ATENEO DE MANILA UNIVERSITY

EMS-GT2: AN IMPROVED EXACT SOLUTION FOR THE (l, d)-PLANTED MOTIF SEARCH PROBLEM

A THESIS SUBMITTED TO

THE GRADUATE FACULTY OF

THE SCHOOL OF SCIENCE AND ENGINEERING

IN CANDIDACY FOR THE DEGREE OF

MASTER OF SCIENCE IN

COMPUTER SCIENCE

DEPARTMENT OF INFORMATION SYSTEMS AND COMPUTER SCIENCE

BY

MARK JOSEPH D. RONQUILLO

QUEZON CITY, PHILIPPINES

2016

The THESIS entitled:

EMS-GT2: AN IMPROVED EXACT SOLUTION FOR THE (l, d)-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.	PROCESO L. FERNANDEZ, JR., Ph.D.
Chair	Adviser

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

The Faculty of the Department of Information Systems and Computer Science, School of Science and Engineering, Ateneo de Manila University ACCEPTS THE THESIS entitled:

EMS-GT2: AN IMPROVED EXACT SOLUTION FOR THE (l, d)-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.

Grade : 1
Date : October 24, 2016

ACKNOWLEDGMENTS

First, I would like to thank my God and my Savior Jesus Christ for all the blessings He has given to me and all the lessons He has taught me. I also extend my deepest gratitude to my adviser Dr. Proceso Fernandez for teachings and the knowledge that he has imparted to me. To the Computer Algorithms and Analysis lab people and my peers, thank you for all the support. Lastly, I also would like to thank my panelists Dr. Andrei Coronel, Dr. Julieta Nabos and Mr. Alfredo De Vera.

This work is funded and supported by the DOST's Engineering Research and Development for Technology (ERDT).

Thank you and let's all aim for excellence in our lives. May God bless us all.

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. One variation of the motif finding problem 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 explored a way to improve the d-neighborhood generation by focusing on non empty blocks in the candidate motifs array. We improved the Hamming distance computation by pre-computing mismatch values. 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 17.02%, 18.13%, 25.67%, 14.13% and 21.65% 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 93.97%, 89.02%, 82.24%, 57.29%

and 5.00% for the (9, 2), (11, 3), (13, 4), (15, 5) and (17, 6) synthetic challenge instances respectively.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

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-complete, 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. An l-mer that is common in all sequences in the dataset, considering the number of mutations allowed d, is called a motif.

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 $d_H(x, y) = |\{i \mid x[i] \neq y[i], 1 \leq i \leq l\}|$, where $x[i]$ and $y[i]$ are the i^th characters of l-mers x and y.

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

The underlined characters represent 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. $\texttt{\underline{a}tta\underline{g}ct}$ and $\texttt{\underline{g}tta\underline{c}ct}$ 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 $N(x, d)$ of all l-mers x' with Hamming distance, of at most d, from x, i.e., $d_H(x, x') \leq d$.

Ex. `ccgga`, `ccaaa`, and `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 $\mathcal{N}(S, d)$ 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 $\Sigma = \{\texttt{a}, \texttt{c}, \texttt{g}, \texttt{t}\}$, 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

The EMS-GT algorithm 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. The EMS-GT2 algorithm extends the 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. It also introduces pruning conditions that improves the runtime of the algorithm. The EMS-GT algorithm was developed recently and 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, referred to as the EMS-GT2 algorithm. The objectives of the study are as follows:

1. To explore additional speedup techniques that may be used to 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 compare the performance of the EMS-GT2 against its predecessor EMS-GT and the qPMS9 algorithm using synthetic datasets.

1.3 Research questions

This study aims to answer the main question: How can the EMS-GT be improved to handle challenging instances more efficiently?

These are the supporting sub-questions:

1. 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?
2. Can some Hamming distance pre-computation be developed to further speedup the EMS-GT?
3. 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 [26]. EMS-GT2 is not yet designed to run on multiple processors and, thus, we evaluate the algorithms in a single-processor execution. Finally, 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).

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 Expectation 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.

The Bit-based algorithm [6] generates the neighborhood of each sequence and intersects it to get the set of motif. It 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 part 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.

Given a tuple of l-mers T, PMS8 uses the consensus total distance of T as a lower bound to check if an l-mer M is a common neighbor of the l-mers in T. The consensus total distance of tuple T is computed by $Cd(T) = \sum_{i=1}^{l} k - m_i$, where k is the number of l-mers in the tuple T and m_i is the maximum frequency of any character in column i. 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.

The qPMS9 algorithm is an extension of the PMS8 algorithm. In PMS8, everytime it adds an l-mer in the tuple T, it reorders the remaining string based on the number of alive l-mers left in increasing order. This is improved in qPMS9 by changing the criteria into prioritizing strings with the largest minimum additional distance.

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 two main 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 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. Figure 2.1 shows a snapshot of the compressed bit array data structure. 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;

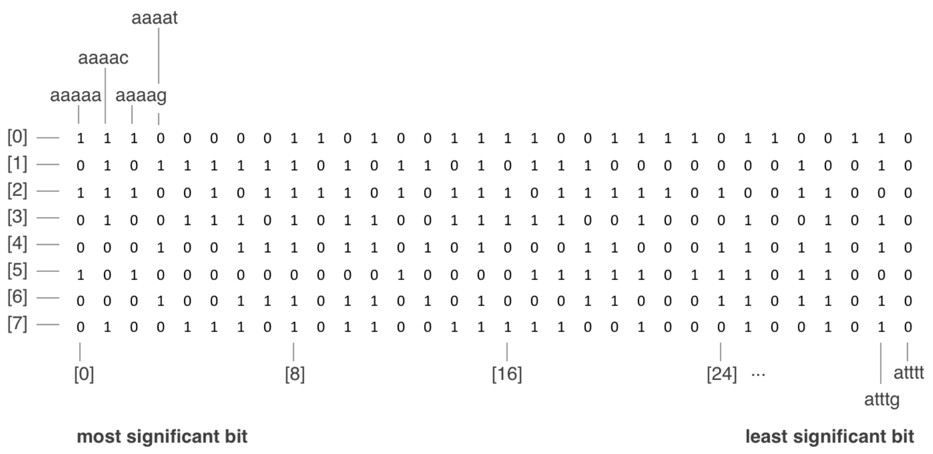
array index = 539 / 32 = 16;

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

of the integer at array index 16.

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 positions. Counting this nonzero pairs of bits result to the hamming distance value. An example of this computation is shown



```
[0] — 1 1 1 0 0 0 0 0 1 1 0 1 0 0 1 1 1 1 0 0 1 1 1 1 0 1 1 0 0 1 1 0
[1] — 0 1 0 1 1 1 1 1 1 0 1 0 1 1 0 1 0 1 1 1 0 0 0 0 0 0 0 1 0 0 1 0
[2] — 1 1 1 0 0 1 0 1 1 1 1 0 1 0 1 1 1 0 1 1 1 1 1 0 1 0 0 1 1 0 0 0
[3] — 0 1 0 0 1 1 1 0 1 0 1 1 0 0 1 1 1 1 1 0 0 1 0 0 0 1 0 0 1 0 1 0
[4] — 0 0 0 1 0 0 1 1 1 0 1 1 0 1 0 1 0 0 0 1 1 0 0 0 1 1 0 1 1 0 1 0
[5] — 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 1 1 0 1 1 1 0 1 1 0 0 0
[6] — 0 0 0 1 0 0 1 1 1 0 1 1 0 1 0 1 0 0 0 1 1 0 0 0 1 1 0 1 1 0 1 0
[7] — 0 1 0 0 1 1 1 0 1 0 1 1 0 0 1 1 1 1 1 0 0 1 0 0 0 1 0 0 1 0 1 0
```

Figure 2.1. Example of 4^5 search space showing its first 8 rows with random flag values.

below:

Ex. `aacgt` maps to `0000011011`

$\underline{\texttt{tacgc}\text{ maps to }\texttt{1100011001}}$

XOR produces $\underline{\texttt{11}}\texttt{000000}\underline{\texttt{10}}$ = 2 mismatches.

(Note, the mismatches are counted per pair)

Recursive neighborhood generation

The Generate phase of the algorithm produces the d-neighborhood of a string sequence by generating the d-neighborhood of all l-mers in that sequence. The 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. Figure 2.2 illustrates the recursion tree of the neighborhood generation. Given this, the expected size of $N(x, d)$ can be computed using the equation:

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

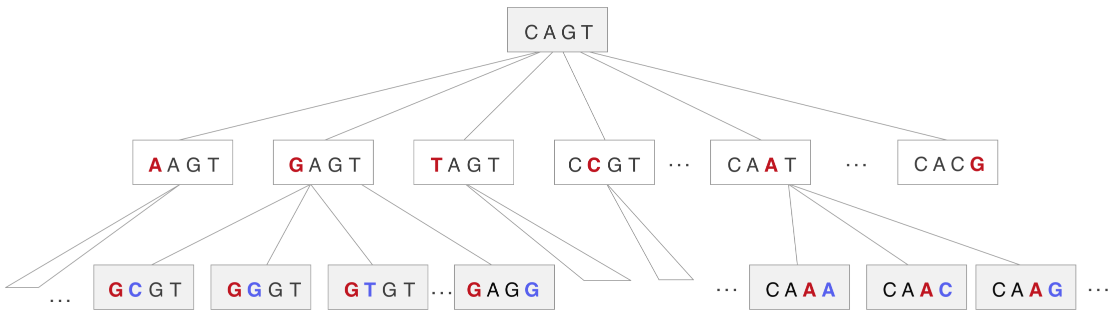


Figure 2.2. Recursively generating the neighborhood of l-mer CACGT with $d = 2$

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 compressed 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.

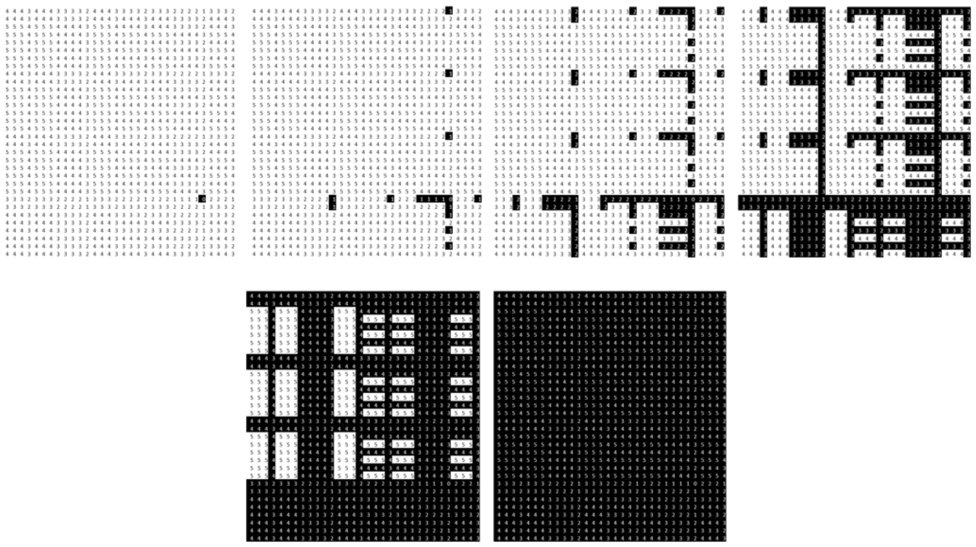


Figure 2.3. Bit patterns followed by blocks of size $4^5 = 32 \times 32$ in the bit-based representation of $\mathcal{N}(\texttt{acgtacgtacgt}, 5)$. Black signifies a 1. There are $(5+2) = 7$ possible patterns—the empty pattern (all 0s) is not shown. Images from [25]

The speedup-technique 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 $d' = d - d_H(y, y')$. As defined in the study, 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.

EMS-GT with this speedup technique has proven its competitiveness against

Algorithm 2.1 BLOCK PATTERN GENERATION

Input: Block degree k

Output: 3D bit-array $\mathcal{P}$ containing all possible non-trivial block patterns

```
1: $\mathcal{P}[\,][\,][\,] \leftarrow \{\}$                ▷ retrieve a pattern $P$ as $\mathcal{P}[z][d - d_{y'}]$
2: for $z \leftarrow 0$ to $4^k$ do
3:     for $j \leftarrow 1$ to $k - 1$ do
4:         for $z' \leftarrow 0$ to $4^k$ do
5:             if $dH(z, z') \leq j$ then
6:                 $\mathcal{P}[z][j][z'] \leftarrow 1$
7:             else
8:                 $\mathcal{P}[z][j][z'] \leftarrow 0$
9:             end if
10:        end for
11:    end for
12: end for
13: return $\mathcal{P}$
```

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 improve the EMS-GT algorithm so that it outperforms qPMS9 for challenges up to (17, 6).

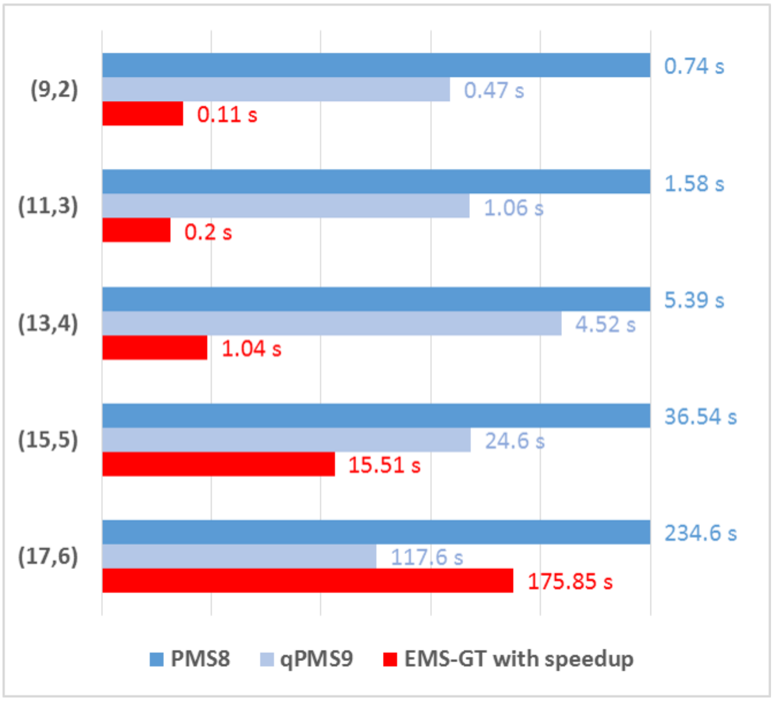


Figure 2.4. Improved EMS-GT's performance vs. PMS8 (baseline) and qPMS9. Image from [25]

CHAPTER III

METHODOLOGY

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

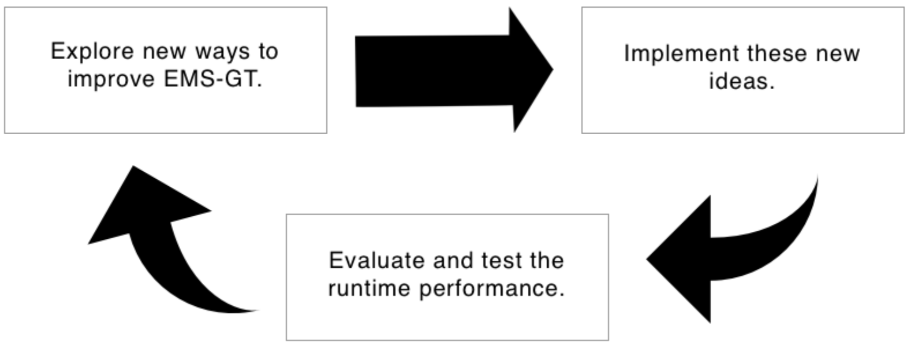

Explore new ways to improve EMS-GT.
Implement these new ideas.
Evaluate and test the runtime performance.

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 following 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 unnecessary variables that may affect the evaluation of the algorithms, it was converted to C++. This is the same language used in the development of the PMS8 and qPMS9 algorithms.

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 problem. 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 $1 \leq n'' \leq n'$. 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.

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 technique and pre-computed hamming distance values over different n'' values.

3.3 Evaluation

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 fast candidate motif elimination, (4) EMS-GT with fast candidate motif elimination and improved Hamming distance computation and (5) EMS-GT with block boolean flags, fast candidate motif elimination and improved Hamming distance computation. We also evaluated EMS-GT2 with its predecessor EMS-GT and with the qPMS9 algorithm.

3.4 Datasets

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

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 datasets that were previously used in the earlier implementation of EMS-GT. The real datasets are the sets of promoter sequences of yeast (*Saccharomyces cervisiae*) [27] and sets of orthologous sequences of different gene families of eukaryotes. The study [17] used these real datasets.

CHAPTER IV

RESULTS AND DISCUSSION

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, 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, given a height block value h, the v-th block boolean flag array is defined as:

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

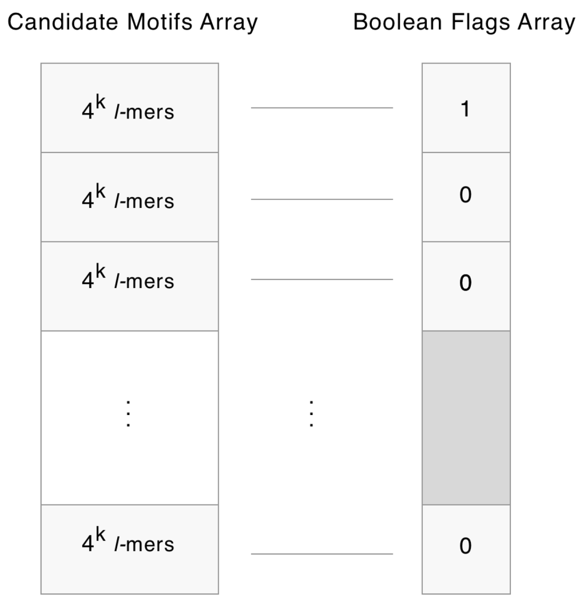


Figure 4.1. Assigning boolean flags for every 4^k l-mers block in the candidate motifs array.

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 we can skip the block bits setting by checking if the block is empty using its assigned block boolean flag.

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 4.2 illustrates a part of the search space's representation.

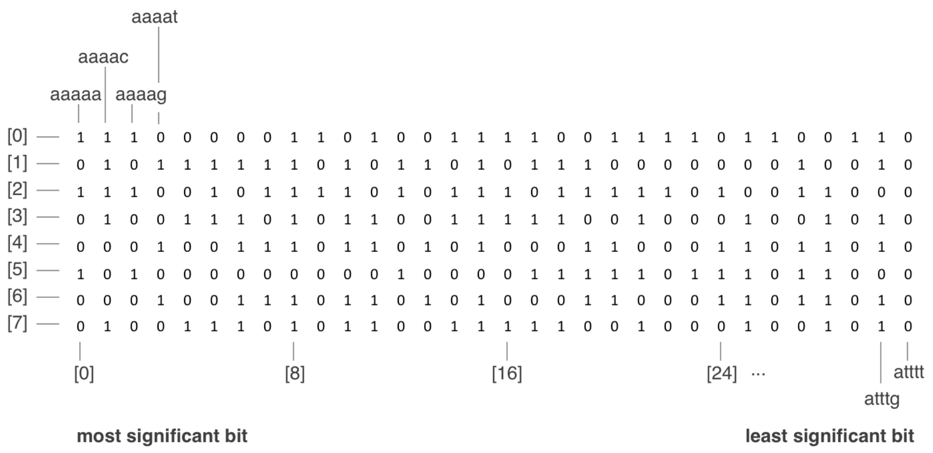


Figure 4.2. First 8 rows of the 4^5 search space with random flag values.

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 4.3 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

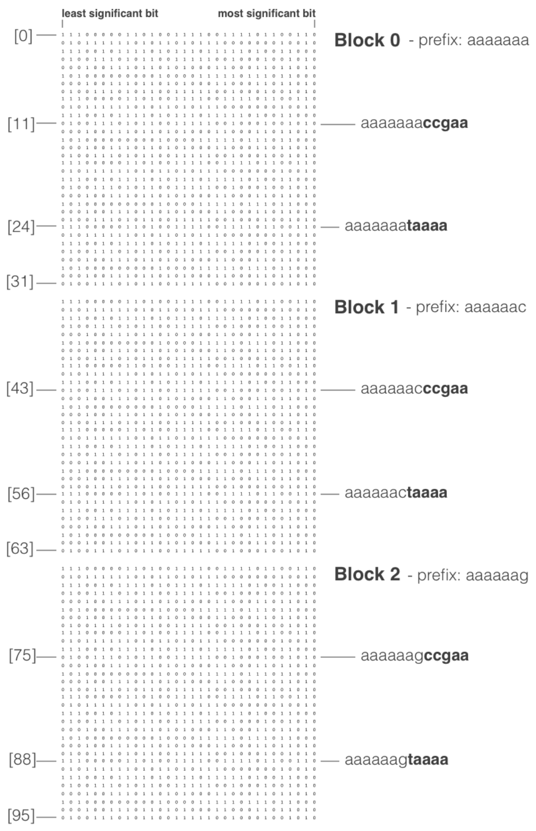


Figure 4.3. Illustration of the block partitioning of a 4^{12} search space. There are 4^5 l-mers in each block and has a height of 32. The illustration shows the first 3 blocks only.

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 4.4 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 $N(y, d)$

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 $d_H(y, z) > d$. □

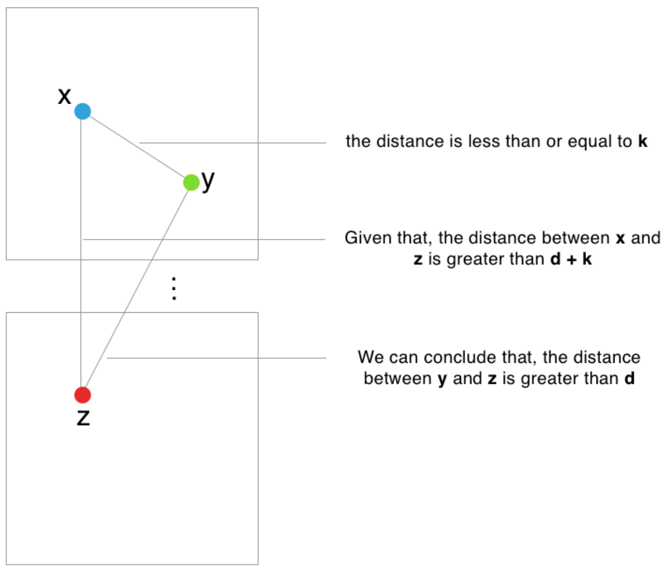

x
y
z
the distance is less than or equal to **k**
Given that, the distance between **x** and **z** is greater than **d + k**
We can conclude that, the distance between **y** and **z** is greater than **d**

Figure 4.4. 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.

4.3 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 for the original EMS-GT, the hamming distnace of two binary represented l-mers, can be efficiently computed using 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 4.5 shows a snapshot of this table of pre-computed number of non-zero pairs of bits.

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.

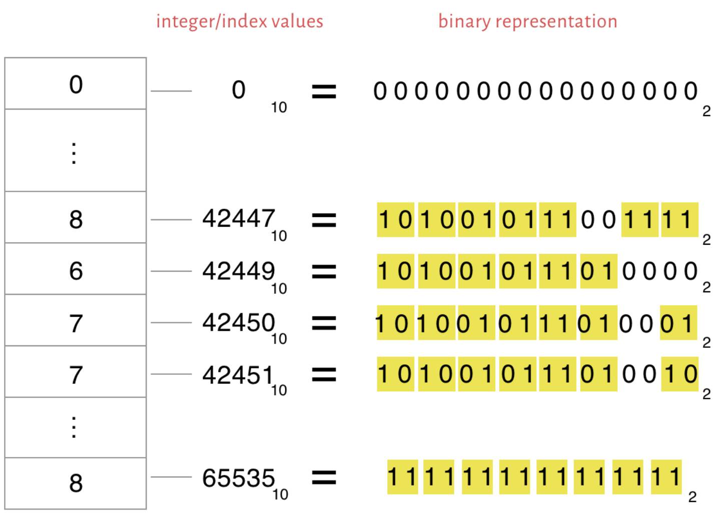


Figure 4.5. An illustration of the pre-computed number of non-zero pairs of bits up to 65535 integer value for the hamming distance value computation.

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 4.6 shows a demonstration of the improved hamming distance computation.

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 hamming distance between any given pair of l-mers.

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** $d_H(u, v)$
```

4.4 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 fastest 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 12.77%, 18.13%, 2.15%, 5.03% and 4.23% for (9, 2), (11, 3), (13, 4), (15, 5) and (17, 6) challenge instances respectively.

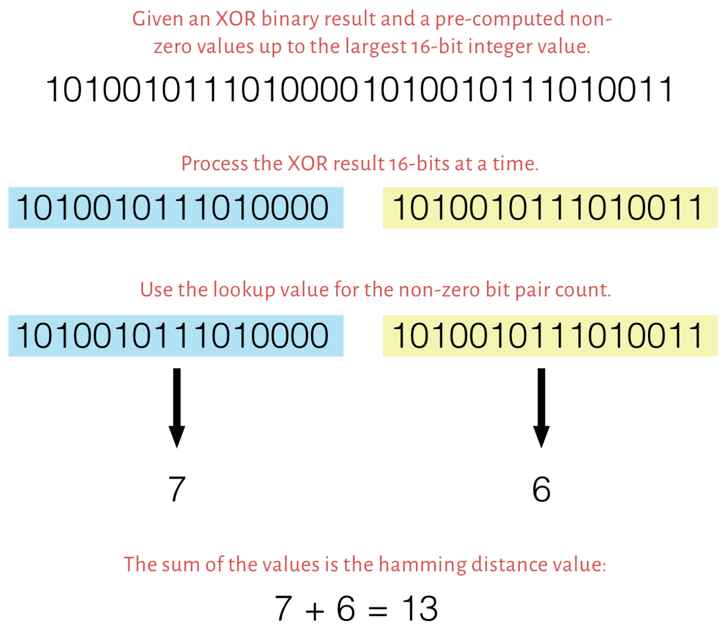

Given an XOR binary result and a pre-computed non-zero values up to the largest 16-bit integer value.
10100101110100001010010111010011
Process the XOR result 16-bits at a time.
1010010111010000
1010010111010011
Use the lookup value for the non-zero bit pair count.
1010010111010000
1010010111010011
7
6
The sum of the values is the hamming distance value:
7 + 6 = 13

Figure 4.6. A demonstration of the improved hamming distance value computation.

4.4.1 Evaluation of block boolean flag (BBF) technique

Table 4.1 shows the runtime performance of block boolean flag technique with its corresponding runtime reductions from the original EMS-GT.

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

(l, d)	**EMS-GT**	**EMS-GT with BBF**	**% speedup**
(9, 2)	0.047 s	0.053 s	-
(11, 3)	0.171 s	0.189 s	-
(13, 4)	1.020 s	0.951 s	6.76 %
(15, 5)	12.304 s	11.684 s	5.03 %
(17, 6)	143.341 s	132.195 s	7.77 %

Table 4.1. EMS-GT with block boolean flags technique (BBF).

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 4.7 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.

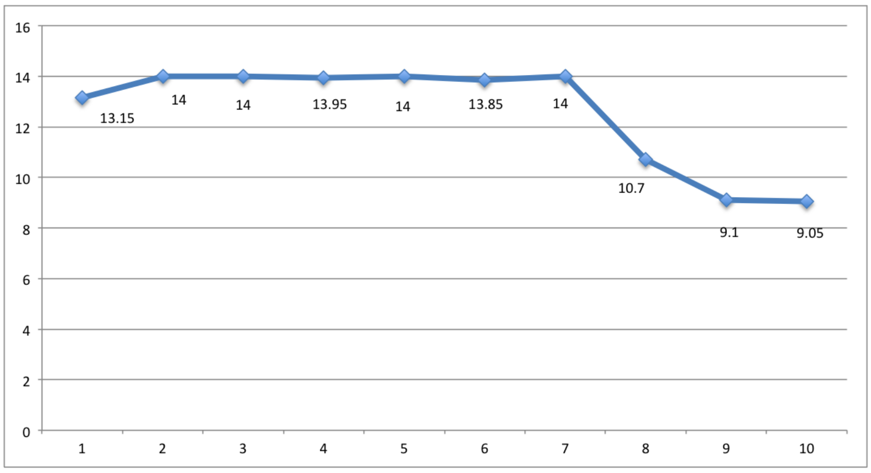


Figure 4.7. Runtime (s) of d-neighborhood generation of sequence for all sequence in Generation Phase using (17, 6) problem instance.

Additionally, we incorporated the improved hamming distance computation technique with block boolean flags technique resulting into faster runtime performance, which is shown in Table 4.2.

(l, d)	**EMS-GT**	**EMS-GT with BBF and IHD**	**% speedup**
(9, 2)	0.047 s	0.041 s	12.77 %
(11, 3)	0.171 s	0.124 s	27.49 %
(13, 4)	1.020 s	0.767 s	24.80 %
(15, 5)	12.304 s	11.195 s	9.01 %
(17, 6)	143.341 s	131.153 s	8.50 %

Table 4.2. EMS-GT with block boolean flag technique (BBF) and improved hamming distance computation (IHD).

4.4.2 Evaluation of fast candidate motif elimination (FCE) technique

Runtime performance of fast candidate motif elimination technique is shown in Table 4.3. Fast candidate motif elimination is used in 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.

EMS-GT with fast candidate motif elimination technique alone only improved the runtime in (13, 4) and (17, 6) only by 2.15% and 4.23% respec-

(l, d)	**EMS-GT**	**EMS-GT with FCE**	**% speedup**
(9, 2)	0.047 s	0.058 s	-
(11, 3)	0.171 s	0.188 s	-
(13, 4)	1.020 s	0.998 s	2.15 %
(15, 5)	12.304 s	12.822 s	-
(17, 6)	143.341 s	137.276 s	4.23 %

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

tively, 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 of the algorithm with these speedup techniques. Table 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)	0.047 s	0.039 s	17.02 %
(11, 3)	0.171 s	0.140 s	18.13 %
(13, 4)	1.020 s	0.758 s	25.67 %
(15, 5)	12.304 s	10.565 s	14.13 %
(17, 6)	143.341 s	112.311 s	21.65 %

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

Finally, Table 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)	0.047 s	0.041 s	12.77 %
(11, 3)	0.171 s	0.137 s	19.88 %
(13, 4)	1.020 s	0.842 s	17.45 %
(15, 5)	12.304 s	10.826 s	12.01 %
(17, 6)	143.341 s	112.046 s	21.83 %

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

4.5 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 instance where $l \leq 17$. Table 4.6 shows the runtime results between EMS-GT2 vs EMS-GT, while Table

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	0.039 s	17.02 %
(11, 3)	0.171 s	0.140 s	18.13 %
(13, 4)	1.020 s	0.758 s	25.67 %
(15, 5)	12.304 s	10.565 s	14.13 %
(17, 6)	143.341 s	112.311 s	21.65 %

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)	0.647 s	0.039 s	93.97 %
(11,3)	1.276 s	0.140 s	89.02 %
(13,4)	4.269 s	0.758 s	82.24 %
(15,5)	24.737 s	10.565 s	57.29 %
(17,6)	118.226 s	112.311 s	5.00 %

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 outperforms the qPMS9 in all of the (l, d)-challenge instances where $l \leq 17$. The efficiency of qPMS9 algorithm improves as the l value increases, due to the fewer l-mer tuple combinations to consider. The performance of EMS-GT improves as l value decreases, since there are fewer candidate motifs to consider. This is one reason why the previous implementation of EMS-GT was unsuccessful in beating qPMS9 in $(17, 6)$-challenge instance. The EMS-GT maintains a parameter n' that directly affects the number of candidate motifs left to process in the Test phase. Fixing the value of n' for all challenge instances fails to maximize the efficiency of the additional speedup techniques used in the Test Phase. Thus the re-evaluation of the optimal n' value, for all challenge instances considered in this study, helped improved the overall runtime performance of EMS-GT2. The parameter fine tuning returned the optimum n' values 10, 10, 9, 8, and 7 for the (9, 2), (11, 3), (13, 4), (15, 5) and (17, 6) challenge instances respectively.

(l, d)	**EMS-GT**	**qPMS9**	**EMS-GT2**
(9,2)	0.047 s	0.647 s	**0.039 s**
(11,3)	0.171 s	1.276 s	**0.140 s**
(13,4)	1.020 s	4.269 s	**0.758 s**
(15,5)	12.304 s	24.737 s	**10.565 s**
(17,6)	143.341 s	118.226 s	**112.311 s**

Table 4.8. EMS-GT, qPMS9 and EMS-GT2 runtime evaluation.

Aside from evaluating the EMS-GT2 algorithm on synthetic datasets, it was also run using real datasets. EMS-GT2 algorithm was able to find quickly the motifs in set promoter sequences of yeast (Saccharomyces cerevisiae) [16] as shown in Table 4.9. Table 4.10 shows the result of the algorithm run on real datasets involving orthologous sequences of different gene families of eukaryotes.

Transcription Factor/ Dataset Size [t, n]	**Published & Detected Motif**	(l, d) **Used**	**Run Time**
PHO4	CACGTG	(6, 2)	0.05 s
$[4, 250]$	CACGTT	(6, 2)	0.05 s
MCB	ACGCGT	(6, 2)	0.03 s
$[5, 250]$			
PDR3	TCCGTGAA	(8, 2)	0.04 s
$[7, 250]$	TCCGCGAA	(8, 2)	0.04 s

Table 4.9. Test on promoter sequences of Saccharomyces cerevisiae.

Even though the implementation of EMS-GT2 can only run in (l, d)-challenge instances where $l \leq 17$ due to a significant amount of memory requirement, studies have shown that typical length of motifs is around 10 base pairs (bp) [26].

4.6 Difficulties and Failures Encountered

This study explored numerous ways to improve the EMS-GT algorithm. In line with this, the study also faced numerous difficulties and failures in developing the speedup techniques for the algorithm.

A pruning technique was developed in the d-neighborhood generation of a sequence. The idea is that, while recursively generating all possible prefix given an l-mer x in a sequence, the algorithm can prune some recursive calls if x has d-neighbor in the set of already processed l-mers in that sequence. To implement this technique, the algorithm has to check at each level of the recursion tree if the node l-mer is one of the already processed d-neighbors to prune. The technique's efficiency depends on the number of d-neighbor and their distance value with the current l-mer. The problem is that for all (l, d)-challenge instances considered in this study, the number of d-neighbors to prune and their distance number with the current l-mer failed to compensate for the additional computational time the technique introduced.

Additionally, loading the pre-computed block patterns in a file, instead of generating it everytime, and iteratively generating the neighborhood instead of recursive generation did not improve the runtime performance of the algorithm significantly.

Gene Family/ Dataset Size [t, n]	**Published & Detected Motif**	(l,d) **Used**	**Run Time**
Insulin	AAGACTCTAA	(10, 4)	0.09 s
$[8, 500]$	GCCATCTGCC	(10, 3)	0.05 s
	CTATAAAG	(8, 2)	0.05 s
	GGGAAATG	(8, 2)	0.03 s
Metallothionein	GCTATAAA	(8, 3)	0.05 s
$[26, 590]$	CATGCGCAG	(9, 3)	0.03 s
	TTTGCACACG	(10, 3)	0.06 s
	TAACTGATAAA	(11, 5)	0.63 s
	TACACTCAG	(9, 3)	0.03 s
	CAGGCACCT	(9, 3)	0.03 s
	GTACATTGT	(9, 3)	0.06 s
c-myc	GTTTATTC	(8, 1)	0.01 s
$[7, 1000]$	TTGCTGGG	(8, 2)	0.03 s
	GGCGCGCAGT	(10, 3)	0.06 s
	CAGCTGTTCC	(10, 3)	0.07 s
	CCCTCCCC	(8, 1)	0.01 s
	AGCAGAGGGCG	(11, 4)	0.23 s
	GGCGTGGG	(8, 3)	0.04 s
	ATCTCCGCCCA	(11, 3)	0.10 s
c-fos	GAGTTGGCTG	(10, 3)	0.04 s
$[6, 700]$	GTTCCCGTCAATC	(13, 5)	1.68 s
	CACAGGATGT	(10, 4)	0.06 s
	AGGACATCTG	(10, 4)	0.05 s
	TACTCCAACCGC	(12, 4)	0.22 s
Growth Hormone	GGGAGGAG	(8, 3)	0.03 s
$[16, 380]$	ATTATCCAT	(9, 4)	0.06 s
	TTAGCACAA	(9, 3)	0.03 s
	GTCAGTGG	(8, 3)	0.03 s
	ATAAATGTA	(9, 4)	0.06 s
	TATAAAAAG	(9, 3)	0.05 s
	TCATGTTTT	(9, 4)	0.05 s
Histone H1	CAATCACCAC	(10, 3)	0.06 s
$[4, 650]$	AAACAAAAGT	(10, 3)	0.06 s

Table 4.10. Test on orthologous sequences of genes of eukaryotes.

CHAPTER V

CONCLUSION

In this study we have presented EMS-GT2, an improved exact solution for the planted-motif search problem. This study 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 pre-computed 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 17.02%, 18.13%, 25.67%, 14.13% and 21.65% 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 93.97%, 89.02%, 82.24%, 57.29%, and 5.00% 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 Rhone-alpes, 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.

APPENDIX A

Source Code Repository of the EMS-GT2 algorithm

The source codes and codes used for experimentations are stored in github repository. The link for the repository is https://github.com/markronquillo/thesis.

APPENDIX B

ECBA 2016 Conference Paper

Accepted and Presented

EMS-GT2: An Improved Exact Solution for the (*l, d*)-Planted Motif Problem

Mark Joseph D. Ronquillo and Proceso L. Fernandez

Ateneo de Manila University, Loyola Heights, Quezon City

Author Note

Mark Joseph D. Ronquillo, Department of Information System and Computer Science, Ateneo de Manila University.

Proceso L. Fernandez PhD., Department of Information System and Computer Science, Ateneo de Manila University.

The study was funded by DOST-ERDT scholarship grant. Aside from the financial support, the sponsors do not have any involvement in the study.

Contact Info: markronquillo23@gmail.com

Abstract

Finding DNA motifs is a widely studied area in the field of Computational Biology. Motifs signify different information that is useful for biologists. 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. 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 20.3%, 15.8% and 22.6% for the (*l, d*) challenge instances (13, 4), (15, 5) and (17, 6) respectively. It also outperforms qPMS9, having runtime reductions of 91.6%, 79.3%, 82.0%, 59.4% and 9.7% for the (9, 2), (11, 3), (13, 4), (15, 5) and (17, 6) synthetic challenge instances respectively.

Keywords: planted (*l, d*)-motif problem, bit-based, exact enumerative algorithm

EMS-GT2: An Improved Exact Solution for the (*l, d*)-Planted Motif Problem

DNA motif finding is a well-studied topic in computational biology. A motif is a short pattern of interest that occurs in large amount of biological data. Detection of these motifs often leads to new biological discoveries. This may lead to finding transcription factor binding sites that help biologists understand gene functions, understand human diseases, identify potential therapeutic drug targets and conclude commonalities from different species.

There are many variations of the motif search problem, such as the Simple Motif Search (SMS), the Edit-distance-based Motif Search (EMS) and the Planted Motif Search (PMS) which is also known as (*l, d*)-motif search. This study focuses on an exact enumerative algorithm for the PMS problem.

Planted Motif Search is formally defined in (Rajasekaran, Balla, & Huang, 2005) 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 are not 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 (Evans, Smith, & Wareham, 2003).

Some of the terms used in this study are defined below:

- An *l*-mer is a string of length *l* in a DNA sequence of length m where $l < m$.
- The Hamming distance d_H between two *l*-mers, of equal length, is equal to the number of positions where they have mismatches. *Ex.* d_H(acttgca, actaaga) = 3.
- 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*.

• The *d*-neighborhood of an *l*-mer *x* is the set *N*(*x*, *d)* of all *l*-mers with at most *d* Hamming distance from x. i.e., $d_H(x, x') \leq d$. Ex. ccgga, ccaaa, and gctta are all in *N*(cctta, 2), where $l = 5$

• The *d*-neighborhood of a sequence *S* is the set ***N***(*S*, *d*) of all *d*-neighbors of all *l*-mers in sequence *S*. Ex. ***N***(aattacg, 2) = *N*(aatta, 2) ∪ *N*(attac, 2) ∪ *N*(ttacg, 2) where $l = 5$.

In this study we introduce EMS-GT2, an improvement of EMS-GT that is an exact enumerative algorithm for the planted motif problem consisting of two phases -- Generate and Test. The EMS-GT2 algorithm leverages on the block processing of candidate motifs during its Test phase. The original EMS-GT algorithm was previously evaluated using (*l*, *d*)-challenge instances (9, 2), (11, 3), (13, 4), (15, 5) and (17, 6). It was already showed in (Sia, Nabos, & Fernandez Jr., 2015) that the current implementation of EMS-GT is faster than the then state-of-the-art algorithms PMS8 and qPMS9 for all (*l, d*)-challenge instances mentioned except in (17, 6). Even though that EMS-GT2 algorithm can only solve (*l, d*)-instances when $l \leq 17$, studies has shown that DNA motif lengths are usually around 10 bp (i.e., base pairs, or the number of characters in the string representation) in eukaryotes and 16 bp in prokaryotes (Stewart, Hannenhalli, & Plotkin, 2012). Thus, the EMS-GT2 algorithm is still significant in practical biological applications.

Literature Review

Motif finding has been studied extensively in the previous years. Numerous algorithms have been made for motif finding and for PMS. Each of these algorithms may be categorized as either an approximate algorithm or an exact algorithm. Approximate algorithms, although they are fast, do not guarantee the exact solution all the time. Heuristic algorithms for PMS that perform local search such as Gibbs Sampling (Lawrence et al., 1993), Expectation Maximization (EM) (Bailey, Williams, Misleh, & Li, 2006), Projections (Buhler & Tompa, 2001; Huo, Zhao, Stojkovic, & Liu, 2009) have been 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. They then iteratively refine the alignment until a certain criteria is met. MEME (Bailey et al., 2006) is a tool for motif finding that implements Expected Maximization. WINNOWER (Pevzner, Sze, et al., 2000) 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 (Keich & Pevzner, 2002), PatternBranching, ProfileBranching (Price, Ramabhadran, & Pevzner, 2003) and CONSENSUS (Hertz & Stormo, 1999).

Exact algorithms are not as fast as approximate algorithms but are guaranteed to return the correct answer for any given problem instance. 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 or *l*-mers from all string sequences. This approach is called sample- driven. Another approach is called pattern-driven that checks from $|\Sigma|^l$ possible *l*-mers which are the motifs over a base alphabet Σ.

Rajasekaran et al. proposed a series of exact algorithms for the (*l, d*)-motif search problem. The algorithm PMS1 (Rajasekaran et al., 2005) 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 a set of motifs. PMSi and PMSP (Davila, Balla, & Rajasekaran,

2006) 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 (Davila, Balla, & Rajasekaran, 2007) works very similar to PMP 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 (Dinh, Rajasekaran, & Kundeti, 2011) and PMS6 (Bandyopadhyay, Sahni, & Rajasekaran, 2014) extend the ideas of PMS1 and PMSPrune. PMS5 generates the common neighborhood of three *l*-mers from different sequences at a time and uses Integer Linear Programming for the pruning process. PMS6 only differs from PMS5 in the way it determines the three *l*-mers. Quorum PMS (qPMS) 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* where $q \leq n$. We can see that a qPMS problem is equal to PMS when $q = n$. The qPMS7 (Dinh, Rajasekaran, & Davila, 2012) 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 (Nicolae & Rajasekaran, 2014) is an algorithm that combines the sample-driven approach and the pattern-driven approach. First, it chooses *k*-tuple *T* of *l*-mers from *k* different sequences and it makes sure that all *l*-mers in *T* have a common neighbor. Each *l*-mer that belongs to the common neighborhood of the tuple *T* is checked if it appears in the remaining $n - k$ sequences. One of the current state-of-the-art algorithms, qPMS9 (Nicolae & Rajasekaran, 2015), improves the sample-driven approach of PMS8 by prioritizing *l*-mers that are highly distant from those already in the tuple, resulting in a smaller size of common *d*-neighborhood to test and enables the algorithm to process the quorum version of the PMS problem.

Several exact algorithms solve the PMS problem using suffix trees and other related data structures. RISO (Carvalho et al., 2005), RISOTTO (Pisanti, Carvalho, Carvalho, Sagot, & Marsan, 2006), SPELLER (Sagot, 1998) and SMILE are all exact algorithms that use suffix trees. MITRA (Eskin & Pevzner, 2002) improves the excessive memory requirement of sample-driven approach by using a mismatch tree. Two other algorithms that have some similarity with our algorithm are the Voting algorithm and Bit-based algorithm. Voting algorithm (Chin & Leung, 2005) 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 (Dasari, Desh, & Zubair, 2010) 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 $|\Sigma|^l$ to represent the neighborhood of a sequence and uses the integer representation of an *l*-mer to flag if it is 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.

The EMS-GT Algorithm

The Exact Motif Search - Generate and Test algorithm (EMS-GT) for the planted motif search problem is composed of 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 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 checking if *c* has at least one *d*-neighbor in each of the remaining *n – n'* string sequences. The phases are formally defined below.

(a) Generate candidates

This step operates on the first *n*' sequences. The intersection of the *d*-neighborhood of each sequence results to the set of candidate motifs *C*.

$$C = \boldsymbol{N}(S_1,d) \cap \boldsymbol{N}(S_2,d) \cap ... \cap \boldsymbol{N}(S_{n'},d). \quad (1)$$

(b) Test candidates

Each candidate motif in *C* is evaluated if it has a *d*-neighbor in all of the remaining *n – n'* string sequences. If a candidate motif passes the test, it is then included in the set of motifs *M*.

Speedup Strategies. To further improve the algorithm performance, we introduced a number of speedup strategies, and these techniques are defined in this section:

1. Integer mapping of *l*-mers**.**

   The *l*-mers are represented using binary representation of integer values. 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.

2. Bit-based set representation and *l*-mer enumeration.

   The EMS-GT implementation 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.

3. *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. An illustration on how the algorithm accesses the bit flag is provided 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.

4. *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 positions. Counting this nonzero bit pair 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)

5. *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 *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 starting at the $p + 1$ 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* (see Figure 1). Given this, the expected size of *N*(*x*, *d*) can be computed using the equation:

$$|N(x,d)| = \sum_{i-0}^{d} \binom{l}{i} 3^i \qquad (2)$$

6. Block-based optimization for neighborhood generation .

The way our implementation of the algorithm represents the neighborhood of 4^l bit flags array opens up a new way to improve the generation of neighborhood *N*(*x*,*d*). The algorithm maintains an array of 32-bit integers where each bit represents an *l*-mer. Setting the bit value to 1 means that the *l*-mer is in *N*(*x*, *d*), otherwise 0 means it is not. The improvement generates the neighborhood by blocks of size *k* where $0 < k < l$ instead of per bit. It is observed that dividing the bit array *N* into 4^k consecutive blocks results to blocks conforming to one of the $(k + 2)$ possible block patterns. For each possible *k*-mer, block patterns are pre-generated according to the remaining number of allowed mismatch *d'* where $0 \leq d' \leq d$. There is no need to pre-generate the block pattern for 0 value since it only corresponds to a block pattern where exactly one *l*-mer is set and the block pattern for *d* value since it corresponds to a block pattern where all bits are set.

Given an *l*-mer *x*, the generation of *N*(*x*, *d*) now divides the *l*-mer to its $(l - k)$-length prefix *y* and its *k*-length suffix *z*. The algorithm then generates all prefixes in *N*(*x*, *d*) by recursively generating *N*(*y*, *d*). For each prefix $y' \in N(y, d)$, a block pattern is applied to the neighborhood array based on the remaining number of allowed mismatch *d'* computed using $d - d_H(y, y')$ and the suffix *z*. This is based on the observation that the distance between two *l*-mers is equal to the sum of the distance between their prefixes and the distance between their suffixes. The run time complexity for generating the *d*-neighborhood of *l*-mer *x* is now reduced to $O(4^k \sum_{i=0}^{l-k} \binom{l-k}{i} 3^i)$.

Methodology

This section states how the EMS-GT2 and the proposed speedup techniques were implemented and evaluated using synthetic datasets for some challenge (*l*, *d*) instances. A study (Nicolae & Rajasekaran, 2015) proposed that (*l*, *d*) instances where *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) are categorized as *challenging instances*. The following (*l*, *d*) instances (9, 2), (11, 3), (13, 4), (15, 5) and (17, 6) are the only challenge instances where *l* is between 9 and 17, and these challenge instances were used for the evaluation of the algorithms.

Datasets

Algorithms that solve PMS (Pevzner et al., 2000; Nicolae & Rajasekaran, 2014, 2015) 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.

Implementation

The EMS-GT2 maintains a 4^l motif search space that is represented by array of bits for space efficiency. The previous speedup technique (used in EMS-GT) exploits this manner of representing the search space by generating the neighborhood of an *l*-mer by blocks instead of per bit. In this study, we took advantage of this block-processing approach 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, then 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*.

Parameter Fine Tuning

The EMS-GT2 defines an integer value *n'* ($1 < n' < n$) that divides the dataset into two set of sequences. The first *n'* sequences are used in the Generate phase while the remaining is assigned to the Test phase. Previous experimentations (Sia et al., 2015) showed that it is efficient for the algorithm to set the value of *n'* to 10. Technically, *n'* dictates how big is the size of the set of candidate motifs *C* to be evaluated if they are in the remaining $n - n'$ sequences. In line with this, 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'* range from 5 to 10 in this experiment, since we only want to make the candidate motif set large enough for our speedup technique to take effect.

Table 1 shows the ideal *n'* value for (*l*, *d*)-challenge instances. 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 problem. For (13, 4), (15, 5) and (17, 6), different *n'* were used in the evaluation which are 9, 8 and 7 respectively.

Table 1

Runtime evaluation of the algorithm with speedup over different n' values.

Sequence	**(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	**119.94 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

Evaluation

For evaluation of the algorithms, we compared the EMS-GT2 to EMS-GT and the algorithm qPMS9. We used the challenging (*l*, *d*) instances defined in (Davila et al., 2007; Nicolae & Rajasekaran, 2015). The instances used in the evaluation are the following: (9, 2), (11, 3), (13, 4), (15, 5) and (17, 6).

Results

This section describes EMS-GT2 with the two proposed speedup techniques and briefly discusses the observation where the idea for the improvement originated. Runtime evaluation of EMS-GT2 is also discussed in this section.

Faster Candidate Motif Elimination through Block Processing

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* does not have any *d*-neighbor, then 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 (see Figure 2 for the illustration of the search space).

In EMS-GT, each testing of candidate motif is independent of each other. We propose here a new 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 $(4^k/32)$ (see Figure 3 for an example of a partitioned search space). Additionally, any two *l*-mers within a block has at most *k* hamming distance value between them.

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 $(n' \le i \le n)$, 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 them for testing the remaining candidate motifs in the block along with the other *l*-mers in the remaining sequences in $\{S_{n'}, S_{n'+1}, ..., S_n\}\backslash\{S_i\}$. The theorem below formalizes the main property used in this speedup technique.

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 $N(y, d)$

Proof. 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 $d_H(y, z) > d$.

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 32 rows of 32 bit flags representing a total of 4^5 *l*-mers.

Pre-computation of Mismatch Values

The EMS-GT2 uses the hamming distance computation heavily during the Test phase. As discussed earlier for the original EMS-GT, the hamming distance of two binary represented *l*-mers, can be efficiently computed using the boolean operator XOR. In EMS-GT2, instead of repeatedly counting this nonzero pairs of bits every time we compute the hamming distance, we use a pre-computed lookup table to help reduce computational time. 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 $l >= 10$ (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 requires $b = 2l'$ number of bits. Then we determine the hamming distance by looking up the nonzero counts in the XOR results, *b* number of bits at a time, as described in Algorithm 1. 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 hamming distance between any given pair of *l*-mers.

Performance of EMS-GT with speedup techniques

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 20 synthetic datasets for each (*l*, *d*)-challenge instance where $l \leq 17$. Table 2 shows the runtime results between EMS-GT2 vs. EMS-GT while Table 3 shows the runtime results between the EMS-GT2 vs. the qPMS9.

Table 2.

EMS-GT and EMS-GT2 runtime evaluation.

(*l, d*)	EMS-GT	EMS-GT2	% speedup
(9, 2)	0.04 s	0.05 s	-
(11, 3)	0.17 s	0.26 s	-
(13, 4)	1.03 s	0.82 s	20.3%
(15, 5)	12.39 s	10.43 s	15.8%
(17, 6)	143.87 s	111.22 s	22.6%

The additional speedup techniques become more effective as the *l* value in the (*l*,*d*)-instance grows. For every (*l*,*d*)-challenge instances mentioned where $l \geq 13$, EMS-GT2 has improved the runtime over the EMS-GT by at least 15%. Unfortunately, the speedup techniques in EMS-GT2 failed to compensate for their additional overhead computations in both (9, 2) and (11, 3) challenge instances and failed to improve the overall runtime of the implementation. Previous implementations of the original EMS-GT failed to beat qPMS9 in (17, 6)-challenge instance. The proposed EMS-GT2 not only produced improved runtimes but also succeed in beating the qPMS9 in this challenge instance. The improved EMS-GT is now faster than the state-of-the-art 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 the typical length of motifs is around 10 base pairs (bp) (Stewart et al., 2012) anyway.

Table 3.

EMS-GT2 and qPMS9 runtime evaluation.

(*l, d*)	qPMS9	EMS-GT2	% speedup
(9, 2)	0.60 s	0.05 s	91.6%
(11, 3)	1.26 s	0.26 s	79.3%
(13, 4)	4.58 s	0.82 s	82.0%
(15, 5)	25.73 s	10.43 s	59.4%
(17, 6)	123.17 s	111.22 s	9.7%

Conclusions

We have presented EMS-GT2, an improved exact solution for the planted-motif search problem. EMS-GT2 was able to efficiently test candidate motifs within the same block by filtering out *l*-mers using a property of the search space array that we have discovered and proven in this paper. 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 original algorithm's performance on (13, 4), (15, 5) and (17, 6) and was able to beat qPMS9 in all (*l*, *d*)-challenge instances where $l \leq 17$.

References

Bailey, T. L., Williams, N., Misleh, C., & Li, W. W. (2006). MEME: discovering and analyzing DNA and protein sequence motifs. *Nucleic Acids Research*,*34*(Web Server issue), W369–W373. http://doi.org/10.1093/nar/gkl198. 369-373

Bandyopadhyay, S., Sahni, S., & Rajasekaran, S. (2014). Pms6: a fast algorithm for motif discovery. *International Journal of Bioinformatics Research and Applications*, 10(4-5), 369-383.

Buhler, H., & Tompa, M. (2001). Finding motifs using random projections. *In Proceedings of the fifth annual International Conference on Computational Biology* (pp. 69 – 76). New York, NY, USA: ACM. doi: 10.1145/369133.369172.

Carvalho, A. M., Freitas, A. T., Oliveira, A. L., Rhone-alpes, I., Bernard, U. C., & I, L. (2005). A highly scalable algorithm for the extraction of cis-regulatory regions. *In Proc. APBC`05*. Imperial College Press.

Chin, F. Y., & Leung, H. C. (2005). Voting algorithms for discovering long motifs. *In Proceedings of the third Asia-Pacific Bioinformatics Conference (APBC)* (p. 261-271)

Dasari, N. S., Desh, R., & Zubair, M. (2010). An efficient multicore implementation of planted motif problem. *In High performance computing and simulation (HPCS), 2010 international conference on* (pp. 9–15).

Davila, J., Balla, S., & Rajasekaran, S. (2006). Space and time efficient algorithms for planted motif search. In V. N. Alexan- drov, G. D. van Albada, P. M. A. Sloot, & J. Dongarra (Eds.), Computational Science – ICCS 2006: 6th international conference, reading, uk, may 28-31, 2006. Proceedings, Part II (pp. 822–829). Berlin, Heidelberg: Springer Berlin Heidelberg

Davila, J., Balla, S., & Rajasekaran, S. (2007). Fast and practical algorithms for planted (l, d) motif search. *Computational Biology and Bioinformatics, IEEE/ACM Transactions on*, 4(4), 544–552.

Dinh, H., Rajasekaran, S., & Davila, J. (2012, 07). qPMS7: A fast algorithm for finding, (l,d)-motifs in DNA and Protein Sequences. *PLoS ONE, 7(7),* 1-8.

Dinh,H.,Rajasekaran,S.,& Kundeti,V.K.(2011).Pms5:an efficient exact algorithm for the (l ,d)-motif finding problem. *BMC Bioinformatics*, 12(1), 1–10. doi: 10.1186/1471-2105-12-410

Eskin, E., & Pevzner, P. A. (2002). Finding composite regulatory patterns in DNA sequences. *Bioinformatics, 18* (suppl 1), S354–S363.

Evans, P. A., Smith, A. D., & Wareham, H. (2003). On the complexity of finding common approximate substrings. *Theoretical Computer Science*, 306(1-3), 407 - 430. Retrieved from http://www.sciencedirect.com/science/article/pii/S0304397503003207 doi: http://dx.doi.org/10.1016/S0304-3975(03)00320-7

Hertz, G. Z., & Stormo, G. D. (1999). Identifying DNA and protein patterns with statistically significant alignments of multiple sequences. Bioinformatics, 15(7), 563–577.

Huo, H., Zhao, Z., Stojkovic, V., & Liu, L. (2009). Combining genetic algorithm and random projection strategy for (l, d)-motif discovery. *In Bio-inspired computing*, 2009. BIC-ta'09 (pp. 1–6).

Keich, U., & Pevzner, P. (2002). Finding motifs in the twilight zone. *Bioinformatics*, 18(10), 1374-1381.

Lawrence, C. E., Altschul, S. F., Boguski, M. S., Liu, J. S., Neuwald, A. F., & Wootton, J. C. (1993). Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment. *Science*, 262(5131), 208–214.

Nicolae, M., & Rajasekaran, S. (2014). Efficient sequential and parallel algorithms for planted motif search. *BMC Bioinformatics*, 15(1), 34.

Nicolae, M., & Rajasekaran, S. (2015). qPMS9: An efficient algorithm for quorum Planted Motif Search. *Scientific reports*, 5.

Pevzner, P. A., Sze, S.-H., et al. (2000). Combinatorial approaches to finding subtle signals in DNA sequences*. In ISMB* (Vol. 8, pp. 269–278).

Pisanti, N., Carvalho, A. M., Carvalho, R. M., Sagot, M.-F., & Marsan, L., (2006). RISOTTO: Fast extraction of motifs with mismatches*. In Proceedings of the 7th Latin American Theoretical Informatics Symposium*, 3887 of LNCS:757-768 (pp. 757–768).

Price, A., Ramabhadran, S., & Pevzner, P. A. (2003). Finding subtle motifs by branching from sample strings. *Bioinformatics*, 19(suppl 2), ii149-ii155.

Rajasekaran, S., Balla, S., & Huang, C. (2005). Exact algorithms for planted motif challenge problems. Journal of Computational Biology, 1117-1128.

Ronquillo, M. (2016). (*l, d)*-challenge instances for the Planted Motif Search Problem. Retrieved from *https://www.researchgate.net/publication/305828021_l_d-challenge_instances_for_the_Planted_Motif_Search_Problem* . DOI: 10.13140/RG.2.1.2692.8884

Sagot, M.-F. (1998). Spelling approximate repeated or common motifs using a suffix tree.

Sia, A., Nabos, J., & Fernandez Jr., P. (2015). An Efficient Exact Solution to the (l,d)-planted motif problem. *8th AUN/SEED-Net Regional Conference on Electrical and Electronics Engineering*.

Stewart, A. J., Hannenhalli, S., & Plotkin, J. B. (2012). Why transcription factor binding sites are ten nucleotides long. *Genetics*, 192(3), 973–985.

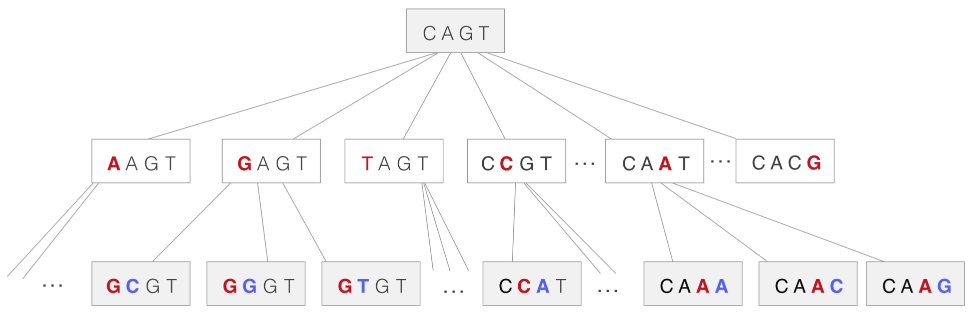


Figure 1: Illustration of the recursive generation of neighborhood.

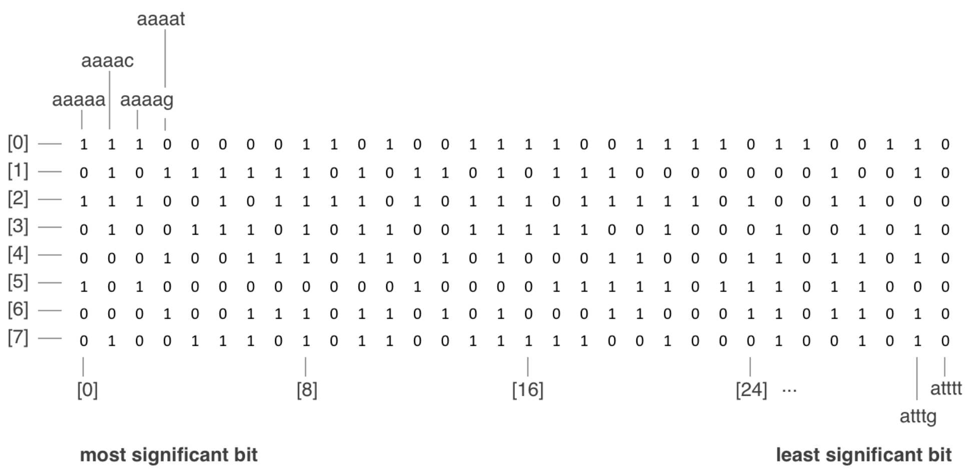


Figure 2. First 8 rows of the 4^5 search space with random flag values.

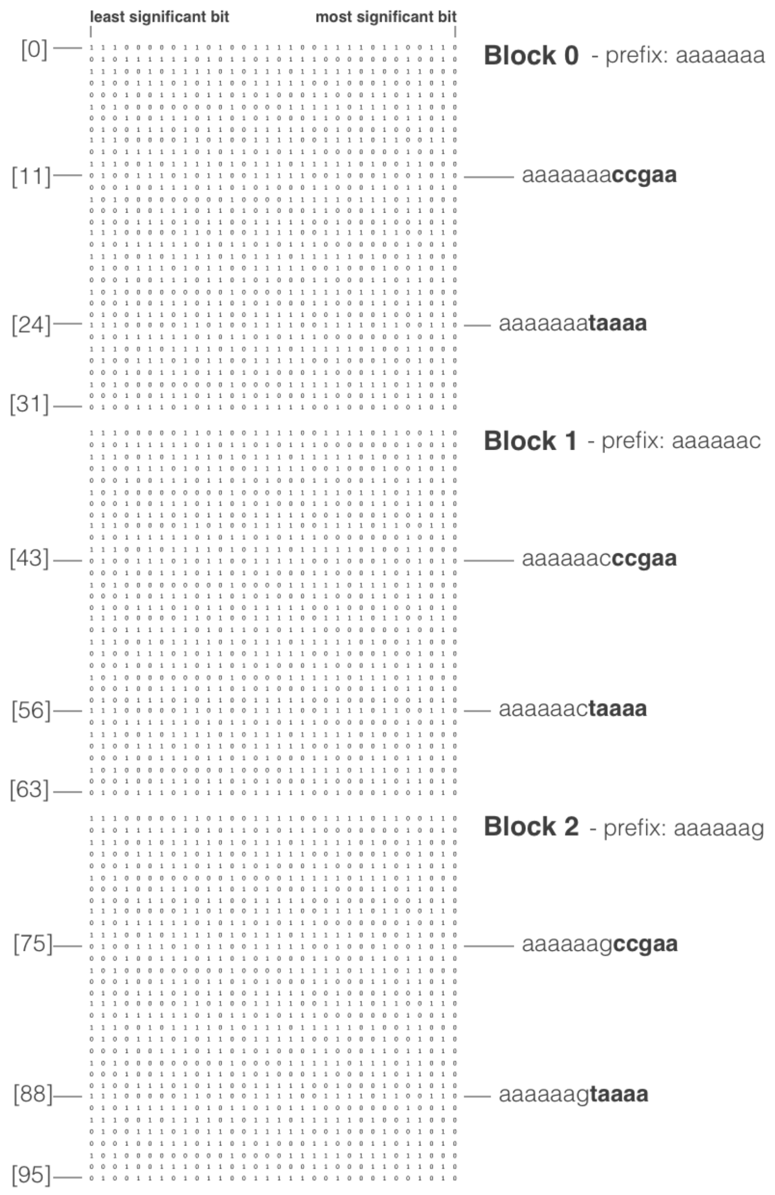


Figure 3. Illustration of the block partitioning of a 4^{12} search space. There are 4^5 *l*-mers in each block and has a height of 32. The illustration shows the first 3 blocks only.

Algorithm 1 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:     $l \leftarrow z\&((1 << 18) - 1)$ ▷ process 18 bits at a time
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 $d_H(u,v)$
```

Algorithm 1. Shows the improved way of computing the mismatch value for the Hamming distance computation