An Improved Exact Solution for Planted Motif Search Problem

A Thesis

Presented to the

Faculty of the Graduate School

Ateneo de Manila University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

By

Mark Joseph D. Ronquillo

2016

The thesis entitled:

An Improved Exact Solution for Planted Motif Search Problem

submitted by **Mark Joseph D. Ronquillo** has been examined and is recommended for oral defense.

MARLENE M. DE LEON, Ph.D. Chair	PROCESO L. FERNANDEZ, JR., Ph.D. Adviser

EVANGELINE P. BAUTISTA, Ph.D.
Dean
School of Science and Engineering

The Faculty of the Graduate School of Ateneo de Manila University accepts the THESIS entitled:

An Improved Exact Solution for Planted Motif Search Problem

submitted by **Mark Joseph D. Ronquillo** in partial fulfillment of the requirements for the Degree of Master of Science in Computer Science.

_ 1 Member	_____ 1 Member
_ 1 Member	_____ PROCESO L. FERNANDEZ, JR., Ph.D. Adviser

_____
EVANGELINE P. BAUTISTA, Ph.D.
Dean
School of Science and Engineering

Grade: **Very Good**

Date: 11 November 2013

Abstract

Finding DNA motifs is a widely studied area in the field of Computational Biology. Motifs signify different information that are useful for biologists like discovering transcription factor-binding sites and helps them identify potential therapeutic drug target. There are several variations of the motif finding problem, and one of these is called the (l, d)-motif search or Planted Motif Search problem (PMS). In this paper, we propose the EMS-GT2 algorithm, an extension of the Exact Motif Search - Generate and Test (EMS-GT) which is an exact enumerative algorithm for PMS. In EMS-GT2, we incorporated a new speedup technique that is based on an important property that we have discovered, which we prove in this paper, and which has enabled a more efficient block-processing of candidate motifs. We also improved the Hamming distance computation in EMS-GT2 by pre-computing mismatch values. We also explored a way to improve the d-neighborhood generation by focusing on non empty blocks in the candidate motifs array. Our C++ implementation of EMS-GT2 running on synthetic data for several PMS challenge instances demonstrate that it is competitive with both the EMS-GT and qPMS9, the two current best exact solutions for PMS. In particular, EMS-GT2 is able to reduce the run-times of EMS-GT by 18.33%, 25.46%, 19.88% 14.81% and 22.03% for the (l,d) challenge instances (9, 2), (11, 3), (13, 4), (15, 5) and (17, 6) respectively. It also outperforms qPMS9, having runtime reductions of 94.41%, 91.65%, 85.40%, 59.22% and 5.8% for the

(9, 2), (11, 3), (13, 4), (15, 5) and (17, 6) synthetic challenge instances respectively.

TABLE OF CONTENTS

CHAPTER

CHAPTER I

Introduction

DNA motif finding is a well studied topic in computational biology. Motif is a short patterns of interests that occurs in large amount of biological data. Detection of these motifs often leads to new biological discoveries. These may lead to finding transcription factor-binding sites that helps biologists understand gene functions. Additionally, it helps understand human diseases, identify potential therapeutic drug target and conclude commonalities from different species.

There are many variations of the motif search problem such as Simple Motif Search (SMS), Edit-distance-based Motif Search (EMS) and Planted Motif Search (PMS) also known as (l, d)-motif search. This study focuses on the exact enumerative algorithm for the PMS problem. Planted Motif Search is formally defined in [23] as *Input are t sequences of length n 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.* In solving the PMS problem, traditional string matching won't be efficient since these biological motifs are not typically exact but are subject to mutations. As a matter of fact, PMS has already been proven to be NP-hard, which means that it is very unlikely to

have an algorithm that solves it in polynomial time [12].

In this study we introduce EMS-GT2 an improved version of the EMS-GT algorithm that implements a series of speedup techniques that use the block processing, introduced in [25], in different areas of the algorithm. EMS-GT2 also improves the way it computes the hamming distance by pre-computing the non-zero pairs of bits for every integer of a given number of bits.

1.1 Context of the Study

This section formally defines the (l, d)-planted motif search problem and some related terms that are commonly used in this study.

DEFINITION 1. *l*-mer

An ***l*-mer** is a string of length l over a specified alphabet Σ. In this study, Σ = {a, c, g, t} since we are dealing with DNA sequences. Formally, an l-mer is an element of the set Σ^l.

Ex. agagt is a 5-mer and agagtca is a 7-mer.

DEFINITION 2. Hamming distance d_H

The **Hamming distance d_H** between two l-mers x and y is equal to the number of positions where the l-mers have mismatches. Formally, the hamming distance

between x and y is $d_H(x, y) = |i|x[i] \neq y[i], 1 \leq i \leq l$ where $x[i]$ is the i^th character of l-mer x.

Ex. $d_H(\underline{\texttt{a}}\texttt{ctt}\underline{\texttt{g}}\texttt{ca}, \underline{\texttt{t}}\texttt{ct}\underline{\texttt{aa}}\texttt{ca}) = 3$.

The underlined characters represents the mismatch positions.

DEFINITION 3. *d*-neighbor

An l-mer x is considered a ***d*-neighbor** of another l-mer y if the Hamming distance between the two is at most d, i.e., $d_H(x, y) \leq d$.

Ex. $\underline{\texttt{a}}\texttt{tta}\underline{\texttt{g}}\texttt{ct}$ and $\underline{\texttt{g}}\texttt{tta}\underline{\texttt{c}}\texttt{ct}$ are d-neighbors,

if $d \geq 2$.

DEFINITION 4. *d*-neighborhood of an *l*-mer *x*

The ***d*-neighborhood of an *l*-mer *x*** is the set $\boldsymbol{N(x, d)}$ of all l-mers x' with Hamming distance, of at most d, from x. i.e., $\boldsymbol{d_H(x, x') \leq d}$.

Ex. $\texttt{ccgga}$, $\texttt{ccaaa}$, and $\texttt{gctta}$ are all in $N(\texttt{cctta}, 2)$

DEFINITION 5. *d*-neighborhood of a sequence *S*

The ***d*-neighborhood of a sequence *S*** is the set $\boldsymbol{\mathcal{N}(S, d)}$ of all l-mers that belongs in the neighborhood of at least one l-mer in the sequence S.

Ex. $\mathcal{N}(\texttt{aattacg}, 2) = N(\texttt{aatta}, 2) \cup N(\texttt{attac}, 2) \cup N(\texttt{ttacg}, 2)$

if $l = 5$.

DEFINITION 6. (l, d) **Planted Motif Problem**

Given a set of n sequences over $\Sigma = \{\mathtt{a}, \mathtt{c}, \mathtt{g}, \mathtt{t}\}$, where each sequence is of length m, two integer values l and d corresponding to the length of planted motif and the number of allowed mutations respectively, the goal is to find a motif of length l. It is also given that a d-neighbor of the motif is planted in each of the n sequences exactly once at a random position.

DEFINITION 7. EMS-GT, EMS-GT2 and qPMS9

EMS-GT is an exact enumerative algorithm, with Generate and Test phase, that uses the pattern-driven approach and quickly filters the 4^l search space by generating the neighborhood of sequences and intersecting them. EMS-GT2 improved EMS-GT by improving the Test phase of the algorithm. The qPMS9 algorithm, one of the state-of-the-art algorithm, uses a combined sample-driven approach and pattern-driven approach and introduces pruning conditions that improves the runtime of the algorithm. EMS-GT was developed recently is better than previous state of the art qPMS9 on some challenge instances.

1.2 Objectives of the study

This study aims to improve the runtime performance of the EMS-GT algorithm and propose a new version of the algorithm, which is the EMS-GT2 algorithm. The objectives of the study are as follows:

1. To explore more speedup techniques that improve the runtime performance of the EMS-GT algorithm.
2. To evaluate the runtime performance of the proposed EMS-GT2 algorithm using synthetic datasets.
3. To evaluate EMS-GT2 against its predecessor EMS-GT and the qPMS9 algorithm using synthetic datasets.

1.3 Research questions

This study aims to answer the following questions:

- How can the block-related partitioning of the set of l-mers in the EMS-GT be exploited to improve the Generation and Test phases of the EMS-GT?
- Can some Hamming distance pre-computation be developed to further speedup the EMS-GT?

- How does the proposed improved version compare with the state-of-the-art motif search algorithms EMS-GT and qPMS9?

1.4 Significance of the study

Motif signifies important information that are useful in the field of Computational Biology. Finding these motifs, which is already proven as NP-Complete problem, requires creative, fast and efficient algorithms. Improving an already established and competitive algorithm in the motif finding is one way to contribute to this field.

EMS-GT algorithm is competitive and able to beat state-of-the-art algorithms like PMS8 and qPMS9 in a certain range of problem instances. Improving EMS-GT in such a way that it can beat other algorithms in different problem instances will make it a more viable option in practical motif finding applications. In addition, the improved way of hamming distance computation and the discovered properties of the data structure that EMS-GT2 use may prove useful in other algorithm solutions that use such data structure.

1.5 Scope and limitations

This study focuses on improving the existing EMS-GT by exploring and optimizing different areas in the algorithm and thus proposing the EMS-GT2 algorithm.

Since EMS-GT2 requires significant amount of memory when l is sufficiently large, each new speedup technique was evaluated using all synthetic datasets of (l, d)-challenge instance where $l < 18$. Generally this is acceptable since the motif lengths in actual biological applications is around 10 base pairs (bp) only [**?**]. Finally, EMS-GT2 is not yet designed to run on multiple processors and, thus, we evaluate the algorithms in a single-processor execution.

CHAPTER II

Review of Related Literature

Motif finding has been studied extensively in the previous years. Numerous algorithms have been made for motif finding and for PMS. These algorithms are categorized as either approximate algorithms or exact algorithms. This section discusses related algorithms that solves the (l, d)-planted motif problem including the EMS-GT algorithm.

2.1 Approximate Algorithms

Approximate algorithms, although they are fast, do not guarantee the exact solution all the time. Heuristic algorithms that perform local search such as Gibbs Sampling, Expectation Maximization (EM), Projections etc. are previously explored in the literature. Most of these algorithms initially work on a tuple of alignment positions that corresponds to l-mers across different string sequences in the dataset. Then they iteratively refine the alignment until a certain criteria has met. MEME [1] is a tool for motif finding that implements Expected Maximization. Other approximate algorithms that use local search are GARP [14], GibbsDNA [16] and Random Projection [3, 14]. WINNOWER [20] reduces the PMS problem to finding a large clique in a multipartite graph. Instead of

looking for the motif directly, the algorithm applies a winnowing technique to remove spurious edges that trims the graph representation making it easier to find the motif. Other approximate algorithms are MULTIPROFILER [15], PatternBranching, ProfileBranching [22] and CONSENSUS [13];

2.2 Exact Algorithms

Although they may not be as fast as approximate algorithms, exact algorithms return the correct answer all the time. Furthermore, these exact algorithms can be categorized based on their approach in solving the problem. One approach is to generate a common neighborhood out of all $(m - l + 1)^n$ possible positions representing l-mer tuples from all string sequences. This approach is called sample-driven. Another approach is called pattern-driven that checks from Σ^l possible l-mers which are the motifs.

Many exact algorithms solve the PMS problem using suffix trees and other related data structures. RISO [4], RISOTTO [21], SPELLER [24] and SMILE are all exact algorithms that use suffix trees. MITRA [11] improves the excessive memory requirement of sample-driven approach by using mismatch tree. Two other algorithms that have some similarity with our algorithm are the Voting algorithm and Bit-based algorithm. Voting algorithm [5] maintains a hash table that tracks the number the occurrence of every possible l-mer and makes sure

that every l-mer is only counted once in each sequence. An l-mer is considered a motif if its total occurrences is equal to the total number of sequences in the dataset.

Bit-based algorithm [6] generates the neighborhood of each sequence and intersects it to get the set of motif. Unlike PMS1, the Bit-based algorithm maps every l-mer to its corresponding integer value. It uses an array of size Σ^l to represent the neighborhood of a sequence and uses the integer representation of an l-mer to flag if its a member of the array. It generates the neighborhood of all sequences and merges it using the logical operator AND. The resulting array represents the set of motif.

2.2.1 PMS Algorithms Series

A series of exact algorithms for the (l, d)-planted motif search problem was developed by Rajasekaran et al. The algorithm PMS1 [23] is one of these algorithms. PMS1 solves the problem by enumerating the d-neighborhood of all the sequences in the dataset and intersects them, the result is the set of motifs. PMSi and PMSP [7] are algorithms based on PMS1. PMSi improves the memory space requirement of PMS1 by processing only two sequences at a time. PMSP works by generating all the d-neighborhood of each l-mer in the first sequence and testing each d-neighbor if it exists in the remaining sequences. PMSPrune

[8] works the same as PMP but with some improvements. It generates the neighborhood of an l-mer using a branch-and-bound approach and implements a pruning strategy to speedup the testing of l-mers. Succeeding algorithms like PMS5 [10] and PMS6 [2] extend the ideas of PMS1 and PMSPrune. PMS5 generates the common neighborhood of three l-mers from different sequences at a time and uses ILP for the pruning process. PMS6 only differs from PMS5 in the way it determines the three l-mers.

Quorum PMS is a generalized version of the (l, d)-motif search problem. Instead of finding an l-mer that exists in all n sequences, it only considers up to q. We can see that a qPMS problem is equal to PMS when $q = n$. The qPMS7 [9] is one algorithm that solves the qPMS problem. Algorithm qPMS7 is a generalized version of qPMSPrune (quorum version of PMSPrune) combined with the pruning strategy of PMS5 algorithm.

PMS8 and qPMS9 Algorithm

The PMS8 [18] algorithm improved its predecessors by using certain pruning conditions in common neighbor generation of a tuple of l-mers. The algorithm is composed of 2 parts, the sample driven part and the pattern driven part. The sample driven patrt generates a tuple of size k from the first k string sequence. The algorithm generates the tuple by choosing an l-mer x in sequence S_i. After

including an l-mer in the tuple, the algorithm filters all l-mers that has a distance of greater than $2d$ from x in $S_{i+1}...S_n$ sequences. If the filtering step results to at least one empty row in any sequences in the dataset, it discards x then it proceeds by choosing the next l-mer in S_i. If the size of the tuple reached a certain threshold, the algorithm proceeds to the pattern driven part by generating the common neighbors of the l-mers in the tuple. In computing the common neighborhood, the algorithm traverses a tree of possible l-mers considering some pruning conditions.

PMS8 solves the (l, d)-planted motif problem by repeatedly doing the sample driven part and pattern driven part starting from all possible l-mers in the first sequence.

2.2.2 EMS-GT Algorithm

The Exact Motif Search - Generate and Test algorithm for the planted motif search problem, first developed by Nabos [17], is composed of main two phases, the Generate phase and the Test phase. The Generate phase takes the first n' number of string sequences in the dataset and generates the set d-neighborhood one sequence at a time then intersects it. This accumulates and outputs the set of candidate motifs $\mathcal{C}$ and is composed of l-mers that have at least one d-neighbor in each of the first n' sequences. The Test phase evaluates each candidate motif

$c \in C$ by comparing c if it has at least one d-neighbor in each of the remaining $n - n'$ string sequences.

These phases are formally defined below:

(a) Generate phase

This phase operates on the first n' sequences. The intersection of the d-neighborhood of each sequence will result to the set of candidate motifs C.

$$C = \mathcal{N}(S_1, d) \cap \mathcal{N}(S_2, d) \cap ... \cap \mathcal{N}(S_{n'}, d). \tag{2.1}$$

(b) Test phase

Each candidate motif in C will be evaluated if it appears in all of the remaining $n - n'$ string sequences. If a candidate motif pass the test, it is then included in the set of motifs M.

Data structures and the how an algorithm deals with the data commonly drive the performance of an algorithm. The EMS-GT algorithm uses a compressed bit-flag array for fast candidate motif elimination. Some key techniques that EMS-GT uses are defined in this section.

Integer mapping of l-mers

EMS-GT converts l-mers into its corresponding integer values. To achieve this, each character in the l-mer is translated using 2 bits (a=00, c=01, g=10, t=11).

Ex. `actg` maps to `00011110` and has an integer value of 30

Bit-based set representation and l-mer enumeration

The EMS-GT maintains a 4^l array for enumerating all the possible l-mer values. The l-mer's integer value is used as the index value for the array. It uses the value of 1 if the l-mer is a member of the set, else it sets the value to 0.

Bit-array compression

To efficiently store these l-mers and save memory space, EMS-GT implements an approach that compresses the search space array using integer value bit flags. Instead of one l-mer per index value, the implementation can flag up to 32 l-mers (since we are using 32-bit integers) per index value. The explanation on how the algorithm accesses the bit flag is defined below:

Ex. `gacgt` maps to `1000011011` = 539 in decimal.

bit position = 539 mod 32 = 27;

array index = 539 / 32 = 16;

The bit flag for `gacgt` is in the 27^{th} least significant bit

of the integer at array index 16.

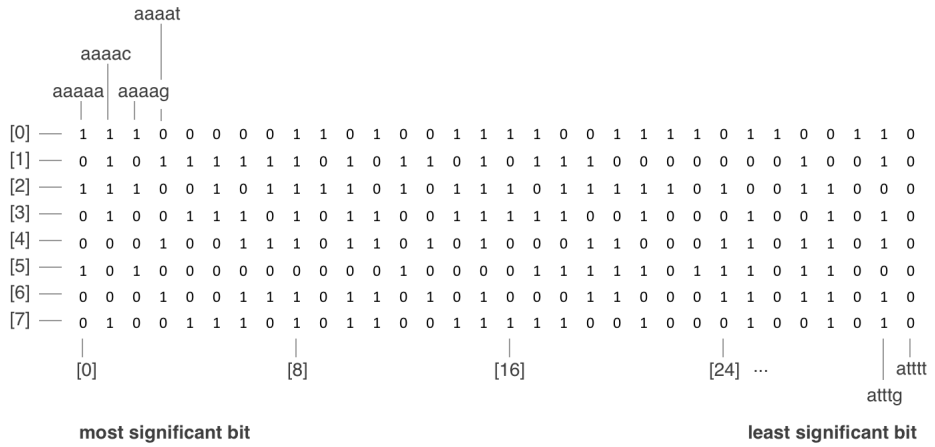

Figure 2.1. First 8 rows of the 4^5 search space with random flag values.

XOR-based Hamming distance computation

The mapping of an l-mer to its integer value has an additional advantage in computing for mismatch positions. Applying the boolean operator exclusive-or (XOR) between two integer values will return another integer value that contains nonzero value for mismatch position. Counting this nonzero positions result to the hamming distance value. An example of this computation is shown below:

Ex. aacgt maps to 0000011011

tacgc maps to 1100011001

XOR produces 1100000010 = 2 mismatches.

(Note, the mismatches are counted per pair)

Recursive neighborhood generation

The Generate step of the algorithm produces the d-neighborhood of a string sequence by generating the d-neighborhood of all l-mers in that sequence. Our implementation of EMS-GT uses a recursive approach for generating the d-neighborhood of an l-mer. The recursive generation can be visualized by a tree $\mathcal{T}(x)$ of height d that is generated in depth-first manner. Each node is a tuple of (w, p) where w is an l-mer and p corresponds to a position in the l-mer $0 \leq p \leq l$. At a given node (w, p) and $p \neq l$, three children nodes are generated where each node is variant of w that has a different character in $p + 1$ position. The root node is $(x, 0)$ and any l-mer in nodes at depth t has a hamming distance of t from the l-mer x. Given this, the expected size of $N(x, d)$ can be computed using the equation:

$$|N(x,d)| = \sum_{i=0}^{d} \binom{l}{i} 3^i \tag{2.2}$$

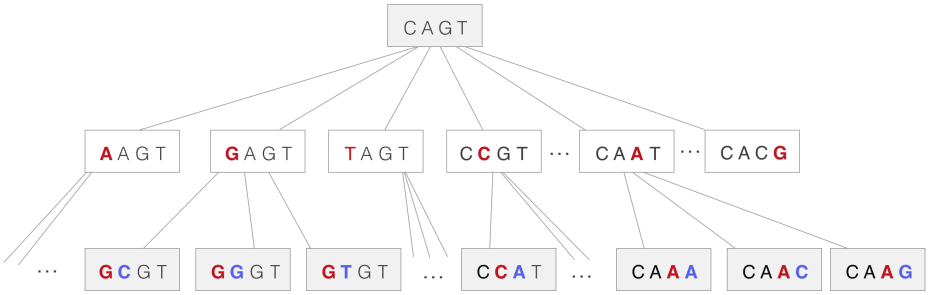

Figure 2.2. Recursively generating the neighborhood of l-mer CACGT with $d = 2$

Block-based optimization for neighborhood generation

The way EMS-GT represents the d-neighborhood N_x of l-mer x opens up a new way to improve the generation of neighborhood. N_x is represented by a compresed 4^l bit flags array, where value of 1 corresponds to set membership, 0 if otherwise. A previous study by Sia [25] improved the runtime performance in generating N_x. If N_x is partitioned into blocks of 4^k bits each, where $k < l$, each block will conform into (k + 2) bit patterns. By pre-computing these patterns, the algorithm can build the N_x by blocks of bits instead of one bit at a time.

The algorithm divides the l-mer x into its $(l - k)$-length prefix y and suffix z of length k. With the block patterns of 4^k l-mers generated, the algorithm recursively generates all possible prefix of x. For each prefix y' generated, the algorithm applies the corresponding block pattern in N_x based on z and the remaining number of allowed mutations d', where $d' = d - d_H(y, y')$. Specifically,

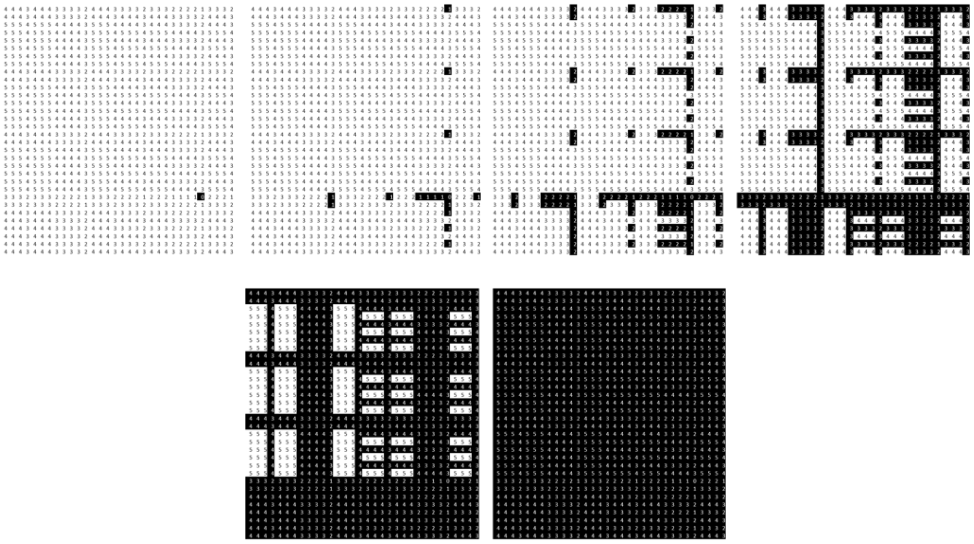

Figure 2.3. Bit patterns followed by blocks of size $4^5 = 32 \times 32$ in the bit-based representation of $\mathcal{N}(\texttt{acgtacgtacgt}, 5)$. Black signifies a 1. There are $(5+2) = 7$ possible patterns—the empty pattern (all 0s) is not shown. Images from [25]

EMS-GT builds the $\mathcal{N}_S$ using these steps:

1. Initialize $\mathcal{N}_S$ as an array of 4^l bits set to zero, and select a value for k.

2. Pre-generate *Pattern*(z, d_z) for all $z \in \Sigma^k$ and all $d_z \in \{1, ..., k-1\}$ to serve as bit masks for blocks. Note that block patterns for $d_z = 0$ (one bit set) and $d_z = k$ (all bits set) will not require bit masks.

3. For each l-mer $x = yz$ in sequence S: take each neighbor y' of y, find the block in $\mathcal{N}_S$ whose prefix is y', and compute the allowable suffix mismatches $d_z = d - d_H(y, y')$ within this block. Then,

   (a) if $d_z = 0$, set the bit at position z in the block;

(b) if $d_z \geq k$, set all bits in the block to 1;

(c) otherwise, mask *Pattern*(z, d_z) onto the block.

Choosing the optimum value for k is important in this speedup technique. The k value determines the runtime complexity of this technique since k value determines the size of the block patterns. When k value is higher, there are fewer prefix to generate recursively but each block bits setting is large. When k value is lower, block bits setting is small but the algorithm has to recursively generate a larger number of prefix value of l-mer x. The optimum k value used in the study is 5.

Algorithm 2.1 BLOCK PATTERN GENERATION
Input: block degree k
Output: 3D bit-array $\mathcal{P}$ containing all possible non-trivial block patterns

```
1: P[ ][ ][ ] ← {}                          ▷ retrieve a pattern P as P[z][d − d_y']
2: for z ← 0 to 4^k do
3:     for j ← 1 to k − 1 do
4:         for z' ← 0 to 4^k do
5:             if dH(z, z') ≤ j then
6:                 P[z][j][z'] ← 1
7:             else
8:                 P[z][j][z'] ← 0
9:             end if
10:        end for
11:    end for
12: end for
13: return P
```

EMS-GT with this speedup technique has proven its competitiveness against algorithms qPMSPrune, qPMS7, PMS8 and qPMS9. Previous experimentations

showed that EMS-GT with this speedup technique outperforms PMS8 in challenge instances (9, 2), (11, 3), (13, 4), (15, 5) and (17, 6). Compared to qPMS9, the improved EMS-GT is faster on all challenge instances mentioned except (17, 6). This study aims to outperform qPMS9 for challenges up to (17, 6).

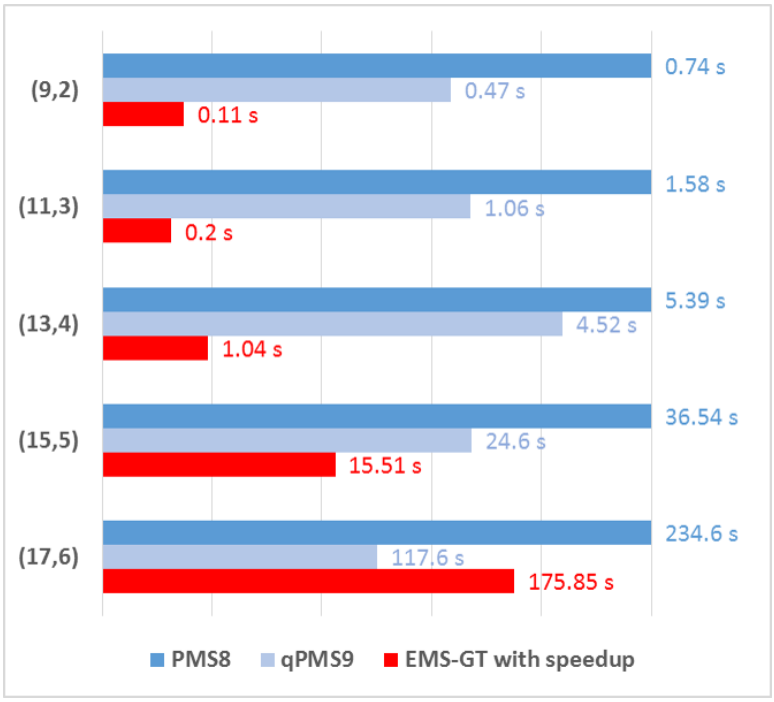

Figure 2.4. Improved EMS-GT's performance vs. PMS8 (baseline) and qPMS9. Images from [25]

CHAPTER III

Methodology

This section describes how the speedup techniques were explored and describes how we come up with the proposed EMS-GT2 algorithm. This also describes the process in evaluating the speedup techniques as well as other algorithms that solves the (l, d)-planted motif problem.

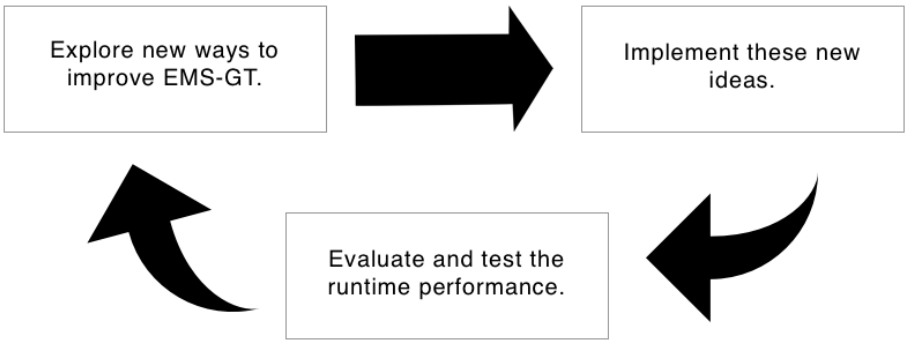

Figure 3.1. Speedup technique development cycle.

The study aims to improve the algorithm by pre-computation of values and exploring other usage of the block-processing technique. The speedup techniques were developed using the cycle shown in Figure 3.1.

3.1 Improving the EMS-GT algorithm

Originally, EMS-GT was implemented using Java, but for the purpose of eliminating variables that may affect the evaluation of the algorithms, it was converted to C++.

A previous study [25] improved the neighborhood generation of an l-mer by the use of block bits settings in the neighborhood array instead of one bit at a time. The Generation phase quickly filters the candidate motifs array as it processes the first n' sequences, leaving numerous empty blocks of l-mers in the candidate motifs array. It was observed that at some point in the Generation phase, some of the block settings are not necessary anymore since that block is already empty in the candidate motifs array. We improved the algorithm by maintaining boolean flags for those empty blocks and then we skip all block bit settings for those blocks. Additionally, the block-processing procedure was found useful in testing of candidate motifs. The Test phase checks if a candidate motif c is in the remaining $n - n'$ sequences by comparing if there is at least one l-mer in each sequence that is within d-distance from c. If a candidate motif x is eliminated for failing to have a d-neighbor in some input sequence S_i, that it is possible to reduce the testing for another candidate motif y on the same sequence S_i, if y is within the same k-block as x.

Lastly, hamming distance computation was also improved using a pre-computed lookup values. Given an XOR result, instead of counting nonzero pairs of bits, we use the lookup table to get the number of mismatches.

3.2 Parameter Fine Tuning

The EMS-GT algorithm defines an integer value n' ($1 \leq n' \leq n$) that divides the dataset into two smaller set of sequences. The first n' sequences are used in the Generate phase while the remaining sequences are assigned to the Test phase. Previous experimentations [25] showed that it is efficient for the algorithm to use $n' = 10$. Technically, the n' parameter is related to the candidate motifs array $\mathcal{C}$ size that will be evaluated in the remaining $n - n'$ sequences.

In this study, we re-evaluated the ideal value for n', since two of the speedup techniques introduced here requires sufficiently large candidate motifs array for them to take effect. We run an experimentation that records the average runtime of the algorithm with the speedup techniques over 5 tests and having different values for n'. The values for n' ranges from 5 to 10 in this experiment.

Table 3.1 shows the ideal n' values for each (l, d)-challenge instances mentioned. For (l, d) instances where $l \leq 11$ the ideal value for n' is still 10. This is true due to the efficiency of the Generate phase in small instances of the prob-

lem. For (13, 4), (15, 5) and (17, 6) problem instances, different n' values were used in the evaluation which are 9, 8, and 7 respectively.

n'	$(9, 2)$	$(11, 3)$	$(13, 4)$	$(15, 5)$	$(17, 6)$
5	0.07 s	0.42 s	2.37 s	17.86 s	148.62 s
6	0.06 s	0.29 s	1.54 s	12.64 s	116.01 s
7	0.05 s	0.21 s	1.01 s	10.70 s	**111.94 s**
8	0.05 s	0.18 s	0.84 s	**10.38 s**	119.82 s
9	**0.03 s**	0.14 s	**0.75 s**	11.01 s	132.10 s
10	**0.03 s**	**0.13 s**	0.76 s	11.90 s	146.10 s

Table 3.1. Runtime evaluation of the algorithm with fast candidate elimination and pre-computed hamming distance values over different n' values.

Additionally, the block flags speedup technique also defines another parameter n'' where $1 < n'' < n'$. This parameter represents the sequence number where the algorithm will start using the block flags. We also run an experimentation to assess the best value for the parameter n''. Table 3.2 shows the runtime performance of EMS-GT with the block flags speedup technique using different n'' values. The experiments showed that the ideal value for n'' in instances (9, 2), (11, 3), (13, 4), (15, 5) and (17, 6) are 8, 10, 7, 7 and 7 respectively.

3.3 Evaluation

We first assess different combinations of speedup techniques for EMS-GT. The combination with the fastest runtime performance will become the proposed

n''	$(9, 2)$	$(11, 3)$	$(13, 4)$	$(15, 5)$	$(17, 6)$
2	0.039 s	0.138 s	1.006 s	14.255 s	149.369 s
3	0.034 s	0.136 s	1.067 s	13.888 s	143.152 s
4	0.038 s	0.129 s	0.947 s	13.274 s	137.835 s
5	0.036 s	0.121 s	0.969 s	12.972 s	134.219 s
6	0.037 s	0.119 s	0.887 s	12.776 s	132.306 s
7	0.037 s	0.120 s	**0.878 s**	**12.422 s**	**132.207 s**
8	**0.032 s**	0.115 s	0.928 s	12.451 s	135.140 s
9	0.036 s	0.106 s	0.891 s	12.775 s	138.997 s
10	0.035 s	**0.104 s**	0.914 s	13.054 s	143.720 s

Table 3.2. Runtime evaluation of the algorithm with the block boolean flags strategy and pre-computed hamming distance values over different n'' values.

EMS-GT2. The different combinations are composed of the following: (1) EMS-GT with block flags, (2) EMS-GT with block flags and improved Hamming distance computation, (3) EMS-GT with faster candidate motif elimination, (4) EMS-GT with faster candidate motif elimination and improved Hamming distance computation and (5) EMS-GT with block boolean flags, faster candidate motif elimination and improved Hamming distance computation.

We evaluated each of the speedup techniques introduced in this study. We combined these speedup techniques and determined which combination yields the fastest runtime performance, which will be incorporated in the proposed EMS-GT2 algorithm. Specifically, these different combinations are the following: (1) EMS-GT with block boolean flags, (2) EMS-GT with block boolean flags and improved Hamming distance computation, (3) EMS-GT with faster candi-

date motif elimination, (4) EMS-GT with faster candidate motif elimination and improved Hamming distance computation and (5) EMS-GT with block boolean flags, faster candidate motif elimination and improved Hamming distance computation.

We compared EMS-GT2 to its predecessor EMS-GT and the algorithm qPMS9 using a set of challenging (l, d) instances. An (l, d) problem instance is said to be challenging if d is the largest integer value for which the expected number of motifs of length L would occur in the input by random chance and does not exceed a constant value (500) [19]. The challenge instances used in the evaluation are the following: (9, 2), (11, 3), (13, 4), (15, 5) and (17, 6).

3.4 Datasets

The algorithm EMS-GT2 was evaluated using both synthetic data sets and real data sets.

3.4.1 Synthetic Datasets

Algorithms that solve PMS [20, 18, 19] use a dataset containing 20 string sequences where each nucleotide is in $\Sigma = \{a, c, g, t\}$. Each string sequence is 600 base pairs (bp) long and each nucleotide is randomly generated with equal chance of being selected. A motif is then generated and for each string sequence

in the dataset, a d-neighbor is planted at a random position. In this study, a generator was run to produce a dataset of PMS problem instances with this configuration, and this dataset was then used to evaluate the algorithms. Furthermore, a converter program was used to translate the dataset into FASTA format in order to execute qPMS9.

3.4.2 Real Datasets

The EMS-GT2 algorithm was also evaluated using real data sets that were previously used in the earlier implementation of EMS-GT. The real data sets are the sets of promoter sequences of yeast (*Saccharomyces cervisiae*) [27] and sets of orthologous sequences of different gene families of eukaryotes.

CHAPTER IV

Results and Analysis

This section describes the speedup techniques introduced in this study and briefly discusses the observations where the idea for the improvements originated. This also describes the speedup techniques incorporated in the EMS-GT2 algorithm. Lastly, evaluation of runtime performance of the EMS-GT2, EMS-GT and qPMS9 was also discussed in this section.

4.1 Block boolean flags technique in the Generate phase (BBF)

The EMS-GT algorithm follows the pattern-driven approach where it exhaustively tests the 4^l bit-array of possible motifs. As discussed earlier, EMS-GT quickly filters the 4^l bit-array in the Generate phase by intersecting the d-neighborhood of the first n' sequences. EMS-GT maintains two 4^l bit-array during the Generate phase. The first array stores the remaining candidate motifs $\mathcal{C}$ and the second array stores the d-neighborhood of the current sequence $\mathcal{N}$. For every sequence in the first n' sequences, we generate its d-neighborhood by blocks of bits settings. Then we intersect the d-neighborhood of the sequence with the candidate motifs array, the result will become the new candidate motifs array. We observed that at some point in the Generation phase, candidate motifs array

has numerous empty blocks of l-mers. Applying block bits setting in this part of the neighborhood array will be useless since we already know that there is no candidate motifs in that block. An efficient way is to focus the neighborhood generation on those blocks where there are still candidate motifs. We have implemented a block boolean flags technique that tracks these empty blocks in the candidate motifs array.

4.1.1 Maintaining block boolean flags for the candidate motifs array

We divide the candidate motifs array $\mathcal{C}$ into groups of 4^k l-mers. The k value should be the same with the one used in the block neighborhood generation strategy. In this study, we used $k = 5$. For each group of 4^k l-mers, we assigned a boolean flag that tells if the group is empty or not. Everytime we update the contents of the candidate motif array, we also update the boolean flags for empty blocks in the candidate motifs array. Given a candidate motifs array of size $4^l/32$, we maintain an array of block boolean flags $\mathcal{B}$ of size $(4^l/32)/(4^k/32)$. Figure 4.1 illustrates the way of grouping these 4^k l-mers and its corresponding block boolean flags.

Formally, for any row value v and block height value h, the block boolean

flags array is defined as:

$$\mathcal{B}[v] = \begin{cases} 1 & \text{if candidate motif exists in } \mathcal{C}[v * h, (v * h) + h), \\ 0 & \text{otherwise.} \end{cases} \tag{4.1}$$

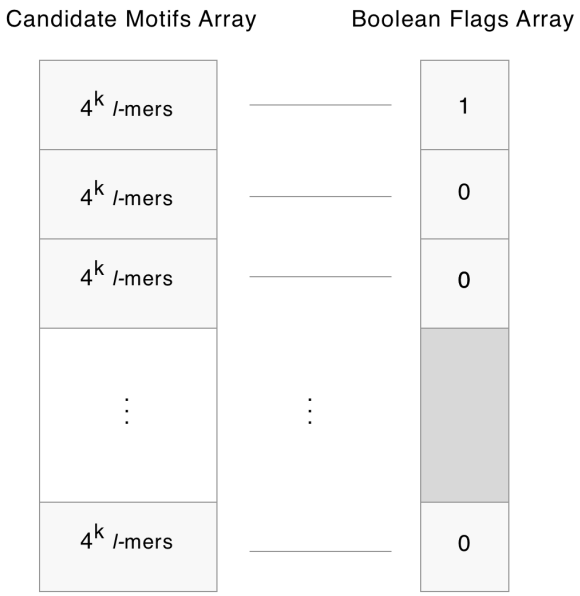

Figure 4.1. Assigning boolean flags for every 4^k l-mers block in the candidate motifs array.

4.1.2 Usage of block boolean flags strategy

The improved d-neighborhood generation of l-mer uses a pre-generated block patterns of size 4^k l-mers. Given these patterns, we recursively generate all possible prefix, with length $l - k$, of the given l-mer. These prefix values represents the row start value where the block bits setting will be applied. For each prefix value, we check if can skip the block bits setting by checking if the block is empty

using its assigned block boolean flag. If the prefix value is not equal to the row start value of the block the boolean flag is representing, then we need to also check the next boolean flag if it is empty since the block bits setting will overlap, unless it is the last block in the array. Figure 4.2 shows an illustration for these scenarios.

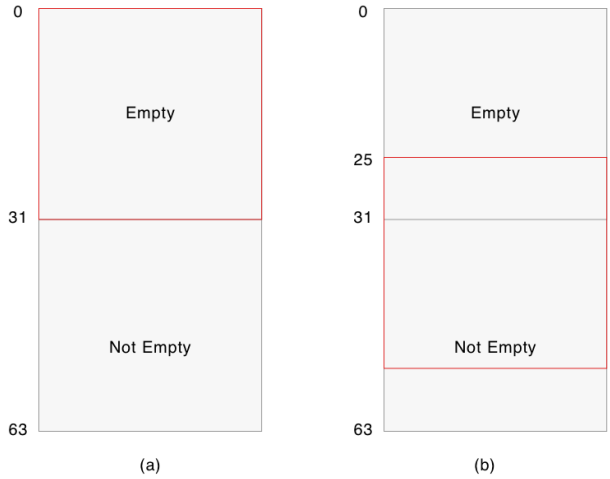

Figure 4.2. (a) Shows a skipped block bit setting with prefix value of 0; (b) Shows a block bit setting with prefix value of 25.

Block boolean flags strategy checks if a block is not empty before applying the block bits setting. This approach is useful when there are sufficiently large number of empty blocks in the candidate motifs array. We introduced a new parameter n'', where $1 < n'' < n'$, that is the sequence number where we will start using the block boolean flags strategy. In this study, we use for problem instances (9, 2), (11, 3), (13, 4), (15, 5) and (17, 6) the n'' value of 8, 10, 7, 7 and

7 respectively.

4.2 Fast Candidate Motif Elimination through Block Processing (FCE)

The EMS-GT algorithm tests candidate motifs in a brute-force approach. A candidate motif is tested by checking if it has at least one d-neighbor in each of the remaining $n - n'$ sequences. In testing a candidate motif c, if there is a sequence S_i in the remaining $n-n'$ sequences where c doesn't have any d-neighbor, candidate motif c is automatically eliminated.

In our implementation, the search space is represented by a compressed bit array and the l-mers are enumerated alphabetically. L-mers that are near each other do not differ that much. We used this observation in improving the way the algorithm tests the candidate motifs. Figure 4.3 illustrates a part of the search space's representation.

In EMS-GT, each testing of candidate motif is independent of each other. We proposed a speedup technique that processes these candidate motifs by blocks. We first partition the search space by blocks containing 4^k l-mers. This results into l-mers that share the same $(l - k)$-prefix characters where $2 < k < l$. Since every row in the search space represents exactly 32 l-mers (32-bit integers), the height of every block is computed using $(4^k/32)$. Additionally, any two l-mers within a block has at most k hamming distance value between them. Figure 4.4

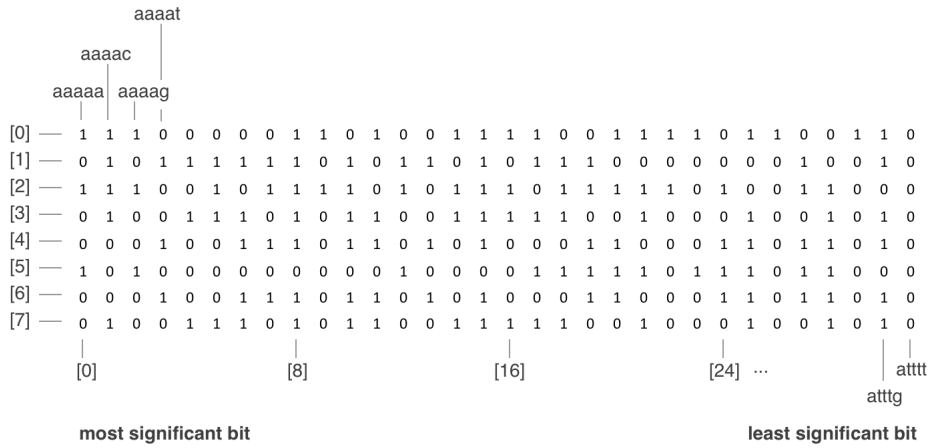

Figure 4.3. First 8 rows of the 4^5 search space with random flag values.

shows an example of partitioned search space showing blocks of l-mers sharing a common prefix string.

We process the testing of candidate motifs now by blocks. If candidate motifs x and y are within a block and x has been eliminated as a candidate motif in sequence S_i $(n' \leq i \leq n)$, we can filter out l-mers $z \in S_i$ where $d_H(x, z) > d+k$. We collect the remaining l-mers in S_i and use it for testing the remaining candidate motifs in the block along with the other l-mers in the remaining sequences in $S'_n, S'_n + 1, ..., S_n - S_i$. The theorem below formalizes the main property used in this speedup technique. Figure 4.5 illustrates the proof of the Theorem 1.

Theorem 1. *Let x and y be l-mers in a block in the search space containing 4^k l-mers. Let d be the number of allowed mutations in the problem instance. Let z be another l-mer. If $d_H(x, z) > (k + d)$ then $d_H(y, z) > d$ and therefore z is not in*

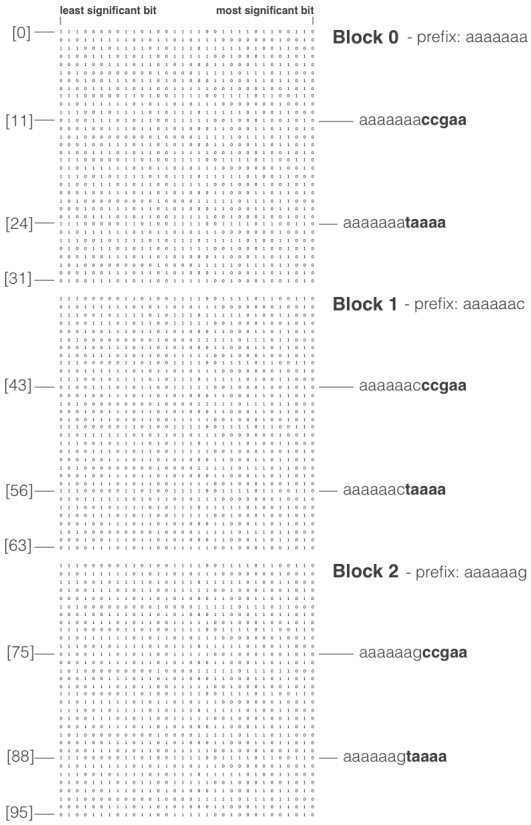

Figure 4.4. Illustration of the block partitioning of a 4^{12} search space. There are 4^5 l-mers in each block and has a height of 32. The illustration shows the first 3 blocks only.

$N(y, d)$

Proof of Theorem 1. Using proof by contradiction, we first suppose that $d_H(y, z) \leq d$. Since the l-mers x and y belong to the same block, then $d_H(x, y) \leq k$. We use these bounds in the triangle of inequality $d_H(x, z) \leq d_H(x, y) + d_H(y, z)$ to derive the result $d_H(x, z) \leq k + d$. This result contradicts the given condition that $d_H(x, z) > (k + d)$. Hence, we are sure that $d_H(y, z) > d$. □

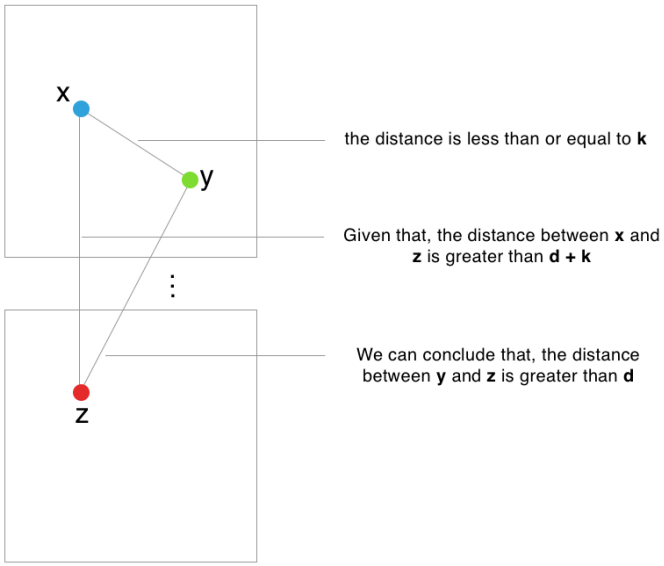

Figure 4.5. Illustration of the proof for the property used in the fast candidate motif elimination.

The k-value affects the number of l-mers that is filtered in a sequence. The lower its value, the larger the number of filtered l-mers and faster candidate motif testing will be. But since k also affects the number of l-mers in a block, the lower its value, the fewer the candidate motifs that might benefit from the speedup technique. In our implementation we use $k = 5$ where every block is 32 rows of 32 bit flags representing a total of 4^5 l-mers.

4.2.1 Improved Hamming distance computation by pre-computation of mismatch values (IHD)

The EMS-GT2 uses the hamming distance computation heavily during the Test phase. As discussed earlier for the original EMS-GT, the hamming distnace of two binary represented l-mers, can be efficiently computed usign the boolean operator XOR. In EMS-GT2, instead of repeatedly counting this non-zero pairs of bits every time we compute the hamming distance, we use a pre-computed lookup table to help reduce computational time. Figure 4.6 shows a snapshot of this table of pre-computed number of non-zero pairs of bits.

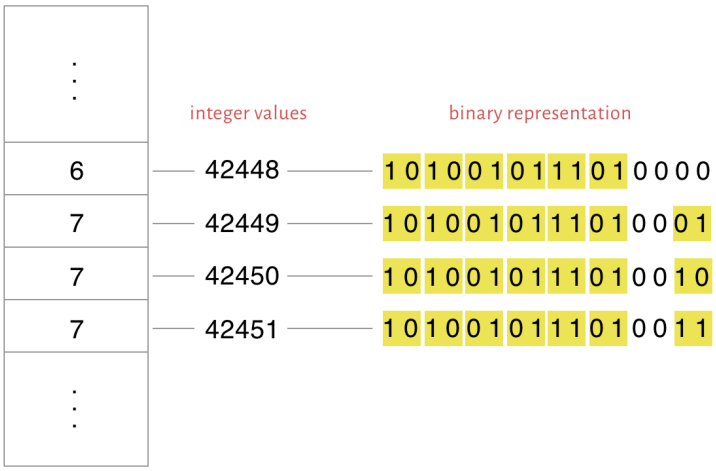

Figure 4.6. An illustration of the pre-computed number non-zero pairs of bits up to a certain integer value for the hamming distance value computation.

A naive pre-computation of these nonzero pair counts for all possible l-mers (each represented by $2l$ bit values) will introduce an unacceptable overhead

computation time when l is sufficiently large, i.e., when $l \geq 10$ (based on actual runs on our current machine configurations). A more efficient approach is to pre-compute only up to l-mers of length $l' < l$, which required $b = 2l'$ number of bits. Then we determine the hamming distance by looking up to the nonzero counts in the XOR results, b number of bits at a time, as described in Algorithm 1. To illustrate the new process, Figure 4.7 shows a demonstration of the improved hamming distance computation.

Algorithm 4 HAMMING DISTANCE COMPUTATION USING PRE-COMPUTED MISMATCH VALUES

Input: l-mer mappings u and v and
MC ▷ *array of pre-computed count of mismatch positions*
Output: Hamming distance $d_H(u, v)$

1: $d_H(u, v) \leftarrow 0$
2: $z \leftarrow u \oplus v$
3: **while** $z > 0$ **do**
4: $l \leftarrow z\&((1 << 18) - 1)$
5: $d_H(u, v) \leftarrow d_H(u, v) + MC[l]$
6: $z \leftarrow z >>> 18$ ▷ *shift 18 bits to the right*
7: **end while**
8: **return** $d_H(u, v)$

In our experimentation the maximum required bits to represent l-mers is 34 bits (for (17, 6)-instance). Given this, we pre-compute up to 18 bits ($l' = 9$) values only and use the lookup table twice for the computation of the actual hammign distance between any given pair of l-mers.

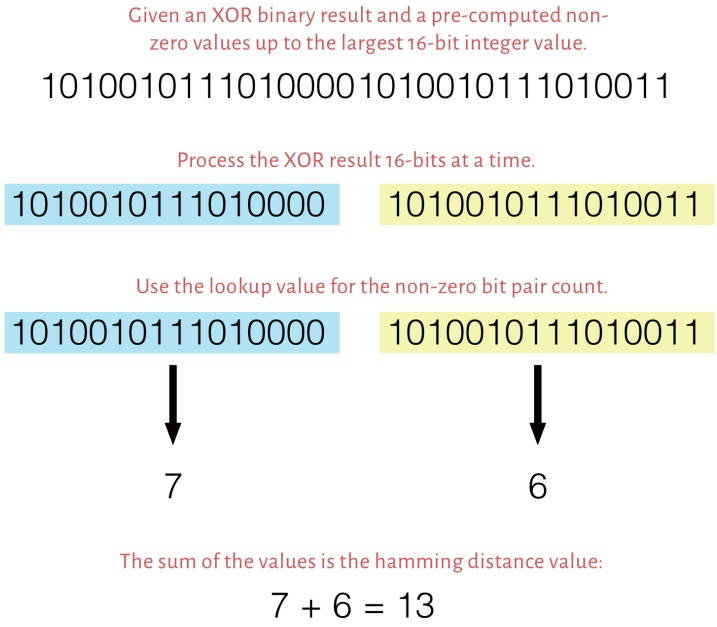

Figure 4.7. A sample demonstration of the improved hamming distance value computation.

4.3 Performance of EMS-GT with speedup techniques

Every runtime evaluation using these speedup techniques was run using Intel Xeon, 2.10 Ghz machine. The performance of each algorithm was averaged over 20 synthetic datasets for each (l, d)-challenge instance where $l \leq 17$. This section shows the runtime performance of EMS-GT with different combinations of speedup techniques introduced in this study. The version of EMS-GT that yielded the faster runtime performance was used in the proposed EMS-GT2 algorithm. Overall, these speedup techniques improved the runtime performance

of EMS-GT algorithm with at least 15%, 20.37%, 5.87%, 6.50% and 5.17% for (9, 2), (11, 3), (13, 4), (15, 5) and (17, 6) challenge instances respectively.

4.3.1 Evaluation of block boolean flag (BBF) technique

Table 4.2 shows the runtime performance of block boolean flag technique with its corresponding runtime reductions from the original EMS-GT.

(l, d)	**EMS-GT**	**EMS-GT with BBF**	**% speedup**
(9, 2)	0.0468 s	0.049 s	-
(11, 3)	0.168 s	0.172 s	-
(13, 4)	1.180 s	1.107 s	6.21 %
(15, 5)	12.767 s	11.937 s	6.51 %
(17, 6)	143.215 s	131.776 s	7.99 %

Table 4.1. EMS-GT with block boolean flags strategy.

The block boolean flags technique is efficient when there are significant amount of empty blocks in the candidate motif array. Using this technique, the generation of d-neighborhood can focus on those blocks with remaining candidate motifs.

Choosing the value for the parameter n' plays a valuable role in this speedup technique. The more sequences processed during the Generation phase the smaller the size of the candidate motifs become and thus the number of empty blocks increases. As we process more sequence in the Generate phase, using the block boolean flags technique, the runtime in generating the d-neighborhood

of a sequence decreases. Figure 4.8 shows the average runtime in generating the d-neighborhood of a sequence per sequence number in (17, 6) problem instance. In evaluating the block boolean flags technique we use $n' = 10$, which is the optimum value used also in the previous studies.

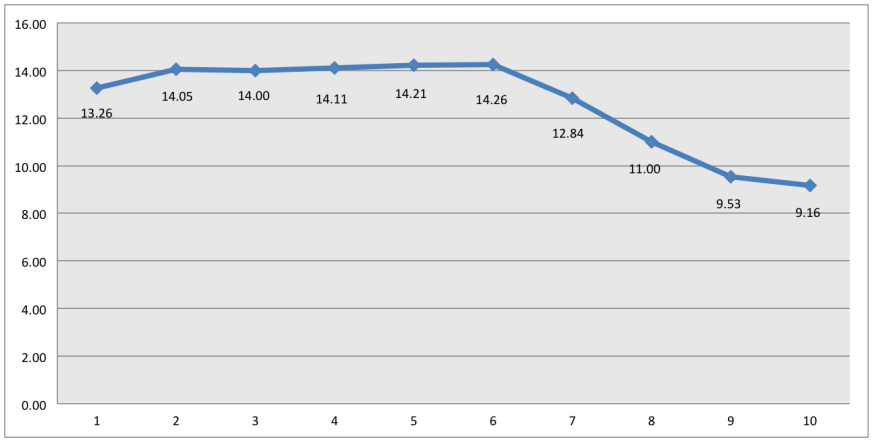

Figure 4.8. Runtime (s) of d-neighborhood generation of sequence for all sequence in Generation Phase using (17, 6) problem instance.

Additionally, the we incorporated the improved hamming distance computation technique with block boolean flags technique resulting into faster runtime performance, which is shown in Table 4.**??**.

(l, d)	**EMS-GT**	**EMS-GT with BBF and IHD**	**% speedup**
(9, 2)	0.0468 s	0.035 s	25.00 %
(11, 3)	0.168 s	0.115 s	31.91 %
(13, 4)	1.180 s	0.930 s	21.27 %
(15, 5)	12.767 s	11.532 s	9.68 %
(17, 6)	143.215 s	130.841 s	8.64 %

Table 4.2. EMS-GT with BBF strategy and improved hamming distance computation.

4.3.2 Evaluation of fast candidate motif elimination (FCE) technique

Runtime performance of fast candidate motif elimination technique is shown in Table 4.4.3. Fast candidate motif elimination is used the Test phase of the algorithm. This technique is efficient if there are a significant number of candidate motifs left in a block, unlike the block boolean flags technique that is efficient when there are numerous empty blocks in the candidate motifs array. This technique uses different values for n' which are 10, 10, 9, 8 and 7 for the mentioned (l, d) challenge instances respectively.

(l, d)	**EMS-GT**	**EMS-GT with FCE**	**% speedup**
(9, 2)	0.0468 s	0.051 s	-
(11, 3)	0.168 s	0.172 s	-
(13, 4)	1.180 s	1.340 s	-
(15, 5)	12.767 s	13.362 s	-
(17, 6)	143.215 s	135.799 s	5.17 %

Table 4.3. Runtime comparison of EMS-GT and EMS-GT with fast candidate motif elimination.

EMS-GT with fast candidate motif elimination technique alone only improved the runtime in (17, 6) by 5.17%, but with the improved hamming distance computation the runtime performance has greatly increased in all of the challenge instances used in the experimentations. The Test phase uses the hamming distance computation heavily and choosing the right sequence to start the Test phase greatly affects the runtime with these speedup techniques. Table 4.4.4

shows the runtime performance of EMS-GT with fast candidate motif elimination technique and the improved hamming distance computation.

(l, d)	**EMS-GT**	**EMS-GT with FCE and IHD**	**% speedup**
(9, 2)	0.0468 s	0.038 s	18.33 %
(11, 3)	0.168 s	0.126 s	25.46 %
(13, 4)	1.180 s	0.946 s	19.88 %
(15, 5)	12.767 s	10.876 s	14.82 %
(17, 6)	143.215 s	111.660 s	22.03 %

Table 4.4. Runtime comparison of EMS-GT and EMS-GT with fast candidate motif elimination and improved hamming distance computation.

Finally, Table 4.4.5 shows the runtime evaluation of EMS-GT with all of the speedup techniques having n' values used in the fast candidate motif elimination experiments. For all the (l, d) challenge instances used except (17, 6), EMS-GT with fast candidate motif elimination and improved hamming distance computation is still faster than EMS-GT with all speedup techniques. EMS-GT with all speedup techniques failed to compensate for the computational overhead it has introduced in the block boolean flags technique. Given this result, we used the fast candidate motif elimination and improved hamming distance computation only in the proposed EMS-GT2 algorithm.

(l, d)	**EMS-GT**	**EMS-GT with FCE, BFF and IHD**	**% speedup**
(9, 2)	0.0468 s	0.040 s	15.0 %
(11, 3)	0.168 s	0.134 s	20.37 %
(13, 4)	1.180 s	1.112 s	5.88 %
(15, 5)	12.767 s	11.252 s	11.87 %
(17, 6)	143.215 s	111.634 s	22.05 %

Table 4.5. Runtime comparison of EMS-GT and EMS-GT with all speedup techniques.

4.4 Runtime performance comparison of EMS-GT2, EMS-GT and qPMS9

The EMS-GT2, EMS-GT and qPMS9 were evaluated in terms of actual runtime on an Intel Xeon, 2.10 Ghz machine. The performance of each algorithm was averaged over synthetic datasets for each (l, d)-challenge isntance where $l \leq 17$. Table 4.4.6 shows the runtime results between EMS-GT2 vs EMS-GT while Table 4.4.7 shows the runtime results between EMS-GT2 vs the algorithm qPMS9.

(l, d)	**EMS-GT**	**EMS-GT2**	**% speedup**
(9,2)	0.047 s	0.038 s	18.33 %
(11,3)	0.168 s	0.126 s	25.46 %
(13,4)	1.181 s	0.946 s	19.88 %
(15,5)	12.768 s	10.876 s	14.81 %
(17,6)	143.215 s	111.660 s	22.03 %

Table 4.6. EMS-GT and EMS-GT2 runtime evaluation.

The additional speedup techniques become more effective as the l value in the (l, d)-instance grows. For every (l, d)-challenge instances mentioned, EMS-GT2 has improved the runtime over the EMS-GT for at least 14.8%.

(l, d)	**qPMS9**	**EMS-GT2**	**% speedup**
(9,2)	0.68 s	0.038 s	94.41%
(11,3)	1.51 s	0.126 s	91.65%
(13,4)	6.48 s	0.946 s	85.40%
(15,5)	26.67 s	10.876 s	59.22%
(17,6)	118.54 s	111.660 s	5.8%

Table 4.7. EMS-GT2 and qPMS9 runtime evaluation.

Previous implementations of EMS-GT failed to beat qPMS9 in $(17, 6)$-challenge instance. The proposed EMS-GT2 not only produces improved runtimes but also succeed in beating the qPMS9 in all of the (l, d)-challenge instances where $l \leq 17$. Even though the implementation of EMS-GT can only run in (l, d)-challenge instances where $l \leq 17$ because of computer memory constraint, studies have shown that typical length of motifs is around 10 base pairs (bp) [26].

CHAPTER V

Conclusions

In this study we have presented EMS-GT2, an improved exact solution for the planted-motif search problem. EMS-GT2 was able to introduce speedup techniques both on the Generate and Test phases of the algorithm. This study introduced a speedup technique that uses boolean flags for empty blocks in the candidate motifs array for an efficient d-neighborhood of a sequence generation in the Generate phase. In the Test phase, EMS-GT2 was able to efficiently test candidate motifs in block by filtering out l-mers using a property of the search space array that we have discovered. A proof of this property is also provided. Lastly, we improved the way of hamming distance computation by using a precomputed lookup table. However, to maximize the efficiency of these speedup techniques, we must use the optimum value for n'. Overall, the optimum value for n' is 10, 10, 9, 8, and 7 for the (9, 2), (11, 3), (13, 4), (15, 5) and (17, 6) challenge instances respectively.

EMS-GT2 algorithm, which implements the fast candidate motif elimination and improved hamming distance computation, was able to improve the runtime performance of EMS-GT with runtime reductions of 18.33%, 25.46%, 19.88%, 14.81% and 22.03% for (9, 2), (11, 3), (13, 4), (15, 5) and (17, 6) challenge

instances respectively.

The previous implementation of EMS-GT already outperforms the state-of-the-art algorithm qPMS9 in (l, d)-challenge instances $(9, 2)$, $(11, 3)$, $(13, 4)$ and $(15, 5)$ but failed to beat qPMS9 in $(17, 6)$-challenge instance. EMS-GT2 improved the algorithm's performance on $(13, 4)$, $(15, 5)$ and $(17, 6)$ and was able to beat qPMS9 in all (l, d)-challenge instances where $l \leq 17$. EMS-GT2 outperforms the qPMS9 algorithm with runtime reductions of 94.41%, 91.65%, 85.40%, 59.22%, and 5.8% for (9, 2), (11, 3), (13, 4), (15, 5) and (17, 6) challenge instances respectively.

Possible areas of improvement in this study includes:

- Further exploration of usage of the block-processing procedure in other parts of the algorithms.
- By default EMS-GT algorithm uses the first n' sequences in the Generate phase without considering the rest of the sequences, pre-processing and analysis of the sequences may lead to runtime improvements
- Early testing of candidate motifs to increase the number of empty blocks can improve the block boolean flags technique.

In conclusion, the speedup techniques introduced in this study can be useful in different algorithms that uses the same data structure. Improvement in

the Hamming distance computation can also be useful in solutions that uses such computations.

BIBLIOGRAPHY

[1] Timothy L. Bailey, Nadya Williams, Chris Misleh, and Wilfred W. Li. Meme: discovering and analyzing dna and protein sequence motifs. *Nucleic Acids Research.*, pages 369–373, 2006.

[2] Shibdas Bandyopadhyay, Sartaj Sahni, and Sanguthevar Rajasekaran. Pms6: a fast algorithm for motif discovery. *International Journal of Bioinformatics Research and Applications*, 10(4-5):369–383, 2014. PMID: 62990.

[3] Jeremy Buhler and Martin Tompa. Finding motifs using random projections. In *Proceedings of the Fifth Annual International Conference on Computational Biology*, RECOMB '01, pages 69–76, New York, NY, USA, 2001. ACM.

[4] Alexandra M. Carvalho, Ana T. Freitas, Arlindo L. Oliveira, Inria Rhonealpes, Universit Claude Bernard, and Lyon I. A highly scalable algorithm for the extraction of cis-regulatory regions. In *In Proc. APBC05*, pages 273–282. Imperial College Press, 2005.

[5] Francis Y.L. Chin and Henry C.M. Leung. Voting algorithms for discovering long motifs. In *Proceedings of the Third Asia-Pacific Bioinformatics Conference (APBC)*, pages 261–271, 2005.

[6] Naga Shailaja Dasari, Ranjan Desh, and Mohammad Zubair. An efficient multicore implementation of planted motif problem. In *High Performance Computing and Simulation (HPCS), 2010 International Conference on*, pages 9–15. IEEE, 2010.

[7] Jaime Davila, Sudha Balla, and Sanguthevar Rajasekaran. *Space and Time Efficient Algorithms for Planted Motif Search*, pages 822–829. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[8] Jaime Davila, Sudha Balla, and Sanguthevar Rajasekaran. Fast and practical algorithms for planted (l, d) motif search. *Computational Biology and Bioinformatics, IEEE/ACM Transactions on*, 4(4):544–552, 2007.

[9] Hieu Dinh, Sanguthevar Rajasekaran, and Jaime Davila. qpms7: A fast algorithm for finding (, d)-motifs in dna and protein sequences. *PLoS ONE*, 7(7):1–8, 07 2012.

[10] Hieu Dinh, Sanguthevar Rajasekaran, and Vamsi K. Kundeti. Pms5: an efficient exact algorithm for the (, d)-motif finding problem. *BMC Bioinformatics*, 12(1):1–10, 2011.

[11] Eleazar Eskin and Pavel A Pevzner. Finding composite regulatory patterns in dna sequences. *Bioinformatics*, 18(suppl 1):S354–S363, 2002.

[12] Patricia A. Evans, Andrew D. Smith, and H.Todd Wareham. On the complexity of finding common approximate substrings. *Theoretical Computer Science*, 306(1-3):407 – 430, 2003.

[13] Gerald Z Hertz and Gary D. Stormo. Identifying dna and protein patterns with statistically significant alignments of multiple sequences. *Bioinformatics*, 15(7):563–577, 1999.

[14] Hongwei Huo, Zhenhua Zhao, Vojislav Stojkovic, and Lifang Liu. Combining genetic algorithm and random projection strategy for (l, d)-motif discovery. In *Bio-Inspired Computing, 2009. BIC-TA'09*, pages 1–6. IEEE, 2009.

[15] U. Keich and P.A. Pevzner. Finding motifs in the twilight zone. *Bioinformatics*, 18(10):1374–1381, 2002.

[16] Charles E Lawrence, Stephen F Altschul, Mark S Boguski, Jun S Liu, Andrew F Neuwald, and John C Wootton. Detecting subtle sequence signals: a gibbs sampling strategy for multiple alignment. *science*, 262(5131):208–214, 1993.

[17] Julieta Q. Nabos. *New Heuristics and Exact Algorithms for the Planted DNA (l, d)-Motif Finding Problem*. PhD thesis, Ateneo de Manila University, 2015.

[18] Marius Nicolae and Sanguthevar Rajasekaran. Efficient sequential and parallel algorithms for planted motif search. *BMC bioinformatics*, 15(1):34, 2014.

[19] Marius Nicolae and Sanguthevar Rajasekaran. qpms9: An efficient algorithm for quorum planted motif search. *Scientific reports*, 5, 2015.

[20] Pavel A Pevzner, Sing-Hoi Sze, et al. Combinatorial approaches to finding subtle signals in dna sequences. In *ISMB*, volume 8, pages 269–278, 2000.

[21] Nadia Pisanti, Alexandra M. Carvalho, Ra M. Carvalho, Marie-France Sagot, and Laurent Marsan. Risotto: Fast extraction of motifs with mismatches. In *Proceedings of the 7th Latin American Theoretical Informatics Symposium, 3887 of LNCS:757-768*, pages 757–768. Springer-Verlag, 2006.

[22] Alkes Price, Sriram Ramabhadran, and Pavel A. Pevzner. Finding subtle motifs by branching from sample strings. *Bioinformatics*, 19(suppl 2):ii149–ii155, 2003.

[23] S. Rajasekaran, S. Balla, and C.H. Huang. Exact algorithms for planted motif challenge problems. *Journal of Computational Biology*, pages 1117–1128, 2005.

[24] Marie-France Sagot. Spelling approximate repeated or common motifs using a suffix tree, 1998.

[25] Aia Sia, Julieta Nabos, and Proceso Fernandez Jr. An efficient exact solution to the (l,d)-planted motif problem. *8th AUN/SEED-Net Regional Conference on Electrical and Electronics Engineering*, 2015.

[26] Alexander J Stewart, Sridhar Hannenhalli, and Joshua B Plotkin. Why transcription factor binding sites are ten nucleotides long. *Genetics*, 192(3):973–985, 2012.

[27] Jian Zhu and Michael Q. Zhang. Scpd: a promoter database of the yeast saccharomyces cerevisiae. *Bioinformatics*, 15(7):607–611, 1999.