

6.2 Future work

This work only deals with motif finding in DNA sequences. It can be extended in the future for motif finding in protein sequences. The current algorithms for motif finding in protein
sequences are very computationally intensive and have high execution times. An efficient hardware design for this problem would be very useful.

The design is currently implemented on an Altera FPGA. During synthesis and placement of the design, the default set of optimizations has been selected in Quartus II. The computation time could be further reduced by using proper logic optimizations. If critical paths are optimized using optimization techniques like factorization and equation flattening, the design would operate at a higher frequency.

In this work, we have used the Expectation Maximization algorithm for motif finding. Based on the analysis carried out by Hu et al. [38], it has been observed that efficient motif finding can be achieved by combining two or more algorithms. This work can be extended to use other algorithms along with Expectation Maximization to achieve better results.

We have used LFSR for random number implementation on hardware. Using multiple LFSR’s or multiple cellular automata can help in further increasing the performance of the design.
BIBLIOGRAPHY


[56] LFSR Diagram.


APPENDIX A

Statistical parameters for the software tool MEME

In this appendix, the results of all the seven statistics observed for the software tool MEME are summarized. The results provided in the appendix are referred to in Section 5.4.

The MEME software tool is run on 12 data sets. In the 12 data sets picked for the analysis, 6 data sets are of type ‘Generic’ and the other 6 are of type ‘Markov Chain’. Each of the 6 data sets (Markov, Generic) contain 2 data sets from human, 2 from yeast and 2 from mouse. On each of the data sets, the tool was executed 10 times. A total of 120 experiments were conducted on the design. All the simulations were carried out on a Cygwin platform running on Windows OS, with a 1.8 GHz Pentium dual core processor consisting of 2GB RAM. The main aim of this analysis is to compare the hardware design with MEME in terms of prediction accuracy and reliability. The statistical parameters as explained in Chapter 2 are calculated for each of these 120 runs. We picked these parameters based on the analysis of Tompa et al. [18] during comparison of various motif matching tools.

After calculation of the statistical parameters for each run, we also calculate mean, minimum value, maximum value, standard deviation and 95% confidence interval. The results for the software tool MEME have been tabulated below. The results for the hardware design are in Section 5.4 along with comparisons of the overall average values for MEME with the hardware design.
### Nucleotide level sensitivity (MEME)

<table>
<thead>
<tr>
<th>Data Set No.</th>
<th>Minimum value</th>
<th>Maximum value</th>
<th>Mean</th>
<th>Standard deviation</th>
<th>Confidence Interval (95%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.090</td>
<td>0.109</td>
<td>0.101</td>
<td>0.005</td>
<td>(0.096,0.105)</td>
</tr>
<tr>
<td>2</td>
<td>0.090</td>
<td>0.109</td>
<td>0.099</td>
<td>0.006</td>
<td>(0.094,0.103)</td>
</tr>
<tr>
<td>3</td>
<td>0.100</td>
<td>0.109</td>
<td>0.106</td>
<td>0.002</td>
<td>(0.104,0.108)</td>
</tr>
<tr>
<td>4</td>
<td>0.092</td>
<td>0.109</td>
<td>0.100</td>
<td>0.005</td>
<td>(0.096,0.104)</td>
</tr>
<tr>
<td>5</td>
<td>0.090</td>
<td>0.099</td>
<td>0.095</td>
<td>0.003</td>
<td>(0.093,0.097)</td>
</tr>
<tr>
<td>6</td>
<td>0.101</td>
<td>0.107</td>
<td>0.104</td>
<td>0.001</td>
<td>(0.103,0.105)</td>
</tr>
<tr>
<td>7</td>
<td>0.091</td>
<td>0.105</td>
<td>0.097</td>
<td>0.004</td>
<td>(0.094,0.100)</td>
</tr>
<tr>
<td>8</td>
<td>0.100</td>
<td>0.109</td>
<td>0.104</td>
<td>0.003</td>
<td>(0.102,0.107)</td>
</tr>
<tr>
<td>9</td>
<td>0.090</td>
<td>0.099</td>
<td>0.094</td>
<td>0.003</td>
<td>(0.092,0.096)</td>
</tr>
<tr>
<td>10</td>
<td>0.100</td>
<td>0.108</td>
<td>0.104</td>
<td>0.003</td>
<td>(0.102,0.106)</td>
</tr>
<tr>
<td>11</td>
<td>0.090</td>
<td>0.109</td>
<td>0.100</td>
<td>0.006</td>
<td>(0.095,0.105)</td>
</tr>
<tr>
<td>12</td>
<td>0.090</td>
<td>0.099</td>
<td>0.095</td>
<td>0.003</td>
<td>(0.093,0.098)</td>
</tr>
</tbody>
</table>
### Nucleotide level positive predictive value (MEME)

<table>
<thead>
<tr>
<th>Data Set No.</th>
<th>Minimum value</th>
<th>Maximum value</th>
<th>Mean</th>
<th>Standard deviation</th>
<th>Confidence Interval (95%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.167</td>
<td>0.198</td>
<td>0.181</td>
<td>0.012</td>
<td>(0.172, 0.190)</td>
</tr>
<tr>
<td>2</td>
<td>0.160</td>
<td>0.178</td>
<td>0.168</td>
<td>0.005</td>
<td>(0.164, 0.172)</td>
</tr>
<tr>
<td>3</td>
<td>0.164</td>
<td>0.179</td>
<td>0.170</td>
<td>0.005</td>
<td>(0.167, 0.174)</td>
</tr>
<tr>
<td>4</td>
<td>0.180</td>
<td>0.199</td>
<td>0.189</td>
<td>0.007</td>
<td>(0.183, 0.194)</td>
</tr>
<tr>
<td>5</td>
<td>0.167</td>
<td>0.193</td>
<td>0.179</td>
<td>0.009</td>
<td>(0.172, 0.186)</td>
</tr>
<tr>
<td>6</td>
<td>0.182</td>
<td>0.197</td>
<td>0.188</td>
<td>0.005</td>
<td>(0.184, 0.192)</td>
</tr>
<tr>
<td>7</td>
<td>0.165</td>
<td>0.194</td>
<td>0.177</td>
<td>0.010</td>
<td>(0.169, 0.184)</td>
</tr>
<tr>
<td>8</td>
<td>0.183</td>
<td>0.198</td>
<td>0.192</td>
<td>0.005</td>
<td>(0.188, 0.197)</td>
</tr>
<tr>
<td>9</td>
<td>0.162</td>
<td>0.193</td>
<td>0.176</td>
<td>0.009</td>
<td>(0.169, 0.182)</td>
</tr>
<tr>
<td>10</td>
<td>0.160</td>
<td>0.191</td>
<td>0.175</td>
<td>0.012</td>
<td>(0.166, 0.184)</td>
</tr>
<tr>
<td>11</td>
<td>0.161</td>
<td>0.174</td>
<td>0.169</td>
<td>0.003</td>
<td>(0.166, 0.172)</td>
</tr>
<tr>
<td>12</td>
<td>0.161</td>
<td>0.190</td>
<td>0.176</td>
<td>0.011</td>
<td>(0.168, 0.184)</td>
</tr>
<tr>
<td>Data Set No.</td>
<td>Minimum value</td>
<td>Maximum value</td>
<td>Mean</td>
<td>Standard deviation</td>
<td>Confidence Interval (95%)</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------</td>
<td>---------------</td>
<td>--------</td>
<td>--------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>1</td>
<td>0.05</td>
<td>0.069</td>
<td>0.061</td>
<td>0.006</td>
<td>(0.057,0.066)</td>
</tr>
<tr>
<td>2</td>
<td>0.051</td>
<td>0.059</td>
<td>0.055</td>
<td>0.003</td>
<td>(0.053,0.057)</td>
</tr>
<tr>
<td>3</td>
<td>0.053</td>
<td>0.066</td>
<td>0.059</td>
<td>0.004</td>
<td>(0.056,0.062)</td>
</tr>
<tr>
<td>4</td>
<td>0.059</td>
<td>0.068</td>
<td>0.063</td>
<td>0.003</td>
<td>(0.061,0.065)</td>
</tr>
<tr>
<td>5</td>
<td>0.054</td>
<td>0.069</td>
<td>0.059</td>
<td>0.008</td>
<td>(0.053,0.064)</td>
</tr>
<tr>
<td>6</td>
<td>0.063</td>
<td>0.078</td>
<td>0.067</td>
<td>0.002</td>
<td>(0.065,0.068)</td>
</tr>
<tr>
<td>7</td>
<td>0.054</td>
<td>0.072</td>
<td>0.063</td>
<td>0.005</td>
<td>(0.059,0.067)</td>
</tr>
<tr>
<td>8</td>
<td>0.051</td>
<td>0.067</td>
<td>0.058</td>
<td>0.006</td>
<td>(0.054,0.063)</td>
</tr>
<tr>
<td>9</td>
<td>0.051</td>
<td>0.057</td>
<td>0.054</td>
<td>0.002</td>
<td>(0.052,0.055)</td>
</tr>
<tr>
<td>10</td>
<td>0.060</td>
<td>0.075</td>
<td>0.0654</td>
<td>0.002</td>
<td>(0.063,0.067)</td>
</tr>
<tr>
<td>11</td>
<td>0.056</td>
<td>0.067</td>
<td>0.062</td>
<td>0.003</td>
<td>(0.059,0.064)</td>
</tr>
<tr>
<td>12</td>
<td>0.050</td>
<td>0.059</td>
<td>0.055</td>
<td>0.003</td>
<td>(0.052,0.058)</td>
</tr>
</tbody>
</table>
### Nucleotide level correlation coefficient (MEME)

<table>
<thead>
<tr>
<th>Data Set No.</th>
<th>Minimum value</th>
<th>Maximum value</th>
<th>Mean</th>
<th>Standard deviation</th>
<th>Confidence Interval (95%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.073</td>
<td>0.087</td>
<td>0.082</td>
<td>0.005</td>
<td>(0.078, 0.086)</td>
</tr>
<tr>
<td>2</td>
<td>0.069</td>
<td>0.086</td>
<td>0.080</td>
<td>0.004</td>
<td>(0.076, 0.083)</td>
</tr>
<tr>
<td>3</td>
<td>0.068</td>
<td>0.079</td>
<td>0.075</td>
<td>0.002</td>
<td>(0.072, 0.077)</td>
</tr>
<tr>
<td>4</td>
<td>0.081</td>
<td>0.090</td>
<td>0.085</td>
<td>0.002</td>
<td>(0.083, 0.087)</td>
</tr>
<tr>
<td>5</td>
<td>0.071</td>
<td>0.085</td>
<td>0.079</td>
<td>0.004</td>
<td>(0.076, 0.082)</td>
</tr>
<tr>
<td>6</td>
<td>0.080</td>
<td>0.093</td>
<td>0.085</td>
<td>0.002</td>
<td>(0.084, 0.087)</td>
</tr>
<tr>
<td>7</td>
<td>0.070</td>
<td>0.083</td>
<td>0.077</td>
<td>0.004</td>
<td>(0.074, 0.080)</td>
</tr>
<tr>
<td>8</td>
<td>0.072</td>
<td>0.078</td>
<td>0.074</td>
<td>0.002</td>
<td>(0.072, 0.076)</td>
</tr>
<tr>
<td>9</td>
<td>0.071</td>
<td>0.081</td>
<td>0.075</td>
<td>0.002</td>
<td>(0.073, 0.077)</td>
</tr>
<tr>
<td>10</td>
<td>0.070</td>
<td>0.089</td>
<td>0.077</td>
<td>0.006</td>
<td>(0.073, 0.082)</td>
</tr>
<tr>
<td>11</td>
<td>0.082</td>
<td>0.089</td>
<td>0.084</td>
<td>0.002</td>
<td>(0.082, 0.086)</td>
</tr>
<tr>
<td>12</td>
<td>0.073</td>
<td>0.090</td>
<td>0.080</td>
<td>0.005</td>
<td>(0.076, 0.085)</td>
</tr>
</tbody>
</table>
## Binding site level sensitivity (MEME)

<table>
<thead>
<tr>
<th>Data Set No.</th>
<th>Minimum value</th>
<th>Maximum value</th>
<th>Mean</th>
<th>Standard deviation</th>
<th>Confidence Interval (95%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.092</td>
<td>0.115</td>
<td>0.102</td>
<td>0.007</td>
<td>(0.097,0.108)</td>
</tr>
<tr>
<td>2</td>
<td>0.093</td>
<td>0.116</td>
<td>0.101</td>
<td>0.007</td>
<td>(0.096,0.107)</td>
</tr>
<tr>
<td>3</td>
<td>0.096</td>
<td>0.114</td>
<td>0.107</td>
<td>0.007</td>
<td>(0.102,0.112)</td>
</tr>
<tr>
<td>4</td>
<td>0.111</td>
<td>0.122</td>
<td>0.118</td>
<td>0.004</td>
<td>(0.114,0.121)</td>
</tr>
<tr>
<td>5</td>
<td>0.091</td>
<td>0.114</td>
<td>0.104</td>
<td>0.009</td>
<td>(0.098,0.111)</td>
</tr>
<tr>
<td>6</td>
<td>0.094</td>
<td>0.108</td>
<td>0.101</td>
<td>0.005</td>
<td>(0.097,0.105)</td>
</tr>
<tr>
<td>7</td>
<td>0.110</td>
<td>0.120</td>
<td>0.117</td>
<td>0.003</td>
<td>(0.115,0.119)</td>
</tr>
<tr>
<td>8</td>
<td>0.091</td>
<td>0.117</td>
<td>0.105</td>
<td>0.007</td>
<td>(0.100,0.115)</td>
</tr>
<tr>
<td>9</td>
<td>0.093</td>
<td>0.111</td>
<td>0.101</td>
<td>0.006</td>
<td>(0.096,0.105)</td>
</tr>
<tr>
<td>10</td>
<td>0.100</td>
<td>0.111</td>
<td>0.107</td>
<td>0.004</td>
<td>(0.104,0.110)</td>
</tr>
<tr>
<td>11</td>
<td>0.092</td>
<td>0.116</td>
<td>0.099</td>
<td>0.008</td>
<td>(0.093,0.105)</td>
</tr>
<tr>
<td>12</td>
<td>0.097</td>
<td>0.118</td>
<td>0.108</td>
<td>0.006</td>
<td>(0.104,0.113)</td>
</tr>
</tbody>
</table>
## Binding site level positive predictive value (MEME)

<table>
<thead>
<tr>
<th>Data Set No.</th>
<th>Minimum value</th>
<th>Maximum value</th>
<th>Mean</th>
<th>Standard deviation</th>
<th>Confidence Interval (95%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.178</td>
<td>0.189</td>
<td>0.184</td>
<td>0.004</td>
<td>(0.181, 0.187)</td>
</tr>
<tr>
<td>2</td>
<td>0.181</td>
<td>0.190</td>
<td>0.187</td>
<td>0.003</td>
<td>(0.184, 0.189)</td>
</tr>
<tr>
<td>3</td>
<td>0.171</td>
<td>0.196</td>
<td>0.178</td>
<td>0.005</td>
<td>(0.184, 0.182)</td>
</tr>
<tr>
<td>4</td>
<td>0.183</td>
<td>0.191</td>
<td>0.185</td>
<td>0.002</td>
<td>(0.183, 0.187)</td>
</tr>
<tr>
<td>5</td>
<td>0.169</td>
<td>0.185</td>
<td>0.177</td>
<td>0.006</td>
<td>(0.182, 0.181)</td>
</tr>
<tr>
<td>6</td>
<td>0.181</td>
<td>0.186</td>
<td>0.184</td>
<td>0.001</td>
<td>(0.183, 0.185)</td>
</tr>
<tr>
<td>7</td>
<td>0.179</td>
<td>0.192</td>
<td>0.185</td>
<td>0.004</td>
<td>(0.181, 0.188)</td>
</tr>
<tr>
<td>8</td>
<td>0.182</td>
<td>0.194</td>
<td>0.187</td>
<td>0.003</td>
<td>(0.184, 0.190)</td>
</tr>
<tr>
<td>9</td>
<td>0.181</td>
<td>0.193</td>
<td>0.187</td>
<td>0.004</td>
<td>(0.184, 0.190)</td>
</tr>
<tr>
<td>10</td>
<td>0.180</td>
<td>0.191</td>
<td>0.184</td>
<td>0.003</td>
<td>(0.181, 0.187)</td>
</tr>
<tr>
<td>11</td>
<td>0.183</td>
<td>0.190</td>
<td>0.186</td>
<td>0.002</td>
<td>(0.184, 0.188)</td>
</tr>
<tr>
<td>12</td>
<td>0.171</td>
<td>0.187</td>
<td>0.179</td>
<td>0.005</td>
<td>(0.174, 0.183)</td>
</tr>
</tbody>
</table>
## Binding site level average site performance (MEME)

<table>
<thead>
<tr>
<th>Data Set No.</th>
<th>Minimum value</th>
<th>Maximum value</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.113</td>
<td>0.135</td>
<td>0.124</td>
</tr>
<tr>
<td>2</td>
<td>0.115</td>
<td>0.137</td>
<td>0.126</td>
</tr>
<tr>
<td>3</td>
<td>0.114</td>
<td>0.133</td>
<td>0.124</td>
</tr>
<tr>
<td>4</td>
<td>0.129</td>
<td>0.147</td>
<td>0.138</td>
</tr>
<tr>
<td>5</td>
<td>0.110</td>
<td>0.130</td>
<td>0.120</td>
</tr>
<tr>
<td>6</td>
<td>0.115</td>
<td>0.137</td>
<td>0.126</td>
</tr>
<tr>
<td>7</td>
<td>0.127</td>
<td>0.144</td>
<td>0.135</td>
</tr>
<tr>
<td>8</td>
<td>0.113</td>
<td>0.136</td>
<td>0.125</td>
</tr>
<tr>
<td>9</td>
<td>0.115</td>
<td>0.137</td>
<td>0.126</td>
</tr>
<tr>
<td>10</td>
<td>0.120</td>
<td>0.14</td>
<td>0.130</td>
</tr>
<tr>
<td>11</td>
<td>0.114</td>
<td>0.137</td>
<td>0.126</td>
</tr>
<tr>
<td>12</td>
<td>0.115</td>
<td>0.134</td>
<td>0.124</td>
</tr>
</tbody>
</table>