Finding Planted Motif in Biological sequence Using ATGC Counter Map Algorithm

by

S.M. Iqbal Morshed
Exam Roll: Curzon Hall–530
Registration No: H–894
Ashraful Hasan Nazir
Exam Roll: Curzon Hall–517
Registration No: H–1404
4^{th} year(Honors)
Session: 2008–2009

A Project submitted in partial fulfilment of the requirements for the degree of Bachelor of Science in Computer Science and Engineering

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
UNIVERSITY OF DHAKA

July 2013

Declaration

We, hereby, declare that the work presented in this project is the outcome of the investigation performed by me under the supervision of Dr. S. M. Tareeq, Associate Professor, Department of Computer Science and engineering, University of Dhaka. We also declare that no part of this project has been or is being submitted elsewhere for the award of any degree or diploma.

Countersigned

.....

(Dr. S. M. Tareeq)

Supervisor

Signature

.....

(S. M. Iqbal Morshed)

.....

(Ashraful Hasan Nazir)

Abstract

A Motif is a conserved pattern thought to exist in several biological sequences e.g. DNA, RNA or Proteins. Motif discovery is an important problem in biology. For example, it is useful in the detection of transcription factor binding sites and transcriptional regulatory elements that are very crucial in understanding gene function, human disease, drug design, etc. As finding motif is one of the most important and challenging tasks in computational biology several different algorithms have been proposed in the literature. In this paper we introduce our novel and efficient ACM (ATGC Counter Map) algorithm to solve motif finding problem using a new datastructure named ATGC Counter Map. Experimental results shows that our algorithm outperforms existing algorithm by solving instances that previous algorithms failed to solve and reducing run time for large motif sequences.

Acknowledgements

First of all, we are thankful and expressing our gratefulness to Almighty Allah who offers us His divine blessings, patient, mental and psychical strength to complete this project work.

We are deeply indebted to our project supervisor Dr. S. M. Tareeq, Associate Professor, Department of Computer Science and Engineering, University of Dhaka. Her scholarly guidance, important suggestions, endless patience, constant supervision, valuable criticism, and enormous amount of work for going through our drafts and correcting them, and generating courage from the beginning to the end of the research work has made the completion of the project possible.

Last but not the least; we are highly grateful to our parents and family members for their support and constant encouragement, which have always been a source of great inspiration for us.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

As the pace of large-scale DNA sequencing outstrips the rate at which genomic sequence can be analyzed in the laboratory, it falls increasingly to computational techniques to digest and analyze these large datasets of sequenced DNA. One analysis, which is frequently required, is the discovery of repetitive patterns, called motifs, in the sequence data. These sets of similar subsequences tend to indicate regions that have the same, or similar, biological function, and therefore hint at the purpose and structure of uncharacterized sequence. This work treats the problem of discovering a certain type of motif.

In section 1.2 we define our thesis problem. In section 1.3 we discuss why we choose that problem. We clarify our objectives in section 1.4.We point out our contribution in section 1.5. Finally, we discuss about the layout of the thesis in section 1.6.

1.2 Problem Definition

In this thesis we provide an algorithm for PMS(Planted Motif Search) problem. PMS is stated as follows. n sequences and two integers l and d are provided as input. For simplicity, we assume that the length of each sequence is m. The problem is to identify all strings of M of length l such that M occurs in each of the n sequences with at most d mismatches. Formally, string M has to satisfy the following constraint: there exists a string M_i of length l in sequene i, for every $i(1 \leq i \leq n)$, such that the number of mismatches between M and M_i is less than or equal to d. String M is called a motif. For example, if the input sequences are GCGCGAT, CAGGTGA and CGATGCC; $l = 3$ and $d = 1$, then GAT and GTG are some of the motifs.

1.3 Motivation towards the Work

We choose this problem for several reasons.

- Motif finding helps us to solve many biological problem.For instance, identification of patterns in nucleic acid sequences has helped in determining open reading frames, identifying promoter elements of genes, identifying intron/exon splicing sites, locating RNA degradation signals etc. In case of protein sequence, pattern identification helped in domain identification, protease cleavage site location, signal peptide identification, protein degradation element identification, short functional motif discovery etc.

- Motif finding can help in drug design. Many drugs are designed by analyzing the DNA of the bacteria. Motif finding can help understand certain traits of gene by finding certain pattern

- One of the most important reason for dealing with motif finding is- it is an NP-complete problem. There is no polynomial time solution to this problem.

So, scientists are coming up with different kinds of algorithm to solve the problem as efficiently as possible.

1.4 Objective of the Research

The main aims behind this research work are:

1. **Understanding previous works on motif finding:** The motif search problem has attracted many researchers over last two decades. In literatures, many versions of the motif finding problem have been enumerated. Understanding those literature is one of the objective of the research.

2. **Provide a new data structure/algorithm:** After studying previous literature we try to provide novel data structure/algorithm to solve motif finding problem.

3. **Provide an efficient solution algorithm to outperform other algorithms:** We try to provide efficient solution to the motif finding problem which we hope to outperform existing algorithms. Our objective is also to solve the motif finding problem for challenging instances where number of mismatches are greater than 4.

1.5 Our Contribution

1. We provide a novel algorithm named ACM(ATGC Counter Map) to solve planted motif finding problem.

2. Our algorithm outperforms MITRA, PMS2 and previous algorithms in terms of runtime where l greater than 12

3. Our algorithm solves (19,5) and (21,5) instances which MITRA, PMS2 and previous algorithm than these failed to solve.

1.6 Layout of the Thesis

The research work is carried out in order to achieve the objectives just described in several stages as chapters. A brief overview of the contents of the chapters is provided as follows:

Chapter 2 describes the literature Review.

Chapter 3 describes our novel algorithm ACM.

Chapter 4 discusses the result achieved from the implementation of our algorithm.

Chapter 5 provides the concluding remarks

Chapter 2

Literature Review

2.1 Overview

The problem of identifying meaningful patterns (i.e., motifs) from biological data has been studied extensively due to its paramount importance. In literatures, many versions of the motif finding problem and many different algorithms to solve them have been enumerated. In this chapter we review the previous literature relevant to our thesis work. We also show our relevance with those literature.

In section 2.2 we discuss different types of motif finding problem. After that in section 2.3 we discuss different types of algorithm to solve a specific type of motif finding problem named PMS(Planted Motif Search) problem. Finally in section 2.4 we discuss three previous algorithm relevant to our research.

2.2 Different Types of Motif Finding Problem

Motif search is an important problem in biology. This problem in general requires finding short patterns of interest from voluminous data. Several variants of this motif search problem have been identified in the literature. In this chapter we are interested in the following three versions.

2.2.1 Problem Definitions

Problem 1. Input are n sequences of length m each. Input 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. (This problem is also known as the *planted (l, d)-motif search problem.*)

Problem 2. The input is a database DB of sequences $S_1, S_2, ..., S_n$. Input also are integers l, d, and q. Output should be all the patterns in DB such that each pattern is of length l and it occurs in at least q of the n sequences. A pattern U is considered an occurrence of another pattern V as long as the edit distance between U and V is at most d. We refer to this problem as the *edited motif search problem.*

Problem 3. A pattern is a string of symbols (also called residues) and ?'s. A "?" refers to a wild card character. A pattern cannot begin or end with ?. AB?D, EB??DS?R, etc. are examples of patterns. The length of a pattern is the number of characters in it (including the wildcard characters). This problem takes as input a database DB of sequences. The goal is to identify all the patterns of length at most l (with anywhere from 0 to $\lfloor l/2 \rfloor$ wild card characters). In particular, the output should be all the patterns together with a count of how many times each pattern occurs. Optionally a threshold value for the number of occurrences could be supplied. We refer to this problem as the simple motifs search problem.

2.2.2 Our chosen problem:PMS

Among the above mentioned problem category, we choose *problem 1* as our thesis problem. It is called Planted Motif Search problem or PMS in short.

2.3 Different Types of Algorithm to solve PMS

Though the planted motif search problem (Problem 1) is defined for arbitrary sequences, in the literature it is usually assumed that the input consists of DNA sequences (and hence the alphabet size is 4). We also make this assumption. Numerous papers have been written in the past on the topic of Problem 1

2.3.1 Categorization on the basic approach applied

2.3.1.1 Pattern based approach

pattern-based algorithms predict the motif (as a sequence of residues) itself. Several pattern based algorithms are known. Examples include PROJECTION [2], MULTIPROFILER [5], MITRA [3], and PatternBranching [9]. PatternBranching (due to Price, Ramabhadran and Pevzner [9]) starts with random seed strings and performs local searches starting from these seeds.

2.3.1.2 Profile based approach

Profile based algorithms predict the starting positions of the occurrences of the motif in each sequence. Examples of profile-based algorithms include CONSENSUS [4], GibbsDNA [7], MEME [1], and ProfileBranching [9]. The performance of profile-based algorithms are 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. For the (15, 4) challenge problem, these algorithms have the following performance coefficients (respectively): 0.2, 0.32, 0.14, and 0.57. The run times of these algorithms for this instance are (respectively, in seconds): 40, 40, 5, and 80.

2.3.2 Categorization on the exactness

2.3.2.1 Approximation Algorithm

Algorithms that may not output the correct planted motif always are refereed to as Approximation algorithm. Examples include Bailey and Elkan [1], Lawrence et al. [7], Rocke and Tompa [11]. These algorithms employ local search techniques such as Gibbs sampling, expectation optimization, etc.More algorithms have been proposed for Problem 1 by the following authors: Pevzner and Sze [8], Buhler and Tompa [2]. The algorithm in [8] is based on finding cliques in a graph and the algorithm of [2] employs random projections. These algorithms have been experimentally demonstrated to perform well. These are approximation algorithms as well.

2.3.2.2 Exact Algorithm

Algorithms that always output the correct answer are referred to as exact algorithms.Many exact algorithms are known. However,as pointed out in [2], these algorithms "become impractical for the sizes involved in the challenge problem". Exceptions are the MITRA algorithm [3] and the PMS algorithms of Rajasekaran, Balla and Huang [10]. These algorithms are pattern-based and are exact. MITRA solves for example the (15, 4) instance in 5 minutes using 100 MB of memory [3]. This algorithm is based on the WINNOWER algorithm [8] and uses pairwise similarity information. A new pruning technique enables MITRA to be more efficient than WINNOWER. MITRA uses a mismatch tree data structure and splits the space of all possible patterns into disjoint subspaces that start with a given prefix. The same (15, 4) instance is solved in 3.5 minutes by PMS [10].

2.3.3 Comparison among different types of algorithm

A profile-based algorithm could either be approximate or exact. Likewise a pattern-based algorithm may either be exact or approximate. Algorithms that are exact

are also known as exhaustive enumeration algorithms in the literature. It is noteworthy here that the profile-based algorithms such as CONSENSUS, GibbsDNA, MEME, and ProfileBranching take much less time for the (15, 4) instance [25]. However these algorithms fall under the approximate category and may not always output the correct answer. Some of the pattern-based algorithms (such as PROJECTION,MULTIPROFILER, and PatternBranching) also take much less time [9]. However these are approximate as well (though the success rates are close to 100

2.3.4 Type of our proposed ACM algorithm

Among the above mentioned category in section 2.3, our algorithm falls under the category of **Pattern based approach**. Because it outputs the motif, not just the starting position of Motif l-mers . Our algorithm is also **exact algorithm**. Because it outputs the exact motif. It's output is not based on approximation.

2.4 Important Motif Finding Algorithm relevant to our research

Part of our ACM algorithm is to create graph to find clique. To do so, we have used techniques described in MITRA. So in section 2.4.0.2 we briefly discuss MITRA algorithm. MITRA did improvement on the WINNOWER algorithm. So, we discuss WINNOWER algorithm too in section 2.4.0.1. In chapter we compared our algorithm with state-of-art PMS2 algortihm.We discuss PMS2 algorithm in section 2.4.0.3 .

2.4.0.1 WINNOWER algorithm

The algorithm of Pevzner and Sze[8] (called WINNOWER) works as follows. If A and B are two instances (i.e., occurrences) of the motif then the hamming distance between A and B is at most 2d. In fact the expected hamming distance between

A and B is $2d - \frac{4d^2}{3l}$. algorithm constructs a collection C of all possible l-mers in the input. A Graph G(V,E) is then constructed. Each l-mer in C will correspond to a node in G. Two nodes u and v in G are connected by an edge if and only if the hamming distance between the two l-mers is at most 2d and these l-mers come from two different sequences.

Clearly, the n instances of the motif M form a clique of size n in G. Thus the problem of finding M reduces to that of finding large cliques in G. Unfortunately, there will be numerous 'spurious' edges (i.e., edges that do not connect instances of M) in G and also finding cliques is NP-hard. Pevzner and Sze [8] employ a clever technique to prune spurious edge. Pevzner and Sze [8] observe that the graph G constructed above is almost random and is multipartite. They use the notion of an extendable clique. If $Q = \{v_1, v_2, ..., v_k\}$ is any clique, node u is called a neighbor of Q if $\{v_1, v_2, ..., v_k, u\}$ is also a clique. In other words, Q can be extended to a larger clique with the inclusion of u. A clique is called extendable if it has at least one neighbor in every part of the multipartite graph G. The algorithm WINNOWER is based on the observation that every edge in a maximal n-clique belongs to at least $n2Cn - k$.

WINNOWER is an iterative algorithm where cliques of larger and larger sizes are constructed. The run time of the algorithm is $O(N^{2d+1})$ where $N = nm$. But in practice the algorithm runs much faster.

2.4.0.2 MITRA algorithm

MITRA(MIsmatch TRee Algorithm)[3] uses a mismatch tree data structure to split the space of all possible patterns into disjoint subspaces that start with a given prefix. By splitting the pattern space, MITRA keeps reducing the pattern discovery into smaller sub-problems similarly to the SPELLER algorithm (Sagot, 1998). MITRA also takes advantage of pairwise similarity between instances.

These similarities can be used to construct a graph where each vertex is an l-mer in the sample and there is an edge if the two l-mers are similar(e.g., differ in no more than 2d positions). An (l, d) k pattern will correspond to a clique of size k in this graph. This type of approach is the basis of the WINNOWER algorithm. In fact, we show that we can impose stronger conditions on the graph for the existence of a pattern than simply a clique of size k.

Incorporating pairwise similarity into the sample driven approach

The key new ingredient of MITRA is an insight that the information about pairwise similarities between instances of the pattern can significantly speed up the sample-driven approach.MITRA took advantage of this information using an insight from Pevzner and Sze (2000). variant of the MITRA algorithm is called MITRA-Graph.

MITRA constructs a graph that models this pairwise similarity.Given a pattern P and a sample S it constructs a graph G(P, S) where each vertex is an l-mer in the sample and there is an edge connecting two l-mer if P is within d mismatches from both l-mers . For an (l, d) k pattern P the corresponding graph contains a clique of size k.Given a set of patterns ρ and a sample S,define a graph G(ρ, S) whose edge set is a union of edge sets of graphs G(P, S) for $P \epsilon \rho$.Each vertex of G(P, S) is an l-mer in the sample and there is an edge connecting two l-mers if there is a pattern $P \epsilon \rho$ that is within d mismatches from both l-mers . If for a subspace of patterns ρ, an existence of a clique of size k can be ruled out, then it can proved that the subspace is empty. To implement this idea MITRA-Graph keeps list of the edges of the graph G(P, S) at each node of the tree and efficiently updates this list while traversing the tree.

2.4.0.3 PMS2 algorithm

An improved algorithm called PMS2 is given in [10]. 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 $(l-k+1)$ k-mers for $(d \leq k \leq l)$ 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 i_j 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 $i_j + lk$. We can get an l-mer putting together these k-mers 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 $i_j, i_j+1, ..., i_j+lk$ such that all of these motifs are present in all of the input sequences (with a hamming distance of at most d). Assume that $M_{i_j+r}$is one motif of length k that starts from position $i_j + r$ of S_i that is also present in every input sequence for $(0 \leq r \leq l-k)$. If the last $k1$ residues of M_{i_j+r} are the same as the first $k1$ residues of $M_{i_j} + r + 1$ for$(0 \leq r \leq lk1)$, 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 $(d+c)$-mers 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-mers (as described above) in this phase. In the second phase we check, for each l-mer M' collected in

the first phase, if M' is a correct answer or not. Finally we output all the correct answers.

First we observe that the algorithm PMS1 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 could be run (without exceeding the core memory of the computer and within a reasonable amount of time).

Chapter 3

ACM Algorithm

In this section we describe our ACM(ATGC Counter Map) algorithm. In section 3.1, we introduce some definitions and notations to help understand our algorithm. In section 3.2 We provide in details description of our algorithm.

3.1 Notations and definitions

Definition 3.1. A string $x = x[1]...x[l]$ of length l is called an *l-mer*. Set of all l-mers from input sequence is denoted as S.

Definition 3.2. Given two strings x and y of equal length, we say the *Hamming distance* between x and y, denoted by $d_H(x, y)$ is the number of mismatches between them.

Definition 3.3. A string C_x is called *ATGC count* of an l-mer x if C_x is of the form <number of A in x>.<number of T in x >.<number of G in x >.< number of C in x >. For example, ATGC count of l-mer AATCCG is 2.1.1.2 . We also denote any ATGC Count as C instead of C_x.

Definition 3.4. Given two ATGC Count C_x and C_y where l is same, we say the *Count distance* between C_x and C_y, denoted by $d_C(C_x, C_y)$ is the total number of difference in number of A,T,G and C. For example, Let, C_x = 1.3.2.1 and C_y = 2.1.2.2, then $d_C(C_x, C_y) = (1 + 2 + 0 + 1) = 4$.

Definition 3.5. A set of ATGC Count C_{y_i} is called *Neighbor of an ATGC Count* C_x if $d_C(C_x, C_{y_i}) \leq 2d$ and C_{y_i} is greater than C_x. It is denoted by $N(C, d)$. For example- neighbor of C=1.1.2.1 for d=1 are { 1.1.3.0, 1.2.2.0, 2.1.2.0, 1.2.1.1, 2.1.1.1, 1.2.2.1 }

Definition 3.6. *ATGC Counter Map* data structure is a map data structure containing $(key, value)$ pair where *key* is ATGC count C, and value is list of l-mer which have same number A,T,G,C as C's internal '.' separated value represents.

Definition 3.7. For a key each l-mer stored in the value is called its *member*.We denote a member as μ. For example, members of C=1.1.2.1 can be patterns ATGGC, GGCAT, TACGG etc.. All patterns have same number A,T,G,C as C represents.

A member contains two list. First one is *Current Neighbor*, which contains all l-mers x such that $d_H(\mu, x) \leq 2d$ and $C_x \geq C_\mu$ where $x \epsilon S$. Second one is *Previous Neighbor*, which contains all$l$-mers x such that $d_H(\mu, x) \leq 2d$ and $C_x < C_\mu$ where $x \epsilon S$.

Definition 3.8. for a key=C, the *Neighbor of the key* are all the key which contains the value from $N(C, k)$,where $k = 1....2d$. for all *neighbor key*, we define key=C as *home key.*

Definition 3.9. For each member μ, *Member group* of μ denoted by γ contains all l-mers x such that $d_H(\mu, x) \leq 2d$ and $x \epsilon S$. Member group contains the member itself, current neighbor, prevuios neighbor of the member.

Definition 3.10. Given a set of n strings $S_1, ..., S_n$ of length m each, a string M of length l is called an (l, d)-motif of the strings if there are at least n strings such that the Hamming distance between each one of them and M is no more than d. M is called an (l, d)-motif

3.2 ATGC Counter Map Algorithm

3.2.1 Overview of the Algorithm

Our ACM algorithm uses ATGC Counter Map to store each l-mer. Then for each key in map we traverse the neighbour of the key. To do this neighbour traversing we use our novel NeighbourTravrase algorithm. By traversing all the neighbour, for each member in key, we store its member group. Then we create a graph for each member group using MITRA [3] algorithm. MITRA [3] algorithm made some improvements on WINNOWER [8] algorithm for creating such graph. Finally, we use clique finding algorithm described in [6] to find clique of size n in the graph. If a clique of size greater than or equals n is found we create profile matrix for the nodes of the clique. From profile matrix, a consensus string σ is formed. If for all nodes of clique, hamming distance between a node and consensus string $d_H(x_i, \sigma), 1 \leq i \leq n$ is not more than d, then this consensus string σ is our desired motif. Otherwise, there is no motif.

So, we describe our ACM algorithm in three parts. In section 3.2.2 we discuss about ATGC Counter Map data structure. In section 3.2.3 we discuss our NeighborTravarse algorithm and how it is used to find the neighbor of a key. We also describe how a member group of a member in the key is stored in this section. And finally in section 3.2.4 we discuss how a graph is created from member group and how clique finding algorithm is implemented on the graph to find the motif.

3.2.2 ATGC Counter Map data structure

ATGC Counter Map data structure uses the map data structure to store each l-mer in S. Map data structure stores $(key, value)$ pair where each key is unique. In ACM algorithm, we use *ATGC count* as key and corresponding list of l-mer(i.e.

members) as value. We build up our *ATGC Counter Map* by the following CreateACM algorithm.

Algorithm 1 Create ATGC Counter Map

```
1: procedure CreateACM(map M)
2:     for each l−mer ϵS do
3:         compute its ATGC count C.
4:         if C is already present in the map as a key then
5:             push l-mer at the back of the member list.
6:         else
7:             insert C in M as key
8:             push the l-mer as value of key=C
9:         end if
10:    end for
11: end procedure
```

For example, insertion of the l-mers ATTGC, GCATT, CTATT in ACM where l=5 is illustrated in figure 3.2.2.

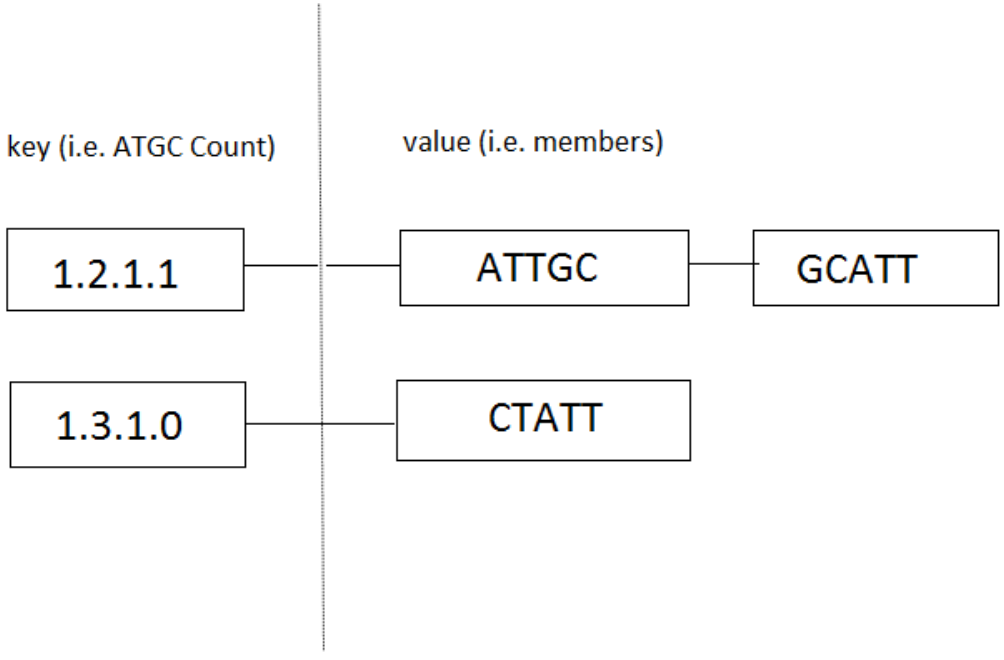

Figure 3.1: ATGC Counter Map.

Complexity of insertion is same as the complexity of inserting into a map which is $O(\log n)$ where n is size of the map. In our algorithm size of the map is total number of ATGC Countvalue.

3.2.3 Travarsing Neighbor

After building up ATGC Counter Map, for each key(i.e. ATGC Count) we traverse its neighbor. We defined neighbor of a key in definition 3.8. While traversing a neighbor, for each member $\mu_i(1 \leq i \leq p)$ in home key we compare all member $\mu_j(1 \leq j \leq q)$ in neighbor key, where p and q are number of members in home and neighbor key respectively. If $d_H(\mu_i, \mu_j)$ is no more than $2d$ then μ_i stores μ_j as its current neighbor member(Definiton 3.7) and μ_j stores μ_i as its previous neighbor member(Definition 3.7).

We optimize our algorithm by defining neighbors as those values which are greater than home key. This optimization works because, we sort the map in ascending order according to key value. So, a neighbor key in current operation will be home key in any of the next operation. So, when we traverse neighbor with greater value, we store member of the home key as the previous neighbor member of the neighbor key where conditions are fulfilled. So, home key doesnt need to traverse smaller neighbors as smaller neighbors members are already stored in previous neighbor member of the home key.

To create neighbor $N(C, d)$ by algorithm, like the example provided in definition 3.5, we have to reduce(or subtract) amount d from position i of the home key, where $2 \leq i \leq 4$ in all possible way and then distribute(or add) d in position/s j of home key,where $1 \leq j \leq 4$ and $j \neq i$, in all possible way.
We have devised a novel recursive algorithm to traverse neighbors. Psudocode of the NeighborTraverse algorithm provided below. In procedure NeighborTravarse

home C_x and ATGC Counter Map M is provided as parameter.

Algorithm 2 Travarsing neighbor key of the home key

```
1: procedure NeighborTravarse(C_x, M)
2:     z = C_x
3:     for all possible Reduction of amount from position i of z,where 2 ≤ i ≤ 4 do
4:         for all possible Distribution of amount from position j of z,where 1 ≤ j ≤ 4 and j ≠ i do
5:             compute a new ATGC Count C_y
6:             if C_y ≥ C_x and C_y is a key in M then CompareMembers(C_x, Cy)
7:             end if
8:         end for
9:     end for
10: end procedure
11: procedure CompareMembers(C_x, C_y)
12:     for all members μ_i ϵ C_x do
13:         for all members μ_j ϵ C_y do
14:             if d_H(μ_i, μ_j) ≤ 2d then
15:                 μ_i.Current Neighbor Member = μ_j
16:                 μ_j.Previous Neighbor Member = μ_i
17:             end if
18:         end for
19:     end for
20: end procedure
```

For example, to traverse the neighbor of 1.1.2.1 with $d = 1$, all possible reduction from position $i, 2 \leq i \leq 4$ would be $\{1.1.2.0, 1.1.1.1, 1.0.1.1\}$. Now for each possible reduction there are different possible distribution. For $\{1.1.2.0\}$ 3 possible distribution, preserving the condition mentioned in line 4, are $\{1.1.3.0, 1.2.2.0, 2.1.2.0\}$. Similarly, $\{1.2.1.1, 2.1.1.1, 1.1.1.2\}$ and $\{2.0.1.1, 1.0.2.1, 1.0.1.2\}$ are the possible distribution of $\{1.1.1.1\}$ and $\{1.0.1.1\}$ respectively, preserving the conditions.
So, for home key $\{1.1.2.1\}$ we get 9 neighbor key 1.1.3.0, 1.2.2.0, 2.1.2.0 1.2.1.1, 2.1.1.1, 1.1.1.2, 2.0.1.1 , 1.0.2.1, 1.0.1.2 According to the condition in line 6 we prune 3 neighbors which are less than home key. which are $\{1112, 1021, 1012\}$. So, finally remaining 6 neighbors are $\{1130, 1220, 2120, 1211, 2111, 2011\}$.

3.2.4 Graph Creation and Clique Finding

As mentioned in section 3.2.3, during the traversal of neighbor key, each member μ of the home key stores its current neighbor member such that for each l-mer in current neighbor member x_i, $d_H(x_i, \mu) \leq 2d, 1 \leq i \leq m$, where m is the size of the current neighbor member. Each member μ also has previous neighbor member which is stored during the traversal process of any previous home key smaller than current home key. For each l-mer in previous neighbor member y_i, $d_H(y_i, \mu) \leq 2d, 1 \leq i \leq n$, where n is the size of the previuos neighbor.
For each member, a member group(definition 3.9) is already formed by fulfilling current neighbor and previuos neighbor. For each member group we create a graph using WINNOWER [8] algorithm.By using WINNOWER algorithm, we construct the following graph. Each l-mer in the member group is a vertex. An edge connects two vertices if the corresponding l-mers have no more than $2d$ mismatches and they are from different sequence.

Improvements done by MITRA [3] algorithm is also incorporated in our ACM alogrithm. MITRA made improvement by removing spurios edges from the graph. At each node of the tree, it removes edges by computing the degree of each vertex. If the degree of the vertex is less than n, it can remove all edges that lead to the vertex since we know it is not part of a clique. It repeats this procedure until it cannot remove any more edges. If the number of edges remaining is less than the minimum number of edges in the clique we can rule out the existence of a clique. If number of edges remaining is greater than or equal to n we apply clique finding algorithm proposed by [6] to find clique. If a clique of size greater than or equals to n is found we create profile matirix for the nodes of the clique. From profile matrix, a consensus string σ is formed. If for all nodes(i.e. l-mer) of clique x_i , hamming distance between a node and consensus string $d_H(x_i, \sigma) \leq d$,then this consensus string σ is our desired motif M. Otherwise, there is no motif.

3.3 Summary

In brief, ATGC Counter Map algorithm separates the space of All l-mer by their ATGC Count Value. Each l-mer is stored in ATGC counter map against it's ATGC count value. When storing of all l-mer finishes, we traverse the neighbor of each key in map. By travarsing we group together all l-mers where each l-mer in a member group is at not more than 2d distance with member. From each member group we create a graph. A clique of size greater than or equal n is searched by clique finding algorithm. If clique is found,we then create its profile matrix and from profile matrix we create consensus string. If all l-mer in clique is not more than d hamming distance with consensus string, then this consensus string is our desired motif. Otherwise, there is no motif.

Chapter 4

Performance Evaluation

In this chapter we will compare performance of ACM algo-rithm with other well-known algorithms PMS2 [10], MITRA Graph [3], MITRA Count [3]. In section 4.1 we discuss on what dataset we did our experiments. In section 4.2 we discuss machine configuration we used for our experiment. In section 4.3 we show our experimental result. Finally, we put our observation in Discussion section in section 4.4

4.1 Dataset

Algorithms for motif search typically tested on random input datasets. Any such dataset will consist of 20 random strings each of length 600(n=20, m=600). We also followed that convention. We applied our Algorithm and other above mentioned algorithm on 20 random strings each of length 600.

4.2 Environment

All the experiments were conducted on a 2.5GHz Intel core i5 processor with 4GB main memory, run- ning on Microsoft Windows 7. All the programs were written in Microsoft/Visual C++6.0.

4.3 Experimental Result

Table 4.3 shows the performance of the algorithms on challenging instances. In table 4.3, the letter '-' means that corresponding algorithm either uses too much memory or takes too long on the challenging instance. In other words, the algorithm cannot solve the challenging instance in the experimental settings. In this table m represents minute/s and s represents second/s.

We see that ACM algorithm outperforms MITRA Count and MITRA graph in

Table 4.1: runtime of Algorithms on different instances

	MITRA Count	MITRA Graph	PMS2	ACM
(11,2)	1 m	1 m	.78 s	24 s
(12,3)	1 m	4 m	15.5 s	276 s
(13,3)	2 m	2 m	20.98 s	38 s
(15,4)	5 m	5 m	217 s	112 s
(17,4)	-	-	216s	29 s
(19,5)	-	-	-	82 s
(21,5)	-	-	-	8 s

all but (12,3) instance. For first three instances i.e. (11,2), (12,3), (13,3) PMS2 outperforms our ACM algorithm. But for length $l \geq 15$ ACM outperforms PMS2 by atleast factor of 2. We also see that ACM can solve instances with $d = 5$ which others fail to solve.

4.4 Discussion

1. **Observation 1:** As the length l increases ACM algorithm outperforms other algorithms.

2. **Observation 2:** While comparing ACM with other algorithms we notice an important fact that, if we increase value of l, keeping value of d same, runtime decreases atleast by the factor of 5. This is may be because, as the value of l increases, number of edges in graph created on member groups decreases. So, our clique finding algorithm can perform faster.

Chapter 5

Conclusion

5.1 Research Summary

In this research work we have devoloped a new algorithm named ATGC Counter Map algorithm. Which outperforms existing algorithm in terms of runtime for larger instances($l \geq 15$). It also solves instances where number of mismatches $d = 5$ which previous algorithms like MITRA, PMS2 failed to solve.

ATGC Counter Map algorithm uses a novel data structure named ATGC counter map which efficiently reduces the search space for neighbor of an l-mer . We also developed a novel algorithm named NeighborTravarse to travarse the neighbor of an l-mer on ATGC Counter Map. We took idea of creating graph from the previous MITRA algorithm. We used clique finding algorithm to find clique in our graph which eventually helped us finding motif.

5.2 Limitations of our work

There are some limitations of our work. Our algorithm failed to solve challenging instances like (15,5),(13,4) etc. We also see inconsistency in the runtime of

our algorithm. Our algorithms performance can decrease if first l-mer of motif (according to ATGC Value) resides in the last part of the map.

5.3 Scope of Future improvement

there are several scopes for future improvement. We can use parallel processing by dividing the task of neighbor finding to different processors. Our algorithm gets slower for clique finding algorithm we used. If we can improve clique finding algorithm we can run our algorithm more efficiently.

Appendix A

Appendix Title Here

Write your Appendix content here.

Bibliography

[1] T L Bailey and C Elkan. Fitting a mixture model by expectation maximization to discover motifs in biopolymers. *Proceedings / ... International Conference on Intelligent Systems for Molecular Biology ; ISMB. International Conference on Intelligent Systems for Molecular Biology*, 2:28–36, January 1994.

[2] Jeremy Buhler and Martin Tompa. Finding motifs using random projections. *Journal of computational biology : a journal of computational molecular cell biology*, 9(2):225–42, January 2002.

[3] Eleazar Eskin and Pavel a Pevzner. Finding composite regulatory patterns in DNA sequences. *Bioinformatics*, 18 Suppl 1(1):S354–S363, 2002.

[4] GZ Hertz and GD Stormo. Identification of consensus patterns in unaligned DNA and protein sequences: a large-deviation statistical basis for penalizing gaps. *Proceedings of the Third International Conference on ...*, (303):201–216, 1995.

[5] U Keich and P a Pevzner. Finding motifs in the twilight zone. *Bioinformatics (Oxford, England)*, 18(10):1374–81, October 2002.

[6] Janez Konc and Dusanka Janezic. An improved branch and bound algorithm for the maximum clique problem Janez Konc and Du zi. *MATCH*, 58:569–590, 2007.

[7] 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 (New York, N.Y.)*, 262(5131):208–14, October 1993.

[8] PA Pevzner and SH Sze. Combinatorial approaches to finding subtle signals in DNA sequences. *ISMB*, 2000.

[9] a. Price, S. Ramabhadran, and P. a. Pevzner. Finding subtle motifs by branching from sample strings. *Bioinformatics*, 19(Suppl 2):ii149–ii155, October 2003.

[10] S Rajasekaran, S Balla, and Chun-hsi Huang. Exact algorithms for planted motif challenge problems. *Proceedings of Asia-Pacific ...*, 2005.

[11] Emily Rocke and Martin Tompa. An algorithm for finding novel gapped motifs in dna sequences. In *Proceedings of the second annual international conference on Computational molecular biology*, RECOMB '98, pages 228–233, New York, NY, USA, 1998. ACM.