BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

DNA Motif Finding Algorithm

Authors
Faisal Bin Ashraf
&
Ali Imam Abir

Department of Computer Science and Engineering (CSE)
Islamic University of Technology (IUT)
Organization of Islamic Cooperation (OIC)
Gazipur-1704, Bangladesh

November 2016

DNA Motif Finding Algorithm

Authors
Faisal Bin Ashraf
ID: 124413
&
Ali Imam Abir
ID: 124453

Supervisor
Prof. Dr. M. A. Mottalib
Head, Department of CSE

Co-Supervisor
Md Sirajus Salekin
Lecturer, Department of CSE

**A thesis submitted to the Department of CSE
in partial fulfillment of the requiremnts for the degree of B.Sc.
Engineering in CSE**

**Department of Computer Science and Engineering (CSE)
Islamic University of Technology (IUT)
Organization of Islamic Cooperation (OIC)
Gazipur-1704, Bangladesh**

November 2016

Declaration of Authorship

This is to certify that the work presented in this thesis is the outcome of the analysis and experiments carried out by Faisal Bin Ashraf and Ali Imam Abir under the supervision of Prof. Dr. M. A. Mottalib, Head of the Department of Computer Science and Engineering (CSE), Islamic University of Technology (IUT), Dhaka, Bangladesh. It is also declared that neither of this thesis nor 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 Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

Authors:

Faisal Bin Ashraf
Student ID - 124413

Ali Imam Abir
Student ID - 124453

Supervisor:

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

Co-Supervisor:

Md Sirajus Salekin
Lecturer, Department of Computer Science and Engineering
Islamic University of Technology (IUT)

Acknowledgment

We would like to express our grateful appreciation for **Prof. Dr. M.A. Mottalib**, Head of the Department, Department of Computer Science & Technology, IUT for being our advisor and mentor. His motivation, suggestions and insights for this thesis have been invaluable. Without his support and proper guidance this research would not have been possible. His valuable opinion, time and input provided throughout the thesis work, from first phase of thesis topics introduction, subject selection, proposing algorithm, modification till the project implementation and finalization which helped us to do our thesis work in proper way. We are really grateful to him.

We are also grateful to **Md Sirajus Salekin**, Lecturer, Department of Computer Science & Engineering, IUT for his valuable inspection and suggestions on our proposal of Bucketing Method for Finding Planted Motif in DNA sequences.

Abstract

DNA contains the information of structure and function of different molecules of any living being. Short repeating patterns in a DNA sequence, called Motifs, are useful to understand and analyze this information. Gene function, drug design etc. has influences of motifs and can be well understand by studying motifs. Transcription Factor Binding Sites, Regulatory elements also can be found using motifs. So motif finding is very important in computational biology. Recent advancements in gene expression analysis already prompt the scientists to introduce a number of motif finding algorithms. Planted Motif Search (PMS) is one of them. It has been found as NP-Hard problem and takes exponential time to calculate. Finding all the possible motifs of the given input set is done by an exact algorithm. But it takes lots of time to calculate. Approximate algorithms always take less time than exact algorithm but do not find all the possible motifs always. We have proposed an approximate algorithm for Planted Motif Search which at first generates all possible motif set and use a bucketing concepts to find out the proper motifs from the whole data set. We use two benchmark data sets of DNA sequences to perform the comparative analysis with other approaches. Experimental results show that our bucketing approach finds more amounts of motifs than the other approximate algorithms and takes less amount of time for about all of the cases than exact algorithms. Most of the time, we get above 80% of the possible motifs of the given input set.

Contents

Acknowledgement i

Abstract ii

Table of Contents iii

1 Introduction 1
- 1.1 Overview . . . 1
- 1.2 Problem Statement . . . 1
- 1.3 Motivation & Scopes . . . 2
- 1.4 Research Challenges . . . 3
- 1.5 Research Objectives . . . 3
- 1.6 Research Contribution . . . 4
- 1.7 Thesis Organization . . . 4

2 Literature Review 5
- 2.1 Approaches . . . 5
- 2.2 Exact Algorithms . . . 6
  - 2.2.1 PMS1 . . . 6
  - 2.2.2 PMS2 . . . 7
  - 2.2.3 PMS3 . . . 9
  - 2.2.4 PMSPrune . . . 11
  - 2.2.5 PMS5 . . . 12
  - 2.2.6 WINNOWER . . . 12
  - 2.2.7 SP-STAR . . . 14
- 2.3 Approximate Algorithms . . . 16
  - 2.3.1 GIBB'S SAMPLING . . . 16
  - 2.3.2 Random Projection . . . 18
  - 2.3.3 RPB-DC . . . 20
- 2.4 Overall Comparison . . . 21

3 Proposed Method 22
- 3.1 Notations and Definitions . . . 22
- 3.2 Proposed Algorithm . . . 22
- 3.3 Improvement of the Proposed Algorithm . . . 27

4 Experimental Results & Discussion 30
- 4.1 Dataset . . . 30
- 4.2 Result Analysis . . . 30
  - 4.2.1 Time Comparison . . . 30
  - 4.2.2 Accuracy . . . 31

4.2.3 Different Instances 32

5 Conclusion and Future Work **33**
5.1 Summary of works 33
5.2 Scope of Future works 33

Chapter : Introduction

This thesis is about finding planted motifs in DNA sequences. In this chapter we discuss about the Overview of the thesis, Definition of the problem, Motivation behind our thesis, Scope of the thesis, Research challenge, Objectives of the thesis, Our contribution and Organization of this book.

1.1 Overview

Cells are the basic structural, functional and biological unit of all living being. Human being has 60 trillion cells of 320 types. Nucleus is the power house of cell where chromosomes are present which contains DNA. DNA is self replicating material which is the carrier of genetic information.Without sequence information encoded as DNA none of the living being could exist. Understanding sequences is therefore important for biological scientists. In this thesis, we study different algorithms for the Motif Discovery Problem and propose an efficient algorithm for this purpose.

A DNA molecule consists of two anti-parallel chains of four types of nucleotides forming a double helix. These nucleotides are - Adenine (A), Cytosine (C), Guanine (G) and Thymine (T). A single strand of this double helix DNA can be expressed as a string over the alphabet $\sum = \{A, C, G, T\}$. Proteins which are special sequence of nucleotides that bind to DNA in a specific manner is called Transcription Factor and this sequence of nucleotides are called Binding Motifs[1]. Motif Discovery Problem arises when DNA contains instances of binding motifs and they are unknown. Example of Binding Motif :

GGCTGCA**CACGT**GTATTGC...ACGATGTCTCGC
ATCGCAT**CACGT**GACCAGT...GACATGGACGGC
TCG**CACGT**GGTGGTACAGT...AACATGACTAAA
CGTTAGGACCAT**CACGT**GA...ACAATGAGAGCG
TAGCC**CACGT**GGATCTTGT...AGAATGGCCTAT

Here, "CACGT" is repeated in all the sequences. "CACGT" is one of the motif for these given DNA sequences.

1.2 Problem Statement

Finding repeating patterns in DNA sequences for a given length considering a given number of maximum mismatch.

The problem statement means we need to find (l, d) motifs from a given set of DNA sequences where,

l = length of motifs needed to be discovered
d = number of maximum mismatch considerable for being a motif

n = number of DNA sequences in the input Data set
m = length of each DNA sequence in the input Data set

Motif Discovery Problem is illustrated below with a sample dataset:

$$\begin{bmatrix} TTACGTACG\underline{\mathbf{AATCG}}AC \rightarrow \text{mismatch with “AATCG”} = 0 \\ \underline{\mathbf{ATTCG}}GTACGTAGAAT \rightarrow \text{mismatch with “AATCG”} = 1 \\ CTTAGGCTGC\underline{\mathbf{TATAG}}T \rightarrow \text{mismatch with “AATCG”} = 2 \\ GTACTG\underline{\mathbf{ACTCG}}CTAGC \rightarrow \text{mismatch with “AATCG”} = 1 \end{bmatrix}$$

Here “AATCG” can be (5,2) motif for this Data set as in every DNA sequence there are sub sequences of length 5 having maximum mismatch 2 with “AATCG”.

1.3 Motivation & Scopes

In biology, sequence motifs are short sequence patterns, usually with fixed lengths, that represent many features of DNA, RNA, and protein molecules. Sequence motifs can represent transcription factor binding sites for DNA, splice junctions for RNA, and binding domains for proteins. Motifs helps to find similarities between different DNA samples by assisting in Sequence Alignment. Analysis of mutation in genomes, disease detection and drug designing requires the assistance of sequence motifs. Thus, discovering sequence motifs can lead to a better understanding of transcriptional regulation, mRNA splicing, and the formation of protein complexes. Furthermore, protein motifs can represent the active sites of enzymes or regions involved in protein structure and stability. Motif discovery is an important computational problem because it allows the discovery of patterns in biological sequences in order to better understand the structure and function of the molecules the sequences represent. Especially, identifying regulatory elements, especially the binding sites in DNA for transcription factor, is important to understand the mechanisms that regulate gene expression. These DNA motif patterns are usually fairly short (5-20 base pairs long) and is known to recur in different genes or several times within a gene [2]. A DNA sequence can have zero, one, or multiple copies of a motif. In addition to these more common forms DNA motifs, there are also palindromic motifs (sub sequence that is exactly the same as its own reverse complement) and gapped motifs (two smaller conserved sites separated by a gap) [3] . The high diversity and variability of motifs make them very difficult to identify.

A large number of algorithms for finding DNA motifs have been developed. These algorithms mostly detect overrepresented motifs and conserved motifs that might be good candidates for being transcription factor binding sites. Algorithms that detect overrepresented motifs deduce motifs by considering the regulatory region (promoter) of several co-regulated or co-expressed genes. Co-regulated genes are known to share some similarities in their regulatory mechanism, possibly at transcriptional level, so their promoter regions might contain some common motifs that are binding sites for transcription factors. Thus, the way to detect these regulatory elements is to search

for statistically overrepresented motifs in the promoter region of such a set of co-expressed genes. However, algorithms that detect overrepresented motifs perform not as well in higher organisms. To overcome this, some algorithms consider conserved motifs from orthologous species. Since selective pressure causes functional sequences to evolve slower than non-functional sequences, well-conserved sites represent possible candidates for DNA motifs. Recent algorithms have also combined the two approaches to achieve improvement in motif finding. As Motif Discovery is an NP-Hard problem and it takes exponential time to discover motifs there is a scope for reducing run-time. Apart from this, large length motif discovery is also a good area to research as finding large length motif still a big challenge as it takes time up to several days for computation.

1.4 Research Challenges

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 and its discovery from a set of 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 discovery, capable of dealing with the enormous amount of data coming from the Bioinformatics community.

As the size of data is huge, we need to design an algorithm such that it can deal with different types of data from different species. It takes exponential time to find motifs. This is one of the major drawback designing any algorithm for this problem. There is a trade off between time complexity and efficiency. For finding all the motifs exist in the data set it takes huge time. On the contrary, run time can be reduced by reducing the efficiency, i.e, finding less amount of motifs than existing motifs. Finding large length motif is also a big challenge in motif discovery as it will take huge amount of time as well as space. As huge amount of data needs to be manipulated and number of motifs for any DNA sequence is huge it consumes a lot of memory. Reducing memory consumption is an important challenge in Motif Discovery Problem.

1.5 Research Objectives

Objective of this thesis is to propose an approximate algorithm for planted motif search problem. As this problem takes exponential time, reducing execution time is the prime objective. Since we propose an approximate algorithm finding as much motif as possible is an issue. We tried to solve the problem for different data samples. Finding motif of different length and mismatch is one of the major concern. Though the data samples of DNA are huge, reducing memory consumption during execution is also our objective.

1.6 Research Contribution

We proposed an algorithm introducing hashing which reduces execution time and can find around 80% motifs of the given samples. Then we improved our algorithm for finding large length motifs and reducing execution time. Our improved algorithm can find upto 30 length motifs with at most 10 mismatches.

1.7 Thesis Organization

In Chapter 1 we have discussed our study in a precise and concise manner. Chapter 2 deals with the necessary literature review for our study and their development so far. In Chapter 3 we have stated the skeleton of our proposed method, proposed algorithm and also the flowchart to provide a detail insight of the working procedure of our proposed method **R**andomly **P**rojected **P**ossible **M**otif **D**iscovery (RPPMD) using Motif prediction and Bucketing. Chapter 4 shows the results and comparative analysis of successful implementation of our proposed method. The final segment of this study contains all the references and credits used.

Chapter : Literature Review

We studied literature related to DNA motif finding problem. In this chapter we discuss about different algorithms proposed in those literature.

2.1 Approaches

Many version of motif search problem has been researched so far. Motifs search problem can be different types –Simple Motif Search (SMS)[4], Edit distance based Motif Search (EMS)[5], Planted Motif Search (PMS)[6, 7, 8].

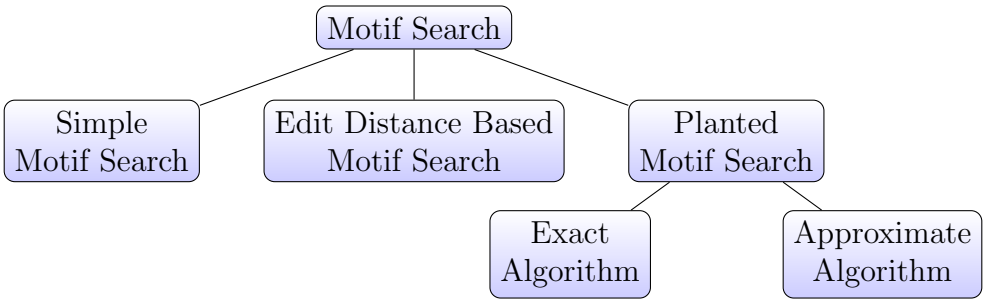


Figure 1: Different Approaches of Motif Search

1. Simple Motif Search (SMS)[4]:

In simple motif search, there is only one input – length of the motif (l). All the motifs of length 1 to l are searched. All the given sequences are checked if they contain any motif of length 1 to l. In this case, maximum number of mismatch considerable is half of length of the motif, i.e., $l/2$.

2. Edit distance based Motif Search (EMS)[5]:

In EMS, there are three inputs – length of motif (l), maximum number of mismatch considerable (d), and a threshold value (q). If any subsequence of length l has occurrences in at least q sequences out of n sequences (n = total number of given sequences) with maximum d number of mismatches, then it will be outputted as a motif.

3. Planted Motif Search (PMS)[6, 7, 8]:

Planted motif search requires two input value – length of motif (l) and maximum number of mismatch considerable (d). That is why it is also called (l, d)-planted motif search problem. All the sequences are checked one by one and if any subsequence is present in all sequences having maximum d number of mismatch then it will be a motif of the given data set.

We have explored further in Planted Motif Search (PMS). There are two types of Planted Motif Search (PMS) algorithm-

- Exact Algorithm
- Approximate Algorithm

Exact Algorithm:
Exact algorithms find out all the possible motifs in the given data set. Finding out all possible motifs is a very complex and time consuming problem. It takes huge amount of time, even days, for finding out the motif of relatively large length motif. But it guarantees that it will find out all possible motifs.

Approximate Algorithm:
Approximate algorithms give emphasis on time. They find out the expected motifs in very efficient time with respect to exact algorithm. But they do not find all possible motifs all the time.

2.2 Exact Algorithms

Some of the Exact algorithms for motif discovery are
PMS1[4], PMS2[4], PMS3[4], PMS4[9], Winnower[10], SP-Star[10], RISOTTO[11], VOTING[12], Pampa[13], PMSPrune[14], PMS5[15], PMS6[16], PMS8[7], qPMS7[17], qPMS9[18].

2.2.1 PMS1

Rajasekaran, Balla, and Huang (2005) give a detailed description of algorithm PMS1 in [4]. From each of the n input sequences (S_1, S_2, ..., S_n) all possible $l-mer$s are generated; let C_i contain every $l-mer$ of S_i. For every $l-mer$ uC_i , all possible $l-mer$s v are generated such that v is at a Hamming distance of at most d from u. Such l-mers v are individually termed neighbors of u and are members of the neighborhood of u. Let Li contain the collection of neighborhoods of all $l-mer$s u from C_i. All sets L_i are alphabetically sorted and duplicates are eliminated. Any $l-mer$ that is an element of every set L_i is added to the solution set M of motifs. Using algorithm PMS1[4], any given PMSP may be solved in $O\left(mn\binom{l}{d}\left(|\sum|-1\right)^d \frac{1}{w}\right)$ time, where w is the word length of the computer.

Consider the following simple algorithm for the planted motif problem:

- Let the input sequences be $S_1, S_2, ..., S_n$.The length of each sequence is m. Form all possible $l-mer$s from out of these sequences.The total number of $l-mer$ is less equal nm. Call this collection of $l-mer$s C. Let the collection of $l-mer$s in S_1 be C

- Let u be an $l-mer$ in C''. For all uC'' 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 $\binom{l}{d}3^d$. Thus the total number of patterns generated is $m\binom{l}{d}3^d$. 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)

- For every pair of $l-mers$ (u, v) with uC and vC'' compute the hamming distance between u and v. Output that $l-mers$ of C'' that has a neighbor (i.e., an l-mer at a hamming distance of d) in each one of the n input sequences. The run time of this algorithm is $O(nm^2\binom{l}{d}3^d)$. If M occurs in one of the input sequences, then this algorithm will run in time $O(n^2m^2l)$.

Another equally simple algorithm considers every possible l-mer one at a time and checks if this $l-mer$ is the correct motif M. There are 4^l possible $l-mers$. Let M'' be one such $l-mer$.We can check if $M''=M$ as follows. Compute the hamming distance between u and M'' for every uC. (Note that C is the collection of all possible l-mers in the input sequences.) As a result we can check if M occurs in each input sequence (at a hamming distance of d). Thus we can identify all the motifs of interest in a total of $O(nml4^l)$ time.

Algorithm PMS1[4] is based on sorting and takes the following form:

- Generate all possible $l-mers$ from out of each of the n input sequences. Let C_i be the collection of $l-mers$ from out of S_i for $1 <= i <= n$.

- For all $1 <= i <= n$ and for all uC_i generate all $l-mers$ v such that u and v are at a hamming distance of d. Let the collection of $l-mers$ corresponding to C_i be C_i'' for $1 <= i <= n$. The total number of patterns in any C_i'' is $O(m\binom{l}{d}3^d)$.

- Sort all the $l-mers$ in every $C_i'', 1 <= i <= n$ and eliminate duplicates in every C_i'' .Let L_i be the resultant sorted list corresponding to C_i''.

- Merge all the L_is $(1 <= i <= n)$ and output the generated $l-mer$ that occurs in all the L_is.

2.2.2 PMS2

An improved algorithm over PMS1[4] called PMS2[4]. Let M be the planted motif. Note that if M occurs in every input sequence, then every substring of M also occurs in every input sequence. In particular, there are at least $l-k+1k-mers$ $(ford <= k <= I)$ such that each of these occurs in every input sequence at a hamming distance of at most d. Let Q be the collection of $k-mers$ that can be formed out of M . There are $l-k+1$ k-mers in Q. Each one of these $k-mers$ will be present in each input sequence at a hamming distance of at most d.

In addition, in every input sequence S_i , there will be at least one position ij such that a $k - mer$ of Q occurs starting from i_j ; another $k - mer$ of Q occurs starting from $i_j + 1; ...;$ yet another $k - mer$ occurs starting from $i_j + l - k$. We can get an $l - mer$ putting together these $k - mer$s that occur starting from each such i_j.

Possibly, there could be many motifs of length k that are in the positions starting from each of $i_j, i_j + 1, ..., i_j + l$–$k$ such that all of these motifs are present in all of the input sequences (with a hamming distance of at most d). Assume that $M_(ij + r)$ is one motif of length k that starts from position $i_j + r$ of Si that is also present in every input sequence,(for $0 <= r <= l - k$) . If the last $k - 1$ residues of $M_(ij + r + 1)$ (for $0 <= r <= l - k - 1$), then we can obtain an $l - mer$ from these motifs in the obvious way. This $l - mer$ is potentially a correct motif. Also, note that to obtain potential motifs (of length l), it suffices to process one of the input sequences (in a manner described above). Now we are ready to describe the improved algorithm.

There are two phases in the algorithm. In the first phase we identify all $(d+c) - mer$s $M_(d + c)$(for some appropriate value c) that occur in each of the input sequences at a hamming distance of at most d. We also collect potential $l - mer$s (as described above) in this phase. In the second phase we check, for each $l - merM'$ collected in the firrst phase, if M' is a correct answer or not. Finally we output all the correct answers.

First we observe that the algorithm PMS1[4] can also be used for the case when we look for a motif M that occurs in each input sequence at a hamming distance of at most d. The second observation is that if c is large enough then there will not be many spurious hits. A suggested value for c is the largest integer for which PMS1[4] could be run (without exceeding the core memory of the computer and within a reasonable amount of time).

We present more details on the two phases

Phase 1:

Solve the planted $(d + c, d) - motif$ problem on the input sequences (with a hamming distance of $<= d$, using e.g., a modified PMS1 []). Let R be the set of all motifs found. Let S_k be one of the input sequences. (S_k could be an arbitrary input sequence; it could be chosen randomly as well.) Find all the occurrences of all the motifs of R in S_k (with a hamming distance of up to d). This can be done, e.g., as follows: form all the $(d+c) - mer$s of S_k (keeping track of the starting position of each in S_k); For each $(d + c) - mer\ u\ \epsilon\ S_k$, find all the $(d + c) - mers\ v$ such that u and v are at a hamming distance of at most d. If R' is the collection of these $(d+c) - mer$s, sort R and R' and merge them; and figure out all the occurrences of interest.

Let S_k be of length m. For every position i in S_k, let L_i be the list of all motifs of R that are in S_k (with a hamming distance of $<= d$) starting from position i. Let A be the $l - mer$ of S_k that occurs starting from position i. Let M_1 be a member of

L_i. If M_2 is a member of $L_i + 1 - (d + c)$ such that the last $2(d + c) - l$ characters of M_1 are the same as the first $2(d + c) - l$ characters of M_2,then we could get an $l - mer$ B by appending the last $l - (d + c)$ residues of M_2 to M_1 (at the end). If the hamming distance between A and B is d, then B is retained as a candidate for the correct motif. We gather all such candidates and check if any of these candidates are correct motifs. Details are given below.

Phase 1:
for $i := 1$ to $m - l + 1$
for every $u \epsilon L_i$
for every $v \epsilon L_{i+l-(d+c)}$
Let the $l - mer$ of S_k starting from position i be A. If the last $2(d + c) - l$ residues of u are the same as the first $2(d+c)$–l residues of v, then form an $l - mer$ B by appending the last $l - (d + c)$ residues of v to u. If the hamming distance between A and B is d, then add B to the list C of candidates.

Phase 2:
for every $v \epsilon C$
Check if v is a correct motif in $O(nml)$ time

For any node u of $L + i$ there can be at most $4^{l-(d+c)}$ candidate motifs. Thus the time needed to get all the candidate motif is $O\left(\sum_{i=1}^{m-l+1} |L_i| \, 4^{l-(d+c)^l}\right)$.

Thus we will get our expected motif. PMS2 [4] is better than PMS1[4].

2.2.3 PMS3

A third algorithm called PMS3 [4] has been given which is more memory saving than PMS2 [4]. This algorithm enables one to handle large values of d. Let $d' = [d/2]$. Let M be the motif interest with $|M| = l = 2l'$ for some integer l'. let M_1 refer to the first half of M and M_2 to the second half. We know that M occurs in every input sequence. Let S $= S_1, S_2, ..., S_m$ be an arbitrary input sequence.

Let the occurrence of M (with a hamming distance of d) in S start at position i. Let $S' = S_i, S_{i+1}, ..., S_{i+l'-1} and S'' = S'_{i+l}, ..., S_{i+l-1}$.

Then, clearly, either

- The hamming distance between M1 and S' is at most d' or
- The hamming distance between M2 and S" is at most d'

Also, note that in every input sequence either M1 occurs with a hamming distance of at most d' or M2 occurs with a hamming distance of at most d' As a result, in at least t' sequences (where $t' = [t/2]$) either M1 occurs with a hamming distance of at most d' or M2 occurs with a hamming distance of at most d'. PMS3 [4] exploits these observations.

Memory Saving:

The way PMS1[4] is described, we first form all possible $l - mer$s from out of all the input sequences, generate all relevant neighbors of these $l - mer$s, sort and merge all of them to identify the generated $l - mer$s found in all the sequences. We can modify the algorithm as follows so as to reduce the memory used.

For PMS1 [4] we generate all possible $l - mer$s from out of the first input sequences S_1. But there exists some extension in this approach.

Extensions:

We consider two variants in this section.

Problem 1(a). Input is n sequences each of length m. The problem is to identify a motif M of length l. It is given that each input sequence has a sub string of length l such that the hamming distance between this substring and M is at most d.

Theorem:

Problem 1(a) can be solved in time $O(nm\sum_{i=0}^{d}\binom{l}{i}3^i\frac{l}{w})$ then this run time is $O(nm\binom{l}{d}3^d\frac{l}{w})$.

Proof:An algorithm similar to PMS1[4] can be devised.

- Generate all possible $l - mer$s from out of each of the n input sequences. Let C_i be the collection of $l - mer$s from out of S_i for $1 <= i <= n$.
- For all $1 <= i <= n$ and for all $u \epsilon C_i$ generate all $l - mer$s v such that u and v are at a hamming distance of at most d . Let the collection of $l - mer$s corresponding to C_i be C_i' for $1 <= i <= n$. The total number of patterns in any C_i' is $O(\sum_{i=0}^{d}\binom{l}{i}3^i)$.
- Sort all the $l - mer$s in every C_i' and eliminate duplicates, $1 <= i <= n$. Let L_i be the resultant sorted list corresponding to C_i'.
- Merge all the $L_is(1 <= i <= n)$ and output the generated (in step 2) $l - mer$ that occurs in all the L_is.

Problem 1(b). Input are n sequences each of length m. The problem is to find all motifs M of length l. A motif M should be output if it occurs in at least ϵ_n of the input sequences at a hamming distance of d. Here ϵ is a fraction specified as a part of the input.

Theorem:

Problem 1(b) can be solved in time $O(nm\binom{l}{d}3^d\frac{l}{w})$.

Proof:

The algorithm to be used is the same as PMS1 [4]. The only difference is that step 4 now becomes:" Merge all the Lis $(1 <= i <= n)$ and output the generated (in step 2) $l - mer$s that occur in at least ϵ_n of the L_is " . The run time remains the same asymptotically.

One could also refine Problem 1(b) to look for motifs that occur at a hamming distance of at most d.

2.2.4 PMSPrune

This algorithm introduces a tree structure for the motif candidates and uses a branch-and-bound algorithm to reduce the search space. Let $S = s_1, s_2, \ldots, s_n$ be a given set of input strings. PMSprune[14] follows the same strategy as PMS0: For every $l-mer$ y in S_1 it generates the set of neighbors of y and, for each of them, checks whether this is a motif or not. Some key steps in the algorithm are:

- It generates the $d-neighborhood$ of every $l-mer$ y in s_1 using a tree of height d. The root of this tree will have y. Every $l-mer$ that is at a distance of 1 from y will be a node in the tree that is at a distance of 1 from the root; every $l-mer$ that is at a distance of 2 from y will be a node in the tree that is at a distance of 2 from the root; and so on.When a node in this tree is visited, check if the corresponding l-mer is an (l,d)-motif. I.e., if the $l-mer$ is x, check if $d_H(x,S)d$. If so, output this $l-mer$. In any case move to the next node in the tree. This tree is explored in a depth first manner.

- If each node in the tree is visited for each $l-mer$ Y in S_1, then the run time of PMSPrune[14] will be at least as much as that of PMS0. PMSPrune[14] uses some pruning conditions to prune sub trees that cannot possibly have any motifs in them.

- For an $l-mer$ x, which corresponds to a node in a sub tree of height h, the algorithm uses the value of $d_H(x,S)$ and h to prune the descendants of x.

- PMSPrune[14] calculates the value of $d_H(x,S)$ for the nodes (x) in the tree in an incremental way, taking into account the way in which the neighborhood is generated.

Overall Process:

The overall process of PMSPrune[14] is given below:

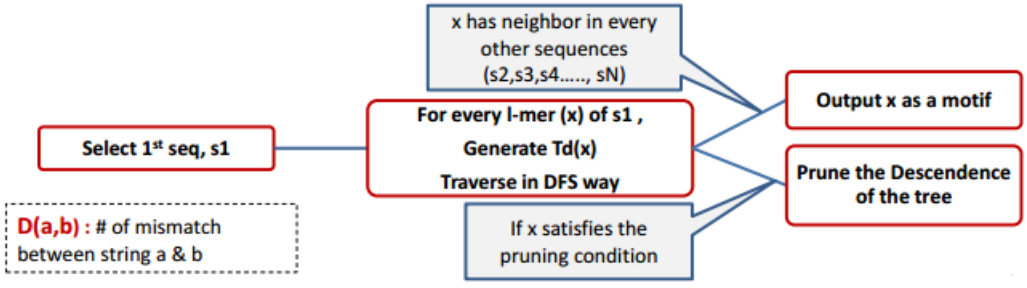


Figure 1: Overall process of PMSPrune[14]

2.2.5 PMS5

PMS5 [15] is an extension of PMS0[19]. If $S = \{S_1, S_2, \ldots, S_n\}$ is a set of strings (not necessarily of the same length). Let $M_d^l(S)$ denote the (l, d)-motifs present in S. Let $S'' = S_2, S_3, \ldots, S_n$. PMS5[15] computes the (l, d)-motifs of S as $\coprod_{L \in S_1} M_d^l(L, S')$. Here L refers to an $l - mer$.

One of the key steps in the algorithm is a subroutine to compute the common d-neighborhood of three $l - mer$s. Let x, y, z be any three $l - mer$s. To compute B_d (x, y, z), PMS5 represents $B_d(x)$ as a tree $T_d(x)$. Each node in this tree represents an $l - mer$ in $B_d(x)$. The root of $T_d(x)$ stands for the $l - mer$ x. $T_d(x)$ has a depth of d. Nodes of $T_d(x)$ are traversed in a depth-first manner. The node and the $l - mer$ it represents may be used.

Interchangeably While the tree is traversed, any node t will be output if t is in $B_d(y) \cap B_d(z)$. When any node t is visited, check if there is a descendant $''$ of t such that $''$ is in $B_d(y) \cap B_d(z)$. Prune the sub tree rooted at t if there is no such descendant. In PMS5[15], the problem of checking if t has any descendant that is in $B_d(y) \cap B_d(z)$ is formulated as an integer linear program (ILP) on ten variables. This ILP is solved in $O(1)$ time. Solving the ILP instances is done as a prepossessing step and the results are stored in a look up table.

2.2.6 WINNOWER

WINNOWER [10] is a graph-theoretic approach which represents a motif as a large clique (complete sub graph) and attempt to solve the clique problem efficiently by filtering.

Problem Reduction:

Given a set of sequences $S = S_1, S_2, \ldots\ldots, S_m$ and suppose we are looking for a (l, d)-motif. We construct a graph G as follows:

- Every vertex in G corresponds to a length-l word in S .
- Consider two words x and y appear in two different sequences in S . x and y are connected by an edge if their hamming distance is at most $2d$. $2d$ is chosen to cater for all possible (l, d)-motif, that "sits" in the middle in terms of distance between the two vertex that has distance $2d$.
- Note that G is a $m - partite$ graph. An example is shown in Figure 2.

Notice that the problem of finding a (l, d)-motif corresponds to finding a clique of size m, where m is the number of strings. Thus, the problem of finding motifs is reduced to the problem of finding large cliques. However, finding cliques is a NP-complete problem. Therefore WINNOWER[10] proposes a method to filter edges which definitely do not belong to any large cliques.

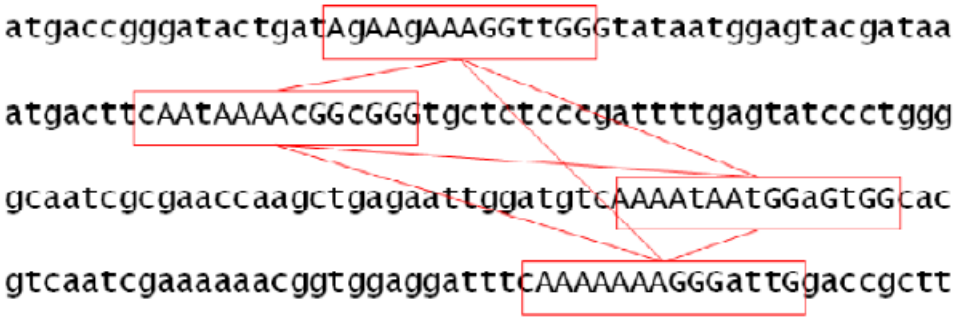


Figure 2: WINNOWER[10] M-partite Graph

Definitions

- A vertex v is a neighbor of a clique C if it is connected to every vertex in this clique (i.e., $C \cup \{v\}$ is a clique).
- A clique is called extendable if it has at least one neighbor in every part of the multipartite graph G.
- An edge is called spurious if it does not belong to any extendable clique of size k.

WINNOWER[10] is an iterative algorithm that converges to a collection of extendable cliques by filtering out spurious edges.

Algorithm:

1. Construct a graph G by:
   - Consider two words x and y appear in two different sequences in S . x and y are connected by an edge if their hamming distance is at most $2d$.
2. Filtering of spurious edges:
   - Filtering weak vertices:
     Vertices that are not supported by a neighbor in every part of G are filtered out.
   - Filtering weak edges:
     Unsupported edges are removed.
   - Filtering weak triangles.
   - If the computation allow filter more.

Example:

An example of finding cliques and removing spurious edges is shown in the following two figures.

Figure 3: Clique in a String

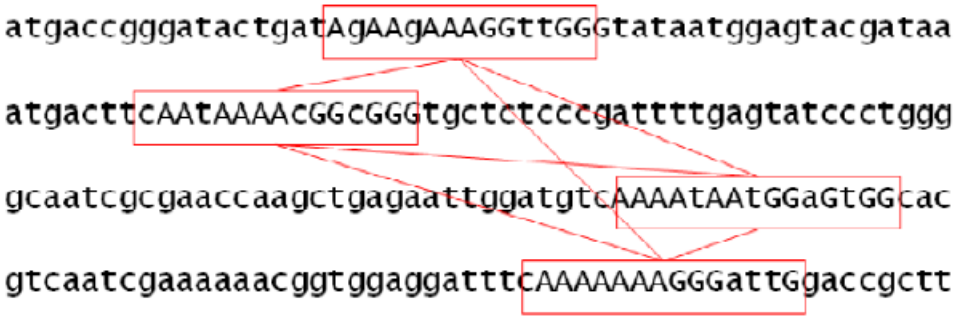


Figure 4: After Filtering Spurious Signal

2.2.7 SP-STAR

SP-STAR [10] is proposed by Pevzner and Sze. First, it chooses a suitable scoring function to access the goodness of a motif. Then, for each $l - mer$ appearing in the sample, find its best instance in each sequence and collect these instances together to form an initial motif. Employ a local improvement heuristic to improve each initial motif.

Definition:

Consider a set of length-l sequences $w_1, w_2, ..., w_n$. Define,

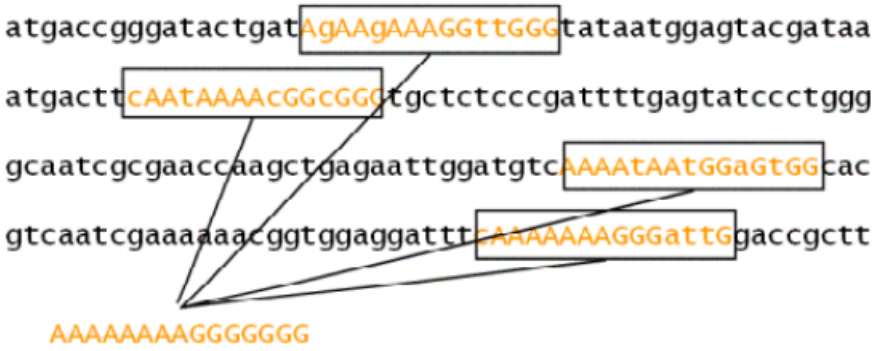
$$SPscore(w_1, w_2, ..., w_n) = \sum_{i,j} \delta(w_i, w_j)$$

Algorithm:

- Let W be the set of length-l words in all n sequences $S_1, S_2...., S_n$
- For any length-l word $w \in W$:
  1. Find the best match $w_1, w_2, ..., w_n$ in each of the n sequence .
  2. Get the consensus word, by counting the most frequently appearing word in each column. Let w be the consensus word.
  3. Repeat steps 1 and 2 until SPscore can not be further improved.

Example:

- Given a string v , locate the string w_i , which is closest to v in each sequence which is shown below.



- Use the most frequent letter in each position to define a new majority string.

	AGAAGAAAGGTTGGG
	CAATAAAACGGCGGG
	AAAATAATGGAGTGG
	CAAAAAAAGGGATTG
Majority string	AAAAAAAAGGGAGGG

- Repeat until there are no more changes in the string v.

The main reason why SP score is chosen over STAR is that it is able to give a better initial estimate of the goodness of a string. For subtle signals, typical distance between signal instances is less than or equal to $2d$, while typical distance between two random best instances from v can be as large as $4d$, where d is the number of mismatches, as shown in the following figure.

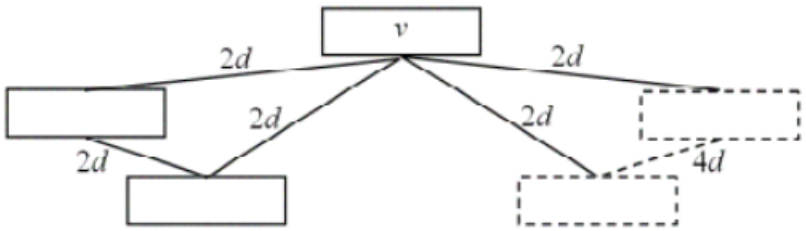


Figure 5: Distances between instances

2.3 Approximate Algorithms

Some of the approximate algorithms for motif discovery are - Gibb's Sampling[20], MEME[21], Random Projection[22], PhyloCon[23], Pattern branching[8], MEME:ZOOPS[24], MITRA[25], PhyME [26], Motion-Motif Graphs[27], PRIORITY- DC[28], MEME:ZOOP-DC[29], RPB-DC[30].

2.3.1 GIBB'S SAMPLING

A stochastic (Monte Carlo) algorithm for motif finding. Works by 'stumbling' onto a few motif instances, which bias the weight matrix, which causes it to sample more motif instances, which biases the weight matrix more until convergence. Not guaranteed to converge to same motif every time run several times, compare results. Works for protein, DNA, RNA motifs.

The Gibbs sampler [20] uses a randomized approach to iteratively improve a motif. Initially, a motif, represented as a PWM, is generated by selecting one random length l segment (substring) from each of the m sequences involved. Then the Gibbs sampler will repeatedly perform a predictive update step and a sampling step to iteratively improve the motif.

Predictive update step: In the predictive update step, one of the selected length l segment, chosen either at random or by some predetermined order, is removed from the selection.

Sampling step: Suppose that the removed segment is from sequence Si. In the sampling step, we add a length l segment from Si to the list of selected segments, which we will describe in more detail.

Firstly, from the remaining m - 1 selected segments, we compute the PWMs for the motif model $\theta(whichisusedtorepresentthemotif)aswellasthebackgroundmodel\theta_0$ (which is used to represent the background distribution of the sequences). The PWM for θ_0 will be computed based on the nucleotides in the sequences that are outside the selected segments, and every column of the PWM for $_0$ will be the same.

Let S_i, j be the length l segment at position j in S_i . We define W_θ to be the PWM for the model $\theta and \mathrm{W}_\theta(j)$ to be the value of the nucleotide at position j in W_θ. Also, we define

$$P(S_{i,j}|\theta) = \prod_{k=1}^{l} W_\theta(S_{i,j}[k], k]),$$

And $P(S_{i,j}|\theta_0)$is defined analogously.

For each segment $S_{i,j}$, we compute a weight $A_{i,j}$, defined to be

$$A_{i,j} = \frac{P(S_{i,j}|\theta)}{P(S_{i,j}|\theta_0)}$$

The weight $A_{i,j}$, after normalization, gives the probability of choosing segment $S_{i,j}$ for addition into the set of selected segments.

When using the Gibbs sampler[20], different run will give different results .Normally the motif is chosen from the best out of a number of runs. Also there are various variants of the standard Gibbs sampling approach. These include the Gibbs motif sampler AlingACE and BioProspector.

Algorithm: Given N sequences of length L and desired motif width W:

1. Choose a starting position in each sequence at random:

   A_1 in Seq 1, A_2 in Seq 2, ..., A_n in Seq N

2. Choose a sequence at random from the set (for example Seq 1).

3. Make a weight matrix model of width W from the sites in all sequences except the one chosen in step 2.

4. Assign a probability to each position in Seq 1 using the weight matrix model constructed in step 3:

   $p = p_1, p_2, p_3, \ldots, p_{L-W+1}$

5. Sample a starting position in Seq 1 based on this probability distribution and set a1 to this new position.

6. Choose a sequence at random from the set (for example Seq 2).

7. Make a weight matrix model of width W from the sites in all sequences except the one chosen in step 6.

8. Assign a probability to each position in Seq 2 using the weight matrix model constructed in step 7.

9. Sample a starting position in Seq 2 based on this dist.

10. Repeat until convergence (of positions or motif model)

2.3.2 Random Projection

Introduced by Buhler and Tompa (2001). It's an approximate approach for finding motif. Suppose we have a planted (l, d)-motif problem on t sequences of$n$ letters. We randomly select k positions out of l positions. This selection is called (l, k) gapped pattern.

For each $l - mer$ in the data we look at its k-subset which is defined by the gapped pattern. This subset is called projection. We use the obtained k-subset as a hash value for the l-mer.

Given a gapped pattern, count all k-length hashes of all $l - mer$s. We cannot avoid collisions, since many $l - mer$s can produce same hash. We group $l - mer$s that hash to the same $k - mer$ in a set called bucket. Random sequences should have the same bucket size for any hashed pattern. Hopefully, some buckets will be significantly over-represented, which will mean that they represent motifs.

Lets see what parameter k should we consider. First of all, for a hash to have a practical meaning, we need $k \leq l - d$. Otherwise, we risk including mutated bases in our hash function.

On the other hand, we want to exclude random sequences. For this we bound k from below. Given a random sequence, calculate expected number of counts in its bucket:

$$E[counts] = \frac{nt}{4^k}$$

Hence if $k > log_4 n \cdot t$, then $E[counts] < 1$ for a random sequence.

Let P be a pattern which represents an (l, d)-motif. We define a planted bucket to be a bucket to which P hashes.

The probability that instance of P in a single sequence hashes to $h(P)$ is

$$Prob[P \to h(p)] = \frac{\binom{l-d}{k}}{\binom{l}{k}} = P_{hit}$$

For $k = 7$ and for $(l, d) = (15, 4)$, $P_{hit} = 0.06$ and we have 20 chances to hit it. The expected number of buckets hit by random sequences is

$$E[|h(P)|)] = \frac{nt}{4^k} + n.P_{hit}$$

Where $n * P_{hit}$ is a probability that expected sequence has got matched with a bucket. This formula tells us that we will see an over-representation in some bucket.

We define a threshold S on bucket counts. Buckets with size above S will be suspects for a planted bucket.The probability that the size of planted bucket will fail to exceed the threshold is:

$$Prob|[h(P)| < S] = \sum_{i=0}^{S-1} \binom{n}{i} (P_{hit})^i (1 - P_{hit})^{n-1} = P_{fail}$$

Above value can be as large as 0.1, which clearly does not help to find the planted bucket.

The solution is to repeat the projection m times. The probability of failing to flag the planted bucket $h(P)$ after m trials is P_{fail}. If we agree to not be able to find motif 1% of time, take

$$m \geq \frac{log(0.01)}{log(P_{fail})}$$

Then 99% of time, the planted bucket gets over the threshold S at least once in m trials.

Having identified the planted bucket we need to figure out what is the motif. The algorithm for doing this is roughly as follows:

- Recover the locations which hash to the planted bucket.
- Construct a profile from these substrings.
- Use an iteration motif finder from this profile.
- If the consensus is an (l, d)-motif, then report.

Random projections[22] algorithm works better than other mentioned algorithms because previous work was largely based on $k - mer$ counts. In other algorithms, every frequent $k - mer$ is used either in some weird graph problem or every $l - mer$ in the sequence is used as an iterative start point as done in WINNOWER [ ], for example. But consecutive patterns just do not get into the twilight zone.

However, situation with real data is a bit different. In real data we have non-uniform letter frequencies, non-independence and mismatch positions much more correlated, since "random data" has non-random profile.

2.3.3 RPB-DC

This method starts with random projection of the input sequences. Once the projected buckets are created and sorted according to the frequency of elements in each bucket, each element of buckets are extended based on Position Specific Prior (PSP) information specifically the DC priors. A popular benchmark data set for motif discovery is used here.

Algorithm:

- Step 1. Take the Bucket with higher number of elements in it.
- Step 2. Take one element of the selected bucket. Here every element is $a(l_min - 1)mer$ which is represented by two numbers - one is the starting index and another is the sequence number.
- Step 3. Select one nucleotide alphabetically ($A < C < G < T$) and extend the selected element in step 2. Now extend all other elements in the same bucket with this nucleotide and star counting the number of matches among the elements of the bucket after extension. Retain the nucleotide for extension which brings maximum number of matches and discard others.
- Step 4. If not a single match is found after extension in step 3 then it will be counted as a mismatch.
- Step 5. Repeat Step 3 and Step 4 until the motif is extended up to l_max or number of mismatches is greater than M or both.
- Step 6. If the extension has been stopped then check if the length of extended motif is within the range from l_min to l_max. If yes then store it in the final list with its length and list of indexes in the sequences where it occurred.
- Step 7. Repeat from Step 2 to Step 6 for every elements in the bucket.
- Step 8. Repeats from Step 1 to Step 7 for every other buckets with elements in them.

This method is not only effective for its higher success rate but also very efficient because it reduces costs by avoiding comparisons of every possible motifs of a predefined length. The buckets created by Random Projection in this method provide the potential starting positions within sequences to start comparison rather than sequential brute force method from the starting of every sequence.

2.4 Overall Comparison

We have compared all the exact and approximate algorithms that we have studied so far in the Table 1. PMS1[4], PMS2 [4], PMS3[4] - these three algorithm can find motif of maximum length of 15 with mismatch 4. PMSPrune[14] can find $(17, 7)$ motifs and PMS5[15] can find up to $(19, 7)$. All of these are exact algorithm. All of these algorithms takes huge memory to execute. They all build tree and traverse the tree in DFS manner to find the motifs.WINNOWER[10], SP-STAR[10] are also exact algorithm.

Table 1: Comparison Among the Studied Algorithms

Algorithm	**Highest Length of MOTIF can be found**	**Space Consumption**	**Approximate /Exact**
PMS1[4]	(15,4)	Highly Consuming	Exact
PMS2 [4]	(15,4)	Highly Consuming	Exact
PMS3[4]	(15,4)	Highly Consuming	Exact
PMSPrune[14]	(17,7)	Highly Consuming	Exact
PMS5[15]	(19,7)	Highly Consuming	Exact
Gibbs Sampling[20]	-	Consuming	Approximate
WINNOWER[10]	(15,4)	Consuming	Exact
SP-STAR[10]	(15,4)	Consuming	Exact
Random Projection[22]	(18,6)	Consuming	Approximate
RPB-DC[30]	-	Consuming	Approximate

Gibbs Sampling[20], Random Projection[22] , RPB-DC[30] - these are approximate algorithm. They cannot find all the motifs that is present in any data set. WINNOWER[10] and SP-STAR[10] are pattern branched algorithms and can find up to $(15, 4)$ motifs. Random Projection[22] and RPB-DC[30] are based on hashing. They build some buckets using hashing and later the find the motifs by expectaion maximization on those buckets. These all are approximate algorithms. So, they take relatively less memory for execution and cannot find all the motifs of the data set.

Chapter : Proposed Method

In this chapter we proposed our algorithm with all necessary definitions and notations for better understanding.

3.1 Notations and Definitions

In this section , some notations and definitions are introduced which will help to describe our algorithm clearly.

Definition 1. *Given a set of sequences* $S = S_1, S_2, S_3, .., S_n$ *over an alphabet set* A, T, C, G *such that* $|S_i| = m$ *and two integer value* l *and* d, $0 \leq d < l < m$, *we define Planted Motif Search(PMS) as to find out a subsequence* x *such that* $|x| = l$ *and every sequence in* S *has a subsequence* x_i *such that* $|x_i| = l$ *and* x *differs from* x_i *in at most* d *places.*

Definition 2. *A string* x *having length of* l *is called an* $l - mer$.

Definition 3. $D_H(x, y)$ *is the number of mismatches between strings* x *and* y *over alphabet set* A, T, C, G.
$D_H(x, y)$:= *number of mismatches between* x *and* y

Definition 4. $B_d(x)$ *is the set of all neighbors of string* x *over alphabet set* A, T, C, G.
$B_d(x) := \{z \,| D_H(x, z) \leq d\}$

Definition 5. C_{bucket} *is the list of those buckets which are obtained after projecting all* $l - mers$ *of candidate motif set along* k *randomly chosen positions.*
$C_{bucket} := \{\ C_{AAAA}, C_{AAAT}, \ldots, C_{TTTT}\ \}$
This given C_{bucket} *is the list of buckets after projecting along 4 randomly chosen position.*

Definition 6. S_{bucket} *is the list of those buckets which are obtained after projecting all* $l - mers$ *of the given input sequences set along* k *randomly chosen positions.*
$S_{bucket} := \{\ S_{AAAA}, S_{AAAT}, \ldots, S_{TTTT}\ \}$
This given S_{bucket} *is the list of buckets after projecting along 4 randomly chosen position.*

Definition 7. S_x *is correspondent bucket of* C_y *, where* $x = y$. *For example,* S_{ATTG} *is the correspondence bucket of* C_{ATTG} .

Definition 8. L_x *is the list of all* $l - mers$ *in* C_x .

3.2 Proposed Algorithm

The idea of our proposed method is based on following observations.

Observation 1: $B_{d(x)}$ for all $l - mer$ x in S_1 of the given sequence set $S = \{S_1, S_2, \ldots, S_n\}$ contains all possible motifs for the set S as well as some more $l - mers$ which are not motif for S.

Observation 2: Bucketing all the $l - mers$ of the given set S along k randomly chosen position, for $k < l - d$, will cluster all those $l - mers$ which have no mismatch

in those k positions in the same bucket.

From observation 1, we realize that we can build a candidate motif set for a given input set S which contains all $l-mer$ neighborhoods of first sequence S_1. Observation 2 states that, using projection along randomly chosen k position will give us the clusters of the input sets $l-mers$.

From the observations, we have proposed a new method of finding DNA planted motif which first generates all candidate motif set and then extract original motifs. Steps of our proposed method are shown in Figure 11 using flowchart.

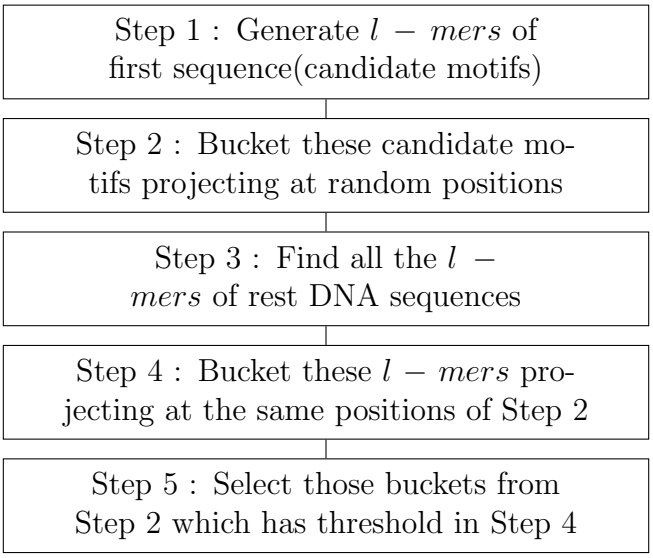


Figure 6: Steps of the proposed method

$Step-1$: Generate all $l-mers$ from first input sequence. The planted motifs are present in this $l-mers$ set. Because any planted motif must be present in first sequence with at most d mismatches. Generated $l-mers$ set from first sequence ensures that the output motifs are present in the set and termed as candidate motif's set.

$Step-2$: Cluster the $l-mers$ from Step-1 in buckets using random projection along k randomly chosen position where $k < l-d$. Then we will get some buckets of all $l-mers$ of candidate motif set, i.e, C_{bucket}.

$Step-3$: Generate a set of all $l-mers$ of all given sequences except the first sequence. These $l-mers$ are the subsequences of the input sequences of length l and their neighbors having maximum d mismatches.

$Step-4$: Project all the $l-mers$ from Step-3 along same k positions which positions were selected for bucketing in Step-2. Thus we will get some buckets of $l-mers$ of input sequences, i.e., S_{bucket} .

$Step-5$: Output all the $l-mers$ contained in bucket C_{bucket} where its correspondent S_{bucket} has size greater than threshold. The $l-mers$ in those C_{bucket} are the output planted motifs of length l having at most d mismatches of the given input

sequences.

$$Motif := \cup_{\forall x} L_x \text{ where size of } S_x \geq S_{threshold}$$

We need to define the threshold value for the buckets of the input sets. We set this value as the average bucket size, i.e.

$$S_{threshold} = \frac{n(m-l+1)}{4^k}$$
where, n = Total number of sequences in S
m = length of each sequence (S_i)
k = randomly chosen positions

The pseudo code of our proposed Algorithm is given as follows.

Algorithm 1 Proposed Algorithm for planted motif search

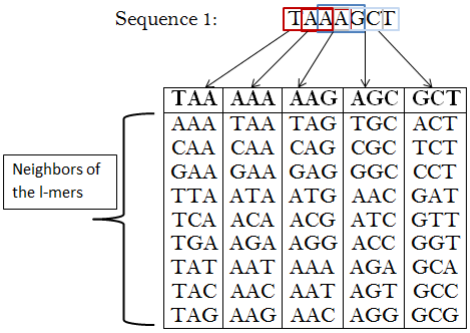
```
1: Q ← φ
2: for each l − mer x of S_1
3:     Compute B_d(x)
4:     Q ← Q ∪B_d(x)
5: end for
6: k = l − d − 1
7: Motif ← φ
8: for 1 to m
9:     generate k random number R_1, R_2, ..., R_k where R_1 ≠ R_2 ≠ ... ≠ R_k
10:    for each l − mer y in Q
11:        String sbucket = y[R_1] + y[R_2]+ ... + y[R_k]
12:        add y into bucket S_sbucket
13:    end for
14:    P ← φ
15:    for 1 to 4^k
16:        if size of S_z ≥ S_threshold then
17:            P ← P ∪ l − mers of C_z
18:        end if
19:    end for
20:    if Motif ← φ then
21:        Motif ← P
22:    else
23:        Motif ← Motif ∩ P
24:    end if
25:
26: end for
27:
```

Example:

A sample data set is given below on which we will explain our proposed method.

Sequence 1: TAAAGCT
Sequence 2: CAAATCT
Sequence 3: TACAGAT
Sequence 4: TAATGCT

1. All the $l-mers$ and their neighbors for $(3,1)$ motif are generated from first sequence.



TAA	AAA	AAG	AGC	GCT
AAA	TAA	TAG	TGC	ACT
CAA	CAA	CAG	CGC	TCT
GAA	GAA	GAG	GGC	CCT
TTA	ATA	ATG	AAC	GAT
TCA	ACA	ACG	ATC	GTT
TGA	AGA	AGG	ACC	GGT
TAT	AAT	AAA	AGA	GCA
TAC	AAC	AAT	AGT	GCC
TAG	AAG	AAC	AGG	GCG

Figure 7: $l-mers$ and their neighbors from Sequence 1

2. Bucket all these neighbors and $l-mers$ hashing in the position 1 and 2.

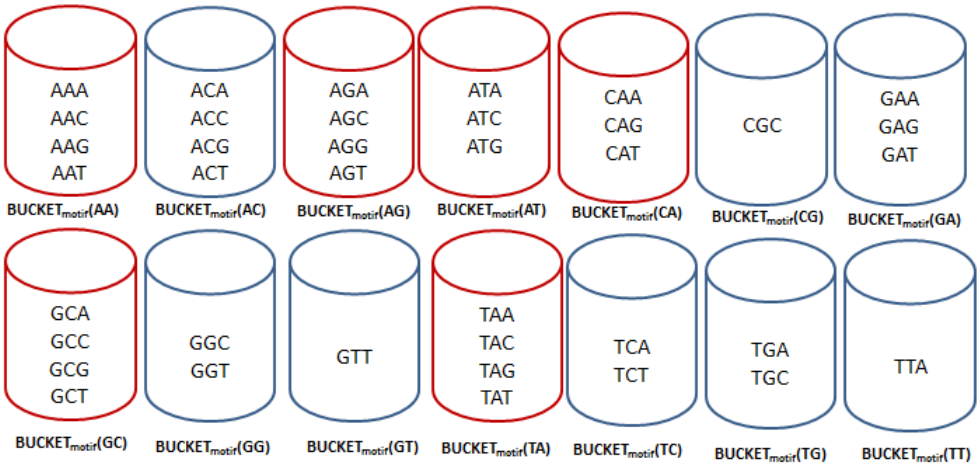


Figure 8: Bucketing $l-mers$ and their neighbors from Sequence 1

3. Generating all the $l - mers$ from all the sequences from the data set.

Sequence 1	TAA, AAA, AAG, AGC, GCT
Sequence 2	CAA, AAA, AAT, ATC, TCT
Sequence 3	TAC, ACA, CAG, AGA, GAT
Sequence 4	TAA, AAT, ATG, TGC, GCT

Figure 9: Generating $l - mers$ from all Sequences

4. Bucketing $l - mers$ of the data set in the same position used in Step-2, i.e, position 1 and 2

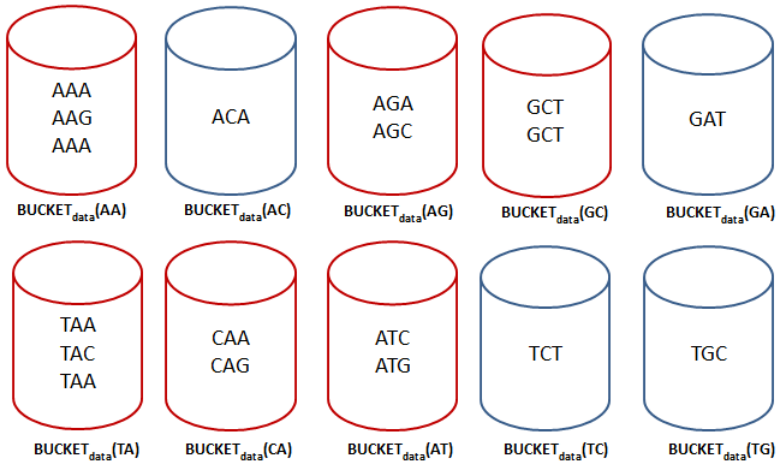


Figure 10: Generating $l - mers$ from all Sequences

5. Now we will calculate the threshold value for the buckets. Here,

$$\text{Total number of } l - mers \text{ from data set} = (7 - 3 + 1) * 4 = 20$$
$$\text{Number of buckets for data set} = 10$$
$$\text{Threshold} = \frac{20}{10} = 2$$

Now, if $BUCKET_{data}(X)$ contains more or equal number of $l - mers$ than threshold, then we will choose the bucket $BUCKET_{motif}(X)$ and add all the $l - mers$ present in the bucket $BUCKET_{motif}(X)$ to the list of the ouput motif. So the output motif list will be,

AAA, CAG, AAC , ATG, AAG , TAG, AAT , TAT, AGA, AGG, AGC, AGT, GCT , GCA, TAA , GCC, TAC , GCG, CAA , CAT, ATC , ATA.

3.3 Improvement of the Proposed Algorithm

Our proposed method works fine for finding motifs of shorter length. For finding motifs of larger length we have improved our proposed method which can find motifs of larger length using our proposed method.

Observation 3: For finding (l, d) motif , first find all the $(\frac{l}{2}, d)$ motifs. Then find two consecutive motif positioned in distance of $\frac{l}{2}$ such that merging these two motif will generate a string of length l and at last checking validity if it can be (l, d) motif.

From Observation 3, we get the idea of merging all the found motifs applying $Algorithm1$ and find motifs for larger length. We will apply $Algorithm1$ for different instances and merge them for finding expected motifs.

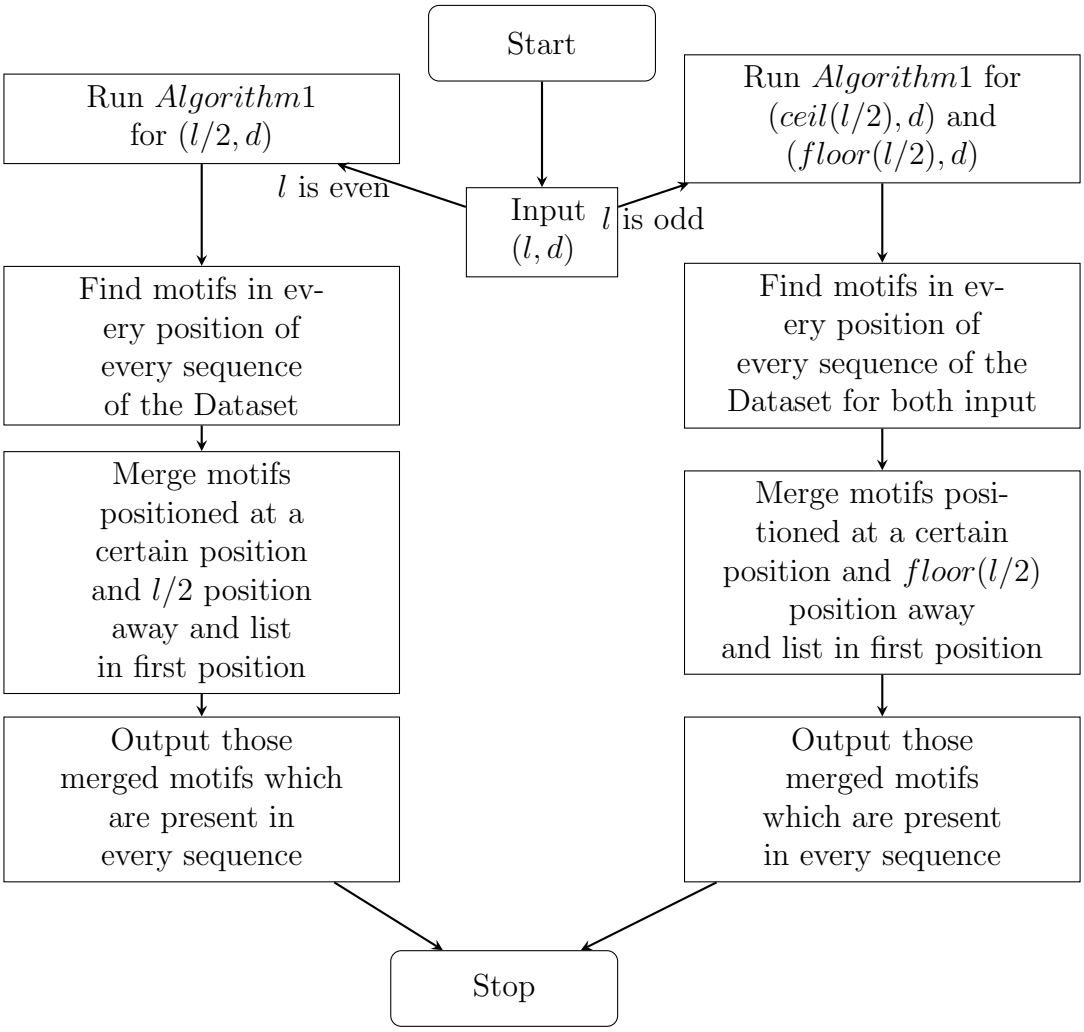


Figure 11: Steps of the proposed method 2

In the Figure 11, the steps of the improved version of our proposed method is shown in a flow chart. We will use our proposed algorithm differently depending on

the value of length of the motif. We will use this approach for finding large length motifs. If the value of the length of motif is even then we will apply our proposed method only once for half of the length. If it is odd then we will apply it two times for two different values of length as shown in the flow chart. Then we will find motifs for every position in the data set. We and merge the motifs in any position with the motifs positioned in half of the desired motif length away. Thus we will get a new data set where every position contains the motif of expected length. Now all those $l-mers$ which are present in every sequence will be our desired motifs of larger length.

The pseudo code of our improved version of the Proposed Algorithm is given as follows.

Algorithm 2 Improved version of Proposed Algorithm for planted motif search

```
1:  input l and d
2:
3:  if l is even then:
4:      Run Algorithm 1 for (l/2, d)
5:      for i = 1 to n
6:          for j = 1 to m
7:              find all the motifs in position j and insert in list L(i, j)
8:      for i = 1 to n
9:          for j = 1 to m
10:             L(i, j) := L(i, j) Cartesian product L(i, j + l/2)
11:
12: else :
13:     Run Algorithm 1 for (⌈l/2⌉, d) output motif in L_1
14:     for i = 1 to n
15:         for j = 1 to m
16:             find all the motifs in position j and insert in list L_1(i, j)
17:     Run Algorithm 1 for (⌊l/2⌋, d) output motif in L_2
18:     for i = 1 to n
19:         for j = 1 to m
20:             find all the motifs in position j and insert in list L_2(i, j)
21:             L(i, j) := L_1(i, j) Cartesian product L_2(i, j + l/2)
22:
23: Output := Common elements in the lists L(i)
24:
```

Explanation with Example:

For the data set provided for the explanation of our proposed algorithm, if we want to find $(6, 1)$ motifs we will first find all the motifs of $(3, 1)$ like we found using our proposed method. Then we will follow the following steps.

1. We have to find all the motifs in every positions in the data set.

Positions	1	2	3	4	5
Sequence 1	AAA, TAA, TAC, CAA	AAA, AAG, ...	....	AGC AGA	GCT .....
Sequence 2	AAA CAA CAG	AAA AAG	....	ATC ATG	GCT ....
Sequence 3	TAA TAC	AAA	.....	AAC ATC	GCT .....
Sequence 4	TAA TAC	AAA AAT	.....	TAC AGC	GCT GCA .....

Figure 12: Finding motifs present in every position of the data set

2. We will merger the motifs of any position with the motifs positions in 3 position away and store the motifs in the first position.

Positions	1	2
Sequence 1	TAAAGC TAAAGA TAAATC	AAAGCT .....
Sequence 2	AAAATC	AAAGCT .....
Sequence 3	TAAATC	AAAGCT .....
Sequence 4	TAAAGC	AAAGCT .....

Figure 13: Merging the motifs for getting larger length motif

3. We will find those motifs which are present in all the sequences and has at most 1 mismatch as we are expected. Thus we can see that, "TAAATC" and "AAAGCT" can be our motif of $(6, 1)$ of the given data set.

Chapter : Experimental Results & Discussion

In this chapter we discuss about the data sets which are used and compare our proposed method with different methods which are already existed. We also show results for different types of data sets.

4.1 Dataset

For time comparison we have used the dataset used in PMS5 [15]. This dataset contains 20 DNA sequences each having 600 nucleotides, i.e, length of each string representing the DNA sequences is 600. This dataset contains at least one motif in every sequences and there can be more.

For executing out method for different instances we have picked the dataset from [31], where there are 52 datasets - 6 from fly, 26 from human, 12 from mouse and 8 from yeast.Number of sequences in the datasets vary from 5 to 35 and length of each sequences vary from 1000 to 2500. All the sequences in the dataset is in FASTA format.

For comparing the accuracy of our method with other approximate algorithms we have used the real data sets from yeast, fly from [26]. From this data set we can see how much accurately our method can find motifs comparing other approximate methods.

4.2 Result Analysis

We have compared our method with other existing approach considering three important aspects - Time, Accuracy and Different instances. In the time comparison we can found how much time our method takes for finding motifs. We have calculated the result for different types of instances so that we can confirm our method works for all instances and the result is shown in the respective section.

4.2.1 Time Comparison

In Table 2, the time comparison of our previous and Developed proposed method with PMS5 [15] is shown. Same data set is used on the same machine for different combination of length and mismatches. From Table 2, we can see that for up to large length like 8, our algorithm outperforms than PMS5 [15] algorithm. Even for more large length like 20 still, our proposed method performs better than PMS5 [15]. Our previous method can calculate motif upto length 10 with 3 mismatches where our modified new method can calculate motif of length 20 with 6 mismatches.Our Developed Proposed Method can find up to 30 length motif with 10 mismatches costing huge memory and time more than 40 hours theoretically.

Table 2: Comparison with PMS5[7] with Proposed Method

(l,d)	**PMS5**	**Algorithm 1**	**Algorithm 2**
(5,2)	35 s	0.141 s	0.13 s
(6,2)	26 s	0.30 s	0.33 s
(7,2)	21s	0.940 s	1.14s
(7,3)	271 s	2.053 s	1.39s
(8,3)	138 s	8.396 s	14.06s
(9,3)	196 s	41.937 s	25.3s
(10,3)	168 s	63.97 s	36.87s
(12,4)	180 s	-	154.24s
(14,6)	3.8m	-	186.2s
(18,6)	1.2 h	-	1.05h
(20,6)	8.6 h	-	6.8h

4.2.2 Accuracy

We have compared our method with different approximate methods on the base of number of motif found. From Table 3, we can see that our method can find about 80% of the motifs of the data set that is used to calculate the accuracy of other approximate algorihms shown in the table. In this case, our method can find more amount of motif than other appoximate algorithms.

Table 3: Comparison of number of discovered motifs of our proposed method and other approximate algorithms on dataset [26]

Algorithms	Found Motif ((%)
PhyloCon[23]	12
PhyME [26]	13
MEME:ZOOPS[24]	25
PhyloGibbs[32]	35
Kellis et al.[33]	36
Converge[34]	44
MEME:OOPS- DC[29]	47
PRIORITY- DC[28]	49
MEME:ZOOP- DC[29]	52
RPB-DC[30]	57
Proposed Method	**80**

4.2.3 Different Instances

In Table 4, different data set from [31] and different value of (l, d) are used to more analysis the performance of the proposed method. We experimented our method on different data sets for ensuring that our method works on different instances and data sets. From this table, we see that our method works very close to exact approach when the data set length is high and number of sequences is also large. And for finding different length motif our method take reasonable amount of time.

Table 4: Experimental results on data set [31]

Dataset	Number of Sequences	Length of Each Sequences	(l,d)	Time (s)	Found Motif (%)
dm01r	5	1500	(6,2)	1.81	87
			(7,2)	5.804	55
			(8,3)	39.55	58
dm05r	3	2500	(6,2)	2.574	99
			(7,2)	7.269	97
			(8,3)	51.537	94
hm01r	18	2000	(6,2)	2.73	93
			(7,2)	7.708	88
			(8,3)	50.135	85
hm03r	10	1500	(6,2)	1.998	74
			(7,2)	6.302	63
			(8,3)	41.565	60
hm20r	35	2000	(6,2)	3.323	98.8
			(7,2)	9.84	98.14
			(8,3)	52.089	97.3
mus04r	6	1000	(6,2)	1.4	76.4
			(7,2)	5.08	60.3
			(8,3)	35.45	60
mus10r	13	1000	(6,2)	1.42	78
			(7,2)	5.383	68
			(8,3)	35.864	60

Chapter : Conclusion and Future Work

In this chapter the total overview of our proposed method is described. Limitations and future works is also discussed.

5.1 Summary of works

In this thesis, we have proposed an approximate method for Planted Motif Search (PMS) problem which is efficient in consideration of finding more number of motifs than other approximate algorithms.It also performs good in terms of time than exact algorithm. Our Proposed method first finds out all the motifs than has a chance to be motif and then it removes the initially considered motifs that are not motif by refining the candidate motifs following the process of hashing. Later, we proposed an improved version of our proposed method which can find larger length motifs. Our proposed finds around 80% of the motifs within less amount of time. As we have generated the candidate motifs, it takes a little bit more time for processing. We faced difficulties in memory consumptions as there was need for generating intermediary data for further processing and it takes relatively more amount of memory. Our method could work more efficiently if we could reduce the memory consumption. After all, our proposed method works relatively good in term of finding more number of motifs and for some cases where the data set contains DNA sequences of large length, it can find motifs close to 100%. As a result, we hope this algorithm will ensure better motif finding approach of a given DNA sequence within a reasonable time to help biologists.

5.2 Scope of Future works

Though our algorithm works well, still it consumes a lot of memory as the size of data set and the intermediate data generation are huge in number. There are scope for future works on our proposed method which includes reducing memory consumption, reducing time for calculation and finding more number of motifs for all instances.

References

[1] A. M. Näär, J.-M. Boutin, S. M. Lipkin, C. Y. Victor, J. M. Holloway, C. K. Glass, and M. G. Rosenfeld, "The orientation and spacing of core dna-binding motifs dictate selective transcriptional responses to three nuclear receptors," *Cell*, vol. 65, no. 7, pp. 1267–1279, 1991.

[2] C. B. Akgül, D. L. Rubin, S. Napel, C. F. Beaulieu, H. Greenspan, and B. Acar, "Content-based image retrieval in radiology: current status and future directions," *Journal of Digital Imaging*, vol. 24, no. 2, pp. 208–222, 2011.

[3] J. Wood, T. Andersson, A. Bachem, C. Best, F. Genova, D. R. Lopez, W. Los, M. Marinucci, L. Romary, H. Van de Sompel, *et al.*, "Riding the wave: How europe can gain from the rising tide of scientific data," *European Union*, 2010.

[4] S. Rajasekaran, S. Balla, C.-H. Huang, V. Thapar, M. R. Gryk, M. W. Maciejewski, and M. R. Schiller, "Exact algorithms for motif search.," in *APBC*, pp. 239–248, 2005.

[5] S. Pal and S. Rajasekaran, "Improved algorithms for finding edit distance based motifs," in *Bioinformatics and Biomedicine (BIBM), 2015 IEEE International Conference on*, pp. 537–542, IEEE, 2015.

[6] H. M. Martinez, "An efficient method for finding repeats in molecular sequences," *Nucleic acids research*, vol. 11, no. 13, pp. 4629–4634, 1983.

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

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

[9] S. Rajasekaran and H. Dinh, "A speedup technique for (l, d)-motif finding algorithms," *BMC research notes*, vol. 4, no. 1, p. 1, 2011.

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

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

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

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

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

[15] H. Dinh, S. Rajasekaran, and V. K. Kundeti, "Pms5: an efficient exact algorithm for the (, d)-motif finding problem," *BMC bioinformatics*, vol. 12, no. 1, p. 410, 2011.

[16] S. Bandyopadhyay, S. Sahni, and S. Rajasekaran, "Pms6: A fast algorithm for motif discovery," *International Journal of Bioinformatics Research and Applications 2*, vol. 10, no. 4-5, pp. 369–383, 2014.

[17] H. Dinh, S. Rajasekaran, and J. Davila, "Qpms7: a fast algorithm for finding (, d)-motifs in dna and protein sequences," *PloS one*, vol. 7, no. 7, p. e41425, 2012.

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

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

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

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

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

[23] T. Wang and G. D. Stormo, "Combining phylogenetic data with co-regulated genes to identify regulatory motifs," *Bioinformatics*, vol. 19, no. 18, pp. 2369–2380, 2003.

[24] T. L. Bailey and C. Elkan, "The value of prior knowledge in discovering motifs with meme.," in *Ismb*, vol. 3, pp. 21–29, 1995.

[25] E. Eskin and P. A. Pevzner, "Finding composite regulatory patterns in dna sequences," *Bioinformatics*, vol. 18, no. suppl 1, pp. S354–S363, 2002.

[26] S. Sinha, M. Blanchette, and M. Tompa, "Phyme: a probabilistic algorithm for finding motifs in sets of orthologous sequences," *BMC bioinformatics*, vol. 5, no. 1, p. 1, 2004.

[27] P. Beaudoin, S. Coros, M. van de Panne, and P. Poulin, "Motion-motif graphs," in *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 117–126, Eurographics Association, 2008.

[28] R. Gordaˆn, L. Narlikar, and A. J. Hartemink, "Finding regulatory dna motifs using alignment-free evolutionary conservation information," *Nucleic Acids Research*, vol. 38, no. 6, pp. e90–e90, 2010.

[29] T. L. Bailey, M. Bodén, T. Whitington, and P. Machanick, "The value of position-specific priors in motif discovery using meme," *BMC bioinformatics*, vol. 11, no. 1, p. 1, 2010.

[30] M. Galib, N. Hasan, M. A. Rahman, and M. A. Mottalib, "Rpb-dc: Motif discovery using randomly projected bucketing (rpb) and discriminative conservation (dc) based prior," *icita.org*, vol. 15, 2015.

[31] M. Tompa, N. Li, T. L. Bailey, G. M. Church, B. De Moor, E. Eskin, A. V. Favorov, M. C. Frith, Y. Fu, W. J. Kent, *et al.*, "Assessing computational tools for the discovery of transcription factor binding sites," *Nature biotechnology*, vol. 23, no. 1, pp. 137–144, 2005.

[32] R. Siddharthan, E. D. Siggia, and E. Van Nimwegen, "Phylogibbs: a gibbs sampling motif finder that incorporates phylogeny," *PLoS Comput Biol*, vol. 1, no. 7, p. e67, 2005.

[33] M. Kellis, N. Patterson, M. Endrizzi, B. Birren, and E. S. Lander, "Sequencing and comparison of yeast species to identify genes and regulatory elements," *Nature*, vol. 423, no. 6937, pp. 241–254, 2003.

[34] K. D. MacIsaac, T. Wang, D. B. Gordon, D. K. Gifford, G. D. Stormo, and E. Fraenkel, "An improved map of conserved regulatory sites for saccharomyces cerevisiae," *BMC bioinformatics*, vol. 7, no. 1, p. 1, 2006.