ISLAMIC UNIVERSITY OF TECHNOLOGY

A MODIFIED ALGORITHM FOR MOTIF DISCOVERY BASED ON RISOTTO AND PROJECTION

Authors:
Marnim Galib (104421)
Nahid Hasan (104434)

Supervisor:
Prof. Dr. M. A. Mottalib
Head of the Department
Department of Computer Science and Engineering
Islamic University of Technology

Co-Supervisor:
Mohammad Arifur Rahman
Lecturer
Department of Computer Science and Engineering
Islamic University of Technology

A thesis submitted in partial fulfillment of the requirements
for the degree of Bachelor of Science in Computer Science and Engineering

Academic Year: 2013-2014

Department of Computer Science and Engineering
Islamic University of Technology.
A Subsidiary Organ of the Organization of Islamic Cooperation.
Dhaka, Bangladesh.

Declaration of Authorship

This is to certify that the work presented in this thesis is the outcome of the analysis and investigation carried out by Marnim Galib and Nahid Hasan under the supervision of Prof. Dr. M. A. Mottalib and Md. Arifur Rahman in the Department of Computer Science and Engineering (CSE), IUT, Dhaka, Bangladesh. It is also declared that neither of this thesis or any part of this thesis has been submitted anywhere else for any degree or diploma. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of reference is given.

Authors:

Marnim Galib (104421)

Nahid Hasan (104434)

Supervisor:

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

Co-Supervisor:

Md. Arifur Rahman
Lecturer
Department of Computer Science and Engineering
Islamic University of Technology (IUT)

Abstract

An important part of gene regulation is mediated by specific proteins, called transcription factors, which influence the transcription of a particular gene by binding to specific sites on DNA sequences, called transcription factor binding sites (TFBS) or, simply, motifs. Such binding sites are relatively short segments of DNA, normally 5 to 25 nucleotides long, overrepresented in a set of co-regulated DNA sequences. There are two different problems in this setup: motif representation, accounting for the model that describes the TFBS's; and motif discovery, focusing in unraveling TFBS's from a set of co-regulated DNA sequences. This thesis proposes a discriminative scoring criterion that culminates in a discriminative mixture of Bayesian networks to distinguish TFBS's from the background DNA. This new probabilistic model supports further evidence in non-additivity among binding site positions, providing a superior discriminative power in TFBS's detection. On the other hand, extra knowledge carefully selected from the literature was incorporated in TFBS discovery in order to capture a variety of characteristics of the TFBS's patterns. This extra knowledge was combined during the process of motif discovery leading to results that are considerably more accurate than those achieved by methods that rely in the DNA sequence alone.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1Overview

Bioinformatics is the application of computer technology to the management of biological information. Computers are used to gather, store, analyze and integrate biological and genetic information which can then be applied to gene-based drug discovery and development. The need for Bioinformatics capabilities has been precipitated by the explosion of publicly available genomic information resulting from the Human Genome Project. There are different fields in bioinformatics like DNA sequencing, DNA Motif finding, Gene selection, Gene prediction, Gene expression, Protein selection etc. Among them we are interested in motif finding or motif extraction problems.

All life on this planet depends on three types of molecule: DNA, RNA, and proteins. DNA, or deoxyribonucleic acid, is a simple molecule consisting four types of bases: adenine (A), thymine (T), guanine (G) and cytosine(C). The nucleotide order of a DNA fragment is known as gene sequence.

Patterns appearing repeated either inside a same string or over a set of strings are important objects to identify. Such repeated patterns are called motifs and their identification is called motif inference or motif extraction. Finding motif in gene sequence is a very important topic in bioinformatics. There are two types of motif, single motifs and structured motifs. In our work, we are only considering the single motifs.

1.2 Problem Statement

There are many motif finding algorithms out there. They are different in strategies, input, output types etc. The main problem of large amount of data is the inclusion of noisy and irrelevant data in the information set. As the datasets become large the number of noisy, redundant and uninformative gene also increases resulting in space-time complexity. So, we are set to find an algorithm that will find out the possible motif with few mismatches with great accuracy. We work on an algorithm called "RISOTTO" which has a great performance in case of accuracy but has poor space-time complexity. So, we combine the "RANDOM PROJECTION" algorithm with RISOTTO to improve its complexity.

1.3 Research Challenges

With the large amount of biological data generated due to DNA sequencing of various organisms, it is becoming necessary to identify techniques that can help in finding useful information amongst all the data. Finding motifs involves determining meaningful short sequences that may be repeated over many sequences in various species. Various approaches for the motif discovery problem have been proposed in the literature. Computational methods for identifying and modeling DNA sequence motifs and regulatory elements have been developed over the past 25 years. These methods either use DNA patterns or position weight matrices to model target DNA-binding sites. Some of these methods have been used successfully to discover the cis-regulatory elements in genes that are thought to be regulated by a common transcription mechanism. Orthogonal information from comparative genomics or co-regulation at the expression level have been incorporated into these methods to identify cis-regulatory sites. Because many of the regulatory elements are functional in vivo only in certain temporal or spatial contexts and require other nearby sites that together function as cis-regulatory modules, methods have also been developed to address composite regulatory elements or regulatory modules that consist of DNA sites bound by multiple regulatory factors. There are now many programs available for DNA motif discovery and putative regulatory element identification, and several have web interfaces for easy use. Bioinformatics is such a research area that is the interface between the biological and computational sciences. Identification of all of the functional elements in genomes, including genes and regulatory elements, is a fundamental challenge now that the complete genomic sequences of a number of prokaryotes

and eukaryotes are available. The ultimate goal of bioinformatics is to uncover the wealth of biological information hidden in the mass of data and obtain a clearer insight into the fundamental biology of organisms. This new knowledge could have profound impacts on fields as varied as human health, agriculture, the environment, energy and biotechnology. The identification of DNA motifs remains an active challenge for the researchers in the bioinformatics domain. In recent years a considerable number of algorithms have been designed for identifying regulatory elements in DNA sequence.

1.4 Motivation

In this large-scale genome sequencing era the main bottleneck to progress in molecular biology is data analysis. The prime objective of this thesis is the investigation of one kind of biological information contained in sequenced data: the motif model and its discovery from a set of co-regulated DNA sequences. A motif is roughly a mathematical representation underlying a transcription factor binding site. More precisely, the main goal of this thesis is the proposal of efficient and effective algorithms for motif representation and discovery, capable of dealing with the enormous amount of data coming from the Bioinformatics community. Such models and algorithms should be able to look in and beyond the DNA sequence alone, preferably gathering information from different sources. This ground in the belief that such diverse information would increase the discriminative power of current tools. To its possible extent, the improvements obtained should be documented by complexity analysis, as well as by experimental results over biologically relevant sequence-sets.

Chapter 2

Literature Reviews

This chapter is divided into different sections. In each section, we will discuss some proposed method for DNA motif finding. Along with their description, we will try to highlight the shortcomings of each of the discussed methods.

2.1 Motif:

To classify and identify the features of DNA, RNA, Protein sequences in molecular biology, motif plays an important role. It has the capabilities to describe and find out the special characteristics of DNA, RNA, Protein sequences. Sequence motifs are becoming important in the analysis of gene regulation day by day. Actually Motifs are short repeating pattern in DNA, RNA, Protein sequences in molecular biology that is conserved during the process of evolution. Motifs are short, recurring patterns in DNA that are presumed to have a biological function. Often they indicate sequence-specific binding sites for proteins such as nucleuses and transcription factors (TF). Others are involved in important processes at the RNA level, including ribosome binding, mRNA processing (splicing, editing) and transcription termination. Basically it has a great impact on biological science. It resides in a DNA, RNA or Protein sequences and plays an important role in molecular biology.

Some common features of motif has been invented to identify it in the long sequences of DNA, RNA or Protein. Some important features that are helpful to identify motif or know about motif are given below:

- Motifs are patterns of length 5 to 20 bases and are repeated over many sequences.
- They are small, have constant size, and are repeated very often.
- They are statistically over-represented in regulatory regions.

Here we have given an example of motif in DNA sequences to understand the characteristic of motif. From the example we can have a brief idea about the short recurring sequence Motif. In the following example, we can see a conserved short sequence (Motif) in few DNA Sequences. From the example we can see that the pattern **ATGCAACT** is unchanged and conserved. This short DNA sequence ATGCAACT is called motif.

CGGGGCT**ATGCAACT**GGGTCGTCACATTC

TTTGAGGGTGCCCAATAA**ATGCAACT**CCA

GG**ATGCAACT**GATGCCGTTTGACGACCTA

AAGG**ATGCAACT**CCAGGAGCGCCTTTGCT

TACATGATCTTTTG**ATGCAACT**TGGATGA

Figure 1: Typical representation for a motif

2.2 Motif Representation

Deterministic models:

There are two main kinds of deterministic models: regular expressions and consensus sequences. The regular expressions used in motif discovery denote a subset of regular languages and are typically composed of exact symbols, ambiguous symbols, fixed gaps or flexible gaps. A consensus sequence represents a collection (neighborhood) of binding site sequences that are at most at a certain Hamming distance of the underlying consensus sequence. Each binding site sequence in this collection is called a motif occurrence or consensus occurrence. The number of mismatches depends largely

on the size of the motif. There are a few variants to these two models. A first alternative imposes a restriction on the location of mismatches along the consensus sequence (Pavesi et al., 2001). That is, a consensus occurrence can present at most a certain number of mismatches in the first i nucleotides, and so on. On the other hand, a second variant takes into account the sum of mismatches between all consensus occurrences and the underlying consensus sequence (Li and Fu, 2005).

Probabilistic models:

The main drawback of deterministic models is that they lose some information, as compared with the collection of binding site sequences from where they are generated. For instance, even if we know that a consensus sequence has at least 2 mismatches within the collection of binding sites, we do not know if there are one or more bases which are specially well conserved, nor, for those bases that are not so well conserved, what kind of mismatches they have. The probabilistic models appeared to overcome such loss of information.

Motif discovery:

Identifying TFBS's is notoriously difficult for both prokaryotic and eukaryotic organisms. There are two major limitations in this task. First, there is a constraint of algorithmic nature, meaning that in general the proposed methods can only be applied to sets of sequences restricted in their length and number. Second, there is a weakness in the models employed for TFBS's, leading to poor TFBS predicting methods. Nevertheless, the subject has gained a renewed interest in the last few years, with the sequencing of the genomes of vertebrates such as man and mouse. The literature on the topic of DNA binding site sequence detection is extensive, and there are several surveys on the subject (Sandve and Drablos, 2006; Tompa et al., 2005; Pavesi et al., 2004a; Stormo, 2000;Vanet et al., 1999; Brazma et al., 1998a). Herein, we concentrate on briefly surveying methods that try to extract conserved single binding sites, or multiple ones, possibly located at constrained distances from one another in a set of co-regulated DNA sequences. Up to

ten years ago, all methods for detecting DNA binding sites considered each such site individually. These methods therefore looked for single motifs, that is, motifs composed of a unique binding site. This includes pattern-based approaches which allowed for wildcards or a limited number of spacers but not for mutations (Brazma et al., 1998b; Tompa, 1999; van Helden et al., 1998). Apart from these, only an approach by Sagot (1998) based on a suffix tree allowed for mutations. Another very popular single motif discoverer is the MEME algorithm (Bailey and Elkan, 1994, 1995a,b), an Expectation Maximization (EM) procedure that identifies motifs with high relative entropy. MEME [4] deals with single and multiple motifs by identifying significant sets of compatible motifs. It works by iteratively building such multiple motifs from single ones ensuring that the occurrence positions of the multiple motifs do not contradict the occurrence positions of the single motifs previously identified. In this, the set of motifs reported must only satisfy compatibility. No constraint, and therefore no statistical value, is put on the distances separating them.

2.3 Related Work:

2.3.1 PMS (Planted Motif Search) [17]:

Inputs are *t* sequences of length *n* each. Inputs also are two integers' *l* and *d*. The problem is to find a motif (i.e., a sequence) *M* of length *l*. It is given that each input sequence contains a variant of *M*. The variants of interest are sequences that are at a hamming distance of *d* from *M*.

Numerous papers have been written in the past on the topic of motif search (PMS).Examples include Bailey and Elkan, Lawrence et al., Rocke and Tompa. These algorithms employ local search techniques such as Gibbs sampling, expectation optimization etc. These algorithms may not output the planted motif always. We refer to such algorithms as approximation algorithms. Algorithms that always output the correct answer are referred to as exact algorithms. More algorithms have been proposed for PMS by the following authors: Pevzner and Sze, Buhler and Tompa. The algorithm of Pevzner

and Sze is based on finding cliques in a graph and the algorithm of Buhler and Tompa employs random projections. These algorithms have been experimentally demonstrated to perform well. These are approximation algorithms as well.

Algorithms for PMS can be categorized into two depending on the basic approach employed, namely, profile-based algorithms and pattern-based algorithms (Price et al.) Profile based algorithms predict the starting positions of the occurrences of the motif in each sequence. On the other hand, pattern-based algorithms predict the motif (as a sequence of residues) itself. Several pattern based algorithms are known. Examples include PROJECTION[5], MULTIPROFILER[8], MITRA[6], and PatternBranching[7]. PatternBranching (due to Price, Ramabhadran and Pevzner) starts with random seed strings and performs local searches starting from these seeds. Examples of profile-based algorithms include CONSENSUS[10], GibbsDNA[9], MEME[4], and ProfileBranching [7]. The performance of profile-based algorithms is specified with a measure called "performance coefficient". The performance coefficient gives an indication of how many positions (for the motif occurrences) have been predicted correctly.

Algorithm:

The algorithm has the following steps: 1) Let the input sequences be $S_1, S_2 \ldots S_t$. The length of each sequence is n. Form all possible l-mers from out of these sequences. The total number of l-mers is $\leq tn$. Call this collection of l-mers C. Let, the collection of l-mers in S_1 be C'; 2) Let u be an l-mer in C'. For all $u \in C'$ generate all the patterns v such that u and v are at a hamming distance of d. The number of such patterns for a given u is $\binom{l}{d}(|\sum| - 1)^d$. Thus the total number of patterns generated is $O(n\binom{l}{d}|\sum|^d)$. Call this collection of l-mers C''. Note that C'' contains M, the desired output pattern (assuming that M does not occur in any of the input sequences); 3) For every pair of l –mers (v,u) with $u \in C'$ and $v \in C''$ compute the hamming distance between u and v. Output that l-mer of C'' that has a neighbor in each one of the n input sequences. The run time of this algorithm is $O(tn^2 l\binom{l}{d}|\sum|^d)$. If M occurs in one of the input sequences, then this algorithm will run in time $O(t^2 n^2 l)$.

2.3.2 RANDOM PROJECTION [5]:

The RANDOMPROJECTIONS algorithm is another randomized approach to motif finding. If an l-long pattern is "implanted" in DNA sequences without mutations, then motif finding simply reduces to counting the number of occurrences of l-mers from the sample (the most common one reveals the implanted pattern). However, motif finding becomes very difficult when the pattern is implanted with mutations. Could we possibly reveal mutated patterns in the same way that we deal with non-mutated patterns, that is, by considering the positions that were not mutated? For example, in any instance of a pattern of length 8 with two mutated positions, six positions remain unaffected by mutations and it is tempting to use these six positions as a basis for motif finding. There are two complications with this approach. First, the six constant positions do not necessarily form a contiguous string: the mutated nucleotides may be in positions 3 and 7, leaving the other six conserved positions to form a gapped pattern. Second, different instances of the pattern can mutate in different positions. For example, three different instances of the mutated pattern may differ in positions 3 and 7, 3 and 6, and 2 and 6 respectively. The key observation is that although the three mutated patterns **X***X*, **X**X**, *X***X** are all different, their "consensus" gapped pattern *XX**XX* remains unaffected by mutations. If we knew which gapped pattern was unaffected by mutations we could use it for motif search as if it were an implanted gap pattern without mutations. The problem, however is that the locations of the four unaffected positions in *XX**XX* are unknown. To bypass this problem, RANDOMPROJECTION [5] tries different randomly selected sets of k (out of l) positions to reveal the original implanted pattern. These sets of positions are called projections. We will define a (k, l)-template to be any set of k distinct integers $1 \leq t_1 < \cdots < t_k \leq l$. For a (k, l)-template $t = (t_1, \ldots, t_k)$ and an l-mer $a = a_1, \ldots, a_l$, define Projection(a,t) $= a_{t1}, a_{t2} \ldots a_{tk}$ to be the concatenation of nucleotides from a as defined by the template t. For example, if a=ATGCATT and t = (2, 5, 7), then Projection(a,t)=TAT. The RANDOMPROJECTIONS [5] algorithm, below, chooses a random (k,l)-template and projects every l-mer in the sample onto it; the

resulting k-mers are recorded via a hash table. We expect that k-mers that correspond to projections of the implanted pattern appear more frequently than other k-mers. Therefore, k-mers that appear many times (e.g., whose count in the hash table is higher than a predefined threshold θ) as projections of l-mers from the sample are likely to represent projections of the implanted pattern. Of course, this is subject to noise and a single (k, l)-template does not necessarily reveal the implanted pattern. The RANDOMPROJECTIONS [5] algorithm repeatedly selects a given number, m, of random (k,l)-template and aggregates the data obtained for all m iterations. As RANDOMPROJECTIONS [5] chooses different random templates, the locations of the implanted pattern become clearer. As compared to other motif finding algorithms, RANDOMPROJECTIONS requires additional parameters: k (the number of positions in the template), θ (the threshold that determines which bins in the hash table to consider after projecting all l-mers), and m (the number of chosen random templates). The RANDOMPROJECTIONS algorithm creates a table, Bins, of size 4^k such that every possible projection (k-mer) corresponds to a unique address in this table. For a given (k,l)-template r, Bins(x) holds the count of l-mers a in DNA such that Projection(a, r) = x.

```
RANDOMPROJECTIONS(DNA, t, n, l, k, θ, m)
 1  create a t × n array motifs and fill it with zeros
 2  for m iterations
 3      create a table Bins of size 4^k and fill it with zeros
 4      r ← a random (k, l)-template.
 5      for i ← 1 to t
 6          for j ← 1 to n − l + 1
 7              a ← jth l-mer in ith DNA sequence
 8              Bins(Projection(a, r)) = Bins(Projection(a, r)) + 1
 9      for i ← 1 to t
10          for j ← 1 to n − l + 1
11              a ← jth l-mer in ith DNA sequence
12              if Bins(Projection(a, r)) > θ
13                  motifs_{i,j} ← motifs_{i,j} + 1
14  for i ← 1 to t
15      s_i ← Index of the largest element in row i of motifs.
16  return s
```

RANDOMPROJECTIONS provides no guarantee of returning the correct pattern, but one can prove that RANDOMPROJECTIONS [5] returns the correct motif with high probability, assuming that the parameters are chosen in a sensible way. The main difference between this toy algorithm and the practical Projection algorithm developed by Jeremy Buhler and Martin Tompa is the way in which this algorithm evaluates the results from hashing all the l-mer projections. The method that we have presented here to choose (s_1, s_2, . . . , s_t) is crude, while the Projection algorithm uses a heuristic method that is harder to trick by accidentally large counts in the array motifs.

2.3.3 RISO:

RISO discovers motifs composed of many binding sites separated by spacers. Each binding site is called a *box*.

Generic parameters:

- Alphabet file: the file name with the alphabet of the motifs (examples are the DNA or protein alphabet).
- FASTA file: the file name of the input sequences (sequences must be in FASTA format).
- Output file: the name of the output file.
- Quorum: the minimum percentage of input sequences where the motif must appear to be extracted.
- Boxes: number of boxes of the motif.

For each box of the motif it is also needed:

- Min length: minimum length of the corresponding box.
- Max length: maximum length of the corresponding box.
- Substitutions: maximum number of substitutions allowed in corresponding box.
- Min spacer length: minimum distance that separates the corresponding box to the next one (if exists).
- Max spacer length: maximum distance that separates the corresponding box to the next.

An example of the input parameters follows:

```
Alphabet file        ../params/alphabet
FASTA file           ../params/dnc_subtilis_330-30.seq
Output file          ../params/b-subtilis-output
Quorum               12
Boxes                2

BOX 1
Min length           6
Max length           6
Substitutions        1
Min spacer length    16
Max spacer length    18
BOX 2
Min length           6
Max length           6
Substitutions        1
```

Table 1: Input parameters of RISO

Interpret the output

For each motif that **RISO-Dynamic** discovers in the input dataset, it prints the following:

- An header with a summary of the input parameters
- Each of the subsequent lines prints the following information:
    - The motif.
    - A numeric representation of the motif.
    - The number of different sequences where the motif appears.
    - The number of occurrences of the motif (allowing repeats and overlaps in the same sequence).
- The number of total motifs founded and the time spent.

```
%%% 2 128/1062 196736 12 12 2 6 6 1 16 18 6 6 1 alphabet ACGT$
============================================================
AAAAAA_AAAAAA 000000-000000 188 703
AAAAAA_AAAAAC 000000-000001 139 357
AAAAAA_AAAAAG 000000-000002 166 410
AAAAAA_AAAAAT 000000-000003 193 461
.....
TTTTTT_TTTTTA 333333-333330 201 507
TTTTTT_TTTTTC 333333-333331 171 441
TTTTTT_TTTTTG 333333-333332 148 384
TTTTTT_TTTTTT 333333-333333 188 700
Nb models: 6419
User time: 45.33 sec.
```

Table 2: Output of RISO

2.3.4 RISOTTO-Single motif extraction [2]:

A single motif is a word over an alphabet. Given an error rate e, a motif is said to e-occur in a sequence if it occurs with at most e letters substitution. The single motif extraction problem takes as input N sequences, a quorum q, d a maximal number e of mismatches allowed, and a minimal and maximal length for the motifs, k_{min} and k_{max}, respectively. The problem consists in determining all motifs that e-occur in at least q input sequences. Such motifs are called valid. An efficient exact algorithm for the extraction of single motifs with mismatches has been introduced in [13] and is based on a suffix tree. In a few words, motifs are considered in lexicographical order starting from the empty word, and they are extended to the right as long as the quorum is satisfied until either a valid motif of maximal length is found (if the k_{max} length is reached), or the quorum is no longer satisfied. In both cases, a new motif is attempted. At each step, all nodes spelling e-occurrences of the current motif are taken in to account. More formally, the algorithm presented in [13] we refer to is sketched in Algorithm 1, where motif m is the one whose extension is being tried.

Algorithm 1 Single motif extraction

ExtractSingleMotif(motif m)

1. **for all** $\alpha \in \Sigma$ **do**
2. **if** $m\alpha$ is valid **then**
3. **if** $|m\alpha| \geq k_{min}$ **then** spell out the valid motif $m\alpha$
4. **if** $|m\alpha| < k_{max}$ **then ExtractSingleMotif**($m\alpha$)

Figure 2 . 1: Single motif extraction

At the beginning ExtractSingleMotif is called on the empty word. The algorithm recursively calls itself for longer motifs built by adding letters (step 4), and considers new ones (step 1) when the extension fails (step 2). A valid motif is spelled out whenever a motif whose length lies within the required minimal and maximal length is being

considered (step 3). The order in which motifs are generated corresponds to a depth-first visit of a complete trie M of all words of length k_{max} on the alphabet. We refer to M as the motif tree. In fact, the algorithm does not need to allocate the motif tree. The only memory requirement is for the suffix tree T. Assuming that the required length of the motif is k (that is $k_{min} = k_{max} = k$), and that at most ***e*** mismatches are allowed, the algorithm has worst case time complexity.

***RISOTTO- Using maximal extensibility of factors*:**

The modification consists in storing information concerning maximal extensibility in order to avoid trying to extend hopeless motifs. Assume that in our virtual depth-first visit of the motif tree, we have found out that motif m can be further extended without losing the quorum up to a length of *MaxExt(m)* only , the latter representing its maximal extensibility . If later on, we are processing a motif m' that has m as a suffix, then the MaxExt(m) information could be useful, as it applies to m' as well because m' can also be extended with at most MaxExt(m) symbols (and possibly less). In particular, we have that if |m'| + MaxExt(m) < k_{min}, then we can avoid any further attempt to extend m' because there is no hope to reach length k_{min} for motifs that have m' as prefix.

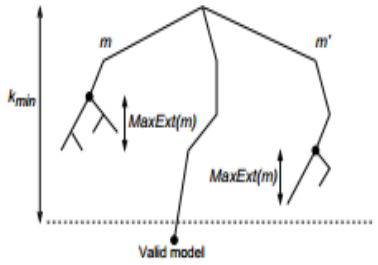

Figure 2 . 2: Example where the extension of m' can be avoided, using MaxExt(m), where m is a suffix of m', because |m'|+MaxExt(m) < k_{min}

Motifs are considered in lexicographical order by a depth-first (virtual) visit of the motif tree M. Every time we stop extending a motif, that is, when we (virtually) backtrack in M, it is either because we found a valid motif of the maximal length, or

because the quorum is no longer satisfied (***m*** does not satisfy the condition at step 2, and we start to consider the next one in lexicographical order). More formally, the analysis the motif m is abandoned both when m is valid and |m| = k_{max}, or m does not satisfy the quorum. In the first case, m is valid, as are all its prefixes, and |m| = k_{max}. No information on the maximal extension of m nor of its prefixes can be of any use because all motifs having a prefix of m as suffix can in general still be extended as much as necessary to reach at least the length k_{min}. For this reason, we set MaxExt(m) = $+\infty$, meaning that m can be extended possibly more than we are computing. In the second case, m does not satisfy the quorum while all its prefixes do. For reasons that will be clearer later, we chose to only use the maximal extensibility information of motifs of length up to k_{min} -1, hence this case can be subdivided in to two subcases. When a motif m cannot be extended anymore and it has not reached the length k_{min}-1, we set MaxExt(m) = 0. If the motif has reached a length h between k_{min}-1 and k_{max}, we set MaxExt(mα|k_{min} -1)=h- *(*k_{min} -1)where (mα|k_{min} -1 is the prefix of length k_{min}*-1 of* mα. Since it can be that MaxExt(mα| k_{min} -1) had already received some value because a previous extension of (mα| k_{min} -1) was interrupted, then we change the value of MaxExt(mα| k_{min} -1) only if we are increasing it, as maximal extensibility of a motif refers to its longest extension. We assume that all maximal extensibility values are initially set to -1, hence the first attribution to MaxExt(mα| k_{min} -1) will always increase its value. In all the cases above, the algorithm does not consider any further extension of m, and backtracks. This backtracking consists in either replacing the last letter of m, or considering a shorter motif which in general shares a prefix with m, if σ|m| was the last letter of the alphabet ∑. In this latter case, the whole sub tree rooted at the node spelling σ1….σ|m|-1, has been (virtually) completely visited. Thus, we have all the information necessary to set the value of MaxExt(σ1….σ|m|-1) according to MaxExt(x)=1 + max 2

MaxExt(x), for all valid motifs x such that |x| < k_{min} -1. If the letter ϭ|m|-1 was the last of the alphabet, then the backtracking goes further. In that case, also the MaxExt information concerning the word ϭ1….ϭ|m|-2 can be lled in in the same way, and so on as long as we (virtually) climb up in the tree. As mentioned before, maximal extensibility information can be used for motifs whose extension is being considered and for which this information could actually prevent some useless attempts. Namely, assume we are trying to extend the motif m=ϭ1….ϭ|m|. Since the motifs are considered by means of a depth-first search on the virtual motif tree, we obviously do not know the value of MaxExt(m) yet. Moreover, we know MaxExt(ϭ2….ϭ|m|). Only if it lexicographically precedes m, that is, it has already been virtually visited in the motif tree. If this is not the case, we check whether MaxExt(ϭ3….ϭ|m|). is already known, and so on, possibly until the singleton ϭ|m|. If they are all lexicographically greater than m, then no maximal extension information can be used for m, but if for any of them MaxExt is known and it holds that the maximal possible extension is not enough to reach k_{min}, then the information is useful as it guarantees that attempting to further extend m is useless.

Algorithm 2 Single motif extraction with maximal extensibility information

ExtractSingleMotif(motif m)

1. **for all** $\alpha \in \Sigma$ **do**
2. $x := m\alpha$
3. **repeat** $x := |x\rangle_{|x|-1}$ **until** ($x = root$ **or** $MaxExt(x) \neq -1$)
4. **if** $x \neq root$ **and** $MaxExt(x) + |m\alpha| < k_{min}$ **then**
5. $MaxExt(m\alpha) := MaxExt(x)$
6. stop spelling $m\alpha$ and **continue**
7. **if** $m\alpha$ is valid **then**
8. **if** $|m\alpha| \geq k_{min}$ **then** spell out the valid motif
9. **if** $|m\alpha| < k_{max}$ **then ExtractSingleMotif**($m\alpha$)
10. **else** $MaxExt(\langle m\alpha|_{k_{min}-1}) := +\infty$
11. **else**
12. **if** $|m\alpha| < k_{min}$ **then** $MaxExt(m\alpha) := 0$
13. **else if** $MaxExt(\langle m\alpha|_{k_{min}-1}) < |m\alpha| - (k_{min} - 1)$ **then** $MaxExt(\langle m\alpha|_{k_{min}-1}) := |m\alpha| - (k_{min} - 1)$
14. **if** $|m| < (k_{min} - 1)$ **then** $MaxExt(m) := 1 + \max_{\alpha \in \Sigma} MaxExt(m\alpha)$

Figure 2 . 3: Single motif extraction with maximal extensibility function

2.3.5 GRISOTTO [3]:

GRISOTTO is a greedy approach to improve combinatorial algorithms, RISOTTO [2] for motif discovery with prior knowledge. Position-specific priors (PSP) have been used with success to boost EM and Gibbs sampler-based motif discovery algorithms. PSP information has been computed from different sources, including orthologous conservation, DNA duplex stability, and nucleosome positioning. Here RISOTTO is extended by post-processing its output with a greedy procedure that uses prior information. PSP's from different sources are combined into a scoring criterion that guides the greedy search procedure.

Methods

GRISOTTO [3] uses the RISOTTO [2] output as starting points of a greedy procedure that aims at maximizing a scoring criterion based on combined prior information. Our approach diverges from EM (used in MEME [4]) and Gibbs sampling (used in PRIORITY [11]) as we do not consider latent variables and do not use a background model. Moreover, instead of maximizing the likelihood, GRISOTTO uses a scoring criterion based on the balanced information of observing the DNA sequences and the priors given a candidate motif. We called this score Balanced Information Score (BIS). Furthermore, GRISOTTO [3] returns the IUPAC string that is best fitted, according to BIS, via a greedy search procedure.

GRISOTTO Algorithm:

Some notations used in GRISOTTO algorithm are given in the table below:

Symbol	Meaning
Σ	alphabet (usually DNA or IUPAC)
f	input sequences
f_i	i-th input sequence
f_{ij}	j-th position of the i-th input sequence
N	number of input sequences
n_i	length of f_i
k	motif size
S_p	p-th prior (in PSP format)
ℓ	number of priors (it can be zero)
S	$S = \langle S_1, ..., S_\ell \rangle$ is the list of all priors
z_{min}	minimum number of motifs expected to be returned by a RISOTTO run
z_{max}	maximum number of motifs expected to be returned by a RISOTTO run
z	number of top motifs post-processed from RISOTTO output
$\mathcal{C}$	the set with the z top motifs to be post-processed by GRISOTTO
m	motif of size k
$m\langle i, \alpha \rangle$	motif m where the i-th position (starting with 0) is replaced by $\alpha \in \Sigma$
ε	empty motif (its BIS score is the minimum possible value)
$f_i[j \dots j + k - 1]$	k-long segment of the i-th input sequence that starts at position j
$S_p[i, j]$	prior probability at the j-th position of f_i
j_i	annotated position for f_i with maximum BIS score for a motif m
P_m	probability distribution given by the PSSM induced by m
α_p	the weight of the p-th prior
λ	coefficient to balance priors and over-representation contribution

Table 3: Definition of terms used in describing the algorithms presented in Methods

Considering that we have a set of N co-regulated DNA sequences hence forward denoted by f =(f_i) i =1,...,N. The length of the each sequence f_i is n_i, that is, f =(f_{ij}) i =1,..., n_i. Moreover, consider that Sp contains some prior information in a PSP format about the domain in study, with p= 1 ...ℓ, where ℓ is the number of priors (eventually zero). We denote by S=⟨S_1,...,S_ℓ⟩the list of all priors. The goal of GRISOTTO [3] is to report a single motif of a fixed size k, that is, an IUPAC string of size k. The IUPAC alphabet is henceforward denoted by Σ. The pseudo code of GRISOTTO is depicted in Algorithm 1. The algorithm starts by running RISOTTO [2] to extract, at least z_{min}, nd at most z_{max}, motifs of size k. From the RISOTTO output, the top Z motifs are collected in a set called **C** (Step 2) and constitute the starting points of the GRISOTTO greedy procedure, called GGP (Step 4). Briefly, GGP starts with a motif $m \in C$ and returns the best fitted motif, according to BIS, by updating each position in m with an IUPAC symbol until no local improvements can be achieved. In Step 5-6 the variable r, which stores the output of the algorithm, is updated whenever the GGP procedure returns a motif with a BIS score higher than the current stored one. Note that in Step 2 the result variable r is initialized

with the empty motif ε. We consider that the empty motif ε has the minimum possible BIS scoring value.

It remains to explain the GGP procedure given in Algorithm 2. The general idea of the algorithm is to process each position of the motif m, received as parameter, in a greedy fashion. Variable i identifies the motif position being processed. It is initialized with the value 0 (Step 1), the first position of m, and it is incremented in a circular way using modular arithmetic (Step 9). GPP terminates when k consecutive positions of the motif m being considered cannot be improved, according to BIS, and so m remains unchanged for a complete k-round. This information is stored in variable t that counts how many consecutive positions of m have not been modified. Variable t is initialized with 0 (Step 1) and controls the outer cycle (Step 2-9), which terminates when t =k. The Boolean flag changed is read in the outer cycle (Step 7) to detect whether the i-th position of the motif has been modified inside the body of the inner cycle (Step 6). It is initialized in each run of the outer cycle with false (Step 3). The inner cycle (Step 4-6) tries to improve the BIS score of m by updating its i-th position with each letter a ∈ Σ. We denote by m ⟨i,α⟩ the motif m where the i-th position of m was replaced by the letter a. Whenever the BIS score of m ⟨i, α⟩ is greater than the BIS score of m(Step 5) three variables are updated:

(i) Motif m is updated to m⟨i, α⟩;

(ii) Variable t is reset to its initial value, forcing a complete k-round from that point on;

(iii) Flag *changed* is turned to *true*.

After the inner cycle, in Step 7, we test whether the i-th position of m was not modified by checking the value of the flag *changed*. If that is the case, variable t is incremented (Step 8). Next, in Step 9, variable i is incremented so that the next position of m can be inspected.

Balanced information score:

Start by noticing that a motif m of size k written in IUPAC can be easily translated into a PSSM with dimension 4 ×k. Moreover, observe that if we had to guess in which position m occurs in sequence fi that would be the position ji that maximizes $P_m(f_i$ $[j_i .. j_i$+ k- 1]) where P_m (w) is the probability of observing the DNA word w by the PSSM induced by m and (f_i $[j_i .. j_i$+ k- 1]) is the k-long segment off i that starts at position j_i. In other words, such j_i annotates the position in which we believe the motif m occurs in fi. Henceforward consider that we annotate for each sequence fi the respective position j_i where m occurs with higher probability (refer to Table 1). Following Shannon, these lf-information of a probabilistic event with probability p is given by - log(p). If the event is very rare, the self-information is very high. On the other hand, if the event has probability close to 1, observing such event gives us almost no information. So, by assuming that m occurs independently in each sequence off, the self-information that m occurs in all sequences off in the annotated positions is given by

$$\sum_{i=1}^{N} = -\log(P_m(f_i\ [j_i .. j_i + \text{ k- 1}])) \qquad (1)$$

Note that the above sum is zero (its minimal value) if the motif m occurs with probability 1 in all annotated positions and, moreover, the sum is not upper-bounded. Considering that the priors are in PSP format, their information can be easily computed from the annotated sequences. Indeed, the self-information given by the prior S_p of observing the annotated position S_{ji}, for all 1≤i ≤N, is computed as

$$\sum_{i=1}^{N} = -\log\ (S_p[\text{i},j_i]) \qquad (2)$$

Where Sp[i, j] is the prior probability stored at the j-th position of the i-th sequence in the S_p PSP file. Having this, it remains to understand how the information from different priors can be combined. Actually, priors come from different sources [4], and some of these sources might have more quality or be more relevant for motif discovery than others. A simple way to heuristically combine prior information is to multiply the contribution of each prior by a constant a p that measures the belief in the

quality/relevance of each prior S_p and consider a balanced sum of all self- information's. In order to keep the resulting value with the same magnitude of each component, we consider a convex combination, that is, $\sum_{p=1}^{l} \alpha p = 1$ .Thus, the combined self-information is computed as

$$\sum_{p=1}^{N} \quad (\alpha p \sum_{i=1}^{N} = -\log(S_p[i, j_i])) \qquad (3)$$

Following a similar idea, we balance with a constant $\lambda \in (0, 1]$ the self-information given by the occurrence of the motif in (1) with the self-information given by the priors in (2), obtaining in this way the following expression.

$$\lambda \sum_{i=1}^{N} -\log(P_m(f_i[j_i \ldots j_i + k - 1])) + (1 - \lambda) \sum_{p=1}^{\ell} \left(\alpha_p \sum_{i=1}^{N} -\log(S_p[i, j_i]) \right) =$$
$$-\sum_{i=1}^{N} \left(\lambda \log(P_m(f_i[j_i \ldots j_i + k - 1]) + (1 - \lambda) \sum_{p=1}^{\ell} \alpha_p \log(S_p[i, j_i])) \right).$$

The closer the above expression is to zero the less (balanced) self-information follows from observing a candidate motif the annotated positions of both the DNA sequences and the priors. Indeed, we expect motifs to occur in the annotated positions of both the DNA sequences and the priors with high probability. Therefore, the goal is to find a motif m that minimizes such information. Next, and for the sake of simplification, we drop the minus sign in (3), that is, we consider the final scoring criterion, called balanced information score (BIS), defined as

$$\text{BIS}(m, f, S) = \sum_{i=1}^{N} \left(\lambda \log(P_m(f_i[j_i \ldots j_i + k - 1]) + (1 - \lambda) \sum_{p=1}^{\ell} \alpha_p \log(S_p[i, j_i]) \right), \qquad (4)$$

And restate our goal to finding a motif m that maximizes (4). Note that BIS (m, f, S) is always non-positive and, therefore, is upper-bounded by 0. For the BIS score in Equation (4) to be well-defined it remains to determine the values of the constant λ and

α_p for all $1 \leq p \leq \ell$. Whenever there is no knowledge about the quality of the priors the values of such constants should be uniform, that is, $\lambda=1/2$ and $\alpha_p = 1/\ell$, for all $1 \leq p \leq \ell$. Usually, it is possible to refine heuristically these constants by evaluating the usefulness of each prior in well-known domains. Finally, it is not obvious how to translate back the combined information into a combined prior that could be used in an EM or Gibbs sampler-based algorithm.

These techniques need that such prior reflects the probability of finding a motif in a certain position of the DNA sequences in order to correctly bias, in each iteration step, the expected log-likelihood of the candidate motif occurring in the positions given by the latent variable. On the other hand, GRISOTTO [3] incorporates prior information in BIS resulting in a theoretical-information scoring criterion that measures the information of observing the candidate motif in the annotated positions of both the DNA sequences and the priors. These annotated positions are computed only once, for each candidate motif, in such a way that the balanced contribution to the BIS score of the DNA sequences and the priors in those positions is maximal. The higher the value of the BIS score, the higher the probability that a candidate motif occurs in the annotated positions of both the DNA sequences and the priors. Therefore, GRISOTTO reports the motif, among all candidate ones, which maximizes the BIS scoring criterion.

GRISOTTO can use different types of prior information. Some of them are given here.

Evolutionary conservation-based priors:

Diverse methods for motif discovery make use of orthologous conservation to assess whether a particular DNA site is conserved across related organisms, and thus more likely to be functional. A comprehensive work along this line was done by PRIORITY researchers [14], where an orthologous conservation-based prior was devised to improve their Gibbs sampler-based motif discovery method. This prior was built in a discriminative way by taking into account not only sequence-sets that were bounded by some profiled TF (the positive set) but also sequence-sets that were not bounded by the

same TF (the negative set). In this way the prior reflects not only the probability that a W-mer at a certain position is conserved but of all the conserved occurrences of this W-mer what fraction occurs in the bound sequence-set. Conserved occurrences are found by searching if a W-mer in a reference sequence also occurs in most of its orthologous ones regardless of its orientation or specific position. For this particular case, the evolutionary conservation-based prior was used for each inter genetic region in S. Cerevisiae and it used the orthologous sequences from six related organisms, namely,S. paradoxus, S. mikatae, S. kudriavzevii, S. bayanus, S. castelliandS. kluyveri. Thepriorwas named discriminative conservation-based prior(DC) and was made available, in a PSP format, at PRIORITY webpage. Herein, we gauge the performance of GRISOTTO [3] when this exact DCprior is incorporated into the BIS scoring criterion. Results comparing GRISOTTO-DC [3] with PRIORITY-DC [16], MEME-DC [17], and other state-of-the-art algorithms, can be found in Table 2. Results show that GRISOTTO-DC correctly predicted 83 motifs out of the 156 experiments, whereas PRIORITY-DC found 77 and MEME:ZOOP-DC81 [4]. We conclude that GRISOTTO [3] performed at least as well as PRIORITY and MEME: ZOOP when the same DCPSP was used. A closer inspection of detailed results of GRISOTTO, in Additional file2 reveals that GRISOTTO-DC found 15 motifs that PRIORITY-DC did not, while PRIORITY-DC found only 10 motifs that GRISOTTODC did not.

Combining priors:

Despite considerable effort to date in developing new potential priors to boost motif discoverers, PSP's from different sources have not yet been combined. Actually, although having some degree of redundancy, because, for instance, the positioning of nucleosomes may be correlated with DNA double helix stability, it is easy to conclude by a closer inspection of the detailed results in. As a matter of fact, PRIORITY researchers have already noticed this fact [15]. However, it is not a trivial task determining how to translate the IS combined information into a PSP that can be used in EM or Gibbs sampler-based algorithms. In order to gauge the potential of combined priors, we

incorporated in the BIS score the three DC, DE and DN priors. We call the final prior combined discriminative prior (CDP). Results show that GRISOTTO-CD [3] Pis the more accurate motif discoverer for the 156 sequence-sets being evaluated. It correctly predicted 93 motifs, while GRISOTTO-DC [3] found 83, GRISOTTO-DE 80 and GRISOTTO-DN77. In this way GRISOTTO CD [3] Pre accomplished an improvement of at least 12% over correct predictions, when compared with GRISOTTO variants considering the priors individually. This raises the overall proportion of successful predictions in 7%, on top of the improvements already attained in the previous sections, over these 156 yeast sequence-sets. Moreover, when comparing GRISOTTO-CDP [3] with state-of-the-art motif discoverers the final proportion of successful predictions was raised to 60%, while the best known previous value, to our knowledge, was 51% attained by MEME-DC [4] . This leads us to conclude that combining priors from different sources is even more beneficial than considering them separately. ChiP-seq data Herein we measure the accuracy of GRISOTTO in TF motif discovery on 13 mouse ChiP-seq data. This data was gathered by Chenet al. where whole-genome binding sites of 13 sequence-specific TFs (Nanog, Oct4, STAT3, Smad1, Sox2, Zfx, c-Myc, n-Myc, Klf4, Essrb, Tcfcp2l, E2f1, and CTCF) were profiled in mouse ES cells using the ChiP-seq approach. Sequences of ±100 bp size from the top 500 binding peaks were selected for each factor, repeats were masked, and the Weeder tool was used to find overrepresented sequences unravelling 12 of the 13 factors We assess the quality of GRISOTTO [3] in discovering motifs from mouse ChiP-seq data with two priors. First, an orthologous conservation-based PSP was used as information for higher organisms is now available. Indeed, there are already such PSP’s for yeast, fly, mouse and even human Second, a binding peak-based PSP was tried as ChiP-seq assays provide an intrinsic positional prior that can be computed from base-specific coverage profiles. This prior has recently been employed in motif discoverers [with success. As for ChiP-chip data, we let GRISOTTO [3] find for a single motif of size 8, since priors were computed for 8-mers. However, as human-curated motifs are not available for this ChiP-seq data, we made only a

resemblance, based on a 6-window match, between the motifs reported by GRISOTTO with those outputted by Chen et al. and MEME for the same data.

Binding peak-based priors:

Huel al. [23] devised prior using coverage profile information provided by the ChiP-seq approach. This ground in the belief that motifs are tightly packed near the peak summit- the location inside each peak with the highest sequence coverage depth. As a result, prior probabilities were set to be proportional to a discretized Student's t-distribution with 3 degrees of freedom and rescaled such that they form a step function with a fixed 25 bp step-size. The prior probabilities are symmetric and centered at the peak summits. As such prior is intrinsically a positional one we built a PSP resuming the described probabilities for the 13 mouse ChiP-seq data and ran GRISOTTO [3]. Our results show that direct use of binding peak-based priors does not help GRISOTTO much. Actually, the motifs reported by this prior were exactly the same a using the uniform prior (recall that for the uniform prior any position in the DNA is likely to contain a motif). Moreover, when combined with the DC prior GRISOTTO [3] reported precisely the same motifs as DC prior alone. These findings suggest that GRISOTTO is unable to retrieve any useful information from the binding peak-based prior. We attributed this to the fact that part of the information contained in the binding peak based prior is already encoded in the BIS score. Indeed, peak summits indicate an overrepresentation of a motif in a certain locus. Such overrepresentation is already weighted in the BIS score. Notwithstanding, it seems reasonable that for short sequences of 200 bp (namely, ±100 bp around the peak summits) the coverage-based prior has no real impact on motif discovery. For longer sequences, the effective resolution of the peak summits seems to provide useful information.

Chapter 3

Proposed Method

3.1 Overall Concept

In our proposed process of finding DNA motif, we merged the Random Projection [5] and RISOTTO [2] algorithm. Thus, we improved the performance of RISOTTO [2] by removing the concept of using suffix tree and injected the hashing or bucketing concept of Random Projection. As, we know that GRISOTTO [3] use the output data of RISOTTO as its input data, now we will feed the output data of our process to GRISOTTO as input. From those input data GRISOTTO will calculate the BIS score as discussed before [section 2.3.5] and find out our desired output motif. We basically kept the GRISOTTO intact and worked on RISOTTO algorithm.

3.2 Proposed Method and Algorithms

Step 1:

Our work starts with finding the most probable starting of our motif. Therefore, we generate all possible 3-mers and create a bucket for each of this 3-mers.Now we will further extend this 3-mer.At each iteration we start with a 3-mer and extend it one nucleotide at a time satisfying the quorum and having a minimum length of k_{min} to k_{max} When we have found that we have a motif that has this length and satisfies a minimum quorum percentage then we believe we have that l-mer which is the most probable motif. Our search for a 3 mer is followed by our proposed algorithm PROJECTED RISOTTO-

PROJECTED_RISOTTO (m, t, n, k_{min}, k_{max}, e)

1. For i ← 1 to t //filling the bucket of 3-mers
2. For j ← 1 to n − l + 1
3. a ← j^{th} l-mer in the DNA sequence
4. store the starting position of j in the bucket list
5. (m_1,m_2 … m_k) = k-most occurring l-mers out of the 4^l buckets.
6. For i ← 1 to k
7. For each symbol α in Σ do
8. x :=m_iα
9. if m_iα is valid then
10. if | m_iα | ≥ k_{min} then spell out the valid model
11. if |m_iα|< k_{max} or |m_iα|< k_{min} then PROJECTED_RISOTTO (m_iα,e)
12. else
13. if e(m_iα) > e then PROJECTED_RISOTTO (m_iα,e)
14. else discard that | m_iα |

Output from this step will be sent as input to the next step of the process.

Step 2:

We then feed the motifs returned by the RISOTTO [2] procedure to the GRISOTTO [3] as input and modify the motif using prior information.

GRISOTTO (DNA sequences f, list of priors S = ⟨S1 ,..., Sℓ ⟩)

a. run RISOTTO(k, z_{min} , z_{max});
b. let r = ε and C be the list of the first z motifs returned in Step 1;
c. for each motif m in C
d. let m = GGP(m, f, S);
e. if (BIS(r, f ,S) < BIS(m, f ,S))
f. let r = m;
g. return r;

GGP (motif m, DNA sequences f, list of priors S =⟨S_1 , ..., S_ℓ ⟩)

1. Let t = 0 and i = 0;
2. While (t <k)
3. Let changed = false;
4. For each a in Σ
5. If (BIS (m⟨i, a⟩, f, S)>BIS(m, f ,S))
6. Let m = m⟨i, a⟩, t = 0 and changed =true;
7. If (not changed)
8. Let t = t + 1;
9. Let i = (i + 1) mod k;
10. Return m;

Thus, this GRISOTTO [3] procedure will modify our motif based on our algorithm and produce the best motif.

3.3 Experimental Data and Result

For testing our algorithm first we implemented it. We implemented it on JAVA platform. After implementation we tested it through some data set. The data set was in FASTA format. There were 20 sequences of DNA and each of them was 600 nucleotides long. Here is the sample of the input sequence:

```
>Sequence1
ACCCTCTTGTATAGCAGCAGATGACCGGGTGTTGCCACTCATAGCCTTCCGATGGAGAGAAGCGCGGGCCACTA
GAAGATAATGTCGGGCCCTTGAGCGCGCCAAGCCCCAGGCATTTGTAGGCAGGTTTCCTCTCCCGCAGGGGCAA
TGTGTACATTCGGTAGAACATAACGCTGGAATTACATTCGCCGCATTACTAGTAAACCGTCCTTTGTAAGGAAGC
CGCCAGGAGTGCGTTAATGGATAGGGTCCGAACGGTCTCAACTAAGTCCACCTTGCGCAGCCAACGCCACAACT
GCCACAGCTTTATCCCGCCTCAGCAGTGGCATGTCTCCAAACCACGGGCAAGCCTGCGATATCAGGCCGCGGAG
TCGTGCCGGAGGATCGTCGCCGTAACGACTGTTCTATACCTACCCTAGGGAATACGGGTCTAATCGAGTATCAGG
GTGGCTAGATAATAGGCGTATTGACGGCTCGCTCATAGGTACCTCAAGAGGTTTTCAGAATATGCACGGCTCAGT
```

Figure 3 . 1: Example of an input sequence

As we divided our algorithm in two steps, this input data set was fed to the first part of the algorithm which is known as the PROJECTED RISOTTO.

First it finds out the occurrences and position of occurrences of all possible 3-mers. As there are total 64 possible 3-mers, it will results for all 64 3-mers. The outputs of this part of the process are shown in figure 3.2:

```
AAA:(1,201) (1,335) (2,30) (2,31) (2,32) (2,82) (2,179) (2,184) (2,304) (2,419) (3,216) (3
ACA:(1,153) (1,165) (1,181) (1,291) (1,300) (1,522) (1,524) (1,532) (1,534) (1,558) (2,26)
AGA:(1,18) (1,55) (1,57) (1,73) (1,76) (1,162) (1,452) (1,492) (1,502) (2,28) (2,34) (2,18
ATA:(1,10) (1,40) (1,78) (1,167) (1,243) (1,355) (1,406) (1,422) (1,454) (1,457) (1,480) (
AAC:(1,164) (1,169) (1,202) (1,253) (1,262) (1,285) (1,293) (1,336) (1,394) (1,531) (2,20)
ACC:(1,0) (1,23) (1,203) (1,272) (1,337) (1,408) (1,412) (1,486) (1,560) (1,576) (2,21) (2
AGC:(1,12) (1,15) (1,42) (1,60) (1,95) (1,104) (1,220) (1,281) (1,302) (1,318) (1,347) (1,
ATC:(1,308) (1,357) (1,383) (1,433) (1,440) (1,569) (2,91) (2,141) (2,175) (2,318) (2,346)
AAG:(1,59) (1,75) (1,103) (1,215) (1,219) (1,266) (1,346) (1,491) (1,540) (1,545) (2,33) (
ACG:(1,170) (1,254) (1,286) (1,340) (1,395) (1,424) (1,469) (1,511) (2,36) (2,113) (2,145)
AGG:(1,110) (1,120) (1,124) (1,140) (1,216) (1,227) (1,245) (1,360) (1,380) (1,417) (1,443
ATG:(1,20) (1,51) (1,81) (1,147) (1,239) (1,327) (1,507) (1,586) (2,45) (2,125) (2,189) (2
AAT:(1,80) (1,146) (1,177) (1,238) (1,421) (1,432) (1,456) (1,504) (1,568) (2,90) (2,140)
ACT:(1,36) (1,70) (1,195) (1,263) (1,294) (1,398) (1,526) (2,54) (2,84) (2,122) (2,162) (2
AGT:(1,198) (1,230) (1,267) (1,321) (1,369) (1,437) (1,518) (1,541) (2,0) (2,8) (2,48) (2,
ATT:(1,114) (1,155) (1,178) (1,183) (1,192) (1,465) (1,579) (2,169) (2,224) (2,251) (2,298
CAA:(1,102) (1,145) (1,261) (1,284) (1,292) (1,334) (1,345) (1,490) (1,544) (2,76) (2,139)
```

Figure 3 . 1: Occurrences of 3-mers

After finding the occurrences of all possible 3-mers we will count the occurrence of all 3-mars on the next stage for extending the 3-mer to find out our desired motif. The counts of the all possible 3-mers are shown in figure 3.3:

```
AAA 165
ACA 195
AGA 187
ATA 180
AAC 197
ACC 200
AGC 193
ATC 179
AAG 199
ACG 170
AGG 206
ATG 191
AAT 180
ACT 201
AGT 173
ATT 178
CAA 184
CCA 188
CGA 197
CTA 190
CAC 182
```

Figure 3 . 2: Count of all possible 3-mers

Now that we have information about the occurrence of all the 3-mers we will extend the 3-mers starting from the most occurring 3-mers. If we want some faster solution with partial information, we may use a limited list of all the 3-mers based on the highest occurrence criteria. The sorted list of the 3-mers are given here:

```
1  GAA
2  GTC
3  AGG
4  CAG
5  CTG
6  ACT
7  GGA
8  TGA
9  TGG
10 ACC
11 TAC
12 AAG
13 CGC
14 GCA
15 AAC
16 CGA
17 ACA
18 AGC
19 GTG
20 GCC
21 GCG
```

Figure 3 . 4: Sorted list of all possible 3-mers

Now we will extend those selected 3-mers by one nucleotide at time. For extending the 3-mers we find out it's occurrences in all the input sequences and count the next position after that 3-mer at each matched location. We find out the position of a 3-mer by using FINDPOSITION here (given in Appendix) and extend the 3-mer one base at a time. So, at each step we find the corresponding positions of a sequence and it is extended in the next step by counting the highest occurring base and adding it to the current sequence. The positions of a sample 3-mer "GAA" is given in the next page:

```
59 75 164 177 219 253 421 504 531
20 30 70 82 148 161 184 188 231 317 490
27 124 130 236 336 369 444 453 540 555 576
83 208 260 309 331 348 387 438 510 558 571 593
22 59 264 268 274 279 383 416 467 506 514 546 575 590
82 124 194 211 216 290 294 345 378 383 393 524 538 575 583
25 36 47 195 228 261 272 310 334 357 371 390 397 501
5 73 87 112 151 253 262 359
30 96 111 122 227 411 522 537
36 68 93 104 158 170 244 273 290 373 400 445 450 453 506
142 197 228 239 243 308 422 459 469 501
95 171 192 304 326 363 374 387 428 480 501
36 117 134 139 206 311 498 502 525
38 44 154 223 257 272 318 361 401 417 435 441 479 483 501 540
13 113 148 360 558
132 243 332 395 524 597
67 95 135 211 286 300 330 414 443 459 479
33 216 229 300 391 420 453 487
171 225 288 297 301 340 361 402 417 430 519 582
4 46 196 232 258 293 334 391 407 425 447 470 512 550 598
```

Figure 3 . 5: Positions of "GAA" in all the sequences

While extending the 3-mers we will consider checking the minimum and maximum length of the motifs which can be given manually. We also consider the quorum percentage which means minimal occurrence of each motif. If the motif length is less than the given maximum motif length (Kmax) and also fulfill the quorum (q) then we will extend the motif. If a motif fulfill the quorum and its length is in between Kmax and Kmin then we will consider it as an output motif. In this algorithm we also considered the number of mismatch. While extending a motif if the motif doesn't fill the quorum for the extended nucleotide then we will consider it as a mismatch and extend the motif from the next nucleotide. Output of this process has been shown in figure 3.6.

```
1  GAACAGAATGCCAAA
2  GTCCCATTCGACACGA
3  AGGAACGCGCTACTGA
4  CAGTCTTACTCC
5  CTGCCTAATGAC
6  ACTGTTCAACGT
7  GGAAGCTCATTCTCGT
8  TGAGACGTAATAACCG
9  TGGAAGCAGCCCGCC
10 ACCTCAGAAGTTGAGA
```

Figure 3 . 6 *:Output motifs of "PROJECTED_RISOTTO"*

Now these output motifs will be given to GRISOTTO [3] process to check with the prior information which we call as the Position Specific Prior. GRISOTTO will check each position of a particular motif with it's prior information in order to find out a motif that's better than the current one. So, GRISOTTO can be used further to resolve the motifs we found by RISOTTO. From this process we will find the final desired motif. The working procedure of GRISOTTO is discussed in section 2.3.5.

3.4 Output Run-time comparison

We compared our output and runtime of our algorithm with several other renowned motifs finding algorithm. From the result of the comparison we found that our output result is very impressive and accurate.

While comparing the running time challenging instances of motif problem we considered the following parameters (DNA, $|\Sigma| = 4$, N = 20 sequences of length n = 600). Problem instances are denoted by (k, m, $|\Sigma|$), where k is the length of the motif implanted with m mismatches.

The run-time comparisons of our proposed method are compared here with some existing well known algorithms here:

Algorithm	**(13, 4, 4)**	**(15,5,4)**
qPMSPrune	45s	10.2 m
PMS5	117s	4.8 m
Voting	104s	21.6 m
RISOTTO	772s	106 m
Proposed Algorithm	**57s**	113 m

Table 4 : Run-*time comparison*

These results shown above proved that our modified algorithm finds out motifs faster than some well-recognized algorithms while it takes the mismatches and quorum percentages into count. Hence, we can claim that our proposed method was implemented successfully.

Chapter 4

Conclusion

4.1 Summary*:*

Through the duration of this thesis work we have been able to propose an improvement of RISOTTO [2] algorithm for single motif extraction including mismatch. We tried improving RISOTTO using random projection method, though we exclude the random part of random projection. Output will be post-processed in a greedy fashion by GRISOTTO [3] algorithm. This will give us better time and space complexity than the original one. Besides, our proposed algorithm will be faster and more efficient for motif finding with longer length and more mismatches. Run-time and space complexity will be more favorable. We also implemented the algorithm in our way and compared the output with the output of other motif searching algorithms. For comparing the algorithms we used same set of data set as input and found that our results are very impressive compared to few other algorithms.

4.2 Future work:

While implementing and simulating our proposed method we faced some problem which has some effect on space and the runtime, we will try to eliminate those efficiently. We still could not process the output of our proposed method to the GRISOTTO [3] algorithm. We will try to implement it in future. We are also thinking some other approach in future. Here, we are just working with the single motif extraction problem GDP can be further improved by using the probability calculation. We will try to extend our algorithm for structured motif extraction in future. Gibbs sampling [9] method can be used as the substitution of random projection. Modification of extensibility function to

improve the result can also be done. Suffix tree which is used by RISOTTO [2] can be substituted by suffix array. Suffix array[18] is a quite new algorithm which can be implemented in RISOTTO [2] instead of suffix tree which was used previously.

Appendix

JAVA Simulation Codes of Proposed Method:

The JAVA simulation codes of our proposed algorithm are given below:

File 1: MOTIF_FINDING.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package pkgfinal;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.DataInputStream;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.io.Writer;

/**
 *
 * @author Marnim
 */
public class MOTIF_FINDING {

    /**
     * @param args the command line arguments
     * @throws java.io.FileNotFoundException
     */
    public static String newline = System.getProperty("line.separator");

    public static void main(String[] args) throws FileNotFoundException, IOException {
```

```
32
33          String strLine;
34          String[] DNA;
35          DNA = new String[21];
36
37          while ((strLine = br.readLine()) != null) {
38
39              for (int j = 1; j < DNA.length; j++) {
40                  DNA[j] = br.readLine();
41              }
42
43          }
44
45          //********* Reading the DNA sequences in DNA Array **********
            String s = null;
47          String m_motif = null;
            String motif = null;
            String last_correct = null;
50
51
52          String[] pos_motifs;
53          pos_motifs = new String[64];
54          //######### Taking the highest buckets in an array ##########
55          String strln;
56          String[] bucketarray;
57          bucketarray = new String[64];
58          FileInputStream istrm = new FileInputStream("sorted_list.txt");
59          BufferedReader brd = new BufferedReader(new InputStreamReader(istrm));
60
61          while ((strln = brd.readLine()) != null) {
62
63              for (int j = 0; j < bucketarray.length; j++) {
64                  bucketarray[j] = brd.readLine();
65                  pos_motifs[j] = bucketarray[j];
66              }
67
68          }
69          istrm.close();
70          PrintWriter output = new PrintWriter(new BufferedWriter(new FileWriter("Possible_Motifs_2.txt")));
71          int mismatch = 0;
72
73          //******* Traverse through the highest buckets
74          for (int i = 0; i < 64; i++) {
75              s = bucketarray[i];
76              motif = bucketarray[i];
77              last_correct = bucketarray[i];
78
79              //**** Loop for extending the motif length
80              extension(s, motif, last_correct, DNA, mismatch, m_motif, output);
81              System.out.println("=======================");
82              mismatch = 0;
83
84          }
85          output.close();
86          in.close();
87
88      }
89
```

```
    public static void extension(String s, String motif, String last_correct, String[] DNA, int mismatch,
91  String m_motif, PrintWriter output) throws IOException {
92      String temp = null;
93      for (int cur_len = 3; cur_len <= 15 && mismatch < 5; cur_len++) {
94
95          System.out.println("--------------------");
96          System.out.println("Current Length:" + cur_len);
97          System.out.println("Current Motif:" + motif);
            String line = "";
99          int max = -1;
100         int maxA = 0, maxC = 0, maxG = 0, maxT = 0;
101         int high_mismatch=4;
102         Find_positions fp = new Find_positions();
103         //System.out.println("Generating positions for string: " + last_correct);
104         fp.genText(last_correct,high_mismatch);
105
106         for (int x = 0; x < 4; x++) {
107             //System.out.println("Reading positions for string: "+last_correct);
108             FileInputStream input = new FileInputStream("Positions.txt");
109             BufferedReader buffer = new BufferedReader(new InputStreamReader(input));
110             int countA = 0, countC = 0, countG = 0, countT = 0;
                int len = 0;
                int pos = 0;
113             char character = 0;
114             int dna_line = 1;
115
116             if (x == 0) {
117                 character = 'A';
118             } else if (x == 1) {
119                 character = 'C';
120             } else if (x == 2) {
121                 character = 'G';
122             } else if (x == 3) {
123                 character = 'T';
124             }
125                 //****** Loop through DNA sequences by selecting A,C,G,T and
126             //****** Check if they are eligible for extending further
127
128             while ((line = buffer.readLine()) != null) {
129                 if (line.trim().length() == 0) {
130                     continue;
131                 }
132                 String[] positions = line.split(" ");
133                 //****** Traversing the positions and increment Count
                    for (int y = 0; y < positions.length; y++) {
135                     //-------------System.out.print(positions[y]);
136                     Integer d = Integer.parseInt(positions[y]);
137                     //-------------System.out.print(d + " ");
138                     len = DNA[dna_line].length();
139                     pos = d + cur_len + mismatch - 1;
140
```

```
                        if (pos < len && DNA[dna_line].charAt(pos) == character) {
                            if (character == 'A') {
                                countA = countA + 1;
                                break;
                            } else if (character == 'C') {
                                countC = countC + 1;
                                break;
                            } else if (character == 'G') {
                                countG = countG + 1;
                                break;
                            } else if (character == 'T') {
                                countT = countT + 1;
                                break;
                            }
                        }
                    }
                        //****** Output Count at each line
                    //System.out.println("A:" + countA + " " + "C:" + countC + " " + "G:" +
                    //countG + " " + "T:" + countT + " ");
                    dna_line++;
                }
                    //****** Final Count After Each Iteration
                //System.out.println("Final Result:" + "A:" + countA + " " + "C:" + countC + " " + "G:" +
                //countG + " " + "T:" + countT + " ");

                    s = motif + character;
                    s = s.substring(0, s.length() - 1);
                } else if (countC >= 12) {
                    s = motif + character;
                    s = s.substring(0, s.length() - 1);
                } else if (countG >= 12) {
                    s = motif + character;
                    s = s.substring(0, s.length() - 1);
                } else if (countT >= 12) {
                    s = motif + character;
                    s = s.substring(0, s.length() - 1);
                }

                //**** Selecting the most frequent outcome among the next base:A,T,G,C
                if (countA > max) {
                    max = countA;
                    m_motif = motif + character;
                } else if (countC > max) {
                    max = countC;
                    m_motif = motif + character;
                } else if (countG > max) {
                    max = countG;
                    m_motif = motif + character;
                } else if (countT > max) {
                    max = countT;
                    m_motif = motif + character;
                }
```

```
                if (x == 0) {
                    maxA = countA;
                } else if (x == 1) {
                    maxC = countC;
                } else if (x == 2) {
                    maxG = countG;
                } else if (x == 3) {
                    maxT = countT;
                }
            }
            if (maxA < 12 && maxC < 12 && maxG < 12 && maxT < 12) {
                mismatch = mismatch + 1;
                s = m_motif;
                motif = s;
                System.out.println("FINAL MOTIF: " + motif);
                s = s.substring(0, s.length() - 1);
                System.out.println("REDUCED STRING TO BE READ: " + last_correct + "   Mismatched positions:" + mismatch);
            } else if (maxA > 16 || maxC > 16 || maxG > 16 || maxT > 16) {
                s = m_motif;
                motif = s;
                last_correct = motif;
            } else {
                s = m_motif;
                motif = s;
            }
        }
        output.println(motif);
    }
```

File 2: FINDPOSITIONS.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

package pkgfinal;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.DataInputStream;
import java.io.FileInputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.Writer;

/**
 *
 * @author Marnim
 */
public class Find_positions {

    public static String newline = System.getProperty("line.separator");


    void genText(String str){
        try {
            FileInputStream in = new FileInputStream("DNA_input.txt");
            FileInputStream input = new FileInputStream("Possible_Motifs.txt");
            BufferedReader buffer = new BufferedReader(new InputStreamReader(input));
            try (DataInputStream in1 = new DataInputStream(in)) {
```

```
32              try (DataInputStream in1 = new DataInputStream(in)) {
33                  String s = str;
                    Writer out = new BufferedWriter(new FileWriter("Positions.txt"));
35                  String strLine;
36                  String word;
                    int linenumber = 1;
38                  int k;
39                  //Read File Line By Line
40                  for (k = 0; k < 1; k++) {
41                      word = s;
42                      linenumber = 1;
                        BufferedReader br = new BufferedReader(new InputStreamReader(in1));
44                      in.getChannel().position(0);
45                      br = new BufferedReader(new InputStreamReader(in));
46                      while ((strLine = br.readLine()) != null) {
47
48                          for (int i = -1; (i = strLine.indexOf(word, i + 1)) != -1;) {
49                              i=i+1;
50                              out.append(i + " ");
51                          }
52                          out.append(newline);
53                          linenumber++;
54                      }
55                  }
56                  in.close();
57                  out.close();
58              }
59          } catch (IOException e) {//Catch exception if any
60              System.err.println("Error: " + e.getMessage());
61          }
62      }
63  }
```

Bibliography

[1] Carvalho AM, Freitas AT, Oliveira AL, Sagot MF: An Efficient Algorithm forthe Identification of Structured Motifs in DNA Promoter Sequences. IEEE/ACM Trans. Comput Biol Bioinformatics2006,3(2):126-140.

[2] Pisanti N, Carvalho AM, Marsan L, Sagot MF: RISOTTO: Fast extraction of motifs with mismatches. In Proc. LATIN'06, Volume 3887 of LNCS. Edited by:JR Correa AH, Kiwi M. Spriger-Verlag; 2006:757-768.

[3] Alexandra M. Carvalho and Arlindo L. Oliveira, GRISOTTO: A greedy approach to improve combinatorial approach to improve combinatorial algorithms for motif discovery with prior knowledge Algorithms for Molecular Biology, 6:13, Apr 2011.

[4] Bailey TL, Bodén M, Whitington T, Machanick P:The value of position specific priors in motif discovery using MEME.BMC Bioinformatics2010,11:179.

[5] J. Buhler and M. Tompa. Finding motifs using random projections. Proc. Fifth Annual International Conference on Computational Molecular Biology (RECOMB), April 2001.

[6] E. Eskin and P.R. Pevzner. Finding composite regulatory patterns in DNA sequences. Bioinformatics S1, 2002, pp. 354-363.

[7] U. Keich and P. Pevzner. Finding motifs in the twilight zone. Bioinformatics 18, 2002, pp. 1374-1381.

[8] T. L. Bailey and C. Elkan. Unsupervised learning of multiple motifs in biopolymers using expectation maximization. Machine Learning 21(1-2), 1995, pp. 51-80.

[9] 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, 1993, pp. 208-214

[10] G. Hertz and G. Stormo. Identifying DNA and protein patterns with statistically significant alignments of multiple sequences. Bioinformatics 15, 1999, pp. 563-577.

[11] A.V. Aho and M.J. Corasick. Efficient string matching: an aid to bibliographic search.Communication of ACM, 18:333–340, 1975.

[12] B. Alberts, D. Bray, J. Lewis, M. Raff, K. Roberts, and J. W atson. Molecular Biology of the Cell. Garland Publishing, New York, 1994.

[13] M.-F. Sagot. Spelling Approximate repeated or common motifs using a suffix tree. In C.L. Lucchesi and A.V. Moura, editors, Proc. Latin '98, volume 1380, pages 111{127, 1998. LNCS.

[14] Gordân R, Narlikar L, Hartemink AJ:A Fast, Alignment-Free, ConservationBased Method for Transcription Factor Binding Site Discovery.Proc. RECOMB’082008,98-11

[15] Gordân R, Hartemink AJ:Using DNA Duplex Stability Information for Transcription Factor Binding Site Discovery.Pacific Symposium on Biocomputing2008, 453-464.

[16] Jones, Neil C., and Pavel A. Pevzner. *An Introduction to Bioinformatics Algorithms*. Cambridge, MA: MIT, 2004. Print.

[17] Sanguthevar Rajasekaran, Sudha Balla, and Chun-Hsi Huang.: Exact algorithms for planted motif challenge problems. *Dept. of Computer Science and Engineering Univ. of Connecticut, Storrs, CT 06269-2155, USA*

[18] Mohamed Ibrahim Abouelhoda, Stefan Kurtz, Enno Ohlebusch: Replacing suffix trees with enhanced suffix arrays,24: 2012