COMPUTATIONAL ANALYSIS OF STRUCTURE AND FUNCTION OF GENOMIC SEQUENCES

by

ABANISH SINGH

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2008

To my parents in gratitude for a lifetime of inspiration and support.

ACKNOWLEDGEMENTS

I would like to thank my supervising professors Dr. Nikola Stojanovic for constantly motivating and encouraging me, and also for his invaluable advice during the course of my doctoral studies. I also wish to thank my committee members Dr. Cedric Feschotte from Biology Department, Dr. Gautam Das and Dr. Jean Gao from CSE Department, and Dr. Subhrangsu Mandal from Biochemistry Department for their encouraging words, thoughtful criticism, interest in my research, and for taking time to serve in my dissertation committee.

I would also like to extend my appreciation to the National Institute of Health and the University of Texas at Arlington for continuous support to my research. I am grateful to all the teachers who taught me during the years I spent in school, in India, UK, and Unites States. I also thank several of my friends and fellow members who have shared their enthusiasm and helped me throughout my graduate studies.

Finally, I would like to express my deep gratitude to my parents and family members, who have encouraged and inspired me throughout my life. I am extremely fortunate to be so blessed, and thank them for their never ending lifetime love, support, sacrifice, and patience.

November 17, 2008

ABSTRACT

COMPUTATIONAL ANALYSIS OF STRUCTURE AND FUNCTION OF GENOMIC SEQUENCES

Abanish Singh, Ph.D.

The University of Texas at Arlington, 2008

Supervising Professor: Nikola Stojanovic

The genetic code consists of long chains of deoxyribonucleic acid (DNA) present in every cell of a living organism. These chains contain both functional and non-functional DNA sequences, and their proportion in the mix varies widely along the tree of life. Generally, more complex organisms tend to feature large amounts of "junk" DNA, whose importance is still subject of a debate in the scientific circles.

The functional sequences include coding sequences (genes) and various types of signals, mostly, but not exclusively, controlling the regulation of coding sequences, i.e. activating and deactivating the expression of genes, during the developmental stage, in response to external stimuli, or during housekeeping activities in a cell or organism. Such expression leads to the production of various ribonucleic acids (RNAs), out of which the most common is messenger RNA (mRNA) which serves as a template for chains of amino acids, or polypeptides. The polypeptides themselves fold and group into proteins, providing structural components and functionalities to the living cells and tissues. Regulatory signals in DNA tend to act as parts of complex networks, whose structure and dynamics have been subject to biomolecular studies for many

decades. Recently, especially after sequencing of several major eukaryotic genomes has been completed, these studies have become increasingly computational. The applied techniques focus on sequence features, such as periodicity, motif over–representation, phylogenetic conservation, sequence or structural homology, or the experimental data about binding effects, patterns of gene co–expression, and, more recently, epigenetic information.

Over the last several years, the search for functional elements in human and other genomes by exploiting motif over-representation became increasingly popular. Although there has been some success in this field, the existing tools are still neither sensitive nor specific enough, usually suffering from the detection of a large number of false positive signals. Given the properties of genomic sequences, some of which we analyze in this document, this is not unexpected, but one can still find interesting signals worthy of further computational and laboratory investigation.

In this thesis we present several algorithms for DNA sequence analysis, and in particular the identification and characterization of short motifs. We start with presenting an efficient algorithm to find significant variable motifs shared within target sequences, generally taken from the upstream regions of co-expressed genes. Various filtering techniques have been applied to this problem in the past, but in our view it is important that we generate complete data, upon which separate selection criteria can be applied, depending on the nature of the sites one wants to locate. Though we primarily intended to develop software to locate the significant motifs based on their over-representation in the given DNA sequences, we also attempted to elucidate why such software often fails in locating the real elements. We have thus performed a study of the repetitive structure and distribution of short motifs in human genomic sequences. In most mammalian species about half of the genome consists of known or readily recognizable repeated elements, and we demonstrate that in addi-

tion to these repeats human genomic sequences feature many short motifs which are significantly over-represented, and that their frequency varies only slightly between random repeat–masked sequences and regions located immediately upstream of the known genes.

Recent studies have established the existence of evolutionary (and thus presumably functional) constraint on only about 5% of the human genome. If a half of it consists of known repeated sequences, that leaves an open question about the source of the remaining 45%, for which we postulate that it should have mostly originated from ancient transpositional or other duplication activity. The original copies could have become so broken over time that they cannot be recognized as such any more, giving rise to seemingly unique sequences which nevertheless share large numbers of greatly over–represented short motifs. We have developed an algorithm, and written software which efficiently associates these motifs and reconstructs the consensus sequences of possible ancient broken repeats. We have found a significant number of new large repeated sequences, in addition to the previously characterized transposable elements and other duplications in the human genome, and we have built their consensus sequences and attempted to characterize them. We believe that in view of a recently proposed model postulating that transposable elements have been a significant source of transcriptional regulatory signals, further study of broken genomic repeats would be very useful.

The software implementing our methods have been made available in the public domain, and we have also developed a web server to enable on–line access to our tools by other investigators.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

The genome of every living organism holds the key hereditary information and consists of long chains of deoxyribonucleic acid molecules known as DNA. These chains are organized in compact units, confined to limited volume, and known as chromosomes. The chromosomes are present in pairs (one copy inherited from each parent) in each cell of an organism and comprise both functional and non-functional DNA. The genome is the complete set of DNA of an organism. It may contain as less as 500 genes in some bacteria to as many as 40,000 in humans [22]. A gene is a basic unit of hereditary information; it is a specific segment of DNA which serves as a unit of function by encoding a particular ribonucleic acid, or RNA. Generally, more complex organisms tend to feature large amounts of "junk" DNA, whose importance is still a subject of debate in scientific circles. The functional sequences include coding sequences (genes) and various kinds of signals–mostly, but not exclusively, controlling the regulation of coding segments, activating and deactivating the expression of genes in response to the needs of the cell or organism, during developmental stage, or in response to external stimuli. Such expression leads to the production of RNAs, most of which serve as a template for chains of amino acids, or polypeptides. The polypeptides themselves fold and group into proteins, providing structural components and functionalities to the living cells and tissues. These proteins may work independently or as part of multi-protein assemblies. The role of regulatory sequences, which includes promoters and supplementary modules targeted by various transcription factors, is to have these proteins produced at the appropriate time and place.

1.1 Biological significance of the motifs search

The gene expression is a process of information transfer from DNA into RNA, further into proteins. It involves two major phases: transcription and translation. A human cell contains a large number of genes (the final number is yet to be confirmed). Some of these genes are expressed all the time and are responsible for controlling run type metabolic functions or respiration; some are expressed when cells enter particular pathways of differentiation; and the others are expressed when the conditions in and around the cells change. Gene regulation is the coordinated control of gene expression. It has attracted much attention of researchers in the recent past, as we have been striving to know the spatial and temporal differences in how genes are expressed, related to developmental control of an organism, cell differentiation and tissue specificity, production of hormones and enzymes, cellular responses to stress such as disease or adverse physical conditions, and a number of other issues [67].

Gene expression is substantially different in bacteria (prokaryotes) and higher organism (eukaryotes), which maintain their DNA in a nucleus. There are several mechanisms used by eukaryotes to regulate the gene expression. These mechanisms are generally of two types: transcription level regulation and post transcription level regulation. The mechanisms of transcription level regulation are responsible for altering the rate of transcription either by activating or silencing the process. They include modification of chromatin structure of the DNA which makes genes hidden from or invisible to transcription, control through special proteins called transcription factors (which bind to DNA sites to cause an enhancing or suppressing activity), or silencing the transcription through genomic imprinting. The post transcription level regulation is done by altering the rate of RNA processing (while still in the nucleus), altering the stability of mRNAs and the rate of their degradation, and by altering the efficiency of ribosomes which translate the mRNA into polypeptides. This regulation

includes the mechanisms such as RNA splicing, control of the amount of gene product by stability of RNA, modulation of translation because of non-coding sequences in mRNA, and use of antisense (siRNA) to stop or slowdown the translation.

It is well known that transcription level regulation, in which the transcription is activated or suppressed by specialized proteins, is the most important mechanism of gene regulation. Each gene has a transcription start site (where RNA polymerase II binds to initiate the transcription) and regulatory regions like enhancers, silencers and promoters. These regulatory regions have to fulfill several requirements in order to serve biological functions in transcriptional control. In general, the promoter is an integral part of a gene and often makes sense only in the context of its own gene. The function of a promoter is to mediate and control the initiation of the transcription of the gene located immediately downstream to it. The promoter sequence is present in only one of the two DNA strands and the presence of a promoter determines which one of the two strands is to be used as a template for copying the DNA to RNA in the transcription process. The RNA polymerase binds at the promoter to initiate this process. The promoter is a necessary region to achieve the transcriptional initiation, but it may not be sufficient to completely determine the regulation. In addition, the polymerase complex which binds the promoter and initiates the transcription is mainly concerned with accurately copying the DNA into RNA, but not with determining from where or when to start. Selecting the location to start the copying is the function of other types of protein called activators. The main players in the promoter activation are specialized proteins called transcription factors. There are varieties of transcription factors in a cell and all of them contain a DNA binding domain, which enables them to bind with genomic DNA, and an activator domain, which can have either an activating or suppressing activity. The factor brings its activator domain to a specific location by binding to genomic DNA at a specific site. The transcrip-

tion factor binding sites are usually short 8-20 nucleotide stretches which are covered by proteins upon binding. Surprisingly, the sequences for binding sites for the same protein can vary considerably.

The transcription factors (TF's) binding sites located in upstream, downstream, or even in intronic regions of eukaryotic genomes are called cis-acting elements. These sites play a significant regulatory role in gene expression. Many of them often occur together, in groups called modules, which denote the set of TF's binding sites located close to each other and which have a cumulative effect on gene expression. By binding the appropriate sites, the transcription factors can increase or decrease the expression of genes. The binding sites causing an increase in gene expression are known as enhancers, and if they cause a decrease in gene expression they are known as repressors or silencers. The promoters are generally not a part of the set of enhancers. They are the binding sites for the proteins which initiate gene transcription process, and enhancing or silencing are the activities apart from the core initiation. It has been found that more then one gene expressed together have similar cis-acting elements, and a set of same TF's can initiate, activate or suppress the regulation of these genes. The phenomenon is called co-regulation. A gene which itself encodes a transcription factor can regulate the expression of many other genes which have the binding sites for its product. This relation can be used to establish a complex regulatory network, where one gene controls the expression of other genes.

Gene expression is regulated by a network of protein-DNA and protein-protein interactions. Most of the transcription factor binding sites occur within a few hundred bases upstream of the transcription start sites of their targets, although many can be quite distant. It is also generally accepted that the binding of transcriptional enzymes is directed by specific motifs in DNA sequences. However, while there are proteins which bind specifically to exact sequences of bases, most transcription factors are

rather promiscuous in their choice of a preferred binding consensus. This indicates that sequence motifs may only be a part of the regulatory signal.

In summary, the TF's binding sites are small DNA motif sequences which play an important roles in gene expression, and they are proximally or distantly located in non-coding regions. The computational prediction and identification of these motifs can help with the understanding of the underlying process of gene regulation.

1.2 Previous work on motifs finding

The search for transcription factor binding sites is one of the most popular subfields of bioinformatics, and many algorithms have been developed over more than a decade of intensive research [66, 94, 57, 46, 44]. Some computational methods were developed even before whole eukaryotic genome sequences became available, for example determining the minimal sequences necessary for transcription in cell-culture based systems, identifying sequences potentially available for transcription factor binding, in vitro approaches to identify sequences that bind various regulatory proteins, and enhancer and promoter tapping studies. But, these methods were slow, difficult, expensive, and not well suited for easy and fast understanding of the gene regulation system. Meanwhile, as the landmark in genomics studies the whole-genome sequences for human [47], and other closely and distantly related species were made available around 2001. Furthermore, a number of computational methods based on sequence comparison and other statistical approaches were developed to identify the transcription factor *cis*-acting binding motifs.

Apart from the laboratory methods, the computational approaches to identify *cis*-acting elements can be concerned with the identification of potential TF's binding sites in isolation, or the identification of *cis*-acting regulatory modules. The examples of the first kind of approaches include the identification of regulatory regions by

comparing genomic sequences, identification of regulatory regions by correlating their presence with gene expression patterns, and identification of motifs based on their over-representation. The methods for the identification of *cis*-acting modules first identify all occurrences of regulatory motifs in the target area, and then use clustering to detect potential modules.

Closely or distantly related species have conserved sequences representing orthologous genes, which are resulted by their evolutionary history or function. The non-coding parts of a genome may have many conserved motifs, too. It is likely that conserved motifs in non-coding sequences are *cis*-acting regions regulating gene expression, although this needs not necessarily be the case. It has been found in various experiments that human [47] and mouse [69] genomes share many regulatory motifs. Wasserman et al. [102] have identified common binding sites in human and mouse using sequence comparison method, and this algorithm has been used by many others. In this approach the homologous genes in two related species are identified and then non-coding DNA, both upstream and downstream, are compared. If we get any conserved motifs in the results, they are likely *cis*-acting elements. This approach has some limitations: two species for which the DNA non-coding sequences are being compared must be close enough to have many homologous regions, and this approach does not give any functional information about the conserved sites.

Tavazoie et al. [94] analyzed the expression pattern of mRNAs of Saccharomyces cervisiae over a series of time points during a cell division cycle. After grouping and clustering the expression profile based on their common expression patterns across many time points and then searching for common upstream DNA motifs, they were able to identify 18 motifs spread over 12 different clusters. This approach also has limitations; clustering may not be a good way to identify co-regulated genes, and motif may occur in various genes not sharing the common expression pattern. Later,

the same group came up with a solution to some of its early shortcomings [57]. To determine the significance of the motifs, the criteria of group specificity and position bias were used. This approach identified a large number of experimentally verified motifs in the *S. cervisiae* genome. Bussemaker et al. [46] proposed another approach known as Regulatory Elements Detection Using Correlation with Expression (REDUCE), which was based on the assumption of a linear additive model for the effect of different regulatory motifs. Each motif may have either a positive or negative influence on the expression level of the gene found downstream of it. This approach was able to detect many experimentally known cis-acting elements as well as to predict some new ones; however, it can account only for 30% of the total signal present in genome-wide expression pattern.

The same research group [44, 45] also proposed a purely computational approach to detect over-represented motifs in a genome of single species. Their algorithm works by iteratively building words from a text, determining the probability of occurrence of all letters of the alphabet in the text, and adding words that have a non-zero probability to the dictionary. Expected frequencies for all combinations of two or more letters are calculated based on that probability. The occurrence frequencies of the words are obtained from the text, and the combinations which occur in the text with given significance are added to the dictionary. To validate this algorithm, it was tested against the novel Moby Dick, by Herman Melville (1851), by removing all non-alphabet characters, concatenating the text, and inserting random junk letters in-between to expand the text three times. This algorithm is known as the Moby Dick algorithm. As a result of this test, 740 out of 1050 most significant words were English words, or composite words. Though DNA sequences are likely to be harder to deal than English words, the algorithm has shown promising results when applied to

600 bp long upstream sequences of the yeast genome. However, this algorithm does not provide any functional annotation of the detected motifs.

The early approaches to regulatory signal finding relied on a rather naive assumption that the target sites of proteins must feature information content sufficient for them to be recognized. Disillusionment soon followed, as any attempt to isolate functional elements in DNA resulted in an enormous number of false positives. Recent methods have thus concentrated on the incorporation of additional information to the raw sequence data. They often relied on phylogenetic conservation [32, 19, 90, 75] or search for clusters whose elements matched experimentally confirmed consensus motifs [19, 90], retrieved from databases such as TRANSFAC [99] or Jaspar [15]. The latter methods exploited the fact that proteins involved in the initiation of transcription rarely, if ever, act in isolation, the basis postulate of the regulatory module finding. In a recent study, structural aspects of TF-DNA interaction has also been exploited to supply the additional information [86] for motif finding.

With the advances in microarray technology large sets of putatively co-expressed genes became available, stimulating the development of new methods to detect conserved motifs in their upstream sequences [57, 19]. It is intuitive that if a group of genes is co-ordinately regulated, it should be controlled by the same transcription factors. From the hypothesis that protein binding is directed by DNA sequence motifs it follows that same motifs should be present in all regulatory sequences, moreover, as a cluster or clusters. This led to the exploitation of motif over-representation in the target sequences. In addition, it has been observed in yeast that the promoter regions are often characterized by multiple occurrences of the same binding motif [55], and it has been postulated that it may be the case in higher eukaryotes, too. These assumptions stimulated the search for over-represented, or "surprise", motifs in related sequences [12, 56].

1.3 Transposable DNA repeat elements

It has been well known, and for a long time now, that genomic sequences, even in large "junk" areas, are not random assemblies of four letters. In a series of seminal studies initiated more than four decades ago, Britten, Davidson and colleagues demonstrated that the nuclear genome of diverse eukaryotes contained a large fraction of repetitive DNA [37], and recent large–scale genome sequencing has established the ubiquitous occurrence of repeats. They can be of a tandem nature, however it is the interspersed sequences which generally represent the major component of repetitive DNA [47]. Most of these are derived from mobile (transposable) genetic elements (TE's), fragments of DNA which can move around and insert into new chromosomal locations, often duplicated in the process. Thus, transposable elements or their remnants tend to represent the single largest component of eukaryotic genomes, accounting, by the latest estimates, for 10% of the tiny genome of the nematode *C. elegans*, about 45% of the human genome and about 80% of the maize genome, to mention just a few (C.Feschotte and E.Pritham in [73]). These transposable elements can be divided into different classes, and a brief description of several major categories is given below:

1. **Tandem array repeats:** These repeats are composed of multiple head to tail repetition of simple sequences. Such repeats are also known as local repeats. They can be further divided into:
   - **Micro–satellites and mini–satellites:** The micro–satellites are also referred to as *simple sequence repeats*, and the mini–satellites are known as *variable number repeats*. These types of repeats are defined by 3 parameters: pattern, length, and number. The length of these repeats is usually 1-6 bp, and sometimes the upper boundary is extended up to 10-13 bp.

- **Satellite and telomeric repeats:** The satellites are arrays of 103 -107 tandemly repeated units located in well defined chromosomal regions, such as centromeres and telomeres. Centromere is the point where the two sister chromatids touch, and telomeres are the ends of a chromosome.

2. **Interspersed repeats:** These are the scattered pieces of the transposable elements which were active in the human genome or genomes of our ancestors. These can be further divided into:
   - **Non LTR retrotransposons:** The retro elements proliferate by the reverse transcription of their RNA, expressed in a host cell. These are also referred as Long Interspersed Repeat Elements (LINE), and their short associates are referred to as Short Interspersed Repeat Elements (SINE). Other examples of non-LTR retrotransposed elements are processed pseudogenes, although strictly speaking they are not transposons.
   - **LTR Retrotransposons:** These are retrovirus-like elements retrotransposed in a genome, and inherited by a host from generation to generation. These are flanked by long terminal repeats (LTRs).
   - **DNA transposons:** The transposable elements which move from one genomic site to another in the form of DNA only, are known as DNA transposons. They work like "cut and paste". Both ends of DNA transposons have terminal inverted repeats (TIRs), which are identified by transposase enzymes to cut the DNA. Most, if not all, transposons encode an enzyme called transposase that acts much like λ-integrase by cleaving the ends of the transposon as well as its target site.
3. **Segmental duplications:** A segmental duplication occurs in a region containing several partially degraded mobile elements of various types followed by

mutation of duplicated regions including insertion, deletion, and rearrangement. Apparently, this kind of repeat seems to be beyond annotation.

Large–scale DNA sequencing has revealed that most of the repetitive DNA is derived from the activity of transposable elements, i.e. sequences that are able to move and replicate within a genome. An important and virtually universal feature of transposons, regardless of their classification, is that the large majority are not capable of further transposition [16, 25]. Most are found as defective copies, which cannot encode the proteins necessary for their movement — for some this is a natural form, while for others it is the result of sequence degradation or deletion at the time of insertion. Consequently, once integrated, most sequences will never transpose again, and they can be regarded as molecular fossils. Regardless of their origin and of the mechanisms responsible for their inactivation, it is widely accepted that fossilized transposons, as a whole, do not assume function to the host. Consequently, inactive copies of transposable elements are progressively eroded by mutations accumulating at a neutral rate until they become unrecognizable, so the largest portion of most multi-cellular and even some single–celled eukaryotic genomes can be regarded as an enormous transposable element graveyard (C.Feschotte and E.Pritham in [73]).

1.4 Identification of repeats in DNA sequences

Traditionally, the first task in the analysis of transposable elements is the assembly of the repeat library. It consists of defining all repeats within the genome, demarcating their boundaries, subdividing related copies into families and reconstructing a consensus sequence representative of each family. The annotation of repeats then includes the mapping (or masking) of regions which correspond to a given family, on the basis of the similarity to their consensus sequences compiled in a depository such as RepBase [52, 51]. The major limitation of this homology–based approach is that

only sequences similar to these previously described will be identified. It could be a satisfactory approach if the query sequence originates from a species which is closely related to one for which a comprehensive catalog of repeats is available. Otherwise, homology–based searches will tend to reveal only these elements which are relatively intact and contain protein–coding sequences similar to those typically found in known transposons (such as reverse-transcriptase and transposase).

Another computational method for identifying transposons consists of *de novo* strategies, designed to identify repeated sequences in a genome based on their copy counts, without resorting to homology with an outside catalog. Two basic approaches have been employed for *de novo* identification of repeats: query vs. query similarity searches and word counting/seed extension. The former relies on genome self–comparison, where an entire genome is aligned to itself. This step generates a series of pairwise or local alignments (using blast [92] or blast–like programs). Pairwise alignments are then converted into multiple alignments, and clustering methods are used to group related sequences into families, based on a pre-set or user–defined threshold of similarity and alignment length [104]. Miropeats [53], RepeatFinder [76], RECON [104] and PILER [85] use this approach. Most of them are computationally intensive, requiring vast memory capacity and substantial processing time, which varies with the size and complexity of the query sequence. Their results are also significantly affected by the relative lack of sensitivity of the programs used for the initial self-comparison (blast, for instance) and subsequent problems related to the clustering methods, which often lead to imprecise definitions of the repeat ends. An alternative and increasingly popular approach involves word counting and seed extension. These methods bypass the need for whole genome alignments by building a set of repeat families starting with short strings (seeds) repeated in the genome. These seeds are progressively extended into a consensus through comparisons of their copies

in the query sequence. RepeatScout [14] and ReAS [89] have been developed implementing this approach. A brief description of four major programs — RECON [104], PILER [85], ReAS [89], and RepeatScout [14] — is given below.

1. **RECON:** This is a *de novo* approach based on single linkage clustering of pairwise alignments, with extensions. These extensions were (1) using multiple alignment information to define the boundaries of individual copies of repeats, and (2) distinguishing homologous but distinct repeat elements for the biologically reasonable clustering. The problems with the single linkage clustering of pairwise alignments were (1) the use of overlap to infer syntopy would cause error if overlapping images were of different length, and (2) inter-family similarity — lumping of related but distinct families together, as many of repeat families could be evolutionary related. As a solution to these problems, the RECON uses the collection of the endpoints in multiple alignment of the images in filtering out elements after the initial definition. The RECON considers two elements to be distinct if the length of non-conserved bases adds up more than certain length of two sequences. While applying a graph based clustering, the significant alignments between same families are represented by primary edges of the graph, and significant alignment between different families are represented by secondary edges. The algorithm of the RECON program works in the following steps:
   - Obtain local alignment of input sequence to itself.
   - Define elements from the obtained alignment:
     - First, define elements using single coverage information.
     - Reevaluate elements after filtering out misleading images.
     - Stabilize the definitions — if an element is split and considered to be composite, elements forming the alignments with the composite

elements will be reevaluated, and this process will continue until all the definitions of the elements stabilize.

- Group elements into the families on the basis of their sequence similarity:
  - Build a graph of the element–family relationship — this relationship is determined and converted in to graph $H(V, E)$, where V = elements and E = relationship, primary or secondary based on the similarity and length of the non-conserved region.
- Find all connected components of graph $H(V, E)$ based on primary edges, and for each connected component, define a family as the set of all elements in the component.

Pros: The RECON was a significant improvement over the single linkage clustering. When first reported 6 years ago, it was an important work that attempted to identify the boundaries of repeats using the aggregation of endpoints from multiple alignment. In real application it became a dominant tool, and was used to construct the library of C.briggsae repeat families.

Cons: Though no asymptotic analysis of CPU/memory usage was given, the approach used in the RECON was inefficient, as it took 39 CPU hours and 750 mb RAM for 9mb sequence. Also, the RECON is unable to recover highly fragmented families in one piece, and the program can fail if its simple assumption about alignment-end-clustering is violated. The program attempts to define the boundaries based on the alignment, but the local alignment usually does not correspond to the biological boundaries. Also, the program can not distinguish between multi copy genes/pseudogenes, segmental duplication, and transposons.

2. **PILER:** This tool also begins with comparing target DNA sequences against itself to identify the local alignments between the regions, much similar to the

RECON, but it differs in the process of the identification and classification of the repeats. The PILER focuses on identifying the subsets of the hits — those form a characteristic pattern of a given type of repeat. It uses the PALS — a program for Pairwise Alignment of Long Sequence, optimized for aligning a sequence to itself — to get the local alignment. The algorithm of the PILER works in the following steps:

- Take an array c[x] of length L, where L is sequence length, and initialize the array to 0s.
- Take each image Q in a set of images, resulted from self–alignment, and increase c[x] by 1 at all corresponding positions of an image. Now c[x] would have copy count of base x.
- Scan array c[x], and check for all piles — i.e. if c[x -1] = 0 & $c[x] > 0$, or if x starts a new pile — and replace the c[x] with sequential pile number.
- Create a data structure for P empty piles — i.e. set of images.
- Again go through each image Q in a set, and find pile sequence number associated to this image by looking p = c[start(Q)], and add image Q to pth pile in the data structure P.

Pros: The PILER searches have high specificity with sensitivity that varies with genomes. The tool avoids combinatorial explosion of local alignments without discarding functional similarities. Also, it attempts to find the boundaries induced by evolutionary process or inferred from biological mechanisms in contrast to other tools, such as RepeatFinder and RECON.

Cons: Because of using incomplete sets of overlapping hits from PALS to reduce computational work, the PILER exhibits very low sensitivity, and, sometimes, it also ends up misidentifying two or three concatenated instances as one.

3. **ReAS:** The ReAS attempts to recover ancestral sequences for transposable elements (TE) from the unassembled reads of whole genome shotguns. In general, in the whole genome shotgun approach, some of the unassembled reads are due to centromeres or telomeres, but Li et al., 2005 claims that in the rice genome many such reads are recent TE's. Unassembled reads are the most informative for TE recovery as they are the least diverged from their ancestral sequences. The ReAS algorithm works in the following steps:
   - Select high depth *K-mers*:
     - The depth is the number of times a K-mer appear in the shotgun data, the copy number is the number of times a *K-mer* appears in an assembled genome, and the coverage is the depth divided by the copy number.
   - Retrieve all of reads that contain a *K-mer*.
   - Assemble these reads in to an initial consensus sequences (ICS).
   - Look for the new *K-mers* at both the right and the left of a consensus (i.e. ICS), and use these *K-mers* as secondary seeds to iteratively extend the ICS until no further extension are possible.
   - The final consensus is a ReAS transposable element.

   Pros: As the ReAS finds TE's using shotgun assemblies, this approach may help in reducing the error in assembling whole genome shotguns.

   Cons: The ReAS works on an assumption that TE's must exist at a high copy number, and must not be so old that they are no longer recognizable. This high copy number assumption is a big constraint for the ReAS. For example, if we look at RepBase TE's, only half of the 17-mers have the depth of 14 — as per the settings which were used for the rice genome — so the other half could never been recovered by the ReAS.

4. **RepeatScout:** This tool is a *de novo* method to find the set of repeat families by extending the seed-motifs to the longer sequences. This tool uses high frequency *l-mers* — short sequences of length l — as the seed-motifs, and greedily extends each seed motif to a progressively longer sequence, following the dynamically inferred alignments between the consensus sequence and its occurrences in a genome. The RepeatScout utilizes an efficient method of similarity search, and enables a rigorous definition of the repeat boundaries. The algorithm works in the following steps:
   - Find all *l-mers* of a given length.
   - Align the sequences having a *l-mer* by greedily extending 1 bp at a time.
   - Discard a sequence after it stops aligning with the consensus.
   - Stop extending when the most sequences no longer align.
   - First extend in the right of a *l-mer*, then extend in the left in similar way.

   Pros: The RepeatScout enables a rigorous definition of the repeat boundaries. The tool works in orders of magnitude faster as compared to other tools like the RECON and the RepeatFinder. Also, it is more sensitive to the RECON, as more than 4% of the C.briggsae genome was identified as repeat which was missed by the RECON.

   Cons: The identification of the repeats depends on the fixed size of a complete *l-mer*. In such case, it would not be possible to recover ancient repeats because those got decayed too much, and no complete *l-mer* could be chosen for the extension. In locating the repeats, the performance of the RepeatScout fully depends on the RepeatMasker. Also, like the RECON, the RepeatScout cannot distinguish between the multi copy genes/pseudogenes, segmental duplication, and transposons.

De novo repeat identification methods are powerful because they generate a catalogue of repeats present within a query sequence without *a priori* knowledge of their characteristics and classification. A major pitfall of these approaches is that they can only be applied to queries representing considerable sequence data, such as whole–genome shotgun sequences. A related problem is the identification of low copy number repeat families. This is because of the increasing number of false positives — including host–gene families and segmental duplications — as the copy number cutoff of the program is decreased. *De novo* repeat identification programs will also tend to miss or split composite repeats — i.e. repeats made of several independently repeated units — or families with highly variable structure.

1.5 Transposable elements and regulatory networks

Barbara McClintock discovered transposable DNA elements early in her career [23], for which she was recognized with Nobel prize in 1983. The TE's are major components of eukaryotic genomes, contributing about 50% of the size of mammalian sequences. It has been known for a while that in order to survive transposable elements replicate themselves faster than the host that carries them, and move around to different positions within the host genome. Many scientists regarded the TE's as "junk" DNA, genomic burden, unnecessary ballast, selfish DNA, or a parasite, but the real functions of transposable elements remain poorly understood. In recent studies, the earlier notions about the TE's have been expanded, and it has been proposed that they have been a significant source of transcription regulatory signals [21, 26]. Some of these studies [80, 21, 81, 54, 17, 71, 26] have suggested that TE's have also had a key role in the evolution of eukaryotic gene regulation. These studies have elucidated several ways in which TE's can influence the expression of a nearby gene. Moreover, a large number of regulatory elements, including the signals which are nor-

mally used by TE's to control their own expression and several TF binding sites, have been identified in active TE's, or in computationally reconstructed sequences of TE's. In addition to creating new regulatory networks the replications and movements of TE's may rewire the existing ones, too. The TE-mediated shuffling and duplication of *cis*-regulatory elements is one way to modify an existing genetic network [33].

1.6 Our contribution

With the availability of sequences of a large number of eukaryotic genomes over the last several years, the efforts to characterize non-coding functional DNA elements have intensified [98]. Comprehensive studies of the effectiveness of many motif-finding tools [63, 70] have shown that while there has been some success in the binding site recognition, the existing methods are not nearly satisfactory. Our ability to predict and identify TF's binding sites in DNA sequences is significantly limited, possibly because of the following:

- These sites may be proximal or distal — their locations are not well understood yet.
- The length of transcriptional protein binding sites is usually very short.
- Gene regulation may also depend on the spatial configuration of DNA and other epigenetic phenomena.
- An isolated site may have no regulatory function.
- Detection of motifs through over-representation or conservation may be hampered by pieces of randomly repeated/conserved sequences.
- Transcription factors generally feature very non-specific binding preferences [59], and that permits for wide variations in the motif consensus, making it difficult to isolate the motifs.

The solution to some of these issues may lie in the simultaneous study of a large number of closely related sequences, which are becoming increasingly available [74]. This was one of the major methods of the ENCODE project [97, 98], which in its pilot phase aimed at the development of high–throughput techniques for the classification of DNA elements within a set of target regions, comprising approximately 1% of the human genome.

It is intuitive that if a group of genes is coordinately regulated, it should be controlled by the same transcription factors. Though we are not still clear how important are DNA motifs per se, common wisdom is that DNA–protein binding activity, to a great extent, is directed by these motifs. Consequently, the same motifs should be present in all observed upstream regulatory sequences of coordinated regulated genes, moreover as a cluster, or multiple clusters representing targets for the transcriptional initiation complexes [19, 77, 90]. This assumption has led to the exploitation of motif over-representation in the target regions.

As a part this thesis, we present an efficient algorithm to find all significant variable motifs in given sequences. Our approach is motivated by the consideration that it is better to detect every motif which appears statistically significant, and upon which separate selection criteria can be applied depending on the nature of the sites one wants to locate. We have written a software MotFi which efficiently searches through multiple upstream sequences and lists all the significant degenerate motifs shared within subsets of these sequences. The underlying algorithm of this software [8] is described in the Chapter 2 of this thesis, and our software is publicly available on our webserver. MotFi takes an input file with sequences in Fasta format, finds the list of significant motifs, and generates a PDF file showing the visualization of the spatial distribution of selected motifs in the input sequences. In order to have a better insight about the functions and possible biological properties of the listed motifs, our

server also supports the matching of the reported motifs in RepBase and JASPAR databases. For detecting weak similarities in many sequences our algorithm favorably compares to other methods, such as Gibbs sampling [27] (as our method can rapidly detect all the significant motifs) or the BEST [28] suite of tools (featuring better sensitivity in our tests on known regulatory motifs described in TRANSFAC [99]).

While working on the MotFi software, we realized that any given selection criteria returns too many results. More puzzling was the observation that a large number of listed motifs showed extremely significant p-values. Though we primarily intended to develop software to locate the significant motifs based on their over-representation in the given upstream DNA sequences, we have also attempted to understand why every search for over-represented motifs returns many results, and why every motif–finding software often apparently fails in locating the real elements. We have thus performed a systematic study of the repetitive structure of DNA and the distribution of short motifs in human genomic sequences. In most higher eukaryotes, about half of the genome consists of known repeated elements, and we demonstrate that in addition to these known repeats human genomic sequences feature many short motifs which are significantly over-represented, as well as that their frequency varies only slightly between random repeat–masked sequences and regions located immediately upstream of the known genes. It is well known that even non-functional parts of a genome are not a random assembly of four letters, so the analysis of statistical features of DNA sequences often involves a non-trivial background model, such as these based on Markov models or hidden Markov models. We have done extensive simulations [4, 6] in which the number of repeated elements in randomly generated synthetic sequences was almost perfectly conforming to the Poisson expectation, but the number of repeated motifs in repeat–masked random intergenic DNA was far greater than expected. In consequence, any search for over-represented sequences is

bound to return many results. Depending on what we search for, most would likely be false positives. This analysis of repetitive structure and distribution of short motifs in selected part of the human genome [4, 6] is included in the Chapter 3 of this thesis.

After looking at the distribution of short motifs, we have also attempted to answer the following question. It is well known that around 50% of the human genome consists repeated DNA and less than 5% is known to be under evolutionary (and thus presumably functional) constraint, so where does remaining 45% DNA in human genome come from? In an effort to answer this question we have postulated that most of such DNA is a result of ancient transpositional activity (the segmental duplication could have also played a role), with copies becoming so broken over time that they cannot be recognized as such any more. These broken copies of repeated long DNA elements could result in the over-representation of short motifs we have seen in human genomic sequences. We have developed an algorithm and written a software tool, RepFi, which efficiently associates these significantly repeated short motifs and locates possible ancient repeats. The underlying algorithm of RepFi has been described in [5, 2]. The RepFi software attempts to mutually associate groups of short significantly over-represented motifs readily found throughout genomic sequences, and construct the consensus of large substantially broken blocks. Our results have been encouraging, and our simulations have confirmed some of our expectations. This work is included in Chapter 4 of this thesis. In Chapter 5 we have presented a method to reconstruct the consensus sequences of the broken repeated elements and to classify them as transposable elements by integrating our approach with a previously developed computational tool for the classification of repeated elements [72]. We have also attempted to find unknown repeats in the entire repeat-masked human genome and we have built a library of consensus sequences of these repetitive segments. We believe that our RepFi software would prove to be a useful tool for the identification

of ancient (broken) transposable elements, those which could have not been located yet, in the human and other genomes. According to a recently proposed model of transposable elements, TE's have been a significant source of transcription regulatory signals [80, 21, 26], and the alignment of human genomic sequences to their orthologous regions in other mammals have provided an estimation of TE exaptation which reveals that fixed TE sequences have indeed been under functional constraints [26]. In view of these studies, the identification and characterization of the broken genomic repeats may benefit further genomic research.

CHAPTER 2
IDENTIFICATION OF SHORT OVER-REPRESENTED VARIABLE MOTIFS IN REGULATORY SEQUENCES

Gene expression is regulated by a complex mechanism involving binding of numerous transcription factors to sites often, but not always, found in the relative proximity to the gene. Because of the protein interactions taking place in *cis*–regulation, their DNA binding sites have been postulated to be multiplied and clustered in these areas. Over the last several years the search for functional elements in human and other genomes by exploiting motif over-representation became increasingly popular. As discussed in the previous chapter, many methods have been developed for predicting the candidate motifs of the transcription factor binding sites. With the increased interest in the search for non–coding functional elements in newly sequenced genomes, the effectiveness of many motif–finding tools has also been evaluated [63, 70]. Not surprisingly, this evaluation has shown that, while there has been some advancement in binding site recognition, the existing methods are not nearly satisfactory, and given the properties of genomic sequences this is not unexpected. In this chapter, we describe a new efficient algorithm for the identification of significant variable motifs in the given upstream sequences of co-regulated genes based on motif repetitions. We have developed methods for the characterization and visualization of these motifs, and for relating the layout of the site clusters in the promoter regions of different genes.

We have attempted to characterize the upstream promoter regions of several sets of known duplicated and co-expressed genes, estimating the similarity of the module layout, and relating it to the environment found in presumably neutral segments of

DNA. In order to avoid over-reliance on either known regulatory motifs (which may be limiting, or even misleading) or extensive sequence conservation (which may be difficult to detect in extensively rearranged sequences), we opted for an approach that builds the list of repeated candidate sites, then filters out those which are unlikely to have been repeated due to a functional constraint (i.e. not at random). This approach thus allows for independent cross–validation with other methods, and may also point to other previously undescribed but significant features present in the promoter regions.

In our approach we aim to detect everything that appears significant and then process the results looking at the number of regions sharing the motif, motif composition , database hits, clustering patterns, or positional conservation. The latter can be sometimes used to locate the motifs using sequence alignments. However, this approach is vulnerable to the quality of the alignment, and may miss quite a few positionally conserved elements simply by not aligning them precisely on the top of one another. We have thus developed a software MotFi which searches through gene regulatory sequences and lists all significant variable motifs shared within subsets of these sequences. This software is available for public use, on a webserver (http://bioinformatics.uta.edu/toolkit/motifs) and for download (http://bioinformatics.uta.edu/toolkit/download). Our algorithm favorably compares to other methods for detecting weak similarities in many sequences, such as Gibbs sampling [27] (as it can rapidly detect all significant motifs) or the BEST [28] suite of tools (featuring better sensitivity in our tests on known regulatory motifs described in TRANSFAC [99]). In identifying the significant variable motifs, over-representation of motifs, either within a single putative promoter region or within multiple promoter regions of co-expressed genes was our primary guide.

2.1 Estimating the over-representation of motifs

In order to understand and estimate the over-representation of repeated motifs, we focused primarily on the answers to three questions:

1. How many times will a pattern of n characters show up in a sequence of length l characters, solely due to chance?
2. How many distinct patterns of n characters will show up m times in a sequence of length l characters, by chance?
3. What should be the minimum length l of a sequence to expect a chance occurrence of a pattern of n characters at least twice?

To answer the above questions we used Poisson distribution. Let $P(n)$ be the probability of occurrences of a pattern of n characters, and λ be the rate of occurrence in a sequence of l characters.

$$P(n) = \left(\frac{1}{4}\right)^n \tag{2.1}$$

$$\lambda = \text{Total occurrences X Probability} \tag{2.2}$$

$$= l\left(\frac{1}{4}\right)^n \tag{2.3}$$

Using the Poisson density, we can compute the probability of m occurrences of this pattern in l characters of the sequence:

$$P(\lambda, m) = \frac{\lambda^m e^{-\lambda}}{m!} \tag{2.4}$$

The m patterns of n characters would comprise m times n characters out of l characters of the sequence. So, we can compute the probability P of such patterns:

$$P = \frac{mn}{l} = \left(\frac{1}{4}\right)^n \tag{2.5}$$

$$m = \frac{l}{n}\left(\frac{1}{4}\right)^n \tag{2.6}$$

We know that the total number of possible distinct patterns of n characters can be 4^n; hence, the rate of occurrence of these patterns in l characters of the sequence, λ,

will be $\frac{l}{4^n}$. Thus, we can compute the probability of a pattern of n characters in l characters of the sequence:

$$P(\lambda, m) = \frac{\lambda^m e^{-\lambda}}{m!} \tag{2.7}$$

Since we know the total number of possible distinct patterns of n characters is 4^n, we can compute the number of such distinct patterns, N, appearing m times in l characters of the sequence:

$$N = \left[\frac{\lambda^m e^{-\lambda}}{m!}\right] 4^n \tag{2.8}$$

If we expect at most $p\%$ chance of seeing a n character pattern at least q times (where $q \geq 2$) in l characters of the sequence, then,

$$\lambda = \frac{l}{4^n} \tag{2.9}$$

$$P(x \geq q) = 1 - P(x = 1) - P(x = 0) \tag{2.10}$$

$$= 1 - \frac{\lambda^1 e^{-\lambda}}{1!} - \frac{\lambda^0 e^{-\lambda}}{0!} \tag{2.11}$$

$$= 1 - e^{-\lambda}(\lambda + 1) \tag{2.12}$$

$$= 1 - e^{-\frac{l}{4^n}} \left(\frac{l}{4^n} + 1\right) \tag{2.13}$$

$$= 1 - e^{-\frac{l}{4^n}} \left(\frac{l}{4^n} + 1\right) \leq \frac{p}{100} \tag{2.14}$$

Using the above equation, we can estimate l, the length of input sequence, based on at most $p\%$ expected chance of seeing a pattern of n characters at least q times in the sequence.

Even assuming a pure Poisson model discussed above, the number of short repeats expected by chance is large in any significant sequence interval. Ideally, when examining m upstream sequences of related genes we would like to set the length such that m or more occurrences of even a short motif would be significant. Unfortunately,

Table 2.1. Maximal size L_i of individual upstream regions necessary to guarantee 0.01 level significance for a motif of length k repeated m or more times in n sequences.

	k = 3	**k = 4**	**k = 5**	**k = 6**	**k = 7**	**k = 8**
m = 2	4	19	76	304	1216	> 2000
m = 3	9	37	148	595	> 2000	> 2000
m = 4	13	52	210	843	> 2000	> 2000
m = 5	16	65	261	1047	> 2000	> 2000
m = 6	19	76	304	1218	> 2000	> 2000
m = 7	21	85	340	1363	> 2000	> 2000
m = 8	23	92	371	1487	> 2000	> 2000

as shown in Table 2.1, the sequence lengths needed to assure 0.01 significance of a motif of length k in m different segments, if it occurs m or more times, would be unacceptably short. If we would want to include all bases upstream of the genes likely to contain regulatory signals the observed sequences would become so long that a short motif would need to be excessively redundant in order to be recognized as significant. With these sequence lengths many more motifs would start showing up by chance. If we consider a 5' regulatory region of a gene to consist of about 500 bases, then the minimal length of a detectable motif is about 6. These are the numbers for exact repeats — if we permit some variability then we would be forced to either look at much shorter regions, include more co-regulated genes (if any), or seek long motifs only.

2.2 Validating the estimation of the over-representation model

In order to validate the Poisson distribution based model of the over-representation of motifs (discussed in section 2.1), we have performed a study of the distribution of these motifs in human upstream sequences along with synthetic random sequences and random genomic sequences. The synthetic random sequences have been generated by a computer program using four letters A, C, G and T; random genomic sequences

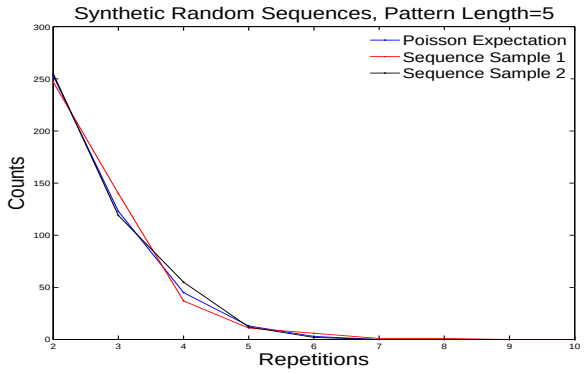

Figure 2.1. Expected and actual occurrences of short repeated motifs of length 5 in synthetic random sequences.

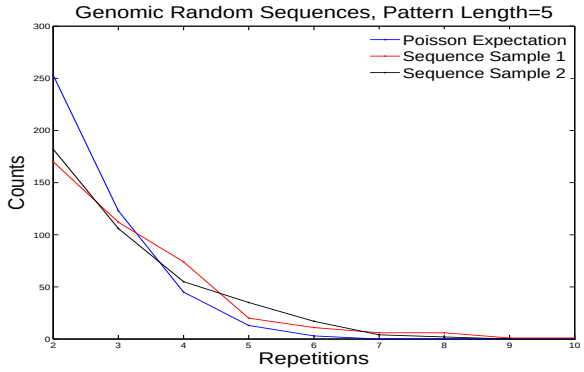

Figure 2.2. Expected and actual occurrences of short repeated motifs of length 5 in random genomic sequences.

are intergenic, interspersed element free sequences extracted from ENSEMBL [35]; and human upstream sequences are upstream sequences of known human genes. We used the Poisson distribution based model — discussed in section 2.1 — to calculate the expected number of repeats. The Figures 2.1, 2.2 and 2.3 show the graph of expected and actual numbers of occurrences of 5 base long motifs (on Y axis) *versus* the number of repeats of 2, 3,..,10 times (on X axis) in two 1000 base long samples of random synthetic, random genomic and human upstream sequences respectively. Figures 2.4, 2.5 and 2.6 show the same distribution for motifs of length 8.

From the results of this experiment it is quite evident that the nature of the distribution of short repeated motifs is different for all three types of sequences con-

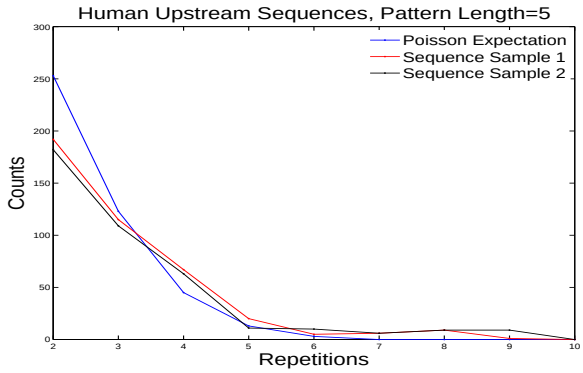

Figure 2.3. Expected and actual occurrences of short repeated motifs of length 5 in human sequences upstream of several known genes.

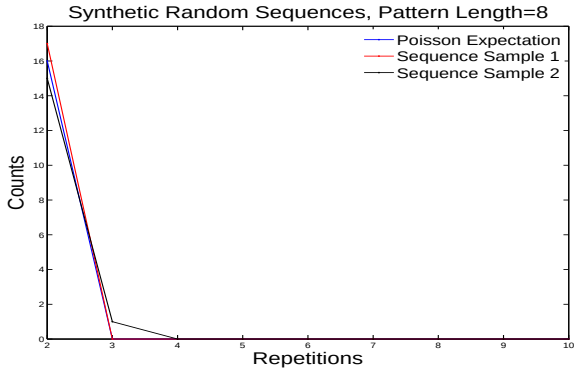

Figure 2.4. Expected and actual occurrences of short repeated motifs of length 8 in synthetic random sequences.

sidered in our experiments. In synthetic random sequences the actual occurrence of short repeated motifs conforms well with the Poisson distribution, validating the estimation of over-representation based on Poisson distribution. However, in the case of random genomic sequences and upstream sequences of the genes the actual occurrence deviates from the Poisson distribution. Chapter 3 includes an extensive study of structure and distribution of short repetitive motifs in different kinds of sequences.

We believe that the deviation in actual occurrences of short repeated motifs against the Poisson distribution expectation has biological significance, since the nature of the repeats is not random. Thus, in this chapter we focus on a study of short repeated motifs in upstream sequences of several sets of co-expressed genes. We have

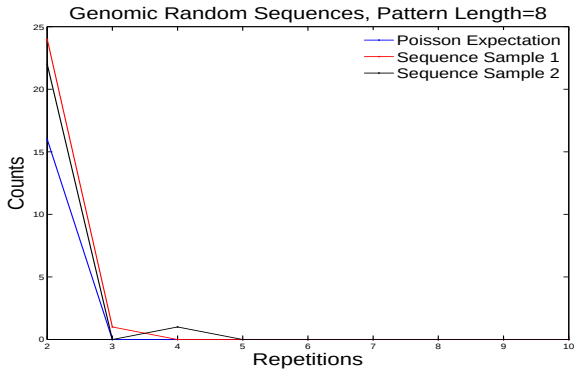

Figure 2.5. Expected and actual occurrences of short repeated motifs of length 8 in random genomic sequences.

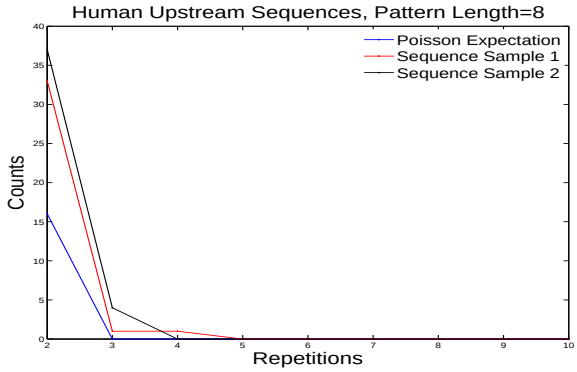

Figure 2.6. Expected and actual occurrences of short repeated motifs of length 8 in human sequences upstream of several known genes.

developed an algorithm and designed software, MotFi, based on this algorithm to identify the most significant variable motifs in the given sequences.

2.3 Algorithm

Our algorithm receives a set of sequences — in which significantly over–represented motifs should be identified, and outputs the list of the motifs filtered by user–definable criteria. These criteria include the following parameters: minimal motif length, minimal number of input sequences where the motifs are present , likelihood of their occurrences by chance, and the permitted degeneracy — the minimal percentage of conserved bases in all occurrences of a motif. This output can be further processed

by a set of tools which we have developed — such as cross–matching, filtering, graphical representation of motif positions, and finding hits in the RepBase and Jaspar databases. We start by identifying all exact repeated strings of length 2 or more in the input sequences on both strands. Using these exact repeats strings as seed sequences, we attempt to identify the variable repeated motifs.

2.3.1 Identification of the exact repeated seeds

To identify the exact repeated seed sequences in the given sequences, we use the suffix tree data structure — the concept was first introduced as a position tree by Weiner in 1973 [82]. The construction of the suffix tree was greatly simplified by McCreight in 1976 [34], and also by Ukkonen in 1995 [39]. Ukkonen provided the first linear-time online-construction of suffix trees, now known as Ukkonen's algorithm. The suffix tree is a well known method, and widely used in similar applications [12, 41, 43, 38, 65, 24, 42, 29].

A suffix tree is a data-structure which stores all suffixes of a string in a graph theoretic way. If $S[1..n] = s_1 s_2 ... s_i ... s_n$ is a string, then $S_i = s_i s_{i+1} ... s_n$ is the suffix of the string S that starts at position i. The suffix tree T for the string $S[1..n]$ is a rooted directed tree with n leaves, and defined as a tree such that:

- There is one to one relationship between the suffixes of S and the paths from the root to the leaves in T.
- All internal nodes in T have at least two children — outgoing edges, with each outgoing edge leveled with different characters of a suffix S_i of the string $S[1..n]$.
- Each edge in T spells a non-empty string.

In order to insure that no suffix is a prefix of another, a terminal symbol— other than the characters of the string $S[1..n]$ — is added at the end of the string $S[1..n]$. For a

string S of length n, there can be at most $n-1$ internal nodes, and $n+(n-1)+1 = 2n$ total nodes, as all internal non-root nodes are branching.

In our implementation of the suffix tree for multiple DNA sequences — as these sequences comprise of four letters A, C, G, and T — we used nodes in the graph with four outgoing pointers and one incoming pointer. We considered the link–list of such nodes as an edge of the graph. Before building the list of suffixes, we appended each input DNA sequence with a special terminal symbol. While searching the suffix tree for the exact repeat strings and their positions in the sequences, we exploited these special terminal symbols — present on the edges of the tree — to find the ends of the input sequences. We have developed an efficient tree traversal algorithm in which each node is visited only once to find all exact repeat strings and their coordinates in the input sequences. The algorithm is based on an elegant use of indexing and stack implementation, and it works in two phases: forward action and backward action. During forward action, it recursively traverses the tree, and records each character of the path in a stack, along with the information about the terminal symbol, if any, associated to a character. The stack based data structure, in addition to recording the characters of the graph edges, also includes two arrays linked to each character — one for recording the sequence ID of a terminal symbol, if any, associated to the character, and the other for recording the distance of the character from the ends, where terminal symbols are present. This data structure also records if there is a branching in the suffix tree at any character. The branching represents internal nodes of the suffix tree. Once forward action is completed for a path of traversal, virtually hitting the endpoint, the backward action uses the information stored in the stack based data structure, and starts removing the characters one by one — based on last in first out — and copying the contents of two arrays to the next character. However, the contents of the array having the distances from the ends are increased by one before

Table 2.2. Variables used in Algorithm 2.3.1

$Temp$	Pointer to start tree traversal
$StackTop$	Variable for stack top, initialized to zero
$STACK$	Stack data structure (array) to store the symbols
Seq	A list associated to each symbol in $STACK$ to store sequence id
Pos	A list associated to each symbol in $STACK$ to store position index
$Symbol$	A character variable to record sequence characters 'A', 'C', 'G', and 'T'
$TerminalSymbol$	Variable for special characters appended to each sequence
$StringSymbol$	Variable for the four sequence characters 'A', 'C', 'G', and 'T'
$TerminalSymbol-$ $SequenceID$	Variable for the sequence id corresponding to a terminal symbol
$NumOfBranches$	Number of branches at a node in the suffix tree
$ExactRepeatString$	String variable to store exact repeated string

copying them to the array of the next character. The schematic representation of the data structure is shown in Figure 2.7, the pseudo-code of the tree traversal algorithm is shown in Algorithm 2.3.1, and the list of variables used in this algorithm is given in Table 2.2.

2.3.2 Identification of the variable motifs

After the original list has been built, we identify the repeats which co-occur at least twice, at a constant distance, and name them potential mates. We use all sites of co-occurrence to build the putative consensus sequence of the variable motif according to the following rules:

1. If the bases at the m-th positions of all loci where the mates have been found are identical, we choose the uppercase character in the consensus, and we record the number of loci as the number of conserved characters.
2. If there is a majority base at the m-th positions of all loci of co-occurrences, we choose the lowercase character in the consensus, and we record the number of occurrences of the majority base as the number of conserved characters.

Algorithm 2.3.1: SUFFIX TREE TRAVERSAL($Temp$)

```
//Forward action
Visit node Temp
STACK[StackTop].Symbol ← Temp.StringSymbol
if TerminalSymbol exists-at-node Temp
  do { n ← 0
       for each TerminalSymbol at-node Temp
         do { STACK[StackTop].Seq[n] ← TerminalSymbolSequenceID
              STACK[StackTop].Pos[n] ← 0
              n++ }
if Branching exists-at-node Temp
  do { if NumOfBranches >= 2
         do {STACK[StackTop].Branch ← TRUE
       for each Branch B
         do { StackTop++
              SUFFIX TREE TRAVERSAL(B)
              //Action after returning from recursive call.
              //Backward action
              if STACK[StackTop].Branch == TRUE
                do { ExactRepeatString ← STACK[0..StackTop].Symbol
                     for each SequenceID S in STACK[StackTop].Seq[ ]
                       do { RepeatPosition-in-Sth-Sequence ← ((Length-of-Sth
                            -Sequence) - (StackTop+1+ STACK[StackTop].Pos[index_S])) }
              //Move the lists from StackTopth position, and append them to the lists at
              // StackTop-1th position.
              for each SequenceID S in STACK[StackTop].Seq[ ]
                do { STACK[StackTop-1].Seq[index'_S] ← STACK[StackTop].Seq[index_S ]
                     STACK[StackTop-1].Pos[index'_S] ← STACK[StackTop].Pos[index_S]+1 }
              Initialize STACK[StackTop] along with lists at StackTopth location.
              StackTop− − }
```

3. If the m-th position of all putative consensus locations does not feature a majority base, we assign 'N' to that position and record the number of occurrences of the most common base as the number of conserved characters[1].

We then determine whether the percentage of conserved characters exceeds a given threshold (95% in our tests). If that holds, we accept the consensus and attempt

[1]Alternatively, we could have used the information content, but it would not be of much help unless we could confirm a bias in the overall distribution of bases.

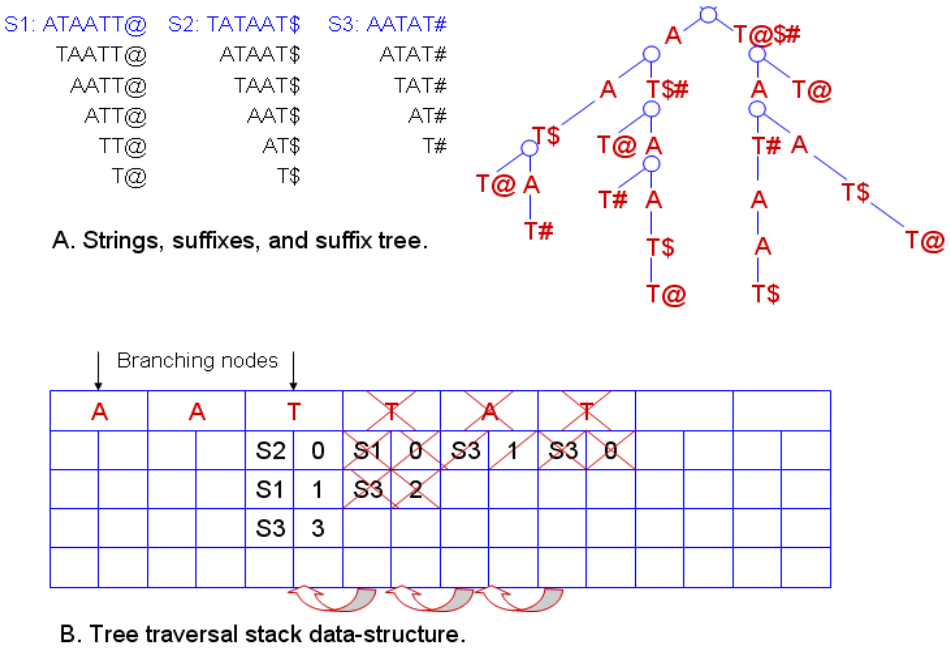

Figure 2.7. An example of strings, suffixes, suffix tree, and tree traversal data-structure. This figure shows three sequences — *S1, S2,* and *S3* — their suffixes, suffix tree, and the data structure, as used by Algorithm 2.3.1, to traverse the suffix tree while identifying the exact repeat strings in the given sequences.

further refinement by recursively searching for additional elements co-occurring with the newly built consensus. The identification of pairs can be efficiently done with the help of an indexing scheme shown in Figure 2.8.

Finally, we combine the consensus motifs with the remaining exact repeats, reindex the list, and attempt to collapse substantially overlapping elements. This is necessary because components of the motifs may have separate occurrences elsewhere, and thus could have been identified as separate, distinct repeats. If these separate occurrences of the motifs permit, their inclusion in the broader consensus is done now. A high–level pseudo–code of our approach is shown in Algorithm 2.3.2, and the list of variables used in this algorithm is given in Table 2.3. This algorithm extends the variable motifs until a given threshold for motif score and p-value are satisfied, and build their consensus.

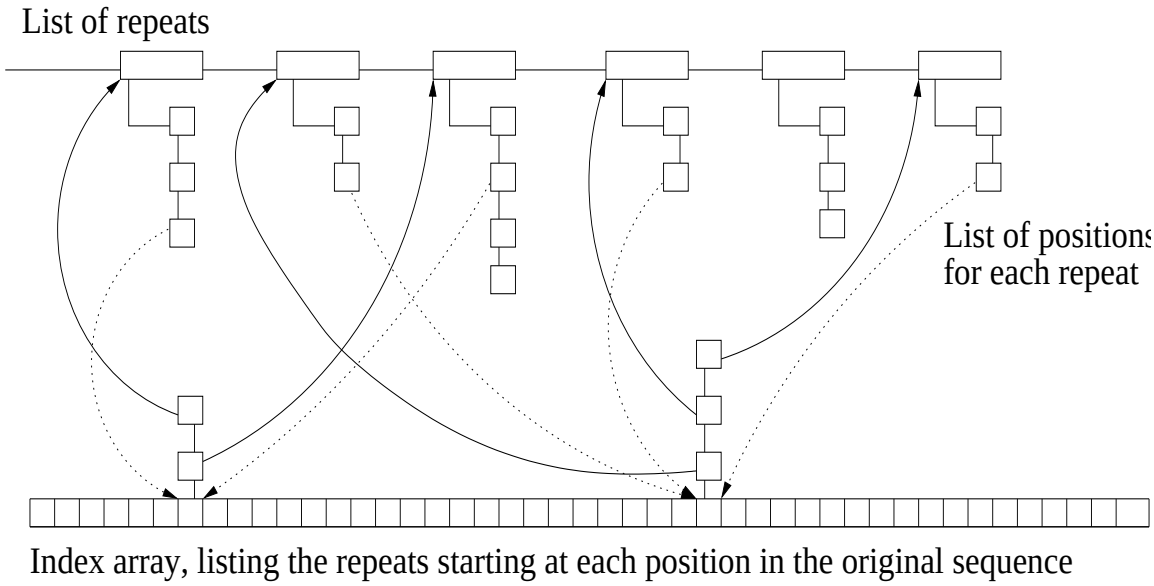

Figure 2.8. A schematic representation of the indexing to find neighboring repeats.

Table 2.3. Variables used in Algorithm 2.3.2

$num\_repeats$	Number of repeats
$num\_positions$	Array for number of repeat positions for each repeat
num_motifs	Number of motifs
$seed$	Exact repeated string
max_gap_size	Maximum gap between two exact repeated seed strings

2.3.3 User–definable significance criteria

Our algorithm finds all repeated motifs with user–definable significance criteria, which includes length of consensus, repeat count, sequence sharing, motif p–value, and significance score. The maximal allowable distance between the anchoring exact repeats is also a parameter to the program, and by default set to 3. No motif copy is lost until the final filtering of the list; however, those which partially overlap and fail to create a consensus scoring above the threshold will be considered separate motifs, and may not be sufficiently significant to be reported. The program uses Shannon's uncertainty, by default set to 1.5, to filter out simple motifs, as an optional parameter. Our program also includes an option to search the motifs in both strands of DNA.

2.3.3.1 Computing significance score for a motif consensus

To establish how significant a motif consensus is, we associate a significance score to it which represent the motif strength between 0 and 1. We calculated this score based on the nucleotide bases participating in the consensus building. Assume that the two given exact strings S_1 and S_2, of length l_1 and l_2, respectively, are present together separated by a gap g at all n different occurrences in a given set of DNA sequences, serving as the left and right anchors of a broken sequence motif. We attempt to build the consensus among all these n co-occurrences of the broken sequence motif. Assume that the motif consensus sequence is given by $C_0C_1C_2...C_{L-1}$, where L denotes the length of the motif. One such example is given below:

Let us assume that S_1 = ACT, S_2 = TAT, $g = 3$, and $n = 5$.

$\Rightarrow l_1 = 3$, and $l_2 = 3$

$\Rightarrow L = l_1 + g + l_2 = 3 + 3 + 3 = 9$

ACTTACTAT : motif sequence at occurrence 1

ACTTCATAT : motif sequence at occurrence 2

ACTTACTAT : motif sequence at occurrence 3

ACTCAGTAT : motif sequence at occurrence 4

ACTGGATAT : motif sequence at occurrence 5

ACTtaNTAT : Consensus motif sequence

Assuming $M_{i,j} \mid 0 \leq i < n$ and $0 \leq j < L$ represents an element (i.e. character) in a matrix M of nucleotides participating in the consensus building for a given motif m. The consensus sequence $C_0C_1C_2...C_{L-1}$ of motif m is built by choosing the most frequent (i.e. majority) nucleotide base at jth column of the matrix M; however in cases of ambiguity, we choose "N" as a wild-card character. We denote the significance

score of m as $Score(M)$. In order to compute the $Score(M)$ for a motif consensus m, we count the frequency of the character C_j in the jth column of the matrix M, and sum these counts together for all the characters in the consensus (i.e. $0 \leq j < L$) to get the count of significant characters in the matrix. In case of wild-card character "N" in the consensus, we add the frequency of any one of the characters causing ambiguity at the jth column of the matrix M to the count of significant characters, and, finally, divide this count by the total characters in the matrix. The mathematical formula for $Score(M)$ is given below:

$$\begin{aligned} Score(M) &= \frac{\text{Significant characters in M}}{\text{Total characters in M}} && (2.15)\\ &= \frac{\sum_{j=0}^{L-1}\sum_{i=0}^{n-1} Freq(C_j)}{nL} && (2.16)\\ &= \frac{\sum_{j=0}^{l_1+g+l_2-1}\sum_{i=0}^{n-1} Freq(C_j)}{n(l_1+g+l_2)} && (2.17)\\ &= \frac{l_1 n + \sum_{j=l_1}^{l_1+g-1}\sum_{i=0}^{n} Freq(C_j) + l_2 n}{n(l_1+g+l_2)} && (2.18)\\ &= \frac{n(l_1+l_2) + \sum_{j=l_1}^{l_1+g-1}\sum_{i=0}^{n-1} Freq(C_j)}{n(l_1+g+l_2)} && (2.19) \end{aligned}$$

2.3.3.2 Estimating p–value for a motif

In order to compute the probabilistic significance of a motif, we used Poisson probability distribution (as shown in Equation 2.4) to model the motif occurrences, and computed the P-Value, the probability that these occurrences of the motif have arisen by pure chance. For a motif having m occurrences, this P-Value is given below:

$$\begin{aligned} \text{P-Value} &= 1 - \text{Sum of probabilities of 0 to m-1 occurrences} && (2.20)\\ &= 1 - \sum_{i=0}^{m-1} P(\lambda, i) && (2.21)\\ &= 1 - \left[\frac{\lambda^0 e^{-\lambda}}{0!} + \frac{\lambda^1 e^{-\lambda}}{1!} + ... + \frac{\lambda^{m-1} e^{-\lambda}}{(m-1)!}\right] && (2.22) \end{aligned}$$

2.3.3.3 Computing uncertainty associated to a motif

As mentioned earlier, we have used Shannon's uncertainty to filter out the simple motifs. Assuming N_b is the count of the base α_b in a motif m_i, where $\alpha_b \in \{A, C, G, T\}$, and N is the sum of the counts of all four bases. The uncertainty $H(m_i)$ for a motif m_i is given below:

$$H(m_i) = -\sum_{b=A}^{T} P(\alpha_b) \log_2 P(\alpha_b) \quad (2.23)$$

$$= -\left[\frac{N_A}{N}\log_2\frac{N_A}{N} + \frac{N_C}{N}\log_2\frac{N_C}{N} + \frac{N_G}{N}\log_2\frac{N_G}{N} + \frac{N_T}{N}\log_2\frac{N_T}{N}\right] \quad (2.24)$$

We calculate the uncertainty associated to each motif in the list, and filter out the motifs which have uncertainty more than a given value, as provided by the user. This approach enables us to filter-out all simple sequences from the list of motifs.

2.3.4 Algorithm performance

Our algorithm to identify variable motifs works in several phases. Although it is very difficult to estimate its performance in a closed form, the running time of the algorithm is the sum of the processing of the suffix tree and the building of the consensus sequences of variable motifs. If we denote the number of sequences in the input by s, and their maximal length by L, then we need $O(sL)$ time to identify the original (anchoring) exact repeats. However, we also need to maintain records of all their positions. This adds another complexity factor of q, the total expected number of positions featuring an occurrence of a repeat. The estimation of q under the Poisson model is given below:

$$\text{Total patterns of i bases} = 4^i \quad (2.25)$$

$$\lambda = \frac{L}{4^i} \quad (2.26)$$

$$P(\lambda, j) = \frac{\lambda^j e^{-\lambda}}{j!} \tag{2.27}$$

$$\text{Patterns of i bases with j occurrences} = (\text{Total patterns})P(\lambda, j) \tag{2.28}$$

$$= 4^i \left(\frac{\lambda^j e^{-\lambda}}{j!}\right) \tag{2.29}$$

$$\text{Total positions of occurrences} = (\text{Occurrences})(\text{Patterns}) \tag{2.30}$$

$$\text{Positions of j occurrences of i bases} = j4^i \left(\frac{\lambda^j e^{-\lambda}}{j!}\right) \tag{2.31}$$

$$\text{Positions of all occurrences of i bases} = \sum_{j=2}^{sL} j4^i \left(\frac{\lambda^j e^{-\lambda}}{j!}\right) \tag{2.32}$$

$$= 4^i \left(\sum_{j=2}^{sL} j\frac{\lambda^j e^{-\lambda}}{j!}\right) \tag{2.33}$$

$$\text{Let all positions of pattern occurrences} = q \tag{2.34}$$

$$q = \sum_{i=2}^{L} 4^i \left(\sum_{j=2}^{sL} j\frac{\lambda^j e^{-\lambda}}{j!}\right) \tag{2.35}$$

$$= \sum_{i=2}^{L} 4^i \left(\sum_{j=2}^{sL} j\frac{\left(\frac{L}{4^i}\right)^j e^{-\frac{L}{4^i}}}{j!}\right) \tag{2.36}$$

$$= \sum_{i=2}^{L} 4^i \left(\sum_{j=2}^{sL} \frac{\left(\frac{L}{4^i}\right)^j e^{-\frac{L}{4^i}}}{(j-1)!}\right) \tag{2.37}$$

The expected number of repeats or patterns (each counted only once) is estimated using Equation 2.8, as shown below:

$$\text{Let expected number of patterns} = r \tag{2.38}$$

$$\text{Patterns of i bases with j occurrences} = 4^i \left(\frac{\lambda^j e^{-\lambda}}{j!}\right) \tag{2.39}$$

$$\text{Patterns of i bases with all occurrences} = \sum_{j=2}^{sL} 4^i \left(\frac{\lambda^j e^{-\lambda}}{j!}\right) \tag{2.40}$$

$$= 4^i \left(\sum_{j=2}^{sL} \frac{\lambda^j e^{-\lambda}}{j!}\right) \tag{2.41}$$

$$\text{All expected patterns} = \sum_{i=2}^{L} 4^i \left(\sum_{j=2}^{sL} \frac{\lambda^j e^{-\lambda}}{j!} \right) \tag{2.42}$$

$$r = \sum_{i=2}^{L} 4^i \left(\sum_{j=2}^{sL} \frac{\lambda^j e^{-\lambda}}{j!} \right) \tag{2.43}$$

$$= \sum_{i=2}^{L} 4^i \left(\sum_{j=2}^{sL} \frac{\left(\frac{L}{4^i}\right)^j e^{-\frac{L}{4^i}}}{j!} \right) \tag{2.44}$$

While it is difficult to express these estimates in a closed form, our simulations over relevant ranges have shown that r was bounded by sL, and q was slightly super-linear to sL (the function we compared it with was logarithmic).

Locating the mates and extending the repeat consensus is primarily dependent on r, since we need to make an attempt for each repeat in our list. If g is the number of positions we allow to separate exact repeats (while still considering them mate candidates), h is the number of repeats examined (and thus the upper bound to the number of mates themselves) and r_i is the number of occurrences of one repeat $\mathcal{R}$, the time necessary to examine the neighborhood at every occurrence of $\mathcal{R}$ is $\sum_{j=1}^{r_i} hs$. Although r_i can theoretically be as large as sL, in practice it would be around q/r on average. If $\mathcal{T}_0$ is the time necessary to recursively refine and extend the initial consensus whenever a co-occurring pair is found, the time necessary to handle the neighborhood of one repeat $\mathcal{R}$ is bounded by $gh\mathcal{T}_0 \sum_{j=1}^{r_i} hs$. As we look for variable motifs with only a very limited number of spaces between their conserved parts, we can consider g as a constant. Due to our indexing scheme (Figure 2.8), the same applies to h. However, for each repeat we still need to look at all of its occurrences. The estimate for $\mathcal{T}_0$ is similar to the time needed to build the original consensus, i.e. proportional to $gh\mathcal{T}_1 \sum_{j=1}^{r_i} hs$, where $\mathcal{T}_1$ is the complexity of further extensions. Since a large number of extensions would imply very large motifs, which are rare under the conditions in which this program has been designed to run, we can consider it as a

very small constant (2 to 3) on average. We shall denote this value by t. The total execution time for the location of mates and building consensuses for one repeat is thus $O((q/r)^t)$, on average. This cost would apply to all r repeats, so the total cost of forming initial motifs would be $O(r(q/r)^t)$.

The final refinement step will on average take time proportional to the final number of motifs. The creation of each consensus can potentially add another repeat to our list if these identified as mates also have separate occurrences that cannot be merged. Consequently, the number of motifs can grow as large as $O(r(q/r)^t)$. However, due to the re-indexing step before the refinement, this last step can be done in time proportional to the number of motifs. Since any motif that has been merged ceased to exist independently, multiple iterative refinements in this step cannot add to the complexity. The total time needed for the execution of our algorithm, on average, is thus $O(r(q/r)^t)$.

The space needed is dominated by that necessary to store all original repeats and the discovered consensus motifs, since the suffix tree uses space proportional to the number of its leaves, sL. As the number of motifs cannot exceed the time needed for their discovery, the space requirement is also $O(r(q/r)^t)$.

Usually t, the exponential factor in our estimates, will be very small (2 or 3), and q/r (the number of occurrences of an average repeat) is a small number, too, so the algorithm would be efficient. However, there are some real–world scenarios under which our program would not perform well. If the sequences examined consist of, for instance, ALU repeats or nearly identical genomic regions, this would result in many extensions, and consequently very large values of t. This would lead to poor performance in terms of both time and space. Fortunately, such sequences are simple to detect and filter before the program is applied, and we have not encountered this problem in practical runs.

2.4 Applications

In order to study the performance of our software, we looked at several eukaryotic datasets, including the CAVEOLIN cluster and MLL target genes.

Our first dataset was taken from the caveolin gene regions of 8 vertebrates. Caveolins encode integral membrane proteins that act as scaffolds to sequester and organize lipids and proteins in signaling complexes [36]. Their expression is highly regulated, and CAV1 and CAV2 are target genes for PPARγ. However, previous studies were unable to identify sequences matching the DR-1 consensus binding element (AGGTCAnnnAGGTCA) of the PPARγ/RXR heterodimer. Indeed, we have found only a fraction of this element, as motif AGGTCACNNAGC, repeated twice in the upstream sequences of CAV2 and CAV3 genes.

We first concentrated on the upstream regions of CAV1, CAV2 and CAV3 genes in the human sequence. With the shortest reportable motif length set to 5, consensus strength at 0.95 and the likelihood of occurring by chance at less than 0.01, there were 416 significant motifs occurring in at least 2 out of 3 sequences, and 126 occurring in all 3. With these numbers in mind, we were interested to determine if any of these motifs were phylogenetically conserved.

Table 2.4. The number of significant short variable motifs discovered in the (left) promoter sequences of 8 vertebrate CAV1 genes, and (right) 8 random synthetic 4-letter sequences, under various stringency levels: the columns represent the minimal number of motif occurrences and the rows show the minimal motif length.

	4/8	**5/8**	**6/8**	**7/8**	**8/8**	**4/8**	**5/8**	**6/8**	**7/8**	**8/8**
5	231	127	92	44	23	100	73	45	21	19
6	224	111	71	35	18	99	72	42	19	18
7	194	93	57	29	13	92	66	39	16	17
8	159	59	29	11	1	74	41	26	4	1

Table 2.5. Highest scoring repeated motifs found in the upstream sequences of caveolin genes. SH — significance in homology; SP — significance in paralogy; OC — occurrences, homology; OP — occurrences, paralogy

Consensus	**SH**	**SP**	**OH**	**OP**	**TRANSFAC hits**	**Hit strength**
CCccCC	0	4×10^{-5}	59	4	human SP1–4	100%
GGctcCC	2×10^{-16}	10^{-4}	39	3	human NF-Atp	100%
					rat MAPF2, YY1	100%
CAtccCT	3×10^{-15}	4×10^{-5}	28	5	human NIP, PEA3	100%
CCACAC	7×10^{-10}	6×10^{-4}	12	4	multiple	80%
CAGgGA	2×10^{-5}	10^{-4}	15	3	multiple	80%
GGgNGA	2×10^{-4}	10^{-7}	16	7	multiple	91.7%
ACttTT	8×10^{-4}	4×10^{-3}	12	6	multiple	80%

Using the Ensembl browser [35], we have extracted 8 sequences of length 500 located upstream of the Caveolin-1 gene in fugu, zebrafish, chicken, mouse, rat, dog, chimpanzee and human. The number of short motifs discovered under varying stringency levels (all highly significant) is depicted in Table 2.4, along with the number of significant motifs discovered in completely random synthetic sequences of 4 DNA letters, which served as a control.

Looking at the intersection of the sets of motifs discovered in the promoter regions of CAV1, CAV2 and CAV3 in the human sequence, on one side, and the motifs discovered in the promoter regions of CAV1 in 8 vertebrates, we again identified large sets under various stringency levels. When we imposed a requirement that a motif must be present in all 3 human genes, and in all 8 vertebrate CAV1, with minimal reportable length of 6, we identified 7 motifs shown in Table 2.5. The top two of these motifs had a perfect hit with the TRANSFAC [99] database, and the remaining could be found as weaker matches. Although our goal was solely to identify the short repeated variable elements in promoter regions of related genes, such TRANSFAC hits may help elucidate their role. The spatial distribution of these motifs in the

upstream regions of human caveolins is shown in Figure 2.9. In general, very few significant repeated motifs exhibit positional conservation, when they can be detected by multiple alignments, and they may thus serve for further refinement of the repeats from our lists.

In another test, we have applied our software to 1000bp of 7 putative target genes of the MLL transcriptional complex (Mixed Lineage Leukemia genes [58]). After filtering for poly-A and other simple sequences, including tandem repeats, our program has identified 27 significant motifs, including several that warrant further experimental study. In an effort to reduce the number of significant motifs, we have also applied our software to 500bp of same the 7 putative target genes and have successfully reduced the number of significant motifs to 11. The layout of these motifs is shown in Figure 2.10.

In another experiment, we considered 1000 bases flanking the 5 region of three genes, CYBB, HBG1 and HBZ (LocusLink Ids 1536, 3047 and 3050), which are co-regulated by the transcription factor CP1, and share the repeated motif CCAAT in their 5 flanking regions. Our software identified 18 significantly repeated short consensus motifs, and one of these consensus sequences has motif CCAAT. This consensus ranked 13^th out of 18 motifs. The layout of these motifs is shown in Figure 2.11 and the motif CCAAT is being represented by red colored star (number 13). However, when we used the BEST suite of programs [28] with same set of flanking regions, CCAAT was not even present in the top 25 motifs.

2.5 Discussion

Our software has, so far, performed well. It was finding the motifs efficiently, and it has shown good sensitivity. Many of the motifs it has identified were also present in the TRANSFAC database of experimentally confirmed sequences.Though

the number of motifs reported by our tool tends to be large, this is a common problem shared by all motif finders, and stems from the structure of genomes, or at least eukaryotic genomes. One weakness of our current approach is that it cannot identify motifs featuring insertions and deletions. While more than one or two such modifications of a consensus would probably result in a loss of binding affinity, permitting some may further improve the sensitivity of our software. For this reason, in one of our experiments done on yeast genes we have failed to recognize a well described common promoter. However, even our present algorithm detects an overabundance of significant motifs, and one can easily imagine that with permitted insertions and deletions this number would substantially increase.

While working with MotFi and carrying out the study described in this chapter, we have observed that any given selection criteria returned large numbers of motifs with extremely significant p-values. Though this observation was a bit puzzling, it also motivated us to perform an extensive study of the repetitive structure and distribution of short motifs in human genomic sequences, in an attempt to know why every motif finding software often failed in locating the real elements. In this study [6] we have described and quantified the remarkable micro-repetitive structure of the human genome, and shown that it is impossible to reliably identify functional motifs solely based on the occurrence counts, even after known repeats have been excluded. This study is described in the next chapter.

Algorithm 2.3.2: IDENTIFICATION OF VARIABLE MOTIFS()

```
// Build the suffix tree
Initialize the tree to empty
for i ← 1 to s          // s: the number of input sequences
  do {Append the branches corresponding to sequence i to the tree
// Find the exact repeats
Traverse the tree depth-first and collect the terminal points
List position and repeat text of each branching internal node
// Build the Index
for i ← 1 to num_repeats
  do { for j ← 1 to num_positions[i]
       do {Create an index array entry at position[j]
// Identify variable motifs by their consensus sequences
for i ← 1 to num_repeats
  do { // Create a list of motifs
       CREATE_CONSENSUS(i)          // Function call listed below
Merge the list of consensus motifs with simple repeats
Rebuild the index of repeat (motif) position
// Refine the list of identified motifs
for i ← 1 to num_motifs
  do { repeat
         Attempt to merge neighbors of i in a single consensus
       until no further refinement possible
// Report the significant motifs
Filter the motifs according to significance and other criteria
```

Algorithm 2.3.3: CREATE_CONSENSUS(*seed*)

```
for i ← 1 to max_gap_size
  do { // Find repeated neighbors for i      (fixed gap size)
       for j ← 1 to num_positions[seed]
         do { List the neighbors of seed at j using the index
              for each neighbor N occurring more than once
                do { Build consensus C of seed and N
                     CREATE_CONSENSUS(C)      // Recursive call
```

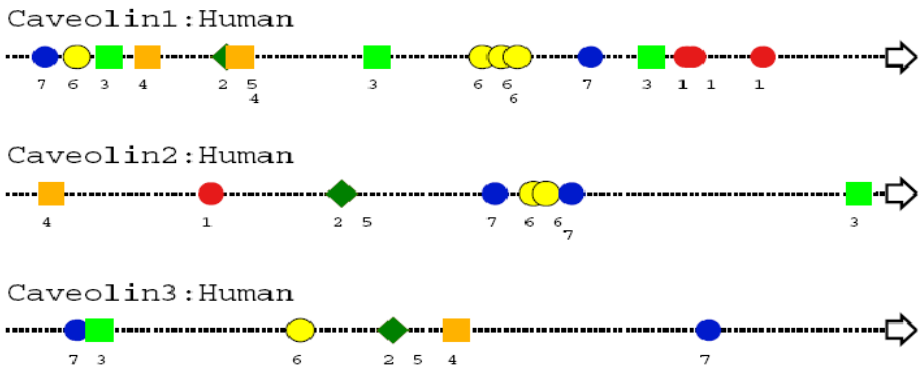

Figure 2.9. The layout of motifs in the upstream sequences of caveolin genes. The overrepresented motifs shown in these sequences are 1. CCccCC, 2. GGctcCC, 3. CAtccCT, 4. CCACAC, 5. CAGgGA, 6. GGgNGA, and 7. ACttTT.

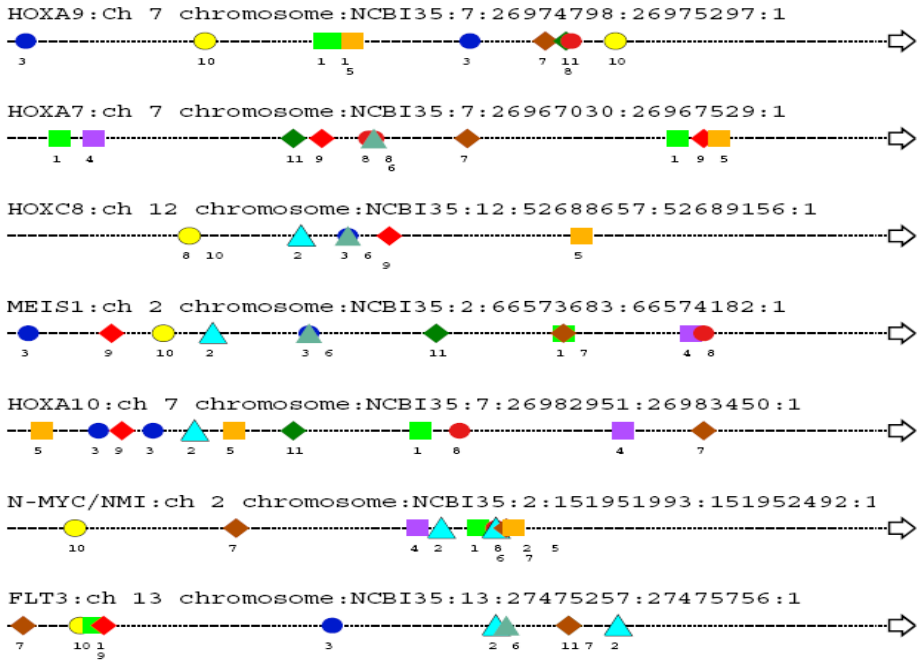

Figure 2.10. The layout of motifs in the upstream sequences of MLL direct target genes. The overrepresented motifs shown in these sequences are 1. CCAgacCAG, 2. CCAgccCTG, 3. GCCgCCA, 4. ATTNcaGGG, 5. GGGNcaTCT, 6. GCCaNCAC, 7. TGGaAGG, 8. AGCcAGC, 9. GCCtGGG, 10. CTCaCCA, and 11. CTGcAAG.

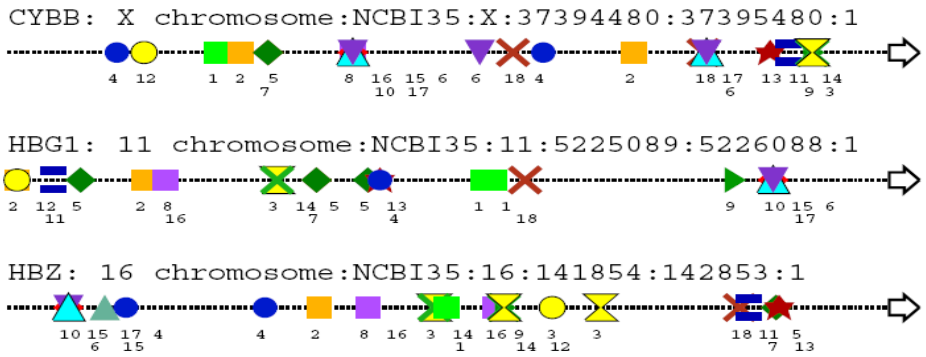

Figure 2.11. The layout of 18 motifs in the upstream sequences of CYBB, HBG1 and HBZ genes which are co-regulated by CP1 factor. The overrepresented motifs shown in these sequences are 1. CTAAaccTTG, 2. TCActaTGT, 3. CAGggCTG, 4. CCAgcCTG, 5. CTCNccTGA, 6. TGGctCAT, 7. CCTCaNcTGA, 8. ACATtaGGT, 9. CAGGgcTGC, 10. GTGGcTCAT, 11. TCTGATAA, 12. ACTGNgNCCT, 13. TGACCAAT, 14. CAGgGCTG, 15. GTGGcTCA, 16. CATNNGGT, 17. TGGcTCAT, and 18. TTGttACT.

CHAPTER 3
MICRO-REPETITIVE STRUCTURE AND DISTRIBUTION OF SHORT MOTIFS IN HUMAN GENOMIC SEQUENCES

As we have discussed in the previous chapter, our every attempt to locate significantly repeated DNA elements returned a long list with extremely significant p-values. In order to achieve better insight into such behavior of human DNA sequences, we have performed an extensive study of the structure and distribution of short motifs in human genomic sequences, and shown that in addition to known long repeated DNA sequences human genome features many short motifs which are significantly over-represented, and that their frequency varies only slightly between random repeat–masked sequences and regions located immediately upstream of the known genes [6, 4]. In this study we have done extensive simulations and analysis of real data in order to identify the short motifs conservation pattern in the human genome, particularly in the ENCODE target regions. While the number of repeated elements in randomly generated synthetic sequences was almost perfectly conforming to the Poisson expectation, the number of repeated substrings in repeat–masked random intergenic sequences was far greater than expected. This bias appears to be genome–wide, as it persisted even when we simultaneously considered many additional randomly collected human sequences, varying in size between 100 and 4,000 characters. Consequently, any search for conserved motifs is bound to return many results, and, depending on what we search for, most would likely be false positives. In order to gain a better perspective on our ability to characterize significant over-represented motifs in different regions of the human genome, we have looked at the number of short (less than 20 bp) repeats, both exact and approximate in various ge-

nomic environments and synthetic sequences. Although studies have been performed regarding the distribution of tandem repeats [61] and larger interspersed repeats [47], we are unaware of any systematic examination of the genome–wide occurrences of very short interspersed motifs.

3.1 Study of the distribution of short exact repeated motifs

This section includes a description of our datasets – both simulation and real genomic, the algorithm for counting the exact motifs in these datasets, and the analysis of the distribution of these motifs.

3.1.1 Datasets

To start with the analysis, we have created 6 different datasets, each consisting of 100 sequences of 500 bases in length. Although we looked at other possible segment lengths, as short as 50, and as long as several thousand, the results on short exact sequences were not substantially different and length 500 was well suited for the consideration of regions immediately 5' to known genes. Although the issue is still unresolved, some studies have shown that most *cis*–acting regulatory elements appear to cluster in the gene upstream regions of about this length [93]. Four of our datasets were synthetic, containing sequences created by assembling A's, C's, G's, and T's using a random number generator on Unix, and sequences generated by second, third, and fifth order Markov Models (MMs), trained on one million bases taken from human chromosome 2 obtained through the Ensembl [35] genome browser. These specific MMs have been selected because the second and the third order are widely used in the simulation of genetic sequences, and the fifth order is popular in gene–finding tools. We were especially interested in the behavior of the second order Markov Model, as it has been used to generate control sequences in a comprehensive evaluation of motif–

Table 3.1. Variables used in Algorithm 3.1.1

$MotifLength$	Length of a motif string
$Sequence$	Input sequence character array
$Hash$	Variable for the hash code of a motif string
$LoPos$	Array index in $Sequence$ corresponding to the right end of a motif string
$HiPos$	Array index in $Sequence$ corresponding to the left end of a motif string
$HiFact$	Variable for the factor corresponding to the left most character of a motif string
$MotifList$	Hash table (array) to store motif counts

finding tools [63]. Strings generated by even higher order MMs were considered in order to confirm trends, but not studied in detail. The remaining two datasets were real DNA sequences: one was constructed from the upstream regions immediately 5' to annotated Ensembl human genes, and the other consisted of random repeat–masked human intergenic sequences. The total length of sequences in each dataset was 50,000 bases (300,000 letters total).

3.1.2 Algorithm for counting exact motifs

In order to count short exact repeated oligonucleotides we used a modification of the Karp–Rabin pattern matching algorithm [88], locating all repeats of specified length in time linear with the size of the sequence. The original Karp–Rabin method was based on numerical keys to code patterns, and we used such keys as indices to a hash table counting the number of occurrences of each motif. The pseudo-code of this modified approach is shown in Algorithm 3.1.1, and the list of variables used in this algorithm is given in Table 3.1. This algorithm returns an array where array indices are codes or numerical keys to the motifs of a given length, and corresponding array values are counts of the motifs occurrences. As we scan a given genomic segment to count the motifs, we increase the array values at the indices corresponding to the motifs.

Algorithm 3.1.1: MODIFIED RABIN-KARP($MotifLength$, $Sequence[0..n-1]$)

```
Hash ← 0
HiPos ← 0
LoPos ← MotifLength - 1
//Compute initial hash
for i ← HiPos to LoPos
  do { if (Sequence[i] == 'A') Hash ← Hash*4
       else if (Sequence[i] == 'C') Hash ← Hash*4 + 1
       else if (Sequence[i] == 'G') Hash ← Hash*4 + 2
       else if (Sequence[i] == 'T') Hash ← Hash*4 + 3
MotifList[Hash] ← MotifList[Hash] + 1
HiFact ← 1
for i ← 1 to MotifLength − 1
  do {HiFact ← HiFact*4
while Sequence[LoPos + 1] != NULL
  do { if (Sequence[HiPos] == 'A') Hash ← Hash
       else if (Sequence[HiPos] == 'C') Hash ← Hash - HiFact
       else if (Sequence[HiPos] == 'G') Hash ← Hash - 2*HiFact
       else if (Sequence[HiPos] == 'T') Hash ← Hash - 3*HiFact
     HiPos ← HiPos + 1
     LoPos ← LoPos + 1
     if ( Sequence[LoPos] == 'A') Hash ← Hash*4
       else if (Sequence[LoPos] == 'C') Hash ← Hash*4 + 1
       else if (Sequence[LoPos] == 'G') Hash ← Hash*4 + 2
       else if (Sequence[LoPos] == 'T') Hash ← Hash*4 + 3
     MotifList[Hash] ← MotifList[Hash] + 1
return (MotifList)
```

3.1.3 Results and analysis

We ran our program separately for motif lengths varying between 4 and 9, and recorded the total numbers of repeated elements in Table 3.2. Since the repeats have been counted separately for each of the 100 sequences in each dataset, the recorded values include the mean (μ) and the standard deviation (σ) for all runs. In addition to the empirically determined counts we have also recorded the expected numbers of the repeats, based on the Poisson model. As it can be seen from the Table 3.2, there were

Table 3.2. The mean numbers (μ) and standard deviations (σ) of repeated patterns of different lengths in different types of nucleotide sequences. Pattern counting has been done over 100 sequences of length 500 in each category.

Pattern Length	Expected Number	Random Synthetic	2^{nd} Order Markov M.	3^{rd} Order Markov M.	5^{th} Order Markov M.	Random Genomic	Upstream Regulatory
4	429.06	$\mu = 425.74$	$\mu = 437.99$	$\mu = 432.84$	$\mu = 432.23$	$\mu = 438.97$	$\mu = 433.92$
		$\sigma = 6.36$	$\sigma = 8.12$	$\sigma = 7.1$	$\sigma = 6.91$	$\sigma = 8.5$	$\sigma = 9.94$
5	193.16	$\mu = 189.18$	$\mu = 237.83$	$\mu = 222.98$	$\mu = 222.27$	$\mu = 261.64$	$\mu = 260.11$
		$\sigma = 15.59$	$\sigma = 17.0$	$\sigma = 16.68$	$\sigma = 15.83$	$\sigma = 33.49$	$\sigma = 30.67$
6	57.46	$\mu = 55.16$	$\mu = 84.33$	$\mu = 74.58$	$\mu = 75.88$	$\mu = 106.62$	$\mu = 115.31$
		$\sigma = 12.51$	$\sigma = 15.16$	$\sigma = 13.27$	$\sigma = 14.66$	$\sigma = 43.5$	$\sigma = 37.72$
7	15.03	$\mu = 14.0$	$\mu = 24.5$	$\mu = 21.82$	$\mu = 23.3$	$\mu = 38.66$	$\mu = 47.54$
		$\sigma = 5.77$	$\sigma = 9.97$	$\sigma = 7.81$	$\sigma = 9.48$	$\sigma = 44.31$	$\sigma = 29.88$
8	3.8	$\mu = 3.12$	$\mu = 7.05$	$\mu = 5.75$	$\mu = 6.87$	$\mu = 15.72$	$\mu = 21.3$
		$\sigma = 2.75$	$\sigma = 5.15$	$\sigma = 4.16$	$\sigma = 5.19$	$\sigma = 44.26$	$\sigma = 21.62$
9	0.95	$\mu = 0.56$	$\mu = 1.94$	$\mu = 1.47$	$\mu = 1.97$	$\mu = 8.57$	$\mu = 11.33$
		$\sigma = 1.17$	$\sigma = 2.42$	$\sigma = 1.92$	$\sigma = 2.25$	$\sigma = 44.04$	$\sigma = 15.67$

only insignificant differences between the models for motifs of length 4[1]. Starting with length 5, an obvious pattern emerges, in which the number of repeats in sequences created by the random number generator correlates with Poisson predictions very well, but none of the other models do. Figure 3.1 shows graph plots between the mean numbers (μ) of repeated patterns and the pattern lengths in different types of nucleotide sequences. Though the visual look of these plots gives an impression of similarity among all the datasets, a statistical analysis does not confirm that. We used chi-square test on the μ values in the columns of Table 3.2 to measure the similarity or correlation between each pair of datasets.

The chi-square test allows us to detect whether two random variables are related, or a model has good fit with the data. To start with, we assert a null hypothesis that the model does fit the data or two variables are related, and we attach a level of

[1]However, when the individual numbers of motifs occurring a particular number of times were taken into account there were considerable differences between the models, as outlined below.

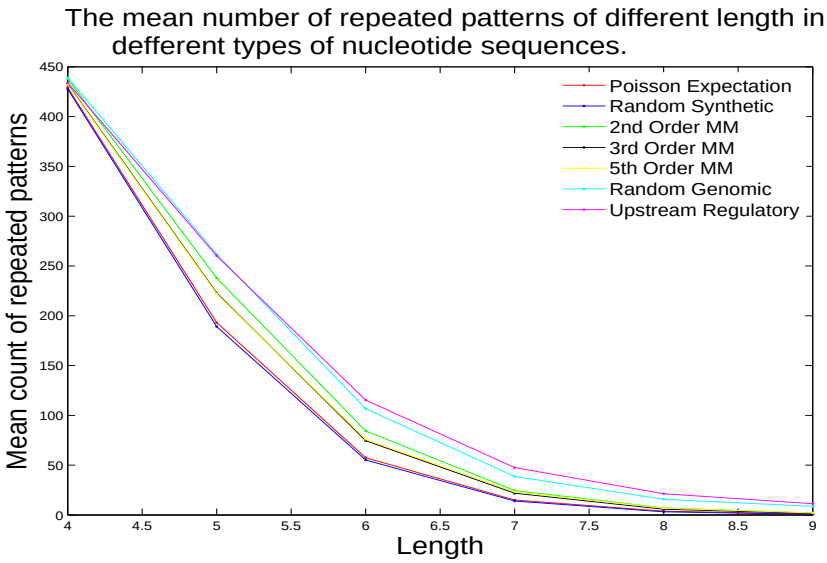

Figure 3.1. The mean numbers (μ) of repeated patterns of different lengths in different types of nucleotide sequences. Pattern counting has been done over 100 sequences of length 500 in each category.

significance to reject this hypothesis. Assuming there are k intervals or categories in a dataset, each with e_i expected value and x_i data value where i varies from 1 to k, the chi-square χ^2 is given by following equation:

$$\chi^2 = \sum_{i=1}^{k} \frac{(e_i - x_i)^2}{e_i} \tag{3.1}$$

With Chi Square, we calculate a value from the data using the above equation, and then compare this value to a critical value from a Chi Square table with degrees of freedom corresponding to that of the data. If the calculated value is equal to or greater than the critical value (table value), the null hypothesis is rejected. If the calculated value is less than the critical value, the null hypothesis is accepted. The degrees of freedom for a dataset is given by the following criteria:

$$\text{Degrees of Freedom} = (\text{No. of Categories in Datasets})\text{- }(\text{No. of Datasets}) \tag{3.2}$$

$$\Rightarrow = k - 1 \quad (\ldots \text{ No. of datasets is normally 1.}) \tag{3.3}$$

Table 3.3. Chi–square confidence levels for the compared data sets, indicating the likelihood that sequences in the compared set pairs (column-wise in Table 3.2) have indeed been drawn from the same distribution. MMn abbreviates n^{th} order Markov model.

	Expected Number	Random Synthetic	2^{nd} Order Markov M.	3^{rd} Order Markov M.	5^{th} Order Markov M.	Random Genomic	Upstream Regulatory
Expected	1.0	> 0.995	< 0.005	≈ 0.02	< 0.005	$<< 0.005$	$<< 0.005$
Random	> 0.995	1.0	< 0.01	≈ 0.2	≈ 0.1	$<< 0.005$	$<< 0.005$
MM2	< 0.005	< 0.01	1.0	≈ 0.6	≈ 0.8	≈ 0.025	< 0.005
MM3	≈ 0.02	≈ 0.2	≈ 0.6	1.0	> 0.995	< 0.005	< 0.005
MM5	< 0.005	≈ 0.1	≈ 0.8	> 0.995	1.0	< 0.005	< 0.005
Genomic	$<< 0.005$	$<< 0.005$	≈ 0.025	< 0.005	< 0.005	1.0	≈ 0.8
Regulatory	$<< 0.005$	$<< 0.005$	< 0.005	< 0.005	< 0.005	≈ 0.8	1.0

Indeed, a chi-square test on the columns of Table 3.2 (μ values), whose results are shown in Table 3.3, confirmed with very high confidence that random synthetic sequence draws from the same distribution as Poisson prediction, but rejected other datasets (except, weakly, the higher order MMs). There were more repeats than expected in all Markov Models and they corresponded well to each other, confirmed by solid p–values. A weak similarity has also been found between the second order Markov Model and random intergenic sequences. This, on one hand, justifies its use in modeling genomic environments, but it also advises caution concerning the use of the MMs in simulations. It was surprising, and somewhat discouraging for the attempts of locating functional elements through over-representation, that random intergenic repeat–masked (and thus, presumably, reasonably unique) sequences featured about the same number of short repeated motifs as sequences taken upstream of the genes. The chi–square test was conclusive on this, with the p–value indicating a strong agreement. As for the correspondence between the real sequences and the models, random genomic sequences appear to have somewhat similar number of repeats as the second order Markov Model, but are otherwise quite distinct from any other

simulated dataset. Overall, real genomic sequences were similar to each other, but not to the models, Markov Models mutually agreed well, but otherwise did not show significant similarity to other datasets, and synthetic sequences corresponded well to the Poisson prediction (a "sanity check"), but their composition was different than that of Markov Model simulated data, and dramatically different than that of the real sequences.

We next analyzed these relationships at a finer granularity, looking separately at each motif length, and for each length separately at the number of motifs repeating n times, where n was varied between 2 and 10 or more (the latter counted together). Although we did full analysis for all motif lengths in our range, the results were similar, and we show the representative sample for motif lengths 4, 7, and 9 in Table 3.4. The graph plots between the mean numbers (μ) of patterns and repeated times (n, where n varies from 2 to 10) for different types of nucleotide sequences of lengths 4, 7, and 9 are shown in Figures 3.2, 3.3, and 3.4 respectively. As before, the sequences generated by using random numbers corresponded to Poisson predictions consistently well, while there was a discrepancy between these two and all other models. There was a somewhat weak mutual agreement between different Markov Models (at least two of the three corresponded to each other in every test), and occasionally between random genomic and gene upstream sequences, but the fit between the synthetic and real data was consistently poor.

In this round of testing we have applied the chi–square test on all combinations of models, separately for each motif length, using the sums of the number of repeats in 100 runs as our samples x_i, where i corresponded to the number of times the motifs have been repeated (so, for instance, in the test for motifs of length 7, x_3 was the count of motifs of length 7 repeated 3 times). This method provided us the sufficient sample size in each category i, which could have otherwise been a problem, having in mind

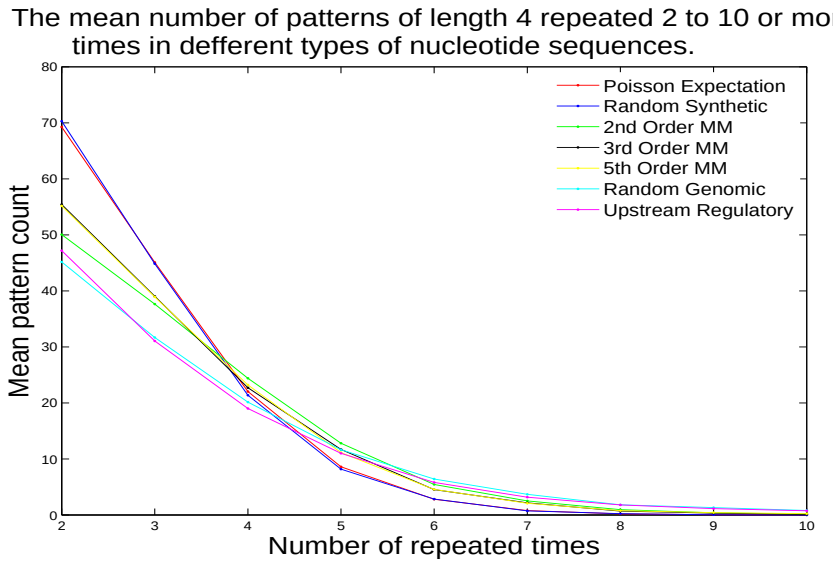

Figure 3.2. The mean numbers (μ) of patterns of length 4 repeated n times in different types of nucleotide sequences, where n varies between 2 and 10 or more. Pattern counting has been done over 100 sequences of length 500 in each category.

the relative scarcity of long exact motifs repeated many times. Unfortunately, for all comparisons except these involving Poisson expectations we needed to estimate the expected values from the data, and thus substantially reduce the number of degrees of freedom, which has made our analysis of longer repeats somewhat unreliable.

Interestingly, while there was a good agreement in the repeat distribution in random genomic and gene upstream sequences for motifs of length 4, the chi–square test failed for every other length. As it can be seen from Table 3.4(Figures 3.3, 3.4) for lengths 7 and 9, gene upstream sequences appear to feature a preference to an increased number of moderately repeated motifs, while random genomic sequences are biased towards smaller numbers of these repeated more dramatically (5 or more times). This pattern was consistent for all considered motif lengths, however the fewer motifs of higher repeat count compensated for the lower number of moderately repeated motifs, resulting in an overall similarity in the overall number of repeated sequences throughout our test sets, regardless of their proximity to the genes. Under

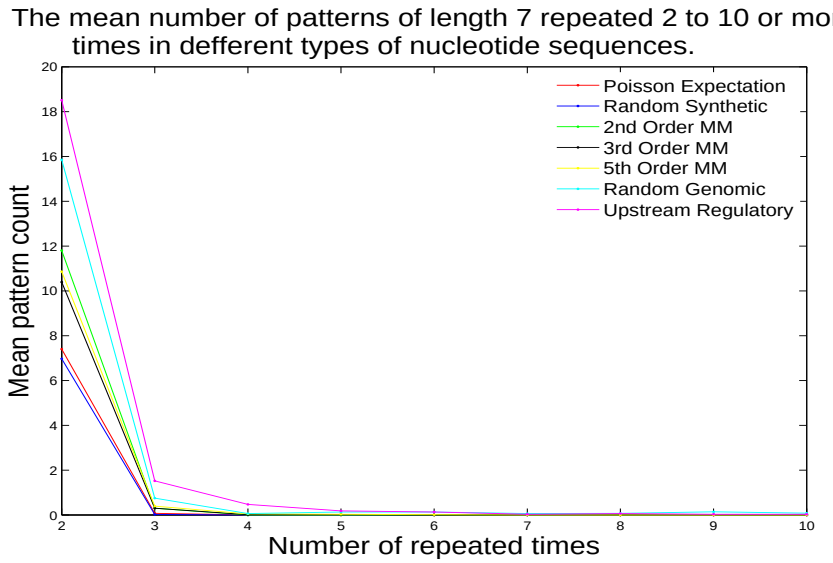

Figure 3.3. The mean numbers (μ) of patterns of length 7 repeated n times in different types of nucleotide sequences, where n varies between 2 and 10 or more. Pattern counting has been done over 100 sequences of length 500 in each category.

any circumstances, the number of short repeated motifs in the genomic sequences was greater than in any of the synthetic models, and far greater (an order of magnitude for longer motifs) than the Poisson expectations.

3.2 Analysis of most common short degenerate motifs

After experiencing this dramatic micro–repetitive structure of the human genome we were interested to find the most common short motifs significantly repeated in the ENCODE regions. Although the program used above was capable of locating short exact motifs in sequences of any length, in linear time, concentrating on perfect conservation appeared to be too restrictive. We have thus used our tool for finding short variable repeated motifs, described in detail in [6].

Briefly, our software starts by locating all exact repeated patterns in given sequences, including dinucleotides. This seeding step is done in time slightly super–linear to the length of the sequences, using the suffix tree data structure [82], which

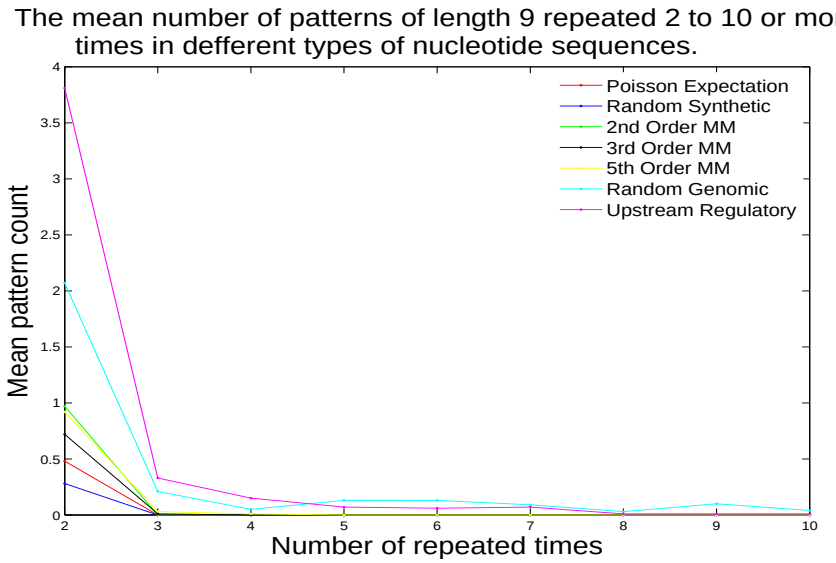

Figure 3.4. The mean numbers (μ) of patterns of length 9 repeated n times in different types of nucleotide sequences, where n varies between 2 and 10 or more. Pattern counting has been done over 100 sequences of length 500 in each category.

has recently been applied to problems similar to ours [41, 43]. After building the original list of repeats, we use an indexing scheme to quickly locate all neighbors of a given (seed) motif, and search for all pairs that appear to be substantially repeated together, at a fixed distance. Whenever such pairs are found we build the tentative consensus of the approximate motif, and recursively try to extend it with additional seed elements and overlaps. The consensus building continues until a certain quality threshold, usually set to 0.9 (90%) or 0.95, cannot be maintained any longer. We label all positions of absolute conservation with uppercase letters, and assign them the weight based on the number of sites participating in the construction of the consensus. Positions featuring a majority character, but occasionally broken with a mismatch are labeled with a lowercase character, whose weight is determined based on the number of sites which agree. When there is no agreement at a position it is signified by character 'N'. The final consensus motif is reported based on the probabilistic evaluation of its length, weight, and the number of occurrences.

Since the program assumes that it has been given a set of sequences, rather than a single one, it considerably reduces the search space by filtering out the motifs which do not appear in the minimal number of distinct segments (a settable parameter). Unfortunately, when the sequences in the input are very similar (such as vertebrate ultra-conserved sequences, or even just homologous sequences from closely related species), this causes a theoretically exponential explosion of the recursive refinement step. However, such situations are rare (after repeat masking, most intergenic sequences do not exhibit good conservation of motifs longer than about a dozen bases), and easily detectable. On average, our software is capable of locating all significantly repeated variable motifs quickly and accurately.

We ran this motif–detection program on the entire set of ENCODE regions, obtained through Ensembl, after masking the repeats. The repeat masking step was done since there was little purpose in trying to find common short repeated motifs in the presence of known long repeat elements. We performed 150,000 runs on 5 to 10 randomly chosen segments of length 1000, setting the program parameters so that only elements which have been found in all segments were reported. We have not excluded known exons from our test data — it simplified the selection and, since exons generally comprise less than 2% of the human genome we believed that they would not significantly affect our results.

Although we recorded only motifs of length 7 and above, their number was in the thousands even after filtering these which were nearly identical or inverse complements of each other, and these featuring extremely simple sequence (all A's, for instance) or tandem repeats. All these motifs do deserve further classification, but at this time we have limited our study only to about two dozen which were statistically least likely to occur by chance. Since our repeat masked ENCODE sequences contained 40,645,510

bases, counting both strands, in a completely random string of this length a motif of size 10, for instance, would be expected to be found about 39 times. We used such considerations as a basis for the selection of the top choices, where the effective length of the string was calculated by assigning different weights to differently conserved positions (1 for an uppercase letter, 0.5 for lowercase and 0 for an 'N' — this could have been further refined by using the exact weights of the identified motifs, but for this study that was not necessary).

In order to do a tentative characterization of the discovered motifs, we have checked them against the human entries in RepBase [52] for possible membership in a known repeat family, and TRANSFAC [99] for a possible functional role. Table 3.5 summarizes this information for 4 longest motifs in our list (the fifth one of that size was $(CTG)_4$, which we filtered out), and Table 3.6 provides the same account for the top 5 motifs after these with two or less G's or C's were removed. The remaining top motifs were either degenerated poly-A's (or poly-T's), or a combination of A's and T's. Although they are also potentially significant, we have not studied them in detail, since poly-A tails are known to be present in many copies in genomic sequences. One explanation for their prevalence is in that they are derived from the terminus of non-LTR retrotransposon repeats. These elements are abundant in the human genome (over 2.3 million copies spanning over one third of the genome) and they are characterized by a stretch of poly-A's at their 3' end [47]. Because of its variable length and rapid mutational degradation, part or all of the 3' poly-A terminus of non-LTR retrotransposons may often remain after repeat masking.

Even as the number of matches our top motifs had in RepBase generally exceeded what would be expected by chance only, these hits were not concentrated in a single repeat class, and thus probably do not represent remnants of a particular mobile element, at least not one of a known classification. Similarly, their TRANS-

FAC matches do not appear to lend strong support to the hypothesis that they may be functional protein binding sites — the examined sequence was human, but most of the hits were in non–human elements, or elements which are common in repeated sequences throughout the mammalian lineage (or even broader). While these motifs are clearly strongly repetitive, and some also likely functional, further studies are needed in order to characterize their nature and origins.

3.3 Discussion

The micro–repetitive structure of the human genome we have described above advises us caution when using over-representation for the determination of functional DNA elements. At present, most motif–finding tools do not solely rely on a single motif frequency, taking into account other features of biological relevance, such as clustering of the motifs, matching against experimentally confirmed consensus patterns, or evolutionary conservation. While each of these approaches has merits, they all have weaknesses. Clustering of over-represented motifs may very well be a consequence of their common source in an ancient repeat (i.e. transposed sequence) not recognized by the RepeatMasker or other repeat–finding tools, such as RepeatScout [14]. Almost all of the most frequent motifs we looked at in ENCODE had hits in RepBase, and often in TRANSFAC, too. On the other hand, methods based on phylogenetic footprints may be overly dependent on positional conservation. The ultimate classification of any DNA segment must be done in the laboratory, and the construction of a detailed map of the repetitive landscape of the genome can be of great help in this process.

The fact that for exact motifs of length 5 or more there was no good chi–square agreement between random genomic and gene upstream sequences is potentially significant. The noticed bias towards more copies of single motifs in intergenic regions may indicate the same origin of the sequences, presumably by many overlapping in-

sertions of DNA mobile elements, but different selection pressures after some of the regions acquired a functional role. Under this scenario, further insertions, which would lead to even more random motif copies in non-functional segments would be harmful in regulatory regions, and thus selected against. TRANSFAC hits in organisms other than human would also point in this direction, as parts of ancient repeats shared among the species may have acquired function in some, but not all, of them. Even if database lookups for functional patterns often result in a large number of false positives, the fact that not every one of our top motifs had a match indicates that a selection of sensible candidates for further study is possible. Our study could have been made more complete in many ways, as we have realized that we needed to map the discovered motifs back to their original genomic locations, looking at their groupings and experimental evidence. If most of the short frequent motifs indeed originate in layered ancient transpositional activity, some of them will show significant patterns of co-occurrence. However, we may have been too far reaching when taking on the analysis of the human genome first. It would be worthwhile to also look at a simpler genome, like that of a worm, and check if one could record similar patterns there.

The study described in this chapter has shown an over-abundance of short motifs in human genomic sequences, but the source of this abundance remained unclear, which motivated us to carry out further study. We have postulated that most of these DNA elements originate in ancient transpositional activity, with copies becoming so broken over time that they cannot be recognized as such any more, and our efforts to confirm this hypothesis are described in the next two chapters.

Table 3.4. The mean numbers (μ) and standard deviations (σ) of repeated patterns of length 4, 7 and 9 in different types of nucleotide sequences. For each motif length, the corresponding rows represent the numbers of motifs repeated n times, where n varies between 2 and 10 or more. Pattern counting has been done over 100 sequences of length 500 in each category.

Length/ Repeats	Expected Number	Random Synthetic	2^{nd} Order Markov M.	3^{rd} Order Markov M.	5^{th} Order Markov M.	Random Genomic	Upstream Regulatory
4/2	69.25	$\mu = 70.28$	$\mu = 50.05$	$\mu = 55.42$	$\mu = 55.16$	$\mu = 45.17$	$\mu = 47.19$
		$\sigma = 6.2$	$\sigma = 6.97$	$\sigma = 6.77$	$\sigma = 7.94$	$\sigma = 8.38$	$\sigma = 9.42$
4/3	45.09	$\mu = 44.86$	$\mu = 37.64$	$\mu = 39.04$	$\mu = 38.93$	$\mu = 31.68$	$\mu = 31.05$
		$\sigma = 5.84$	$\sigma = 5.53$	$\sigma = 5.32$	$\sigma = 4.99$	$\sigma = 8.1$	$\sigma = 7.69$
4/4	22.02	$\mu = 21.36$	$\mu = 24.39$	$\mu = 22.7$	$\mu = 23.21$	$\mu = 20.13$	$\mu = 19.0$
		$\sigma = 3.9$	$\sigma = 4.34$	$\sigma = 4.7$	$\sigma = 4.37$	$\sigma = 4.73$	$\sigma = 5.27$
4/5	8.6	$\mu = 8.17$	$\mu = 12.8$	$\mu = 11.69$	$\mu = 11.12$	$\mu = 11.65$	$\mu = 11.0$
		$\sigma = 2.8$	$\sigma = 3.18$	$\sigma = 3.14$	$\sigma = 3.08$	$\sigma = 3.12$	$\sigma = 3.46$
4/6	2.8	$\mu = 2.83$	$\mu = 5.44$	$\mu = 4.52$	$\mu = 4.56$	$\mu = 6.41$	$\mu = 5.81$
		$\sigma = 1.73$	$\sigma = 2.34$	$\sigma = 1.83$	$\sigma = 1.86$	$\sigma = 2.38$	$\sigma = 2.28$
4/7	0.78	$\mu = 0.73$	$\mu = 2.51$	$\mu = 2.16$	$\mu = 2.09$	$\mu = 3.67$	$\mu = 3.17$
		$\sigma = 0.8$	$\sigma = 1.47$	$\sigma = 1.38$	$\sigma = 1.39$	$\sigma = 2.02$	$\sigma = 1.9$
4/8	0.19	$\mu = 0.23$	$\mu = 0.94$	$\mu = 0.69$	$\mu = 0.77$	$\mu = 1.8$	$\mu = 1.79$
		$\sigma = 0.47$	$\sigma = 1.01$	$\sigma = 0.81$	$\sigma = 1.0$	$\sigma = 1.39$	$\sigma = 1.37$
4/9	0.04	$\mu = 0.03$	$\mu = 0.38$	$\mu = 0.33$	$\mu = 0.41$	$\mu = 1.32$	$\mu = 1.13$
		$\sigma = 0.17$	$\sigma = 0.64$	$\sigma = 0.57$	$\sigma = 0.62$	$\sigma = 1.43$	$\sigma = 1.35$
4/10+	0.01	$\mu = 0$	$\mu = 0.16$	$\mu = 0.12$	$\mu = 0.23$	$\mu = 0.79$	$\mu = 0.74$
		$\sigma = 0$	$\sigma = 0.39$	$\sigma = 0.32$	$\sigma = 0.51$	$\sigma = 1.56$	$\sigma = 1.09$
7/2	7.4	$\mu = 6.97$	$\mu = 11.8$	$\mu = 10.39$	$\mu = 10.86$	$\mu = 15.85$	$\mu = 18.51$
		$\sigma = 2.87$	$\sigma = 4.85$	$\sigma = 3.64$	$\sigma = 4.12$	$\sigma = 6.58$	$\sigma = 9.25$
7/3	0.07	$\mu = 0.02$	$\mu = 0.3$	$\mu = 0.3$	$\mu = 0.39$	$\mu = 0.75$	$\mu = 1.52$
		$\sigma = 0.14$	$\sigma = 0.61$	$\sigma = 0.59$	$\sigma = 0.72$	$\sigma = 1.62$	$\sigma = 2.7$
7/4	0.001	$\mu = 0$	$\mu = 0$	$\mu = 0.01$	$\mu = 0.05$	$\mu = 0.06$	$\mu = 0.47$
		$\sigma = 0$	$\sigma = 0$	$\sigma = 0.01$	$\sigma = 0.22$	$\sigma = 0.24$	$\sigma = 1.11$
7/5	0	$\mu = 0$	$\mu = 0$	$\mu = 0.02$	$\mu = 0.03$	$\mu = 0.13$	$\mu = 0.18$
		$\sigma = 0$	$\sigma = 0$	$\sigma = 0.14$	$\sigma = 0.17$	$\sigma = 0.91$	$\sigma = 0.52$
7/6	0	$\mu = 0$	$\mu = 0$	$\mu = 0$	$\mu = 0.01$	$\mu = 0.13$	$\mu = 0.13$
		$\sigma = 0$	$\sigma = 0$	$\sigma = 0$	$\sigma = 0.01$	$\sigma = 1.01$	$\sigma = 0.44$
7/7	0	$\mu = 0$	$\mu = 0$	$\mu = 0$	$\mu = 0$	$\mu = 0.06$	$\mu = 0.03$
		$\sigma = 0$	$\sigma = 0$	$\sigma = 0$	$\sigma = 0$	$\sigma = 0.42$	$\sigma = 0.17$
7/8	0	$\mu = 0$	$\mu = 0$	$\mu = 0$	$\mu = 0$	$\mu = 0.07$	$\mu = 0.06$
		$\sigma = 0$	$\sigma = 0$	$\sigma = 0$	$\sigma = 0$	$\sigma = 0.6$	$\sigma = 0.24$
7/9	0	$\mu = 0$	$\mu = 0$	$\mu = 0$	$\mu = 0$	$\mu = 0.14$	$\mu = 0.03$
		$\sigma = 0$	$\sigma = 0$	$\sigma = 0$	$\sigma = 0$	$\sigma = 1.3$	$\sigma = 0.17$
7/10+	0	$\mu = 0$	$\mu = 0$	$\mu = 0$	$\mu = 0$	$\mu = 0.08$	$\mu = 0.03$
		$\sigma = 0$	$\sigma = 0$	$\sigma = 0$	$\sigma = 0$	$\sigma = 0.8$	$\sigma = 0.17$
9/2	0.48	$\mu = 0.28$	$\mu = 0.97$	$\mu = 0.72$	$\mu = 0.92$	$\mu = 2.07$	$\mu = 3.81$
		$\sigma = 0.58$	$\sigma = 1.21$	$\sigma = 0.96$	$\sigma = 1.07$	$\sigma = 3.18$	$\sigma = 5.72$
9/3	0	$\mu = 0$	$\mu = 0$	$\mu = 0.01$	$\mu = 0.03$	$\mu = 0.21$	$\mu = 0.33$
		$\sigma = 0$	$\sigma = 0$	$\sigma = 0.01$	$\sigma = 0.17$	$\sigma = 1.6$	$\sigma = 0.9$
9/4	0	$\mu = 0$	$\mu = 0$	$\mu = 0$	$\mu = 0.01$	$\mu = 0.05$	$\mu = 0.15$
		$\sigma = 0$	$\sigma = 0$	$\sigma = 0$	$\sigma = 0.01$	$\sigma = 0.26$	$\sigma = 0.57$
9/5	0	$\mu = 0$	$\mu = 0$	$\mu = 0$	$\mu = 0$	$\mu = 0.13$	$\mu = 0.07$
		$\sigma = 0$	$\sigma = 0$	$\sigma = 0$	$\sigma = 0$	$\sigma = 1.01$	$\sigma = 0.35$
9/6	0	$\mu = 0$	$\mu = 0$	$\mu = 0$	$\mu = 0$	$\mu = 0.13$	$\mu = 0.06$
		$\sigma = 0$	$\sigma = 0$	$\sigma = 0$	$\sigma = 0$	$\sigma = 1.29$	$\sigma = 0.28$
9/7	0	$\mu = 0$	$\mu = 0$	$\mu = 0$	$\mu = 0$	$\mu = 0.09$	$\mu = 0.07$
		$\sigma = 0$	$\sigma = 0$	$\sigma = 0$	$\sigma = 0$	$\sigma = 0.8$	$\sigma = 0.29$
9/8	0	$\mu = 0$	$\mu = 0$	$\mu = 0$	$\mu = 0$	$\mu = 0.03$	$\mu = 0.01$
		$\sigma = 0$	$\sigma = 0$	$\sigma = 0$	$\sigma = 0$	$\sigma = 0.3$	$\sigma = 0.01$
9/9	0	$\mu = 0$	$\mu = 0$	$\mu = 0$	$\mu = 0$	$\mu = 0.1$	$\mu = 0.01$
		$\sigma = 0$	$\sigma = 0$	$\sigma = 0$	$\sigma = 0$	$\sigma = 0.1$	$\sigma = 0.1$
9/10+	0	$\mu = 0$	$\mu = 0$	$\mu = 0$	$\mu = 0$	$\mu = 0.04$	$\mu = 0.01$
		$\sigma = 0$	$\sigma = 0$	$\sigma = 0$	$\sigma = 0$	$\sigma = 0.4$	$\sigma = 0.01$

Table 3.5. Longest repeated consensus motifs in the ENCODE regions, after filtering known repeats, simple sequence and tandem repeats. Uppercase letters indicate positions which were perfectly conserved, lowercase letters represent positions where there was a clear majority character, and N's indicate non-conserved positions. Only 100% TRANSFAC hits are listed. If there was such hit within human factors, others, if any, were omitted. In the absence of a human hit, other factors are indicated by (*) following the factor name.

Motif consensus	Effective length	Expected count in ENCODE	Occurrences in ENCODE	Matches in RepBase	Matches in TRANSFAC
TTTaNaAAAGAAA	11	9.7	135	Multiple (5)	NF-AT1
ATGTNtNTTAAA	9.5	77.5	327	Multiple (5)	MIG (*)
CTGTTTNaNTTT	9.5	77.5	281	Multiple (5)	HNF-3α, HNF-3B
AAAATgNcTTTT	10	38.8	125	Multiple (6)	YY1

Table 3.6. Five most significant repeated consensus motifs in the ENCODE regions, after filtering known repeats, simple sequence, tandem repeats and GC–poor repeats. Uppercase letters indicate positions which were perfectly conserved, lowercase letters represent positions where there was a clear majority character, and N's indicate non-conserved positions. Only 100% TRANSFAC hits are listed. If there was such hit within human factors, others, if any, were omitted. In the absence of a human hit, other factors are indicated by (*) following the factor name.

Motif consensus	Effective length	Expected count in ENCODE	Occurrences in ENCODE	Matches in RepBase	Matches in TRANSFAC
CCCAgNNCTG	7.5	310.1	2692	Multiple (47)	SV40 – unknown (*)
GGGNNcTGGG	7.5	310.1	2621	Multiple (37)	SV40 – unknown (*)
AGANNcAGAA	7.5	310.1	2586	Multiple (81)	—
CTGNNtTCCT	7.5	310.1	2251	Multiple (56)	E74A (*)
AGGNNtGGGG	7.5	310.1	2240	Multiple (42)	—

CHAPTER 4
IDENTIFICATION OF REPEATED SEGMENTS RESULTING FROM CO-OCCURRENCES OF ASSOCIATED SHORT OVER-REPRESENTED MOTIFS

In the study presented in the previous chapter, we have analyzed a remarkable over-representation of many short motifs throughout presumably unique human genomic sequences, as well as (to a lesser extent), Markov model generated sequences trained on human chromosomes. Our findings clearly indicated that: first, all genomic sequences feature dramatically higher numbers of repeated short motifs than one would expect by chance; and second, the differences in numbers of such motifs do not appear to be significant between random intergenic and presumably regulatory sequences upstream of the known genes, despite of the trend that one can notice in the last two columns of Table 3.2. Repeatedly, chi-square tests performed on these columns and other data could show only a mild, but inconclusive, bias.

In several studies carried out more than four decades ago, it has been demonstrated that the nuclear genome of diverse eukaryotes contained a large fraction of repetitive DNA [68, 87, 37]. While more recent insertional events can be readily identified due to the high similarity of the copies, characterization of more ancient insertions remains a challenge. In the human genome, almost half of the sequence is considered unique, but only a small fraction (about 5% of the total) is thought to be functional, whether coding or not. This leaves an open question about the origin and role of the unique non–functional sequence, and we have postulated that the micro–repetitive structure of genomic sequences which we have observed stems mostly from the broken remnants of ancient transposons, which have been degraded

so much that they cannot be recognized as such by present repeat–finding methods. Other than these fragmented transposons, the segmental duplication could have also contributed to overabundance of short motifs.

In this study we present a new algorithm, RepFi (earlier named AssociationFinder [5]), for *de novo* the identification of DNA repeats, which attempts to mutually associate groups of short significantly over-represented motifs, readily found throughout genomic sequences, and construct the consensus of larger, substantially broken elements. Our results have been encouraging and our simulations have confirmed our expectations [5, 2], although we still need to perform biological analysis of many segments we have identified in real genomic data. The goal of this part of the work was in the identification of positions containing a signature of ancient transposon activity, rather than classifying the elements, or even determining their exact boundaries. Once our tool has pointed to these locations, further study is needed to characterize their exact layout and possible evolutionary history, and we provide such study in Chapter 5.

4.1 Algorithm

Our algorithm works in several phases:

1. Locating short motifs v_i' significantly over-represented in the genomic segment $\mathcal{S}$ under consideration.
2. Filtering the list of v_i' in order to retain only motifs v_i which satisfy the minimal criteria of size, structure, and copy count, and mapping them back to their original locations in $\mathcal{S}$.
3. Building a graph $\mathcal{G}$ with v_i as vertices, and weighted undirected edges e_{ij} reflecting repeated co-occurrences of v_i and v_j within predefined windows of width w.

4. Post–processing $\mathcal{G}$ in order to remove e_{ij} which are likely to have arisen by chance.
5. Locating cliques in $\mathcal{G}$, representing groups of motifs which repeatedly co–occur within distance w.
6. Merging the cliques which share a substantial number of motifs into units corresponding to a single repeat element.
7. Mapping the merged cliques back to their genomic locations.

4.1.1 Selecting the seed motifs

We start by counting the number of occurrences of each oligonucleotide of length 5 through 12 in the genomic segment $\mathcal{S}$ under consideration. For this we used a modification of the Karp–Rabin pattern matching algorithm [88], locating all repeats of specified length in time linear with the size of the sequence. The original Karp–Rabin method was based on numerical keys to code patterns, and we used such keys as indices to a hash table counting the number of occurrences of each motif. The pseudo-code in Algorithm 3.1.1 represents the modified version of Karp–Rabin algorithm. After obtaining the counts, we scan through the table in order to identify these motifs v_i' which are most significantly over-represented in $\mathcal{S}$. As mentioned earlier, any genomic sequence features a very large number of such motifs, so we select only these with p-values less than 10^{-5}, as measured by Poisson distribution shown in Equation 2.22. However, we also had to take care that the selected motifs are truly informative, as well as that they make a good mixture of different sizes. The later was necessary because it is expected that more recently inserted copies would share longer motifs, while the ancient ones would have only short pieces in common, and we did not want to overlook any group. We controlled this mixture by maintaining a separate measure of *base count* b so that if e is the expected number

Table 4.1. Variables used in Algorithm 4.1.1

$HashValue$	The integer value used to code a motif string
$MotifLength$	Length of the motif coded by $HashValue$
$Dividend$	Variable used for dividend
$Remainder$	Variable used for remainder
$Motif$	Character array to store the motif string
$Position$	Variable used for array index in the $Motif$ array

of copies of v_i' in $\mathcal{S}$, we require that the actual number of occurrences of v_i' be greater than $e + b$. Although this is a heuristic measure (guided by the expectation that a long motif still has to be present in a sufficient number of repeat copies, no matter what its p–value is) we have fine–tuned it through several thousand test runs under different settings. For large genomic segments (of more than 10 million bases) we have set $b = 1000$.

We scan the table (motif list) returned by Algorithm 3.1.1, and select the indices (hash values) corresponding to the motif counts meeting maximum p-value and minimum copy number conditions, as mentioned above. The algorithm used to convert integer hash values to motif strings is shown in Algorithm 4.1.1, and the list of variables used in this algorithm is given in Table 4.1.

We also looked at the structure of the motifs, and eliminated these consisting of simple sequence (mostly remnants of poly-A tails) and tandemly repeated structure. For simple sequence identification, we measured the overall uncertainty ($-\sum_{c=A}^{T} p_c \log_2 p_c$, by Shannon's information theory, estimating p_c's from base counts within the motif as shown in Equation 2.24), and discarded v_i' with the uncertainty lower than 1.5 bits.

The remaining candidate motifs were scanned for significant mutual overlaps. We have modified the original Smith–Waterman [96] algorithm to produce maximal

Algorithm 4.1.1: CONVERT HASH TO MOTIF STRING($MotifLength, HashValue$)

```
Dividend ← HashValue
Remainder ← 0
Position ← MotifLength - 1
while  Divident > 0
      do { Remainder ← Dividend % 4
           if  (Remainder == 0) Motif[Position] ← 'A'
             else if  (Remainder == 1) Motif[Position] ← 'C'
             else if  (Remainder == 2) Motif[Position] ← 'G'
             else if  (Remainder == 3) Motif[Position] ← 'T'
           Position ← Position -1
           Dividend ← Dividend/4
return (Motif)
```

gap–free local alignments and to get mutual overlap character counts. We eliminated all motifs where mutual overlap with one of higher p–value motif — as measured by Poisson distribution shown in Equation 2.22 — spanned more than 80% of the length of the shorter motif. The remaining motifs comprised our seed set v_i. The pseudocode of the modified Smith–Waterman algorithm is shown in Algorithm 4.1.2, and the list of variables used in this algorithm is given in Table 4.2. The rationale behind the filtering of the similar (overlapped) motifs were to reduce the number of candidate motifs, which would in turn reduce the processing time in mapping the selected motifs to the genomic segment $\mathcal{S}$, building a connectivity graph of these motifs, and allocating the maximal cliques from this connectivity graph.

After the seed motifs v_i have been identified, we scan the original genomic sequence in order to find all positions of their occurrence. For this we used the Aho-Corasick [11] algorithm, matching v_i's all at once. As this algorithm uses time and space linear in size to the sum of the text and patterns we could do the matching efficiently, recording all positions of v_i in $\mathcal{S}$ within seconds.

Table 4.2. Variables used in Algorithm 4.1.2

$Motif_1$	String (character array) for first motif
$Motif_2$	String (character array) for second motif
M	Length of first motif
N	Length of second motif
$F_{i,j}$	Score at the ith row and jth column of the matrix
F_{max}	Largest score in the matrix

Algorithm 4.1.2: MODIFIED SMITH–WATERMAN($Motif_1, Motif_2$)

```
//1. Initialization:
M ← Length of Motif_1
N ← Length of Motif_2
for i ← 1 to N
  do { F_{i-1,0} ← 0
for j ← 1 to M
  do { F_{0,j-1} ← 0
//2. Iterations:
for i ← 1 to N
  do { for j ← 1 to M
       do { if ( Motif_1[i] == Motif_2[j]) F_{i,j} ← F_{i-1,j-1} +1
            else  F_{i,j} ← F_{i-1,j-1}
//3. Find max overlap characters count
 F_max ← 0
for i ← 1 to N
  do {if  (F_max < F_{i,M} ) F_max ← F_{i,M}
for j ← 1 to M
  do {if  (F_max < F_{N,j} ) F_max ← F_{N,j}
return (F_max)
```

The Aho-Corasick algorithm [11] was developed to facilitate bibliographic searches, simultaneously looking for several patterns in a possibly very long text. It also runs in linear time. It combines the ideas of the Knuth-Morris-Pratt algorithm [31] with finite state machines (FSM), and it consists of two phases — constructing a finite state machine for the given set of patterns and then using the machine to process the

string. In the first phase, the algorithm constructs a finite state automaton using the set of patterns based on a lexicographic tree. The number of its states depends on the length of the common prefix in the patterns. It creates a node for every character in each pattern, as long as the character does not extend an already formed prefix. In an extension from the lexicographic tree to a finite state automaton, the Aho-Corasick algorithm computes failure functions, which return a link to a node or state for a particular input at some particular state. The failure links are determined for each state in the automaton. In the second phase, the text that is to be searched is parsed using the automaton constructed in the first phase. A goto function returns a link to the next node on reading an input character, when it is in some particular state. This function is called repeatedly during the parsing. When on reading some input the goto function fails to return a node, then the failure function is invoked. The algorithm also uses an output function for reporting matches and handling special cases of embedded patterns. Typically, the Aho-Corasick algorithm runs in linear time $O(l + n)$, where l is the combined length of all patterns and n is the length of the text. This algorithm has wide applications in many fields due to its high speed and capability of matching multiple patterns.

4.1.2 Building the connectivity graph

Once having the list of v_i's, we look at their associations, i.e. occurring together in clusters at different locations in similar layout, in genomic segment $\mathcal{S}$. For that purpose, we start with defining the *pairings* and *distinct pairings* of motifs:

Definition[Motif pairings] If we denote each distinct occurrence of a motif v_i by v_i^k, we define the set of pairings

$$\mathcal{P} = \{(v_i^k, v_j^l) | k \in [1, n_i], l \in [1, n_j], \\ i, j \in [1, M], i \neq j, \mathtt{d}(v_i^k, v_j^l) \leq w\},$$

where n_i and n_j are the number of occurrences of v_i and v_j in $\mathcal{S}$, respectively, $\mathtt{d}(v_i^k, v_j^l)$ is the distance between motifs v_i^k and v_j^l in the genome, M is the total number of motifs, and w is a pre-defined window size.

Definition[Distinct motif pairings] We form the distinct motif pairings set $\mathcal{DP}$ from $\mathcal{P}$ by replacing all occurrences of $(v_i^k, v_j^{l_1})$ and $(v_i^k, v_j^{l_2})$ with a single pair (v_i^k, v_j^l) (v_j^l unifying all $v_j^{l_m}$ such that $(v_i^k, v_j^{l_m}) \in \mathcal{P}$), and all pairings $(v_i^{k_1}, v_j^l)$ and $(v_i^{k_2}, v_j^l)$ with a single pair (v_i^k, v_j^l). In other words, in $\mathcal{DP}$ each v_i^k connects to one and only one v_j^l, and each v_j^l connects to one and only one v_i^k.

Knowing the exact position of each occurrence of v_i, further on referred to as a vertex, we construct the edges connecting them in the following way. Sliding a window of a pre-defined size w from the beginning of $\mathcal{S}$, we build (and subsequently update) an adjacency list. We have left the widow size as a parameter to our software. However in an attempt to make a good guess about the window size to achieve the best overall results, we have calibrated the window size parameter using simulation data, and these calibration results are discussed in Section 4.1.2.1. Also, our experiments on real genomic data (ENCODE regions) have indicated that in human genomic sequences, the best overall results can be achieved for sizes around 1000. However, this size may vary from one organism to another depending upon the sizes of the repeats in them. This introduces some skepticism about how much our software can be adjusted to identify significantly diverged short elements, such as human ALUs, as the number of associated conserved short motifs within their span might be too small to form a large enough set of significant associations.

As we move the window through $\mathcal{S}$, jumping from one motif v_i to the next v_j (where it is possible that $i = j$), we count the number of occurrences c_{ij} of each $(v_i, v_j) \in \mathcal{DP}$, and at the end we assign c_{ij} as the weights of the edges e_{ij} connecting v_i's and v_j's. After assigning the weights we retain e_{ij} only if the probability of a random association of v_i and v_j, assuming the uniform distribution of motifs in $\mathcal{S}$, is less than 0.05 (this is a parameter to our program, and it can be adjusted with respect to the desired tradeoff between sensitivity and specificity). We do that as follows. As before, let n_i and n_j be the observed numbers of occurrences of motifs v_i and v_j, and let c_{ij} be their observed number of co-occurrences within the intervals of width w. Then, if we denote the total length of $\mathcal{S}$ by L, the expected number of occurrences of v_j in a single interval of width w is:

$$\tau = \frac{\text{Total observed number of occurrences of } v_j}{\text{Total length of } \mathcal{S}} * w \tag{4.1}$$

$$= \frac{n_j}{L} w = \frac{n_j w}{L} \tag{4.2}$$

Using the Poisson distribution (Equation 2.4) with τ rate of occurrences of v_j, we estimate the probability of any interval containing v_i also accommodating one or more v_j's as $1 - \mathsf{e}^{-\tau}$, and the expected number of such intervals as $n_i(1 - \mathsf{e}^{-\tau})$.

$$\text{Probability of no occurrence of } v_j = P(\tau, 0) \tag{4.3}$$

$$= \frac{\tau^0 \mathsf{e}^{-\tau}}{0!} \tag{4.4}$$

$$= \mathsf{e}^{-\tau} \tag{4.5}$$

$$\text{Probability of one or more occurrences of } v_j = 1 - \mathsf{e}^{-\tau} \tag{4.6}$$

$$\text{Expected intervals having } v_i \text{ and one or more } v_j = n_i(1 - \mathsf{e}^{-\tau}) \tag{4.7}$$

Under the Poisson model (Equation 2.4) with $n_i(1 - \mathsf{e}^{-\tau})$ (rate of) expected one or more times co-occurrences intervals of v_i and v_j, the probability of a random variable X representing the chance co-occurrence of v_i and v_j, c_{ij} or more times within the distance of w or less is given by:

$$\begin{aligned} \mathsf{P}(X \geq c_{ij}) &= 1 - \sum \mathsf{P}(0 \leq X \leq c_{ij} - 1) && (4.8) \\ &= 1 - \sum_{k=0}^{c_{ij}-1} P(n_i(1 - \mathsf{e}^{-\tau}), k) && (4.9) \\ &= 1 - \sum_{k=0}^{c_{ij}-1} \frac{[n_i(1 - \mathsf{e}^{-\tau})]^k \mathsf{e}^{n_i(1-\mathsf{e}^{-\tau})}}{k!} \quad \text{(from Equation 2.4)} && (4.10) \end{aligned}$$

For an easy understanding, we show an example of motifs layout, corresponding graph, and the clique in Figure 4.1. This example (in part A) includes five motifs, each shown in different colored boxes in the motifs layout. We scan this motifs layout, and build a graph of five vertices (v1, v2, v3, v4, and v5), each shown in different colored circles. The associations among these motifs in the given layout are shown by the black edges in the graphs. The thickness of these edges represents the edge weight. It is clear from the initial graph that the thin edges e_{23}, e_{24}, and e_{45} have shown up just by chance, and, hence, are removed before attempting to identify the maximal clique. The clique comprises the vertices v1, v3, and v4, indicating a strong association in the co-occurrences of the motifs represented by the color "grey", "red", and "blue".

4.1.2.1 Window size calibration

In order to prepare a dataset for our calibration experiments, we have chosen two Repbase elements, TIGGER2 and L1MCA_5, to insert into the testbed sequence — as described in the Section 4.2 — with different copy numbers, roughly reflecting the

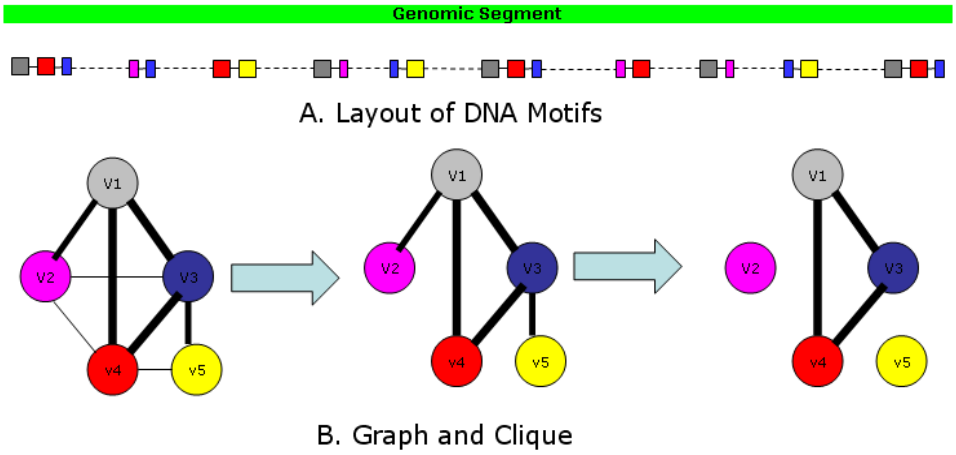

Figure 4.1. Motifs layout, graph, and clique. The top figure (A) shows an example of motifs layout, where colored boxes represent different motifs. The bottom figure (B) shows corresponding initial graph, filtered graph, and the maximal clique. In these graphs, colored circle represent the vertices (i.e. motifs) and black lines represent the edges, where the thickness of an edge represents its weight.

general genomic frequencies of their superfamilies. We introduced random mutations, degrading these sequences at 10, 20 and 30 percent. We have used this dataset in order to determine the best values for the interval width w and probability threshold for c_{ij}'s. We have performed simulation runs with interval width ranging from one half of the estimated repeat element size to 1.5 times the estimated repeat element size, and for the thresholds ranging from 0.1 to 0.001. As expected, probability–wise the best results have been achieved for common statistical significance cutoffs, with p–values around 0.05 and 0.01. Using the threshold of 0.05, we calibrated the window size w with respect to the estimated element size, and we have shown the result of this calibration in Figure 4.2.

We generated data for 10%, 20%, and 30% sequence divergence (since the original insertions of exact copies). While the repeats could have still been recognized with high sensitivity and specificity (and by any method), for divergence up to about 20%, at 30% decay the optimal balance between the sensitivity (at 0.6, i.e. 61%) and

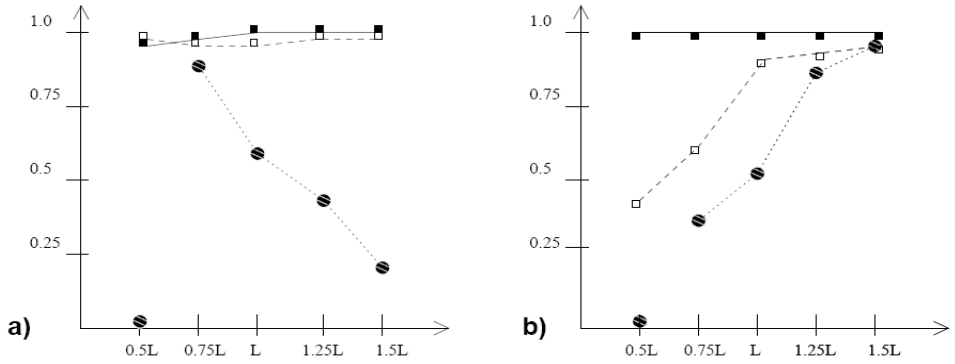

Figure 4.2. Calibration results for the window size w. The left chart (a) shows the results concerning the sensitivity, and the right one (b) specificity. Black box indicates data for 10% sequence decay, white box 20% and patterned circle 30%. L indicates the repeat element length.

specificity (at 0.5) has been achieved for window size matching the size of the inserted elements. However, at that divergence no data could have been obtained at smaller window sizes (of less than 1500 bases) corresponding to windows of half (or less) of the inserted element length, resulting in 0 specificity and 0 sensitivity (as no significant cliques could have been formed, by the process described below). This raises some skepticism about whether our software can be adjusted to identify significantly diverged short elements, such as human ALUs.

Another reason for performing the calibration was our desire to quantify how much the output quality depends on the correct guess of the inserted element size. Up to 20% decay the sensitivity was steadily high (at more than 0.95), however already at this level the specificity was suffering as the window lengths became shorter. The differences were even more dramatic at 30%, when the specificity was rapidly climbing with the window size, but the sensitivity was equally rapidly dropping. In consequence, while an approximate guess of a target element length (within $\pm 10\% - 15\%$ of its correct size) would still generate good results, a less precise one would lead to a substantial bias. However, due to the interplay and monotonicity of sensitivity

and specificity one can decide which side is better to err, depending on the particular circumstances.

4.1.3 Clique allocation and post–processing

After the motif association graph $\mathcal{G}$ has been constructed, we proceed by building the cliques of its vertices.The clique problem is known to be NP–complete, and the approximate solutions would not serve our purpose well, so we used a heuristic algorithm for maximal clique enumeration, which we nevertheless sped up by an efficient implementation. It was based on finding the groups of v_i's forming maximal cliques in $\mathcal{G}$ by locating the intersections among the adjacency lists. This was achieved through indexing and by using the C++ standard template library function `set_intersection`.

In this approach, we start with the graph $\mathcal{G} = (V, E)$, which has N vertices v_i's (i.e. v_1, v_2, ..., v_N). We denote the adjacency list of each vertex v_i by $AdjList(v_i)$ and the kth vertex in this adjacency by $AdjList_k(v_i)$. Assuming that the vertices v_1, v_2, ..., v_N are arranged in decreasing order of degrees or edge counts, we visit each vertex in the graph $\mathcal{G}$ in this order of decreasing degrees and find the intersection between the adjacency list of the vertex being visited and the adjacency lists of all of its neighbors (i.e. all vertices present in the adjacency list of the vertex being visited). If this intersection is not a empty set, we consider it as one maximal clique and denote it by $CLQ(v_i)$. If $CLQ(v_1)$ is the same to the adjacency list of any of the vertex $AdjList_k(v_i)$, we tag this vertex $AdjList_k(v_i)$ to "not to visit" in future for finding the intersection between its adjacency list and the adjacency lists of all of its neighbors. One disadvantage of this approach is that we get duplicate cliques in our list which we filter-out at the end of clique enumeration step. The pseudo-code of this approach is given in Algorithm 4.1.3, and the list of variables used in

Table 4.3. Variables used in Algorithm 4.1.3

N	Number of vertices v_i's in the graph $\mathcal{G}$
$AdjList(v_i)$	Adjacency list of the vertex v_i
$AdjList_k(v_i)$	kth vertex in the adjacency list of the vertex v_i
CLQ	Array of cliques

this algorithm is given in Table 4.3. As mentioned earlier, the algorithm described in Algorithm 4.1.3 may return some similar or closely related cliques. So once the cliques have been formed, we proceed to filter out the similar cliques and merge these likely to correspond to a single consensus of a repeated element. Because the window sizes w used to determine the initial motif associations are in general smaller than the average repeat, without merging this would result in the fragmentation of the element consensuses.

Algorithm 4.1.3: MAXIMAL CLIQUE ENUMERATION($\mathcal{G}$)

```
//1. Initialization:
for each vertex v_i  Visit(v_i) ← TRUE
//2. Clique Enumeration:
for i ← 1 to N
  do { if (Visit(v_i) == TRUE)
       do { CLQ(v_i) ← AdjList(v_i) ∩ [∩_{k=1}^{|AdjList(v_i)|} AdjList(AdjList_k(v_i))]
            for k ← 1 to |AdjList(v_i)|
              do {if (CLQ(v_i) == AdjList_k(v_i)) Visit(AdjList_k(v_i)) ← FALSE
return (CLQ)
```

We estimate the distance between cliques C_i and C_j, viewed as sets of motifs, using the classical formula for Jaccard set similarity, $\mathtt{d}(C_i, C_j) = \frac{C_i \cap C_j}{C_i \cup C_j}$ [79]. Our simulation runs have indicated that the best results are achieved for $\mathtt{d}(C_i, C_j)$ threshold

set at 0.5, which, under optimal circumstances (distinctly different repeat elements with copies degraded less than 10%) gave us 100% accuracy. In general, merging accuracy was highly variable, as the known sequence repeats are members of different families and superfamilies, and as such feature a certain degree of similarity, which for highly degraded copies may lead to incorrect classification. For this purpose one can also apply statistical reasoning concerning a chance occurrence of a clique of a certain size in a random graph, however our graph is far from random, being itself built around associations that were unlikely by chance. In consequence, and guided by the desire to identify highly diverged repeated elements (which would show as a collection of small cliques, rather than one of a substantial size), we strived to consider cliques as small as possible. However, due to an almost certain occurrence of at least one (and usually many more) clique of size 3 still relatively high incidence of these of size 3 in any non–trivial graph with a number of edges at least comparable to the number of vertices (established both through our experiments and by probabilistic considerations), and still a relatively high likelihood of these of size 4, in our analysis we had to limit ourselves only to cliques of size 5 and larger. This is yet another parameter to our software, and we advise the users to estimate it in accordance to the approximate age of the elements desired to be found.

Another, and complementary, problem is fragmentation — one element being represented by more than one clique. If this stems from the physical fragmentation in the genome further analysis (beyond the scope of this study) needs to be done. Otherwise, especially in the cases when window size w is much smaller than the element the fragmentation would call for the collapsing of overlapping cliques. Unfortunately, this opens a Pandora's box of undue extensions, so we preferred to keep our approach clean. In an extension of this work one may want to look at the significant associa-

tions of *cliques*, but that should be done with caution, and only after the mapping of the element positions back in $\mathcal{S}$.

4.1.4 Mapping the cliques on the genome

The last step performed by our software is the mapping of the cliques back to the genome, and we did this using the algorithm for locating the *constrained heaviest segments*. This algorithm, whose early version has been described by Jon Bentley [48], and later used and modified by several authors (with a rigorous treatment of its variants given in [103], [100], and [62], among others), works on arrays of numerical scores, locating areas which "peak" over their environment in terms of their cumulative score, in time linear to the size of the score array. Briefly, a constrained heaviest segment is an interval $\mathcal{I}_{i..j}$ whose cumulative score S_{ij} is greater than or equal to the cumulative score S_{kl} of any of its subintervals $\mathcal{I}_{k..l}$, where $i \leq k \leq l \leq j$, and for which there is no interval $\mathcal{I}_{m..n}$, where $m \leq i \leq j \leq n$ and either $m < i$ or $n > j$, with $S_{mn} \geq S_{ij}$. By keeping track of the local minima and maxima of the cumulative score within the array, counting from its beginning, and updating the information about previous lower minima and higher maxima as the algorithm progresses through the array, one can report all constrained heaviest segments by the time the last array entry is processed (amortized linear time). A high-level pseudo-code of the our variant of constrained heaviest segment algorithm is shown in Algorithm 4.1.4, and the list of variables used in this algorithm is given in Table 4.4.

We use this algorithm for mapping our cliques in the following way. We first assign a slight negative score (-1) to all base positions in $\mathcal{S}$. As we know the genomic positions of each of our v_i, for each clique we go back to the locations of its constituting motifs, and assign a positive score to each base in the motif. The optimal value for this score highly depends on the degradation level of the element represented by the

Table 4.4. Variables used in Algorithm 4.1.4

$Score$	Array of scores
$Score_i$	Score at the i-th location of the $Score$ array
$Upward$	True if score changes from negative to positive in the $Score$ array
$Downward$	True if score changes from positive to negative in the $Score$ array
$PrevCumulativeScore$	Variable for previous cumulative score
$CumulativeScore$	Variable for cumulative score,initialized to 0
N	Constrained heaviest segment count, initialized to 0
$Strat_N$	Start position of the N-th segment
End_N	End position of the N-th segment
$StartScore_N$	Cumulative score at the start of N-th segment
$EndScore_N$	Cumulative score at the end of N-th segment

Algorithm 4.1.4: CONSTRAINED HEAVIEST SEGMENTS($Score$)

for each $Score_i$ in $Score$ array from left to right
do
- $PrevCumulativeScore \leftarrow CumulativeScore$
- $CumulativeScore \leftarrow CumulativeScore + Score_i$
- **if** ($Upward$)
  - **do**
    - $Start_N \leftarrow i$
    - $StartScore_N \leftarrow PrevCumulativeScore$
    - **while** ($Start_{N-1} > PrevCumulativeScore$) $Start_N \leftarrow Start_{N-1}$
- **if** ($Downward$)
  - **do**
    - $End_N \leftarrow i - 1$
    - $EndScore_N \leftarrow PrevCumulativeScore$
    - **while** ($End_{N-1} \leq PrevCumulativeScore$)
      - **do**
        - $End_{N-1} \leftarrow End_N$
        - $EndScore_{N-1} \leftarrow EndScore_N$
        - $N \leftarrow N - 1$

clique, varying from +1 for perfectly conserved copies, to about +15 for very degraded ones (more than 30%). Since the degradation levels of repeat elements in any genome substantially varies, and is not known up front, we have by default set the score to the average value +7, which gave us reasonable results.

Table 4.5. The performance of our software on the ENCODE regions of the human genome.

Tool	Sensitivity	Specificity
RepeatScout	0.435	0.997
PILER	0.172	0.997
RepFi	0.594	0.506

After assigning the scores we identify the constrained heaviest segments in $\mathcal{S}$, and consider them as the locations of the copies of the repeat element represented by the clique. It is worth noting that this method bypasses the need to repeat mask the genome with the established consensus as a separate last step, which is the most time–consuming component in the performance of most *de novo* repeat finders (which generally use RepeatMasker [18] software for this purpose).

4.2 Results

When evaluating the performance of our RepFi software, we were first interested in how well it would locate the known repeats in a relatively well annotated genome (such as human, where the richest libraries are available) and, second, how well it would do in comparison with other *de novo* repeat finders. For this purpose we have looked at the ENCODE [98] regions, repeat masked them with RepeatMasker [18] using the RepBase [52, 51] depository, and compared the masked positions with these annotated by RepFi, RepeatScout [14] and PILER [85]. Using the approach as shown in Figure 4.3, we have counted the number of bases classified as true positives (TP), false positives (FP) and false negatives (FN), then calculated the sensitivity as the ratio $\frac{TP}{TP+FN}$ and specificity as $\frac{TP}{TP+FP}$ [50, 60]. The results of this comparison are shown in Table 4.5.

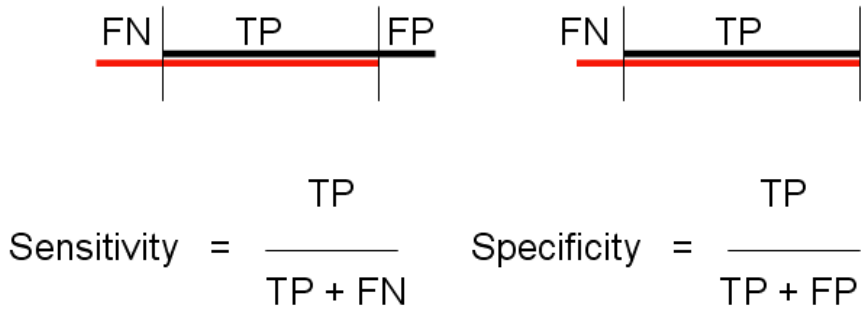

Figure 4.3. Approach to compute sensitivity and specificity: "Red" colored line represents real repeat element and "black" colored line represents computationally identified repeat element. "TP", "FP", and "FN" represent part of real element identified computationally, additional part of computationally identified element, and part of real element missed in computational identification, respectively [50, 60].

Table 4.6. The performance of our software on the simulated data set, experiment 1 (one inserted element, 150 copies). "Sn" stands for "Sensitivity" and "Sp" for "Specificity".

Degradation	PILER Sn	PILER Sp	RepeatScout Sn	RepeatScout Sp	RepFi Sn	RepFi Sp
0%	0.981	0.999	0.979	1.0	0.978	0.982
10%	0.0	0.0	0.979	1.0	0.961	0.968
20%	0.0	0.0	0.979	1.0	0.855	0.906
30%	0.0	0.0	0.0	0.0	0.512	0.516

Our tool was the fastest, taking just about 9.5 minutes for the annotation of the entire ENCODE on a Unix server with dual 3.08Ghz processors and 4Gb of main memory, as opposed to about 1 hour and 10 minutes taken by RepeatScout to create the library, and additional 11.5 hours to repeat mask the regions using the new library as an input to the RepeatMasker. PILER took 41 minutes to construct the consensus elements, and additional 10.5 hours to mask the original genomic segment.

As we were expecting, our sensitivity was better than that of other *de novo* repeated element finding tools, as we are capable of identifying more degraded seg-

ments, even as we require progressively more copies for more divergent elements. On the other hand, it was somewhat surprising that none of *de novo* repeat finders could detect more than 59.4% of the already annotated human repeats (especially PILER, whose overall sensitivity was surprising).

In terms of specificity, our expectations were also confirmed. Since our tool has primarily been built to pinpoint the highly divergent copies — these which cannot be recognized by RepeatMasker (if they are present in its underlying library at all) or other *de novo* repeat finding tools — many of our findings were classified as false positives in this test. However, we expect that some, if not most, of these were actually true positives of elements not detected by other tools.

In order to confirm this, and establish a better measure of the true specificity of our software, we have thus created a well–controlled test environment. It consisted of 1.3 million bases of "testbed" sequence into which we inserted previously characterized repeats. The testbed data has been built to resemble a real genomic environment as faithfully as possible, but at the same time to be free of repeated elements. One third of it was composed of an entirely random assembly of four DNA letters, another third was produced by a second–order Markov model trained on one million bases from human chromosome 2 obtained from the Ensembl [95] genome browser, and the last third has been obtained from real bacterial sequences (*E. coli* from GenBank). As prokaryotic genomes feature little repetition, we have included these segments because we wanted to have a fraction of real genetic data in the testbed sequence, but did not want to bias it too much since there are considerable differences between prokaryotic and eukaryotic DNA.

In the first experiment (which was more of a "sanity check") we have inserted 150 copies of the TIGGER2 element whose consensus sequence is recorded in the RepBase database. While this number of copies in a relatively short "genomic"

Table 4.7. The performance of our software on the simulated data set, experiment 2 (two inserted elements, 150 copies each). "Sn" stands for "Sensitivity" and "Sp" for "Specificity".

Degradation	PILER Sn	PILER Sp	RepeatScout Sn	RepeatScout Sp	RepFi Sn	RepFi Sp
0%	1.0	0.999	1.0	0.999	0.998	0.982
10%	0.0	0.0	0.999	0.999	0.973	0.979
20%	0.0	0.0	0.999	0.999	0.748	0.738
30%	0.0	0.0	0.0	0.0	0.708	0.593

segment might be unrealistic, we can identify substantially degraded elements only if they are present in large copy numbers (while other tools cannot detect them at all), and, besides, TIGGER2 belongs to the DNA transposon class and elements of this class can be found in about 400,000 copies in the human genome. While not all members of this family are so similar to each other to fully justify our number of copies of a single one, we nevertheless found this approach useful to give us rough performance estimates for our software.

We then proceeded to introduce random mutations to both the testbed sequence and the inserted repeated copies, degrading them for 10, 20, and 30 percent. In each of these settings the performance of our software, as well as the comparison with that of the RepeatScout and PILER, in our controlled environment, is shown in Table 4.6. We have omitted the comparison with RepeatMasker, since the repeat we worked with has been taken from its own underlying library, and the RepeatMasker would thus do well in detecting it even at 20% and higher degradation (actually, losing only a small percentage of occurrences even at 30% decay). However, RepeatMasker's performance is not guaranteed at divergence ranges of 30% or higher, while our software can still find useful data with reasonable accuracy.

Table 4.8. The performance of our software on the simulated data set, experiment 3 (three inserted elements, 150 copies each). "Sn" stands for "Sensitivity" and "Sp" for "Specificity".

Degradation	PILER Sn	PILER Sp	RepeatScout Sn	RepeatScout Sp	RepFi Sn	RepFi Sp
0%	1.0	0.999	0.982	0.999	0.998	0.941
10%	0.0	0.0	0.998	1.0	0.939	0.970
20%	0.0	0.0	0.996	0.999	0.724	0.638
30%	0.0	0.0	0.0	0.0	0.646	0.599

The results of this test led us to believe that about half of our declared false positives in the real genomic sequences are actually true hits to degraded elements which have not been previously identified and placed in the RepBase library. On the other hand, the performance of PILER was again disappointing (at least under its default settings), while RepeatScout has shown an abrupt change in behavior, from finding everything to finding nothing, once the sequence divergence fell beyond its tolerance threshold. Since it relies on exactly conserved k–mers in order to seed the possible repeated sequences, it stops finding anything once the conservation of its seeds becomes unlikely (a similar problem has been reported in the past with blast, and experience has proven this to be a serious issue).

However, when RepeatScout finds that there is something repeated in the sequence it further relies on RepeatMasker (by supplying it an enhanced library) to locate the positions of the repeats. Since RepeatMasker is doing such a good job recognizing the sequence, it resulted in high sensitivity and specificity in cases of up to 20% decay, even if RepeatScout itself has found only a small fraction of the elements we have inserted. Without RepeatMasker's aid the RepeatScout's performance would be more gradually declining and would have exhibited sensitivity inferior to our tool

even at a 20% sequence divergence. In order to further corroborate our findings, we have performed two additional simulation experiments. In the experiment number 2, whose results are shown in Table 4.7, we have used the same testbed sequence, but inserted 150 copies of TIGGER2 and 150 copies of L1MCA_5, an L1 non–LTR retrotransposon (also taken from RepBase). In the third experiment we have inserted 3 elements, TIGGER2, L1MCA_5 and MLT1AR (an LTR retrotransposon), 150 copies each — its results are shown in Table 4.8. As it can be seen from the tables, the results of all our simulations were highly consistent, indicating that we have indeed been successful in finding substantially broken elements invisible to other tools (at least when they were present in high enough copy numbers).

4.3 Discussion

The micro-repetitive structure of human and other genomes still leaves us with many puzzles. While it is intuitive to assume that even the unique DNA sequence mostly derives from ancient (and thus hardly recognizable) mobile element activity, substantial further work was necessary to establish that for a fact. Despite the inevitable limitations in the power of any approach due to the mutation amount drowning the signal in noise, especially for short and low–copy–number elements, we anticipated that the tool described in this chapter and its extensions will contribute to resolving this puzzle.

In many respects, the software we describe in this chapter is still a work in progress. It needs to be more extensively evaluated, especially in terms of its parameter settings. We have ran it on many simulated and real genomic datasets (ENCODE regions), yet we still need to fully characterize its performance and the tradeoff between the sensitivity and specificity in the presence of a truly random mixture of transposable elements of different lengths, compositions and copy numbers, and es-

pecially these whose copies may have been broken during the evolution by layered insertions of other elements within their span.

One deficiency of this tool is its lack of ability to establish the exact boundaries of the repeat elements it maps to its cliques. We have considered this to be a separate problem, but much facilitated by the capacity to pinpoint the repeated elements themselves and their locations. On another note, beyond its independent use, our method, in particular, some of the techniques presented in Chapter 2, can also be merged with that of RepeatScout. As previously mentioned, RepeatScout's performance considerably suffers from its quest for exactly repeated k–mers, and we believe that it wold be substantially improved if it would instead use our inexact seed motifs.

An alternative approach would be to build a consensus library for each of our cliques (after initially mapping them to the genome), then submit it to RepeatMasker to search for other occurrences of the element, which we may have missed. Using RepeatMasker adds substantial power to RepeatScout, and it may also improve our results, despite of the loss in time. Masking the repeats is generally not a time-critical task, and spending a few additional hours on it may be worth the wait. After many years RepeatMasker still remains the best tool for finding interspersed repeated elements when good libraries are available, and the best use for *de novo* repeat finders may indeed be in supplying it with new and enhanced element libraries. We have done some work following this idea, and we describe it in the next chapter.

There are two possible improvements to our software which we have investigated and comparatively evaluated. One is in that, instead of using an average weight score in our detection of constrained heaviest segments, we start with very low weights and longer seed motifs (appropriate for well conserved repeat copies), mask the resulting blocks and exclude them from further consideration. That would set aside the best

conserved copies, which would otherwise lead to incorrect mapping if their positions were given greater weight (in an environment where repeat copies lie close to each other the software would tend to merge two or more if they have sufficient weight to bridge the gap between them). After that, we can re-run the software with higher weight factors (and shorter seed motifs), and iteratively continue, setting aside more and more degraded copies, until no further elements can be detected.

Though our approach as described in this chapter has successfully pointed out the signatures of the broken repeated segments, our findings needed further biological analysis. In addition to just mapping the cliques back to the genome, we need to reconstruct the consensus sequences of the broken repeats, hopefully with precise definitions of the ends. Such consensus sequences can then be used for further classification and characterization of the repeated elements. Another problem we still have to address is that of repeat elements which have been broken by an insertion of yet another transposon somewhere within their span. While our current approach would allow for the allocation of fragments of such elements, we still need to work on their correct classification. This is a daunting problem, however the more of the genome we are able to characterize, the better chances we have of eventually understanding the part that remains.

Continuing the study described in this chapter, we present our advancement towards the reconstruction and classification of the sequences of broken repeated elements in the next chapter.

CHAPTER 5
RECONSTRUCTION AND CLASSIFICATION OF THE SEQUENCES OF BROKEN REPEAT ELEMENTS

In the previous chapter we have presented an algorithm RepFi to associate the co-occurrences of the over-represented short sequences (exact motifs of length varying from 7 to 14 bases), and we have shown that this algorithm indeed successfully finds the remnants of broken repeated segments. Our next step was to reconstruct the consensus sequences of these broken repeats and to see how well our repeat consensus sequences could be classified as transposable elements. We were also interested in knowing how well would our approach do in comparison with other *de novo* tools for identifying previously known, as well as unknown repeats, in real genomic data.

So far, many families of repeats have been manually annotated and deposited in databases such as Repbase [52, 51], then used by programs such as RepeatMasker [18] to identify their traces in any given DNA segment. As we have mentioned earlier a majority of these repeats are interspersed, resulting mostly from the activity and accumulation of transposable elements. More than half of the human genomic sequence consists of known repeats, however a very large part has not yet been associated with neither repetitive structures nor functional units. Also, we have postulated that most of the seemingly unique content of mammalian genomes is a result of old transpositions and other duplication activities resulting in copies which have diverged so much that they cannot be recognized as such by current methods any more. The degeneration of these ancient repetitive elements could have resulted in the overabundance of short repetitive sequences (motifs) throughout the human genome which we have described earlier in this thesis.

In this chapter we present our efforts to reconstruct the consensus sequences of the broken repeated elements and to classify them as transposable elements, by integrating a previously developed computational tool for the classification of repeated elements according to biological criteria [72] with our approach [1].

5.1 Methods

We have already described the core method of the identification of degenerated repeated sequences in a given genome in the previous chapter, and we have also postulated that many short motifs which are dramatically over–represented in mammalian genomes derive from ancient repeats which, over time, became so degraded by mutations that they cannot be recognized as copies of the original elements any more (thus creating an impression of a large amount of unique non–functional sequence). However, our core algorithm focuses solely on the identification of the repeated segments. In this chapter we concentrate on our efforts to reconstruct the consensus sequences of these segments, and to classify these consensus sequences as transposable elements, if possible. In addition to these advancements, we have also introduced an iterative approach in order to find repetitive sequences, where we first attempt to locate the most conserved repeated elements, then follow by more degraded ones. Thus, our final approach includes three major steps, as given below:

1. Identification of the repetitive segments.
2. Determining the consensus sequences and building the library of repeat families.
3. Classification of the consensus sequences of repeat families.

5.1.1 Identification of the repetitive segments

We start with the identification of short significantly repeated motifs, and try to associate them into groups which seem to co-occur with suspicious frequency.

Different transposons have copies conserved at different rates [26], and we assume that these which are more similar (presumably corresponding to more recent replication activity) will share long motifs in a relatively stable order, while these which have been substantially degraded will feature random subsets of short motifs, and in a more *ad hoc* manner (due to occasional randomly formed oligonucleotide sequences or multiple insertions at loci close to each other).

We attempt to isolate the repeated sequences in a series of iterative runs by choosing different motif sizes (starting with size 12-14 and decreasing it for every subsequent run). In each iteration, we identify a set of sequences, starting with those featuring most conserved copies and then looking for progressively more degraded elements. At the end of each run we mask the sequences we have identified so far in order to exclude them from further consideration, reduce the size of seed motifs we attempt to cluster, and increase the number of times we need to see the seed motif repeated before we include it in the list used for further consideration. We end this process at about 30% degradation of the copies, when the motifs become so short (7–8 characters) that their over–representation due to being a part of a transposable element becomes dwarfed by the number of chance occurrences (i.e. the variance of the numbers of chance occurrences), and when practically any transposon would feature several copies of the motif. The exact calculation of these numbers and the estimate of the probabilities of associations of such short seeds is a daunting task. So we have established them by running a large number of simulations and recording the effectiveness with which we could recognize longer degraded copies.

We have described various phases of each iterative run to identify the repetitive segments in detail in the previous chapter (Section 4.1), and we have adopted almost all these phases in our final algorithm, except one, i.e. merging the cliques. In the final algorithm we do not attempt to find one clique corresponding to one repeated

segment, as this phase would not be of much advantage. The list of the phases in the identification of the repetitive segments, appropriately modified from the outline in Section 4.1, is given below:

1. Locating short motifs significantly over-represented as candidate components of longer repeated elements.
2. Filtering the list of these short motifs in order to retain only these which satisfy the minimal criteria of size, structure, and copy count, and mapping them back to their original locations in the genomic segment in question.
3. Building a graph with motifs as vertices, and weighted undirected edges reflecting repeated co-occurrences of motifs within predefined windows.
4. Post–processing the graph in order to remove edges which are likely to have arisen by chance.
5. Locating cliques in the graph, representing groups of motifs which repeatedly co–occur within a predefined window.
6. Mapping the cliques back to their genomic locations.
7. Identifying the repetitive segments.

From this point we proceed to group the identified repetitive segments into consensus sequences of individual repeats.

5.1.2 Determining the consensus sequences

The positions where the original cliques mapped to the genome can be used for the determination of the consensus sequence of the corresponding transposon. For well conserved copies resulting from recent insertional events this task is almost trivial, however when copies are degraded more than about 15% it becomes a challenge.

We first looked at the heaviest segments resulting from the mapping of the cliques, and for each pair we calculated their distance as measured by the number of

shared motifs. Since that makes our problem one of the distance between sets, we have first tried traditional measures such as Jaccard distance [79] and several other alternatives, but they have not performed well in our context. We have thus modeled this problem as a drawing of a random variable from the hypergeometric family of distributions.

Motifs within the two segments we are comparing can be viewed as if they belong to the same superset, we can label it $\mathcal{S}$, of cardinality N: this is our original set of seed motifs which we used to build the graph. We can label the segments we are comparing as $\mathcal{S}_1$ and $\mathcal{S}_2$, of cardinality $D_1 =\mid S_1 \mid$ and $D_2 =\mid S_2 \mid$. Thus, $D_1 \leq N$ and $D_2 \leq N$. We can also assume, without loss of generality, that $D_1 \geq D_2$. We model the intersection $\mid S_1 \bigcap S_2 \mid$ as an experiment in which we draw D_2 motifs from $\mathcal{S}$, and consider the probability:

$$\mathrm{P}_k(N, D_1, D_2) \;=\; \frac{\binom{D_1}{k}\binom{N-D_1}{D_2-k}}{\binom{N}{D_2}} \tag{5.1}$$

that the intersection of $\mathcal{S}_1$ and $\mathcal{S}_2$ contains exactly k motifs (i.e. that during the "assembling" of $\mathcal{S}_2$ exactly k motifs came from $\mathcal{S}_1$). Consequently, if $\mathcal{S}_1$ and $\mathcal{S}_2$ share K motifs, the probability that these two segments would share K or more of them is calculated as $P_K = \sum_{k=K}^{D_2} \mathrm{P}_k(N, D_2, D_2)$, and we can adopt it as a measure of distance between $\mathcal{S}_1$ and $\mathcal{S}_2$. We use this distance as a basis for the single linkage clustering of the segments. We set the cut on the resulting tree somewhat heuristically (again optimized through simulations) so that in every cluster we have at least 3 segments. In general, it is better to err in placing too few segments in a single cluster than placing too many, since similar consensus sequences can always be further merged, while consensuses built from only distantly related elements are bound to be poor. After the clusters of segments have been built, we proceed to align the

DNA sequences corresponding to the segments placed in a single cluster. Rather than writing the alignment software ourselves, we have interfaced to CLUSTALW [64], a popular program for constructing multiple alignments. We then looked at the columns of the resulting alignment in order to assign the consensus character to each position (or, better say, ancestral character, since we are determining the most likely sequence of the transposon whose copies we have identified). While many columns do yield a natural consensus character, there is also a large fraction of these featuring substantial ambiguity. Conservatively, one would want to assign the letter 'N' (as a "do not know" or a "wildcard" character symbol) to these positions, but for our purposes that would only lead to complications. Since we would like to use the consensus sequences we build to further mine the genome for similarities not initially discovered by our software, and also to attempt repeat classification, 'N' at any given position would not be of help. In particular, the RepeatMasker program would automatically consider it a mismatch, in addition to true character mismatches, potentially leading to quite a few missed copies. In our tests, sequences with more than 10-15% N's did not yield almost any matches under RepeatMasker's default settings, despite the overall good agreement of other real characters.

For example, Table 5.1 shows a multiple alignment of the six sequences. At the bottom of this alignment, "*" represents a fully conserved column, "." represents a column where we can get a clear majority character, and "?" represents a columns with ambiguity. If we attempt to find the consensus sequence from this multiple alignment, the columns featuring a "*" and "." do not pose a problem as we have a clear choice. However, if we choose "N's" in the columns featuring a "?", our consensus sequence may became very poor. We have thus decided to follow the maximum likelihood principle for determining the consensus characters, and in cases of ambiguity instead of assigning an "N" use evolutionary criteria (i.e. the existing

Table 5.1. Multiple alignment of six sequences and their consensus sequence

G	G	A	G	T	T	T	G	A	G	A	C	C	A	G	C	C	T	G	G	G	C	A	A	C	G	T	G	G	C
G	G	A	G	T	T	T	G	A	G	A	T	C	A	G	C	C	T	G	G	C	C	A	A	C	A	T	G	G	C
G	G	A	G	T	T	T	G	A	G	A	C	C	A	G	C	C	C	G	G	C	C	A	A	C	A	T	G	G	T
G	G	A	G	T	G	C	A	A	T	G	A	C	A	C	G	A	T	C	-	T	C	T	G	C	G	C	A	C	T
G	G	A	G	T	G	C	A	G	T	A	G	C	G	C	A	A	T	C	-	T	C	G	G	C	T	C	A	C	T
G	G	A	G	T	G	T	G	G	T	G	G	C	A	G	G	A	T	C	-	C	T	A	G	C	T	C	A	C	T
*	*	*	*	*	?	.	.	.	?	.	?	*	.	.	.	?	.	?	.	.	.	.	?	*	?	?	?	?	.
G	G	A	G	T	G	T	G	A	G	A	G	C	A	G	C	C	T	G	G	C	C	A	G	C	G	C	G	G	T

knowledge about nucleotide substitution patterns). It is well known that the predominant substitution in vertebrates is the neighbor–dependent and irreversible CpG mehylation deamination process (CpG → CpA/TpG) [84]. Furthermore, studies on pseudogene sequences [40] have shown that, at least in the human genome, relative substitution rates for four nucleotides can be arranged in the following relative order: Substitution(G) > Substitution(C) > Substitution(A) > Substitution(T). We use this information to resolve any ambiguity in cases where a majority character is not clear and we choose the base with the higher substitution rate from the bases (two or more) causing the ambiguity. We treat a gap in the multiple alignment as a character, but choose it as the consensus (insertion/deletion) only if it is present in a significant majority (80% or more) of the aligned sequences. The consensus sequence resulting from this approach for the multiple alignment in Table 5.1 is shown in the bottommost row.

5.1.3 Classification of consensuses

Once the consensus sequence of each cluster of elements has been determined, we proceed to establish its biological properties, and attempt its classification. For that purpose we used the RepClass software [72], developed by our collaborators.

RepClass is a toolset which automatically classifies transposable elements provided by their consensus sequences. It is a high throughput workflow model, leverag-

ing custom scripts and other third party programs such as WU-blast (Warren Gish, 1996–2004, http://blast.wustl.edu), and palindrome and inverted detection from the EMBOSS suite [78], in order to classify transposons in new genomes. RepClass employs a multi–step approach which gathers information using several independent methods, and combines the collected information in order to come up with a tentative transposon classification. In particular, RepClass integrates the results of three independent classification methods: homology, target site duplication (TSD) search, and structural search.

When using homology, the software tries to classify sequence consensuses by comparing them to the already annotated transposon families in RepbaseUpdate [51]. During the target site duplication search, RepClass looks for TSDs which are usually formed at the host site during the transposition. In the structural search it tries to classify the elements by identifying their structural characteristics such as terminal repeats flanking the transposon copies.

Classification of a single transposable element requires a single run of each of the classification methods listed above. Therefore, the classification of elements in a complete genome requires several hundreds of thousands of iterations. That task takes anywhere from several days to several months, depending on the size of the genome and the quality of its assembly, when done sequentially. Since the classification methods used in RepClass are independent and since the classification of one sequence is not dependent on the classification of any other, this process yields to a parallel implementation, which scales well on clusters and grids. We have thus run such implementation of RepClass on the Distributed and Parallel Computing Center (DPCC) cluster at the University of Texas at Arlington. DPCC currently consists of 81 dual processors, 2.667 GHZ and 2.44 GHZ Xeon compute nodes with 2GB of local memory each. The software used a varying number of processors on the

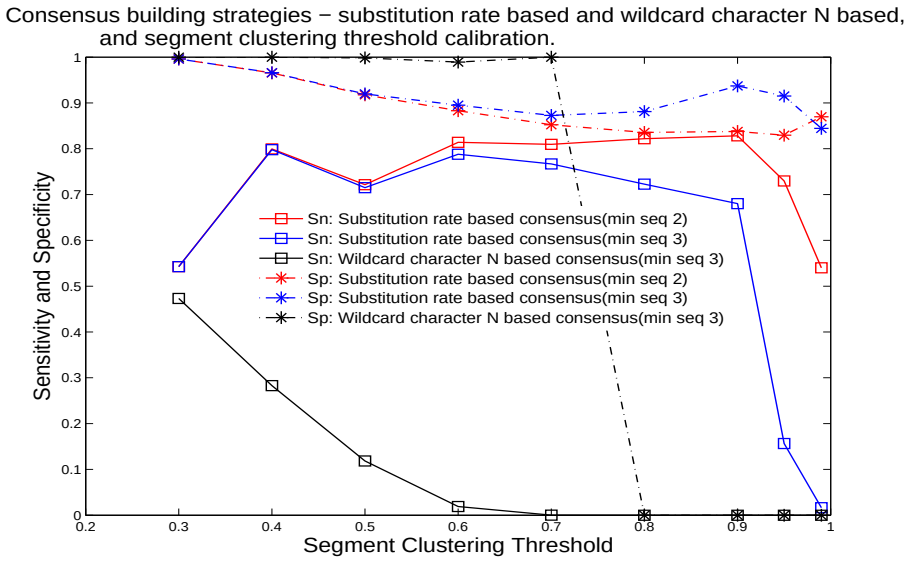

Figure 5.1. Simulation results for consensus building in simulated environment, for varying clustering thresholds and character assignment strategies. "Sn" stands for Sensitivity and "Sp" for Specificity.

cluster for different genomes, from 20, 40, and 60 (different in several runs) processors for the *Drosophila* genome to 100 for human.

The topmost division RepClass attempts to achieve is the classification of the transposons into classes, depending on the presence of the traces of the reverse transcriptase coding region. If they can be recognized the element can be classified as a retroposon, replicating through an RNA intermediate. Otherwise, it is considered a DNA–based. After that, RepClass attempts the identification of the right subclass: for retroposons it looks at non–LTR retrotransposons (LINEs or SINEs), LTR retrotransposons, DIRs and Penelope-s. If DNA–based , it attempts to recognize them as Maverick-s, DNA transposons and Helitron-s. If the subclass has been successfully identified, it goes one step further in an attempt to allocate the correct superfamily. RepClass is as implementation of a distributed cluster and grid based workflow in order to classify transposable elements. However, the details that work have been omitted here, as they concern the work of another research group.

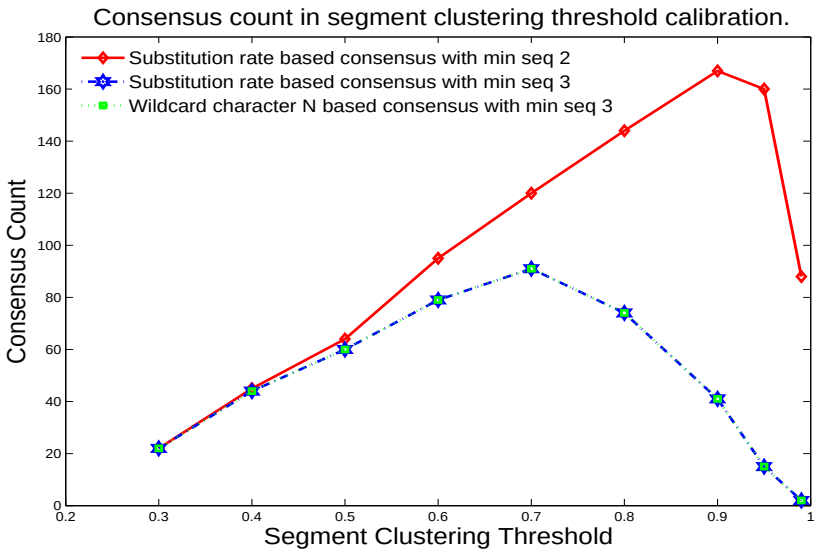

Figure 5.2. Simulation results for consensus counts in simulated environment, for varying clustering thresholds and character assignment strategies.

5.2 Calibration and testing

We have first run our program RepFi on both simulated sequences (providing a well controlled environment in which we knew exactly how many repeats were inserted, and where they were), and real genomic data (human chromosome 21). We have used these sequences to: 1) calibrate the numerous parameters to our software; 2) compare the performance of our software with other publicly available tools, RepeatScout in particular; and 3) to estimate how much our findings agree with the annotations derived from RepBase Update, through use of the RepeatMasker. In addition, we were also interested in how many of the consensuses we built can be classified, and how would these classifications relate to these from manually established genome annotations.

In order to calibrate the mostly orthogonal parameters to various components of our software suite we have run several hundred simulation runs. We have described the calibration of the core repeat finding in the previous chapter, and we here con-

centrate only on our efforts to optimize the consensus building and the subsequent classification. In order to evaluate the effectiveness of our consensus building approach, we ran the measurements in the simulated environment described in Section 4.2, Experiment 2. We finalized that setting by introducing random mutations into the conglomeration, and degraded the sequence for about 30%.

In the simulation runs we have computed the sensitivity and specificity of retrieving the elements we had inserted (measured as the percentage of base positions) for three sets of specifications: 1) substitution rate based approach to consensus building with a minimum of 2 participating sequences (determined by the cutoff in the clustering tree), 2) substitution rate based approach to consensus building with a minimum of 3 participating sequences, and 3) ambiguity character (wildcard character 'N') based approach to consensus building with a minimum of 3 participating sequences. We computed the sensitivity and specificity using the approach illustrated in Figure 4.3. The outcome of this simulation is shown in Figure 5.1. In these simulations we have varied the clustering threshold (defined in the Section 5.1 above), from 0.3 to 0.99. This calibration has indicated that our software achieves the best tradeoff between the sensitivity and specificity at 0.4 and 0.6 threshold with a minimum of 3 sequences in the alignment with maximum likelihood derivation (i.e. with the substitution rate based consensus building approach). The segment clustering threshold in these simulation runs indicates the minimal required sequence similarities between two repeated segments. The higher values of this threshold would result in a smaller number of segments in a cluster, and, in turn, smaller number of copies of the repetitive segments. However, lower values of this threshold would result in a greater number of segments in a cluster, but, on other hand, the higher number of segments in a cluster would also result in a poorer consensus sequence, resulting in a tradeoff between the quality of consensus and the number of repeat copies it is

Table 5.2. Performance comparison of our algorithm and RepeatScout in identifying known and previously characterized repeats.

	RepFi Sn	RepFi Sp	RepeatScout Sn	RepeatScout Sp
Experiment 1	0.5230201	0.8714505	0.4466029	0.9938296
Experiment 2	0.5752110	0.8680427	0.6082956	0.9840795

based on. Though minimal number of two sequences in a cluster is not a preferred scenario, we have chosen this setting only to verify the sequence similarity among two segments in a cluster resulting from a very high segment clustering threshold. As expected with such settings, we obtained a very high number of clusters (thus, consensuses) with the program setting a minimum of two sequences in a cluster. The simulation results for the consensus counts in the simulated environment for varying clustering thresholds and character assignment strategies are shown in Figure 5.2.

In the continuation of this work, we have decided to look at the results of RepeatScout only, since it was both our observation and the result of other related studies [91] that it generally outperforms other tools, including PILER. Moreover, RepeatScout is now emerging as a *de facto* standard in *de novo* repeat annotation, and thus as a benchmark with which any new solution should be compared.

Looking at the real genomic data, we have compared the performance of our algorithm with that of RepeatScout, using human chromosome 21. Since this comparison was mainly targeted towards identifying known and previously characterized repeats, it was easy to calculate the sensitivity (Sn) and specificity (Sp) by repeat masking the chromosome 21 using the RepBase human library and the libraries produced by both programs, then comparing the masked characters. The results are shown in Table 5.2. Again, we used the approach described in Figure 4.3 to calculate the sensitivity and specificity. Experiments 1 and 2 have been performed under

Table 5.3. Comparison of our algorithm and RepeatScout in identifying known and previously characterized repeats. "Families" refer to the determined number of consensus sequences.

	Total Families
RepFi	1088
RepeatScout	422

different parameter settings for both programs. In Experiment 1, we used the l-mer size for RepeatScout same as the smallest motif size (i.e. 7) in our case. The rationale behind it was to run RepeatScout and our software under a similar setting, without any preference on the l-mer size. However, in Experiment 2, we attempted to maximize the sensitivity of both tools by choosing the optimal l-mer size (i.e. 14) in the case of RepeatScout and optimal threshold (i.e. 0.8 or more) for choosing a gap over character in our case. The outcome of the comparison of our algorithm and RepeatScout in identifying known and previously characterized repeats families are shown in Table 5.3. We have also attempted to classify these families as transposable elements using RepClass software. RepClass classified 570 families out of 1088 in our case and 235 families out of 422 in the case of RepeatScout. However, the results of the RepClass classification were purely computational and they have not been validated by a domain expert. We may have received some false positives in this classification; however, that depends solely on the accuracy of RepClass, rather than on the quality of the consensuses we have supplied to it.

The results shown in tables 5.2 and 5.3 indicate that our method achieves results comparable with these of RepeatScout when identifying known, previously characterized repeat families. However, the somewhat lower specificity of our program also indicates that we may have found additional elements which are, presumably, too broken to be readily recognized, and which have thus not been identified yet, either

Table 5.4. Comparison of our algorithm and RepeatScout in identifying old, broken, previously unidentified repeats. "Families" refer to the determined number of consensus sequences.

	Total New Families
RepFi	183
RepeatScout	10

through manual annotation or other available tools. In order to see how well our program would identify broken, old, and previously unknown repeats (and confirm that we are indeed capturing these, and not just noise), we repeat masked chromosome 21 using the RepBase human library, and ran both our software and RepeatScout on the remaining unmasked sequence. The results of this experiment are shown in Table 5.4. It is worth noting that out of 10 repeat elements identified by RepeatScout most were very small (on the order of 100 base pairs or less). In contrast, the elements reported by our software were large, varying between 1000 and 5000 base pairs, on average. The RepClass classified 28 repeat elements out 183 as transposable elements in our case, and none from the RepeatScout's output.

5.3 Identification of repeats in repeat-masked human genome

In the Section 5.2, we have shown that our approach to identifying the unknown repeated DNA elements (i.e. the repeats in addition to those deposited in databases such as Repbase was indeed successful. Although the repeated DNA elements (transposable elements, segmental duplication, and other simple sequences) in the human genome have been already well annotated, the nature of a large part of human DNA is still unknown. Earlier in this thesis we have postulated that apparently unique non–functional DNA could be a result of ancient repeat activities, primarily transposons, but the segmental duplication could have also played a role. In order to understand

the nature of the newly identified repeated elements, we have attempted to characterize them by their copy numbers, lengths, occurrences within the known segmental duplications, their intersection with the 28-way conserved elements in placental mammals (established by publicly available large multiple alignment [101]) , and by their intersection with the nested TE's regions (John Pace, personal communication). We have also used RepClass to classify these newly identified repetitive DNA elements as transposable elements.

5.3.1 Approach

In order to identify new repeats in the repeat-masked human genome, we have used methods as described in Section 5.1. We first repeat-masked the entire human genome (chromosome by chromosome) using RepeatMasker with default settings, and then used the unmasked sequences from each chromosome to identify new repeated DNA segments. The following are the steps used in this experiment:

1. We repeat-masked complete the human genome, chromosome by chromosome.
2. We calculated the percentage of masked bases.
3. We ran our software on each repeat-masked chromosome, and identified the library of the consensus sequences of the repeated segments.
4. We concatenated the libraries built from of all chromosomes in order to assemble a comprehensive library.
5. We self-blasted the consensus sequences in the comprehensive library using WU-blast (Warren Gish, 1996–2004, http://blast.wustl.edu) in order to find the similarities among the consensus sequences resulting from different chromosomes.
6. We filtered out significantly overlapping consensuses by choosing the larger sequences in the blast hits, and built the final library of repeated consensus sequences.

7. We repeat-masked all the pre-repeat-masked-chromosomes (from step 1) of the human genome, and identified the exact copy numbers related to each consensus sequence, using the output of RepeatMasker.
8. We re-calculated the percentage of masked bases as masked by RepFi software, in addition to masked by RepeatMasker in Step 1.
9. We attempted to classify the consensus sequences in the final library as possible transposable elements using RepClass.

All our runs on whole human genome data have been performed on the Distributed and Parallel Computing Center (DPCC) cluster. As RepeatMasker takes a very long time to mask the entire genome in a single run, we have chosen to run it by masking one chromosome at a time, executing all 24 runs (for all human chromosomes) in parallel on different nodes of the DPCC. This enabled us to have the entire human genome repeat-masked in (approximately) 24 hours. In addition, since, RepFi software is memory-expensive, working on the entire human genome in a single run was not even possible. Therefore, we ran RepFi chromosome–wise too, again executing all 24 runs in parallel on the DPCC. The completion of these runs on the most chromosomes took approximately 5 hours.

5.3.2 Results and analysis

After finishing the runs of our software on all human chromosomes and assembling the final library of consensus sequences, we have found a large number (9890) of consensus sequences, varying in size from 356 bases to 5861 bases. Their copy numbers varied from 5 to 1118. These copy numbers have been counted based on the output of RepeatMasker, which confirmed that our consensus sequences were indeed representative of real repeated sequences. In an attempt to characterize them, we have again used RepClass, which classified 314 of our consensus sequences as trans-

Table 5.5. The summary of the results of the extended annotation of the entire human genome.

Measure Description	Values
Total number of new consensus sequences	9890
Shortest length of a new consensus sequence	356bp
Longest length of a new consensus sequence	5861bp
Smallest copy number of a new consensus sequence	5
Largest copy number of a new consensus sequence	1118
Masked percent of the human genome before RepFi run	51.13%
Masked percent of the human genome after RepFi run	55.71%

posable elements. Though it was encouraging to see that a good number of our new elements were classified by RepClass, it was difficult to validate these results in an automated way. Ultimately, the validation would require a manual lookup by a domain expert. We have also measured the percentage of masked bases, both before starting and after finishing the runs of our software on the entire human genome. As a result of this, we have found that the RepFi software has masked 4.6% of bases as part of repetitive segments in addition to the known repetitive contents in the human genome. The summary of these results is shown in Table 5.5.

In order to further characterize the repetitive regions identified by our software, we have intersected these regions with segmental duplication regions, the regions of the 28-way conserved elements in placental mammals, and regions of nested TE's. We then counted the total number of bases in the overlapping pieces of intervals. We used the Galaxy online software (http://main.g2.bx.psu.edu/) to obtain the intersection between the datasets (in Browser Extensible Data – BED format). The summary of these intersections is shown in Table 5.6. Approximately 22% bases from our repetitive regions were identified as previously established segmental duplications, and 8% of bases were found in the 28-way conserved elements record in placental mammals.

We have associated a simple probabilistic significance to these intersections, assuming the following notation:

U = set of total bases in the human genome,

A = set of bases in the repetitive regions identified by our software,

B = set of bases in the record of the segmental duplications in the human genome, and

C = set of bases in the 28-way conserved elements in the placental mammals record.

D = set of bases in nested TE's regions record.

As total number of bases in the human genome (hg18) are 314193916 (i.e. $|U| = 314193916$), we have calculated the probabilities of the sets A, B, C, and D as given below:

$$P(A) = \frac{|A|}{|U|} = 0.01546 \tag{5.2}$$

$$P(B) = \frac{|B|}{|U|} = 0.20147 \tag{5.3}$$

$$P(C) = \frac{|C|}{|U|} = 0.03562 \tag{5.4}$$

$$P(D) = \frac{|D|}{|U|} = 0.00028 \tag{5.5}$$

From our established counts in the intersections, the probabilities of $A \cap B$, $A \cap C$, and $A \cap D$ are given below:

$$P(A \cap B) = \frac{|A \cap B|}{|U|} = 0.00344 \tag{5.6}$$

$$P(A \cap C) = \frac{|A \cap C|}{|U|} = 0.00126 \tag{5.7}$$

$$P(A \cap D) = \frac{|A \cap D|}{|U|} = 0.000036 \quad (5.8)$$

We have also calculated the probabilities of a base being a part of $A \cap B$, $A \cap C$, and $A \cap D$ by chance, assuming that set A is independent of sets B, C, and D:

$$P_{ind}(A \cap B) = P(A)P(B) = 0.00311 \quad (5.9)$$

$$P_{ind}(A \cap C) = P(A)P(C) = 0.00055 \quad (5.10)$$

$$P_{ind}(A \cap D) = P(A)P(D) = 0.0000043 \quad (5.11)$$

The probability $P(A \cap B)$ (Equation 5.6) is approximately equal to the probability $P_{ind}(A \cap B)$ (Equation 5.9), and we conclude that sets A and B are indeed independent, i.e. that we have gotten the intersections just by chance:

$$P(A \cap B) \approx P_{ind}(A \cap B) \quad (5.12)$$

$$\Rightarrow P(B)P(A/B) \approx P(B)P(A) \quad (5.13)$$

$$\Rightarrow P(A/B) \approx P(A) \quad (5.14)$$

However, the probability $P(A \cap C)$ (Equation 5.7) is greater than the probability $P_{ind}(A \cap C)$ (Equation 5.10), i.e.:

$$P(A \cap C) > P_{ind}(A \cap C) \quad (5.15)$$

$$\Rightarrow P(C)P(A/C) > P(C)P(A) \quad (5.16)$$

$$\Rightarrow P(A/C) > P(A) \quad (5.17)$$

The probability $P(A \cap D)$ (Equation 5.8) is greater than the probability $P_{ind}(A \cap D)$ (Equation 5.11), so:

$$P(A \cap D) > P_{ind}(A \cap D) \quad (5.18)$$

$$\Rightarrow P(D)P(A/D) > P(D)P(A) \quad (5.19)$$

$$\Rightarrow P(A/D) > P(A) \quad (5.20)$$

Table 5.6. Intersection of RepFi consensus matching regions with known segmental duplications in the human genome, the 28-way alignment conserved regions in placental mammals, and with nested TE's regions record.

Set Description	Base Pairs
Total bases in RepFi consensus matches (as identified by RepeatMasker): Set A	48,595,993
Total bases in known segmental duplication regions of the human genome: Set B	633,006,079
Total bases in 28-way alignment conserved regions in placental mammals: Set C	111,931,514
Total bases in the nested TEs regions record: Set D	895,662
Intersection in consensus matches and segmental duplication record ($A \cap B$)	10,815,661(22%)
Intersection in consensus matches and 28-way conserved regions ($A \cap C$)	3,986,260(8%)
Intersection in consensus matches and nested TEs regions record ($A \cap D$)	113,247(0.2%)

As shown in Table 5.5, our software has identified 9890 consensus sequences. Since it is very hard to look at each of them manually, we have chosen to analyze only the top consensus sequences individually, at a finer granularity, by looking at their lengths, total copy numbers, copies conserved 80% or more with respect to their lengths, and their intersection with known segmental duplications. We have also looked at the general structure of these sequences manually. We prepared three sets of such consensus sequences: 1) Set 1 — top five sequences in copies conserved 80% or more with respect to their lengths, and classified by RepClass software; 2) Set 2 — top five sequences in copies conserved 80% or more with respect to their lengths, but not classified by RepClass software; and 3) Set 3 — top five sequences in total copies, and classified by RepClass software. For the first set of consensus sequences, as shown in Table 5.7, almost all sequences have shown a good number of very well conserved copies (80% or more with respect to their lengths). However, these copies were present in segmental duplication regions as their intersection with segmental duplication had 100% overlap. These results served as a sanity check for our program, i.e. as proof that it finds the repeated segments if they are present in the given input sequence. At the beginning of our experiments, we have repeat-masked the entire human genome to remove all the known repeated DNA elements. However,

the repeat-masking had not removed the segmental duplications. It is conceivable that some of these segmental duplications exhibit the over-representation of short sequences (i.e. motifs), and hence our software identified them as repeated sequences. Though we masked out all simple sequences and filtered out all other tandem repeats during the RepClass runs, we have noticed that consensus sequence with Sequence ID 9173 exhibited tandem repeats consisting of short sub-sequences.

For the second set of the consensus sequences (top five sequences in copies conserved 80% or more with respect to their lengths, but not classified by RepClass software), as shown in Table 5.8, the intersection with known segmental duplication records have not shown consistency as compared to that of Set 1. This indicates that conserved copies of the consensus sequences are not restricted to segmental duplications. For example, the consensus sequence with Sequence ID 9889 (3030 bases long) was present in 395 well conserved copies; however, only 1.8% bases from these copies had intersection with known segmental duplications. Surprisingly, there was no intersection between the well conserved copies of the consensus sequence with Sequence ID 6940 and the segmental duplications. Moreover, the consensus sequences with Sequence ID 9889 and 8990 also exhibited tandem repeated short sub-sequences. A Blat lookup at UCSC Genome Browser (http://genome.ucsc.edu/) on the consensus sequence (Sequence ID 6940), which showed no overlapping with segmental duplications, has indicated that this consensus sequence is highly similar to the coding region of an RNA gene that encodes a small nuclear RNA (snoRNA). We know that snoRNA genes feature many copies in the human genome, so they could be treated by our software as repeated elements.

Since we have not removed the segmental duplications in the beginning, we have identified several well conserved repeated consensus sequences overlapping these regions. However, our main objective was to locate the broken sequences, and we

believe that these broken sequences would not exhibit well conserved copies. The results shown in Table 5.9 represent our third set of consensus sequences (i.e. top five sequences according to the total number of copies, and classified by RepClass). These sequences are repeated in large copy numbers, but they do not show significant intersections with known segmental duplications. However, the consensus with Sequence ID 2402 was an exception in the sense that it also participated in our Set 1. As shown in Table 5.6, only 22% of bases in our repetitive regions had an intersection with known segmental duplications, and remaining 78% of the bases of these regions are comprised of sequences which are not well conserved in their.

We were also interested to know whether our software would be able to identify the consensus sequences similar to the recently identified 7 families (MARE1, MARE2, MARE3, MER124, MER128, MER135, and X3_LINE) of ancient transposable elements [20]. The consensus sequences of these families are available in RepBase, but they are not included in standard repeat-masking. When we compared, using BLAST, our consensus sequences with the consensus sequences of these families, we have found two families (out of 7) which were showing significant one–to–one hit (with the identity of 68% or more): MER128 with our consensus Sequence ID 1051 and X3_ LINE with our consensus Sequence ID 6338. This indicated that our method was indeed able to find the consensus sequences of some confirmed old transposable elements.

5.4 Discussion

While the performance of our tool on well conserved, and thus very similar, copies of recent transposons is comparable with that of other *de novo* finders (Table 5.2), it clearly over–performs even the best current tools in the identification of highly degenerated repeated elements (Table 5.4). However, while it can locate low–

Table 5.7. The summary of the top 5 RepFi classified consensus sequences (Set 1) having copies matching 80% or more of their lengths.

Sequence Id	Length	Total Copy Number	Copies Matching 80% of Length	Total Bases in all Occurrences	Bases in Copies Matching 80% of Length	SD Intersect. with Copies 80% of Length	SD Intersect. with all Copies
9173	865	73	35	41663	28206	28206(100%)	41191(98%)
7088	2472	79	20	95066	48319	48319(100%)	93782(98%)
2402	3390	235	10	229540	32792	32972(100%)	217543(94%)
940	753	10	9	6557	6510	6510(100%)	6510(99%)
7064	576	14	7	5802	3403	3403(100%)	5802(100%)

Table 5.8. The summary of the top 5 RepFi unclassified consensus sequences (Set 2) featuring copies matching 80% or more of their lengths.

Sequence Id	Length	Total Copy Number	Copies Matching 80% of Length	Total Bases in all Occurrences	Bases in Copies Matching 80% of Length	SD Intersect. with Copies 80% of Length	SD Intersect. with all Copies
9889	3030	606	395	1,340,322	1,155,873	21,169(1.8%)	242,394(18%)
9890	1073	546	353	451,019	356,828	109,031(30%)	148,427(32%)
6913	755	48	42	31,722	30,246	28,066(92%)	29,362(92%)
6953	1775	133	40	104,132	65,594	65,594(100%)	95,304(95%)
6940	817	57	38	33,717	28,636	0	281(.06%)

Table 5.9. The summary of the top 5 RepFi classified consensus sequences (Set 3) with all copies (each copy is a repeat-masked region).

Sequence Id	Length	Total Copy Number	Copies Matching 80% of Length	Total Bases in all Occurrences	SD Intersect. with all Copies
4645	1700	446	0	125428	9146 (7%)
7843	2572	393	1	21163	944(4%)
4451	1338	358	1	97096	407(.4%)
4420	1643	282	1	14083	473(3%)
2402	3390	235	10	229540	217543(94%)

copy number repeats in cases of well conserved elements, it is only capable of finding intensively replicated highly degraded transposons. This places a constraint on its power, however one which we have to accept. When the noise overwhelms the signal there is very little one can do in order to reconstruct it.

Analyzing the on repeat–masked human genome, we have identified a significant amount of repetitive DNA in addition to the known repeated elements; however,

annotating this repetitive DNA remains a challenge. The number of our consensus sequences was so large that we could not look at all of them. Although the RepClass software has classified many of our consensus sequences as transposable elements, validation of these elements is a daunting task. An important evidence one can use to characterize a repetitive sequence as a transposable element is the existence of target side duplications (TSDs) on both of its ends. The identification of the TSDs becomes a slightly easier task if they are conserved at all or many occurrences of the transposable elements; however, TSDs are not generally conserved at all sites. If these TSDs are well conserved motifs, our software would treat them as over-represented motifs, and it would identify them in the consensus sequences. Sometimes even if the TSDs are not over-represented they can still be a part of our consensus sequences because of our relaxed approach to building the consensus sequences (i.e. considering the gap over a character only if the gap shows up more than 80% of cases). To locate the TSDs when they are a part of the consensus sequences, we have divided a consensus sequence into two parts and attempted to find the over-represented motifs using our MotFi software discussed in Chapter 2). MotFi has successfully identified the repeated motifs with very significant p-values in almost all the consensus sequences (except a few which have been dominated by tandem repeats) included in tables 5.7 to 5.9. The size of these motifs varied from 10 bases to 21 bases. Although we have observed conserved occurrences of such motifs in almost all consensus sequences, the spatial distribution and the list of selected motifs for Sequence ID 7088 are shown in Figures 5.3, as a representative outcome. However, we do not have substantial evidence to characterize these motifs as TSDs, as they can be some other signals (such as TF binding sites or part of LTRs) or repeated for some other reason. Consequently, we could not acquire sufficient evidence to annotate our consensus sequences as transposable elements using an automated approach. We believe that a meaningful annotation will require manual

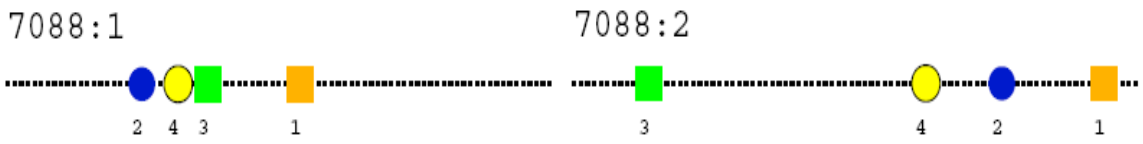

Figure 5.3. The layout of overrepresented conserved motifs (possible candidates for conserved TSDs) in the RepFi consensus sequence 7088. This sequence exhibited copies (occurrences) matching 80% or more of its length and it was classified by RepClass.The duplicated motifs shown in this sequence are 1. AAAACACTTTAAAA, 2. AATNNNTTTTAAAGT, 3. AGAGGCNNNCAGGC, and 4. CATTCNNNTTAAGG.

look up and analysis by a domain expert. Some parts of our repetitive regions have shown the intersection with the 28-way conserved elements in pre–made alignment of sequences from placental mammals. This indicates that in addition to transposable elements and segmental duplications, the evolutionary natural selection may have also played a role in accumulating repetitive DNA in the human genome.

There is still work that should be done to improve our system. We are not content with the setting up of the cuts in the segment clustering tree in such way that it results in a heuristic minimal amount of sequences in each cluster — these cuts should be made so that the resulting clusters precisely correspond to the division of elements into families a biologist would make when manually constructing a library. This is a very difficult task to achieve, as reflected in the poor correspondence of any automatically derived consensus (i.e. by any current tool) to the existing annotations of classes of transposable elements. Nevertheless, a good share of the sequences we discover do classify (although the RepClass classification have not been validated), and even as most of our consensuses would need further work in merging, splitting, trimming or extending this shows that the signal we capture is indeed real.

CHAPTER 6
SUMMARY AND PLANS

In this thesis, we have studied motif over-representation with respect to the baseline Poisson model, and we have presented new efficient algorithms for finding and characterizing significantly repeated motifs in genomic sequences. We have developed two major software tools (MotFi and RepFi) to identify the over-represented motifs, and to associate their co-occurrences and reconstruct the consensus sequences of presumed large broken repeated segments. These software tools are publicly available for download and web access at our website. The web link for MotFi software is http://bioinformatics.uta.edu/toolkit/motifs and for RepFi software is http://bioinformatics.uta.edu/toolkit/repeats. These software tools can be downloaded from http://bioinformatics.uta.edu/toolkit/download, and installed for local use. We are confident that the webserver for our MotFi software would prove to be a useful tool to study the motifs in genomic sequences, especially putative regulatory regions. However, the webserver for RepFi software is a prototype version because finding repeats in short sequences is of limited use, and finding repeats in long sequences requires substantial data transfer, processing time, and data storage space. We are still evaluating the feasibility of the webserver for our RepFi software.

Our MotFi software has generally performed reliably. However, the number of motifs it reported were usually large, even when only the items with highly significant p-values were taken into account. The lengths of these motifs were also relatively large. This may have been expected as long motifs are more likely to achieve high statistical significance, but it nevertheless warrants some further attention. We decided

to record every motif which could possibly represent a signal, rather than relying on filtering during the detection. We could then filter the results according to the number of occurrences, motif composition, database hits (such as JASPAR [15], RepBase [52], and TRANSFAC [99]), clustering patterns, positional conservation, and joint conservation across orthology and paralogy. Often the hits in TRANSFAC database were not conclusive, as we were regularly recording multiple significant hits for most of our top motifs, and the JASPAR database is still small, with not many entries to match.

While applying our MotFi software, we were often finding motifs almost exclusively comprised of A's and T's. We believe that such motifs are remnants of poly-A tails. Finding them was not a surprise, as poly-A tails are known to be present in many copies throughout eukaryotic genomic sequences. They could have been derived by several mechanisms, such as from the terminus of non-LTR retrotransposon repeats. Though these motifs were potentially significant, we have chosen not to study them further, and consequently filtered them out. Although the application of an effective filtering criteria was not an easy task, by generating a large list of motifs we were able to find promising candidates. Indeed, our list of high scoring motifs included many experimentally confirmed sites. Furthermore, our approach successfully found motifs which were missed by other software tools, and these collected in the BEST suite in particular. Our current algorithm works only on multiple input sequences (presumably upstream sequences of the co-expressed genes) and it returns variable motifs occurring in some subsets of these sequences. However, in the future this algorithm may be possible to be extended to accept one long genomic segment (a chromosome or even a complete genome). Another possible extension can be the introduction of a facility to identify motifs featuring small insertions and deletions.

In this work we have observed that, in addition to known repeated elements, the human genome features a micro–repetitive structure characterized by an over-

abundance of short motifs. This result cautions us from using over-representation for the determination of functional DNA elements because that inevitably results in a large number of false positives. It is also difficult to find an adequate model of general genomic environments. The Markov models which are commonly used to simulate the DNA sequences have not shown good correlation with the real data with respect to their micro–repetitive pattern. As a result of this work, we have observed a consistent pattern for all considered motifs lengths indicating that the number of short repeated motifs in the genomic sequences was far greater than in any of the synthetic models. We have also observed that in upstream regulatory sequences of genes, the number of larger motifs (of length 7 to 9) which were repeated less than 5 times was higher in comparison to that of random genomic sequences, whereas the number of such motifs repeated 5 or more times was higher in random genomic sequences. However, the scope of this study was limited, as we have used relatively smaller datasets (100 sequences, each of 500bp, for each dataset). In the future, it would be interesting to expand this study to much larger sequences (of chromosome size, at least).

We have also attempted to establish the source of the noticed over-abundance of short motifs in human genomic sequences. Our RepFi software was able to find many new broken repeated elements, thus confirming our postulate that much of the over-abundance of short motifs throughout human genome is because of broken (presumably old) transpositional and other duplication activities. We have also benchmarked the performance of this software against other existing tools. RepFi has outperformed all other considered tools in identifying broken repeats with high copy numbers. Our simulation runs have shown that our approach to reconstruct the consensus sequences of broken repeated elements based on relative substitution rates of the four DNA bases works well. We have observed that there is a tradeoff between the copy number used to build the consensus and the quality of consensus

sequences. Though it is very hard to characterize a poor quality consensus sequence, one can still see the signature of broken repeats at multiple copies. The boundaries and the length of the putative elements were other important issues related to our consensus sequences. All repeat finding tools suffer from the problems regarding the determination of the accurate repeat boundaries. Our approach to build the consensus was very relaxed, as we choose a gap only in those cases where it was present more than 80% of the aligning copies. This often resulted in very large consensus sequences, extending the elements at both ends. The lengths of our consensus sequences heavily depended on the weight assignment during the constrained heaviest segment identification. Although our iterative approach helped us overcome this problem to some extent, we might still have ended up connecting several elements (represented as segments) into one, many times. More calibration runs could indicate the right weight assignment. However, it is hard to have a well controlled test environment dataset for calibrations of this kind, one which would be similar to the real genomic sequences, in the absence of good understanding of the nature of these sequences. In our testbed sequences for the test environment datasets, we have attempted to make these sequences as similar to real DNA sequences as possible. However, a lot more can be done to improve the quality of these simulations, particularly by choosing an evolutionary model for substitutions, and by increasing the length of the testbed sequences. The test environment sequences can also be improved by choosing a more realistic density of the repeated elements. One may also like to try the insertion and deletion of the repeated elements nested within other ones. However, we do not think that this would affect our computational approach, as constrained heaviest segment algorithm would find these segments even if they are nested in each other.

Unlike other currently available tools, our program can find the locations of repeats in a genome even before their consensus sequences have been reconstructed, and

thus it does not depend on the RepeatMasker to report them. Often this is all that a user wants — to mask the clearly repetitive structures in order to concentrate on more interesting sequences. Furthermore, our software can accomplish the basic masking within minutes, with results comparable to what other tools such as RepeatScout accomplish in hours because of their expense of running the time–consuming RepeatMasker matches. This is not to say that our software runs in an instant — for all but short segments (where looking for repeats would not make much sense anyway) it is indeed computationally expensive. It is just that it performs basic masking an order of magnitude faster. However, there are cases where full annotation of large genomes is needed, such as following a new round of sequencing and assembly. For this purpose our program can be run in extensive mode, determining the consensuses and invoking the RepeatMasker with this catalog (this strategy can detect additional copies which have been omitted when segments were laid out), as well as performing the classification. Depending on the genome size and the power of the computational infrastructure one has, this may take days of computation; however, full *de novo* annotation of complete genomes is not a common task executed daily, and in this context one can afford the wait.

We have also attempted to find unknown repeats in the entire repeat-masked human genome and identified a large number of consensus sequences of newly discovered repetitive segments. Our software was able to identify around 4.6% more interspersed repetitions in the human DNA, in addition to the known repeated elements. Some of our repetitive consensus sequences have shown well conserved copies and shared significant amount of bases with known segmental duplications. However, most of our repetitive regions (approximately 78%) have been found outside the segmental duplication regions. Though these results have confirmed that our software can successfully locate the repetitive elements (both broken and well conserved), char-

acterization of these sequences remains a daunting task. We have used RepClass for an automated classification of our consensuses, and it indeed classified a good number of these as transposable elements. However, we are not aware of any other way to validate these results, except manual annotation by a domain expert. Therefore, the RepClass classification results remained un-validated at this time. In the future, we plan to build a library of reconstructed consensus sequences of the transposable elements, i.e. these which have resulted from ancient repeat activities and which were so far unrecognizable due to their fragmentation over a long period of time. As we have seen in the results discussed in Chapter 5, many of our well conserved consensus sequences overlapped with the known segmental duplications. However, this was not a surprise as ancestral reconstruction of the segmental duplication has revealed punctuated cores in the human genome [105], and these segmental duplications can be annotated using tools like DupMasker [106]. We would also like to rerun our RepFi program after masking all known repeated (transposable) elements and segmental duplications from the human genome. More importantly, it would be interesting to see if the broken repeated elements (and possibly ancient transposons) are somehow associated with gene regulatory networks in human and other genomes. We would also like to know why many of our consensus sequences are overlapping with 28-way conserved track in the pre–made alignments [101]. Finally, we would like to see our tool finding its place as a method of choice for the annotation of repeats in newly sequenced genomes.

APPENDIX A
SOFTWARE ACCESS AND VISUALIZATION OF RESULTS

In chapter 2 of this thesis, we have presented an efficient algorithm to find all significantly over-represented variable motifs in the regulatory sequences of co-expressed genes [8]. The key idea of our approach is to detect everything that appears statistically significant. Further, we apply separate selection criteria depending on the nature of the sites one wants to locate. We have written a software "MotFi" which efficiently searches through multiple gene regulatory sequences and finds all significant variable motifs shared within subsets of these sequences. The software accepts the sequences in Fasta format, finds all significantly over-represented degenerate motifs, and generate a pictorial layout of motifs. Also, in an attempt to characterize the motifs, the software supports the search of the motifs in the RepBase and JASPAR databases.

We have also developed an algorithm to efficiently associate the co-occurrences of significantly over-represented motifs. This software can be used for *de novo* identification of repeats. This work has been described in Chapter 4. Our softwares MotFi and RepFi are publicly available on webserver, both for web access and download. This section describes the usage, architecture and technology of the webserver, and visualization of results.

A.1 Motifs finding software: MotFi

The objective of the MotFi software is to locate significantly over-represented variable motifs in the given sequences. It takes the input file with sequences in Fasta format, finds the list of significant motifs, and generate a PDF file showing the visualization of the selected motifs. In order to characterize the motifs for their possible functional role, the software also provides an option to search these motifs in the RepBase and Jaspar databases.

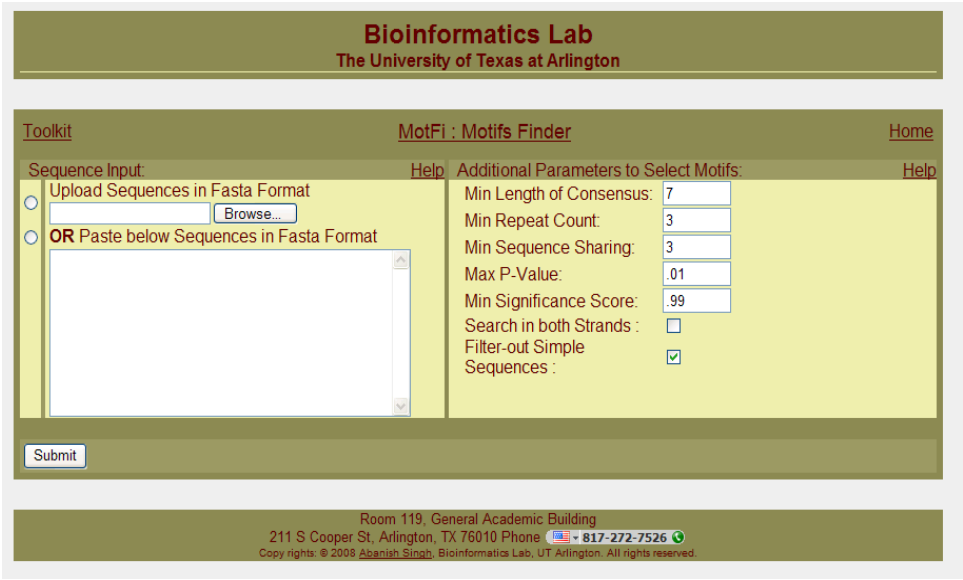

Figure A.1. The web interface of MotFi software.

A.1.1 Web access

The MotFi software can be accessed on the web at http://bioinformatics.uta.edu/toolkit/motifs. The web interface of the software is shown in Figure A.1.1. Using this interface, one may upload the input sequences in Fasta format using two methods: by uploading the file or by pasting the sequences in given text area. After selecting the appropriate radio button, one may browse and upload the file containing the sequences, or copy and paste the sequences in the given text area. An example for the format of input sequences is given below:

>Seq 1

ATCGAGCATTTCCAATAATAAGTGATGAGTCACCAATATCTAACTTTGTATT

>Seq 2

AATAAAGCTGGCGGCCGCGGGCTACTGCCAATATCCTGCGTTTGTGTGCGTG

>Seq 3

GAAACGCCAATCGTGGCTGCCAATATCCCTAAGGAGTGCCTGCCAATATCGC

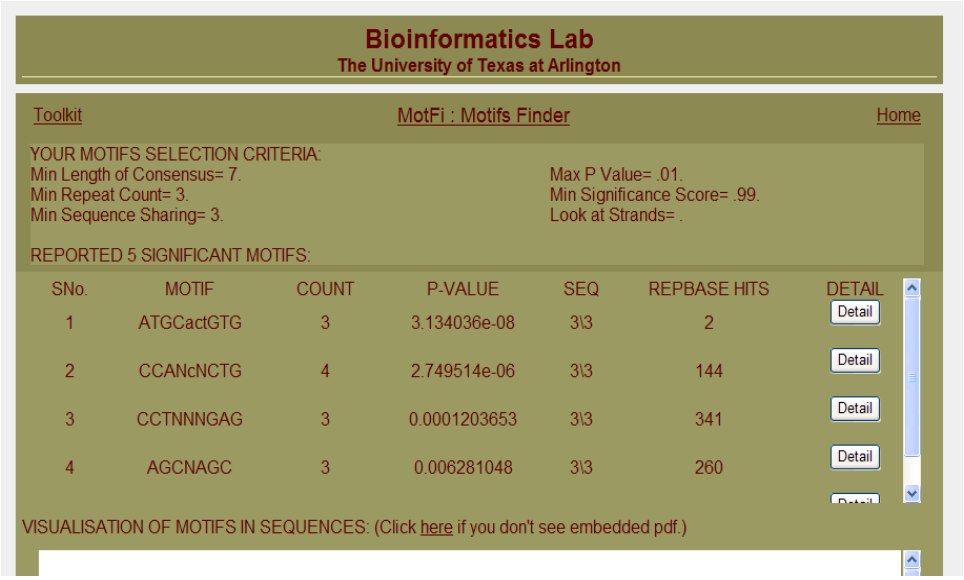

Figure A.2. The web report about significant motifs as generated by MotFi software.

Once the input sequences have been uploaded, the following parameters of the software can be selected to get the desired filtering of motifs:

- Min Length of Consensus: Specify the minimum length of the consensus motifs. For example, the length of CTtcNTA is 7.
- Min Repeat Count: Specify the minimum repeat count of the motif (in all given sequences). For example, a motif may show up 1 time in one sequence, 2 times in next sequence and 1 time in third sequence, so the total repeat count is 4.
- Min Sequence Sharing: The sequence sharing represents the number of different sequences a motif shows up. For example, out of five sequences a motif shows up in sequence 1, 2 , 3 and 5 but not in 4; in this case, the sequence sharing is 4.
- Max P-Value: The P-Value represents the probability of a motif showing up by chance. Specify the maximum acceptable P-Value for the selection of the motifs.

- Min Significance Score: The significance score represents the score of the motif consensus. Choose a minimum acceptable significance score for the selection the motifs.
- Search in both Strands: Select this option if you want that motifs should be looked in both strands.
- Filter-out Simple Sequences: The selection of this option enables the program to filter out simple and low complexity sequences.

Once the input sequences are provided and parameters are configured in the web interface, the software can be run by clicking the "Submit" button. After the software completes it's execution, a web report about the significant motifs, as shown in Figure A.2, is displayed. The upper part of this report includes the significant motifs, their repeat counts, p-values, sequences sharing, count of hits in the RepBase database, and a "Detail" button associated to each motif. The lower part of the report includes the visualization of the layout of selected motifs (in some browsers, the visualization may not show up embedded in the original window but in a new window). A click on the "Detail" button associated with each motif will display a supplemental report showing the detail of the RepBase database hits and JASPAR database matches.

A.1.2 Download

At this time the download for local installation is available only for Unix/Linux systems. The Makefile for these platforms is supplied in the code directory. It is zip-ed and tar-ed. So, it has to be uncompressed first, before running "make". Also, please make sure that you replace the first lines in Perl and Shell scripts with paths appropriate for your system.

The MotFi software can be invoked by running the perl script MotFi.pl with appropriate command line parameters. It accepts the following command line options:

-i <sequences input file>

-o <motifs output file>

-t <path of program files location, example /home/dirA/dirB>

-l <minimum motif length (optional): default=7>

-c <minimum repeat count (optional): default=3>

-n <minimum sequence sharing (optional): default=3 >

-p <maximum motif P-Value: default=.01>

-s <minimum significant score (optional): default=.99 >

-b <1 to choose single strand and 2 to choose both strand (optional): default=1>

-r <0 to choose no filtering of simple sequences and 1 to choose filtering (optional): default=0>

The required parameters are sequences input file, motifs output file, and path of directory where the software has been installed. For example, if one has installed the software in the directory /home/programs/MotFi and invoking the MotFi.pl script from the directory where the input.txt file is located, then:

$perl /home/programs/MotFi/MotFi.pl -i input.txt -o motifs.txt -t /home/programs/MotFi

will execute the software. Also, one may choose other parameters depending upon desired selection criteria as discussed in A.1.1. For the download version, the input sequences format will also be the same as discussed in A.1.1.

A.2 Repeats finding software: RepFi

The RepFi software can be used to identify the repeats in genomic sequences. Though primarily intended to identify to broken repeats, this software can be used

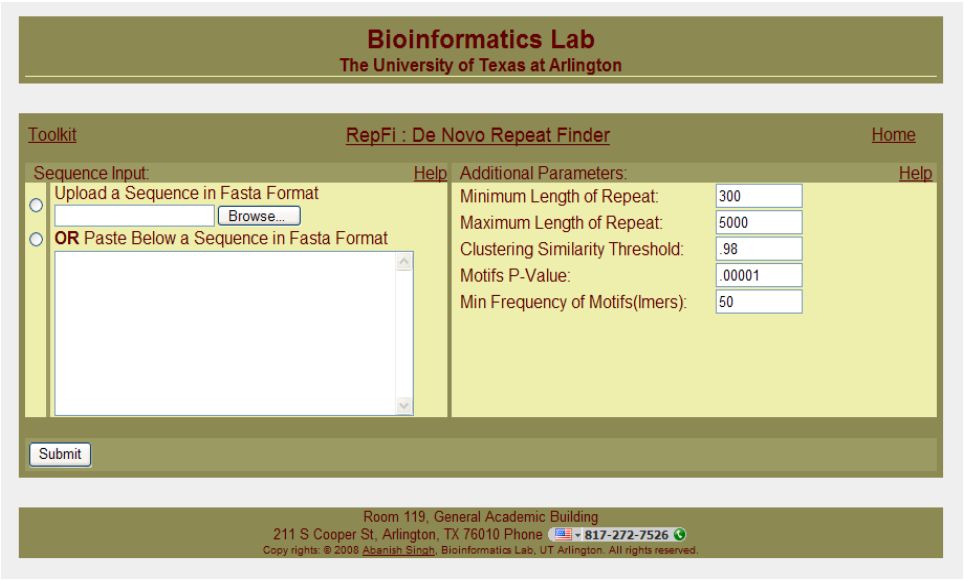

Figure A.3. The web interface of RepFi software.

for *de novo* identification of all repeats. It takes input file with sequence in Fasta format and finds the consensus sequences of repeated elements in the input sequence. For multiple alignment, it uses the clustalw program.

A.2.1 Web access

The RepFi software can be accessed on the web at http://bioinformatics.uta.edu/toolkit/repeats. The web interface of this software is shown in Figure A.3. This prototype web-server can accept the sequence in Fasta format using two methods: by uploading the file or by pasting the sequence in given text area. The file input option can be selected using the appropriate radio button present at left side. The size of the input sequence can be as large as 10MB; however, larger sequences may take longer time and result in browser time out. The appropriate selection of following parameters will help in finding the repeated elements in the given sequence.

- Minimum Length of Repeat: Specify the acceptable minimum length of repeated element.

- Maximum Length of Repeat: Specify the acceptable maximum length of repeated element.
- Clustering Similarity Threshold: This parameter represents similarity threshold required for clustering the sequence segments. A higher threshold would result in higher number of clusters, thus, in higher number of repeated consensus.
- Motifs P-Value: The P-Value represents the probability of a motif showing up by chance. Specify the maximum acceptable P-Value for motifs (lmers) selection. The associations among these selected motifs (lmers) are used in building the repeated elements' consensus. Specify the maximum acceptable P-Value.
- Min Frequency of Motifs (lmers): The motifs frequency represents the count of a motif (lmer) showing up in the given sequence. A motif (lmer) is considered to be over-represented (in this program) if it shows up more than this minimum frequency plus expected count, where expected count is calculated by program itself and the minimum frequency works as base count. The larger value of this option would result smaller number of selected motifs, thus in turn, faster execution of the program. For an idea, our simulation shows that for sequences as large as chr 21, appropriate value of this frequency is 1000, however, for smaller sequences (of length 2-5MB) it could be up to 50, and 0 for very small sequences (of order of few thousands).

Once input sequence is provided and parameters are configured at the web interface, the RepFi software can be run by clicking at the "Submit" button. The execution time of this software depends on the length of the input sequence. Once the execution of the software is completed, the web application displays a report page as shown in Figure A.4. This page includes the consensus sequences of all the repeated elements in the given sequence.

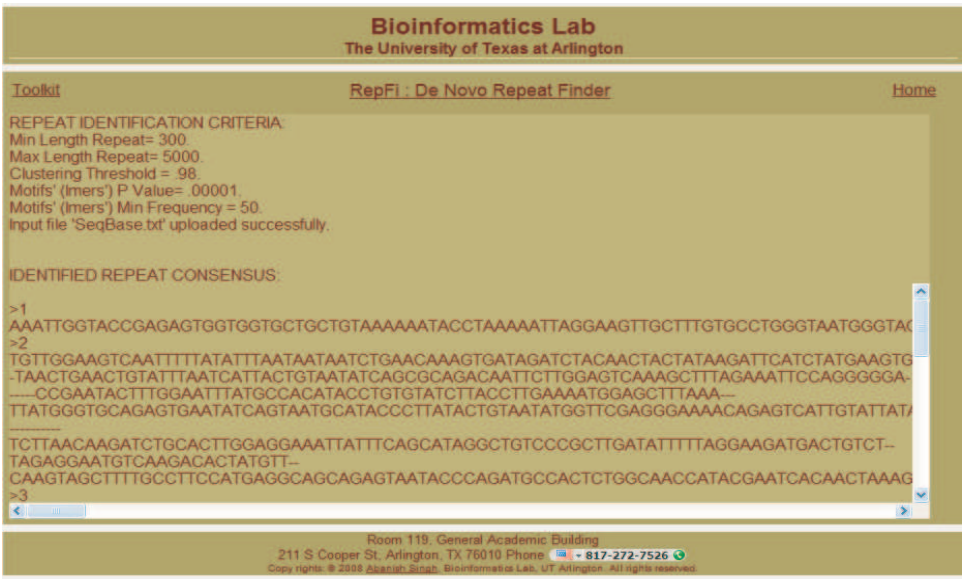

Figure A.4. The web report about repeated elements as generated by RepFi software.

A.2.2 Download

Before running the RepFi software, the one would need to download and install the clustalw it on a server and change the path of it's executable at line 204 (where the clusatalw is being invoked) in the program named Cluster2Alignment2Consensus.cpp. At this time the installation is available only for Unix/Linux systems. The Makefile for these platforms is supplied in the code directory. It is zip-ed and tar-ed, so it has to be uncompressed before running "make". Also, one need to make sure that the first line in the Perl script is replaced by the path appropriate for the server on which the software is being installed.

The program takes input sequence in Fasta format and finds the consensus sequences of repeated element in the input sequence. It accepts following command line options:

-i <sequence input file>

-t <path of program files location, example /home/dirA/dirB>

-l <minimum length of repeat (optional): default=300>

-x <maximum length of repeat (optional):default=5000>

-c <similarity threshold for clustering (optional): default=.98>

-p <maximum motif P-Value: default=.00001>

-f <minimum motif (lmers) frequency (optional): default=50>

-w <window size to find association of motifs (optional): default=1000>

The required parameters are sequence input file and the path for the directory where one has installed the software. For example, if the software has been installed in the directory /home/programs/RepFi and the RepFi.pl script is being invoked from the directory where the "input.txt" file is located, then:

$perl /home/programs/RepFi/RepFi.pl -i input.txt -t /home/programs/RepFi

would execute the software. Also, one may choose other parameters depending upon the desired selection criteria as discussed in A.2.1. As compared to the web access, the download version of RepFi software includes "window size" as an additional optional parameter. The window size represents the span over the sequence where association among the over-represented motifs is to be looked. It is analogous to the average length of repeats. In our simulations, we chose window size 1000. A higher value of window size may slow down the performance of the software.

A.3 Technology

Both MotFi and RepFi softwares are collections of several C++ programs. For download version, these programs are wrapped into Perl scripts. However, for web version, we have chosen HTML forms and PHP scripts to invoke these software's components. In order to pass the UNIX environment variables (such as unlimited stack size to Apache web users) to child process, we have used UNIX shell script to wrap system command along with invocation of component program. All component programs of the software are stored securely in our server. The web data space and

workspace for these programs are configured in a separate area of the same server. The hardware configuration of this server is: dual Xeon processors running at 3.06 Ghz, 2MB cache, 4GB memory, 365GB hard disk storage. The webserver is based on Fedora 2 distribution of Linux with Apache 2.0.49.

A.4 Discussion

We believe that our MotFi software would prove a good tool to study the motifs in regulatory sequences. In several experiments, the software has performed well by finding the motifs efficiently with good sensitivity. The web access of this software would serve as useful tool to the scientific community. The current settings of the software can handle around a dozen sequences, each of size up to 1000bp. The reason behind imposing such restrictions was to make the software run efficiently on our webserver. We are hopeful that these settings would serve the purpose of most users; however, for users with larger datasets, we would be happy to run these datasets locally on our server and provide the results.

In many respects, the RepFi software is a work in progress and it's webserver is still a prototype version. We are evaluating the feasibility of running this software online. The RepFi software presumably works on large datasets and its execution time heavily depends on the distribution of motifs in the input sequence. Though the software has delivered the expected results in most of our tests on the webserver, there could very well be some cases where the browser may timeout and suspend the execution of our software components.

REFERENCES

[1] A. Singh, U. Keswani, D. Levine, C. Feschotte and N. Stojanovic. Reconstructing the sequences of broken transposable elements. In *Proceedings of the 5th Biotechnology and Bioinformatics Symposium (BIOT 2008)*, pp. 27-33, 2008.

[2] A. Singh and N. Stojanovic. An algorithm for finding substantially broken repeated sequences in newly sequenced genomes. In *Proceedings of the International Conference on Mathematical Biology (ICMB 2007)*, American Institute of Physics Conf. Proc. 971, 79, 2008.

[3] A. Singh, C. Feschotte and N. Stojanovic. A Computational Method for DE NOVO Identification of DNA Sequence Repeats. *Cold Spring Harbor Laboratory meeting on Biology of Genomes*, 2007, NY.

[4] A. Singh, C. Feschotte and N. Stojanovic. A study of the repetitive structure and distribution of short motifs in human genomic sequences. *International Journal of Bioinformatics Research and Applications* 3, 523-535, 2007.

[5] A. Singh, C. Feschotte and N. Stojanovic. Micro-repetitive structure of genomic sequences and the identification of ancient repeat elements. In *Proceedings of The IEEE International Conference on Bioinformatics and Biomedicine (BIBM 2007)*, ed. by X. Hu et al: DOI 10.1109/BIBM.2007.56, pp. 165-171. @ 2007 IEEE.

[6] A. Singh and N. Stojanovic. Computational Analysis of the Distribution of Short Repeated Motifs in Human Genomic Sequences. In *Proceedings of the 3rd Biotechnology and Bioinformatics Symposium (BIOT 2006)*, pp. 25-34, 2006.

[7] A. Singh and N. Stojanovic. Systematic Study of Short Variable Repeated Motifs in Human Genomic Sequences. *Dallas Area Bioinformatics and Computational Biology Workshop at the University of Texas Southwestern Medical Center*, 2006, TX.

[8] A. Singh and N. Stojanovic. An Efficient Algorithm for the Identification of Repetitive Variable Motifs in the Regulatory Sequences of Co-expressed Genes. In *Proceedings of the 21st International Symposium on Computer and Information Sciences*, ed. by A. Levi et al: ISCIS 2006, Springer LNCS 4263, pp. 182-191, 2006. @ Springer-Verlag Berlin Heidelberg 2006.

[9] A. Singh and N. Stojanovic. Computational analysis of short degenerate repeated motifs in DNA sequences. *Cold Spring Harbor Laboratory meeting on Genome Informatics*, 2005, NY.

[10] A. Sharma, A. Singh and N. Stojanovic. Software for the analysis of short repeated motifs in promoter regions. *Cold Spring Harbor Laboratory genomic workshop on the Identification of Functional Elements in Mammalian Genomes*, 2004, NY.

[11] A. Aho and M. Corasick. Efficient string matching: An aid to bibliographic search. *Comm. ACM*, 18:333–340, 1975.

[12] A. Apostolico, M.E. Bock, S. Lonardi and X. Xu. Efficient detection of unusual words. *J. Comput. Biol.*, 7:71–94, 2000.

[13] A. Apostolicoa and C. Pizzia. Motif discovery by monotone scores. *Discrete Applied Mathematics* 155 695 706, 2007.

[14] A. L. Price, N. C. Jones, and P. A. Pevzner. *De novo* identification of repeat families in large genomes. *Proceedings of the* 13^{th} *International Conference on Intelligent Systems in Molecular Biology*, pp. 351–358, 2005.

[15] A. Sandelin, W. Alkema, P. Engstrom, W. Wasserman and B. Lenhard. JASPAR: an open-access database for eukaryotic transcription factor binding profiles. *Nucleic Acids Res.*, 32:D91–D94, 2004.

[16] A. Smit. Interspersed repeats and other mementos of transposable elements in mammalian genomes. *Curr. Opin. Genet. Dev.*, 9:657–663, 1999.

[17] A.B. Conley, W.J. Miller, and I.K. Jordan. Human cis natural antisense transcripts initiated by transposable elements. *Trends Genet.* 24(2):53-6, 2008.

[18] A.F.A. Smit, R. Hubley, and P. Green. RepeatMasker at http://repeatmasker.org.

[19] A.G. Jegga, S.P. Sherwood, J.W. Carman, A.T. Pinski, J.L. Phillips, J.P. Pestian and B.J. Aronow. Detection and visualization of compositionally similar *cis*–regulatory element clusters in orthologous and coordinately controlled genes. *Genome Res.*, 12:1408–1417, 2002.

[20] A.J. Gentles, M.J. Wakefield, O. Kohany *et al.* Evolutionary dynamics of transposable elements in the short-tailed opossum Monodelphis domestica. *Genome Research* 17: 992-1004, 2007.

[21] B. G. Thornburg, V. Gotea, and W. Makalowski. Transposable elements as a significant source of transcription regulating signals. *Gene* 365, 104110, 2006.

[22] B. Lewin. Gene VIII. *Pearson Prentice Hall*, ISBN 0-13-145140-5, 2004.

[23] B. McCLINTOCK. The origin and behavior of mutable loci in maize. *Proc Natl Acad Sci U S A* 36 (6): 34455. doi:10.1073/pnas.36.6.344. PMID 15430309, 1950.

[24] B. Phoophakdee and M.J. Zaki. TRELLIS+: an effective approach for indexing genome-scale sequences using suffix trees. *Pac Symp Biocomput.* 90-101, 2008.

[25] C. Feschotte, N. Jiang, and S. Wessler. Plant transposable elements: where genetics meets genomics. *Nat. Rev. Genet.*, 3:329–341, 2002.

[26] C. Feschotte. Transposable elements and the evolution of regulatory networks. *Nature Reviews Genetics*, AOP; doi:10.1038/nrg2337, 2008.

[27] C.E. Lawrence, S.F. Altschul, M.S. Boguski, J.S. Liu, A.F. Neuwald and J.C. Wootton. Detecting subtle sequence signals: a Gibbs Sampling strategy for multiple alignment. *Science*, 262:208–214, 1993.

[28] D. Che, S. Jensen, L. Cai and J.S. Liu. BEST: Binding–site Estimation Suite of Tools. *Bioinformatics*, 21:2909–2911, 2005.

[29] D. He. Using suffix tree to discover complex repetitive patterns in DNA sequences. *Conf Proc IEEE Eng Med Biol Soc.* 1:3474-7, 2006.

[30] D. Zhi, B.J. Raphael, A.L. Price, H. Tang and P.A. Pevzner. Identifying repeat domains in large genomes. *Genome Biology*, 7:R7 (doi:10.1186/gb-2006-7-1-r7), 2006.

[31] D.E. Knuth, J.H. Morris, and V.R. Pratt. Fast pattern matching in strings. *SIAM Journal of Computing*, 6: 323-350, 1977.

[32] D.L. Corcoran, E. Feingold, J. Dominick, M. Wright, J. Harnaha, M. Trucco, N. Giannoukakis and P.V. Benos. Footer: A quantitative comparative genomics method for efficient recognition of *cis*–regulatory elements. *Genome Res.*, 15:840–847, 2005.

[33] E. H. Davidson. The Regulatory Genome: Gene Regulatory Networks in Development and Evolution. *Academic, New York*, 2006.

[34] E. M. McCreight. A Space-Economical Suffix Tree Construction Algorithm. *Journal of the ACM* 23 (2): 262–272. doi:10.1145/321941.321946, 1976.

[35] E. Birney, D. Andrews, M. Caccamo *et al.* Ensembl 2006. *Nucleic Acids Res.*, 34:D561–D453, 2006.

[36] E. Burgermeister, L. Tencer and M. Liscovitch. Peroxisome proliferator–activated receptor-γ upregulates Caveolin-1 and Caveolin-2 in human carcinoma cells. *Oncogene*, 22:3888–3900, 2003.

[37] E. Davidson, D. Graham, B. Neufeld, M. Chamberlin, C. Amenson, B. Hough, and R. Britten. Arrangement and characterization of repetitive sequence elements in animal DNAs. *Cold S Harb. Symp. Quant. Biol.*, 38:295–301, 1974.

[38] E. Ohlebusch and S. Kurtz. Space efficient computation of rare maximal exact matches between multiple sequences. *J Comput Biol.* 15(4):357-77, May 2008.

[39] E. Ukkonen. On-line construction of suffix trees. *Algorithmica* 14 (3): 249–260. doi:10.1007/BF01206331, 1995.

[40] E.F. Zhang and M. Gerstein. Patterns of nucleotide substitution, insertion and deletion in the human genome inferred from pseudogenes. *Nucleic Acids Res.*, 31, 5338–5348, 2003.

[41] E.F. Adebiyi, T. Jiang and M. Kaufmann. An efficient algorithm for finding short approximate non–tandem repeats. *Bioinformatics*, 17:S5–S12, 2001.

[42] F. Gao and M.J. Zaki. Indexing protein structures using suffix trees. *Methods Mol Biol.* 413:147-69, 2008.

[43] H. Bannai, S. Inenaga, A. Shinohara, M. Takeda, S. Miyano. Efficiently finding regulatory elements using correlation with gene expression. *J. Bioinform. Comput. Biol.*, 2:273–288, 2004.

[44] H. J. Bussemaker, H. Li, and E.D. Siggia. Building a dictionary for genomes: Identification of presumptive regulatory sites by statistical analysis. *Proceeding of the National Academy of Science*, 97(18):10096-10100, August 2000.

[45] H.J. Bussemaker, H. Li, and E.D. Siggia. Regulatory elements detection using probabilistic segmentation model. In *Proceeding of the 8th International Confer-*

ence fro Intelligent System for Molecular Biology (ISMB 2000), 67-74, La Jolla, CA, August 2000.

[46] H.J. Bussemaker, H. Li, and E.D. Siggia. Regulatory elements detection using correlation with expression. *Nature Genetics*, 27:167-171, February 2001.

[47] International Human Genome Sequencing Consortium. Initial sequencing and analysis of the human genome. *Nature* 409, 860-921, 2001.

[48] J. Bentley. Programming pearls: algorithm design techniques. *Comm. ACM*, 27:865–873, 1984.

[49] J. Fickett and A. Hatzigeorgiou. Eukaryotic promoter recognition. *Genome Res.*, 7:861–878, 1997.

[50] J. Hecker, J.Y. Yang, and J. Cheng. Protein disorder prediction at multiple levels of sensitivity and specificity. *BMC Genomics*, 9(Suppl 1):S9 doi:10.1186/1471-2164-9-S1-S9, 2008.

[51] J. Jurka, V. Kapitonov, A. Pavlicek, P. Klonowski, O. Kohany, and J. Walichiewicz. Repbase Update, a database of eukaryotic repetitive elements. *Cytogenet. Genome Res.*, 110:462–467, 2005.

[52] J. Jurka. Repbase Update: a database and an electronic journal of repetitive elements. *Trends Genet.*, Vol. 9, pp. 418–420, 2000.

[53] J. Parsons. Miropeats: graphical DNA sequence comparisons. *Comput. Appl. Biosci.*, 11:615–619, 1995.

[54] J. Piriyapongsa, L. Mario-Ramrez, and IK Jordan. Origin and evolution of human microRNAs from transposable elements. *Genetics*, 176(2):1323-37, 2007.

[55] J. van Helden, B. Andre and J. Collado–Vides. Extracting regulatory sites from the upstream region of yeast genes by computational analysis of oligonucleotide frequencies. *J. Mol. Biol.*, 281:827–842, 1998.

[56] J. van Helden. Metrics for comparing regulatory sequences on the basis of pattern counts. *Bioinformatics*, 20:399-406, 2004.

[57] J.D. Hughes, P.W. Estep, S. Tavazoie and G.M. Church. Computational identification of *cis*–regulatory elements associated with groups of functionally related genes in Saccharomyces cerevisiae. *J. Mol. Biol.*, 296:1205–1214, 2000.

[58] J.L. Hess. MLL: a histone methyltransferase disrupted in leukemia. *Trends Mol. Med.* 10:500–507, 2004.

[59] J.P. Balhoff and G.A. Wray. Evolutionary analysis of the well characterized *endo16* promoter reveals substantial variation within functional sites. *PNAS*, 102:8591–8596, 2005.

[60] M. Alexandersson, S. Cawley, and L. Pachter. SLAM: Cross-Species Gene Finding and Alignment with a Generalized Pair Hidden Markov Model. *Genome Res.* 13: 496-502, 2003.

[61] M. Bilgen, M. Karaca, A. N. Onus, and A. G. Ince. A software program combining sequence motif searches with keywords for finding repeats containing DNA sequences. *Bioinformatics*, Vol. 20, pp. 3379–3386, 2004.

[62] M. Csuros. Maximum-scoring segment sets. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 1:139–150, 2004.

[63] M. Tompa, N. Li, T.L. Bailey *et al.* Assessing computational tools for the discovery of transcription factor binding sites. *Nature Biotechnology*, 23:137–144, 2005.

[64] M.A. Larkin, G. Blackshields, N.P. Brown, R. Chenna, P.A. McGettigan, H. McWilliam, F. Valentin, I.M. Wallace, A. Wilm, R. Lopez, J.D. Thompson, T.J. Gibson, and D.G. Higgins. Clustal W and Clustal X version 2.0. *Bioinformatics*, 23, 2947–2948, 2006.

[65] M.H. Schulz, S. Bauer, P.N. Robinson. The generalised k-Truncated Suffix Tree for time-and space-efficient searches in multiple DNA or protein sequences. *Int J Bioinform Res Appl.* 4(1):81-95, 2008.

[66] M.K. Das and H. Dai. A survey of DNA motif finding algorithms. *BMC Bioinformatics*, 8(Suppl 7):S21 doi:10.1186/1471-2105-8-S7-S21, 2007.

[67] M.Q. Zhang. Large-scale gene data analysis: A new challenge to computational biologist. *Genome Research*, 99(8):681-688, August 1999.

[68] M. Waring and R.J. Britten", Nucleotide sequence repetition: a rapidly reassociating fraction of mouse DNA. *Science*, 154, 791–794, 1966.

[69] Mouse Genome Sequencing Consortium. Initial sequencing and comparative analysis of the mouse genome. *Nature* 420, 520-562 doi:10.1038/nature01262, 2002.

[70] N. Li and M. Tompa. Analysis of Computational approaches for motif discovery. *Algorithms for Molecular Biology*, 1:8 doi:10.1186/1748-7188-1-8, 2006.

[71] N. Polavarapu, L. Mario-Ramrez, D. Landsman, J.F. McDonald, and I.K. Jordan. Evolutionary rates and patterns for human transcription factor binding sites derived from repetitive DNA. *BMC Genomics*, 17;9:226, 2008.

[72] N. Ranganathan, C. Feschotte, and D. Levine. Cluster and grid based classification of transposable elements in eukaryotic genomes. In *Proceedings of CCGrid06,* 6^{th} *IEEE Int. Symp. on Cluster Computing and the Grid*, page 45, 2006.

[73] N. Stojanovic (Editor). Computational Genomics: Current Methods. *Horizon Bioscience*, ISBN: 978-1-904933-30-4, 2007.

[74] N. Stojanovic. Computational methods for the analysis of differential conservation in groups of similar DNA sequences. *Genome Informatics* 15(2), 21-30.

[75] N. Stojanovic, L. Florea, C. Riemer, D. Gumucio, J. Slightom, M. Goodman, W. Miller and R. Hardison. Comparison of five methods for finding conserved sequences in multiple alignments of gene regulatory regions. *Nucleic Acids Res.*, 27:3899–3910, 1999.

[76] N. Volfovsky, B. Haas, and S. Salzberg. A clustering method for repeat analysis in DNA sequences. *Genome Biol.*, 2:RESEARCH0027, 2001.

[77] O. Johansson, W. Alkema, W. W. Wasserman, and J. Lagergren. Identification of functional clusters of transcription factor binding motifs in genome sequences: the MSCAN algorithm. *Proceedings of the* 11^{th} *International Conference on Intelligent Systems in Molecular Biology*, pp. 169–176, 2003.

[78] P. Rice, I. Longden, and A. Bleasby. EMBOSS: The European Molecular Biology Open Software Suite. *Trends Genet*,16(6), 276–277, 2000.

[79] P. Jaccard. Distribution de la flore alpine dans le bassin des drouces et dans quelques regions voisines. *Bull. Soc. Vaud. Sci. Nat.*, 37:241–272, 1901.

[80] P. Medstrand, L.N. van de Lagemaat, C.A. Dunn, J.R. Landry, D. Svenback, and D.L. Mager. Impact of transposable elements on the evolution of mammalian gene regulation. *Cytogenet. Genome Res.* 110, 342352, 2005.

[81] P. Medstrand, L.N. van de Lagemaat, C.A. Dunn, J.R. Landry, D. Svenback, and D.L. Mager. Impact of transposable elements on the evolution of mammalian gene regulation. *Cytogenet Genome Res.* 110(1-4):342-52, 2005.

[82] P. Weiner. Linear pattern matching algorithms. *Proceedings of the 14th IEEE Symposium on Switching and Automata Theory*, 1–11, 1973.

[83] P.F. Arndt. Reconstruction of ancestral nucleotide sequences and estimation of substitution frequencies in a star phylogeny. *Gene*, 390, 75–83, 2007.

[84] P.F. Arndt. The history of nucleotide substitution pattern in human. *Biophysics*, 48, Suppl. 1, 2003;

[85] R. C. Edgar and E. W. Myers. PILER: identification and classification of genomic repeats. In *Proceedings of the* 13^{th} *International Conference on Intelligent Systems in Molecular Biology*, pages i152–i158, 2005.

[86] R. Gordan and A.J. Hartemink. Using DNA duplex stability information for transcription factor binding site discovery. *Pacific Symposium on Biocomputing* 13:453-464, 2008.

[87] R.J. Britten and D.E. Kohne. Repeated sequences in DNA. Hundreds of thousands of copies of DNA sequences have been incorporated into the genomes of higher organisms. *Science*, 161, 529–540, 1968.

[88] R. Karp and M. Rabin. Efficient randomized pattern matching algorithms. *IBM J. Res. Development*, Vol. 31, pp. 249–260, 1987.

[89] R. Li, J. Ye, S. Li, J. Wang, Y. Han, C. Ye, H. Yang, J. Yu, and G. Wong. ReAS: Recovery of ancestral sequences for transposable elements from the unassembled reads of a whole genome shotgun. *PLoS Comput. Biol.*, 1:e43, 2005.

[90] R. Sharan, I. Ovcharenko, A. Ben–Hur and R.M. Karp. CREME: a framework for identifying *cis*–regulatory modules in human–mouse conserved segments. In *Proc. of the* 11^{th} *International Conf. on Intelligent Systems in Mol. Biol.*, 283–291, 2003.

[91] S. Saha and S. Bridges and Z.V. Magbanua and D.G. Peterson. Empirical comparison of ab initio repeatfinding programs. *Nucleic Acids Res.*, Electronic version ahead of print, 2008.

[92] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman. Basic local alignment search tool. *J. Mol. Biol.*, 215:403–410, 1990.

[93] S. Khambata–Ford, Y. Liu, C. Gleason, M. Dickson, R. B. Altman., S. Batzoglou, and R. Myers. Identification of promoter regions in the human genome by using

a retroviral plasmid library–based functional reporter gene assay. *Genome Res.*, Vol. 13, pp. 1765–1774, 2003.

[94] S. Tavazoie, J.D. Hughes, M.J. Campbell, R.J. Cho, and G.M. Church. Systematic determination of genetic network architecture. *Nature Genetics*, 22:281-285, July 1999.

[95] T. Hubbard, B. Aken, and K. B. *et al.* Ensembl 2007. *Nucleic Acids Res.*, 35:D610–D617, 2007.

[96] T. Smith and M. Waterman. Identification of common molecular subsequences. *J. Mol. Biol.*, 147:195–197, 1981.

[97] The ENCODE Project Consortium. Identification and analysis of functional elements in 1% of the human genome by the ENCODE pilot project. *Nature* 447, 799816, doi:10.1038/nature05874, 2007.

[98] The ENCODE Project Consortium. The ENCODE (ENCyclopedia Of DNA Elements) Project. *Science*, 306:636–640, 2004.

[99] V. Matys, O.V. Kel–Margoulis, E. Fricke *et al.* TRANSFAC®and its module TRANSCompel®: transcriptional gene regulation in eukaryotes. *Nucleic Acids Res.*, 34:D108–D110, 2006.

[100] W. L. Ruzzo and M. Tompa. A linear time algorithm for finding all maximal scoring subsequences. *7th Intl. Conf. Intelligent Systems for Molecular Biology*, 234-241, 1999.

[101] W. Miller, K. Rosenbloom, R.C. Hardison *et al.* 28-way vertebrate alignment and conservation track in the UCSC Genome Browser. *Genome Res.* 17(12):1797-808, 2007.

[102] W.W. Wasserman and J.W. Fickett. Identification of regulatory regions which confer muscle-specific gene expression. *Journal of Molecular Biology*, 278:167-181, 1998.

[103] X. Huang. An algorithm for identifying regions of a dna sequence that satisfy a content requirement. *Comput. Appl. Biosci.*, 10:219–225, 1994.

[104] Z. Bao and S. Eddy. Automated de novo identification of repeat sequence families in sequenced genomes. *Genome Res*, 12:1269–1276, 2002.

[105] Z. Jiang, H. Tang, M. Ventura, M.F. Cardone, T. Marques-Bonet, X. She, P.A. Pevzner and E.E. Eichler. Ancestral reconstruction of segmental duplications reveals punctuated cores of human genome evolution. *NATURE GENETICS ADVANCE ONLINE PUBLICATION*, doi:10.1038/ng.2007.9., 2007.

[106] Z. Jiang, R. Hubley, A. Smit, E.E. Eichler. DupMasker: a tool for annotating primate segmental duplications. *Genome Res.*, 18(8):1362-8, 2008.

BIOGRAPHICAL STATEMENT

Abanish Singh is an alumnus of the Motilal Nehru National Institute of Technology – Allahabad, India. He completed his Master of Engineering research from Liverpool J M University, Liverpool, UK. Before starting his PhD in Bioinformatics in 2004 at the University of Texas at Arlington, U.S.A., he extensively worked in the area of Computer Networking, and had more than a decade of experience in teaching, research, and project management. His current primary research area is bioinformatics and computational biology, specifically in the genomic sequence analysis and motif discovery .