An Improved Exact Solution for Planted Motif Search Problem

A Thesis

Presented to the

Faculty of the Graduate School

Ateneo de Manila University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

By

Mark Joseph D. Ronquillo

2016

The thesis entitled:

An Improved Exact Solution for Planted Motif Search Problem

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

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

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

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

An Improved Exact Solution for Planted Motif Search Problem

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

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

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

Grade: **Very Good**

Date: 11 November 2013

Abstract

Finding DNA motifs is a widely studied area in the field of Computational Biology. Motifs signify different information that are useful for biologists like discovering transcription factor-binding sites and helps them identify potential therapeutic drug target. There are several variations of the motif finding problem, and one of these is called the (l,d)-motif search or Planted Motif Search problem (PMS). In this paper, we propose the EMS-GT2 algorithm, an extension of the Exact Motif Search - Generate and Test (EMS-GT) which is an exact enumerative algorithm for PMS. In EMS-GT2, we incorporated a new speedup technique that is based on an important property that we have discovered, which we prove in this paper, and which has enabled a more efficient block-processing of candidate motifs. 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.

TABLE OF CONTENTS

CHAPTER

CHAPTER I

Introduction

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

There are many variations of the motif search problem such as Simple Motif Search (SMS), Edit-distance-based Motif Search (EMS) and Planted Motif Search (PMS) also known as (l, d)-motif search. This study focuses on the exact enumerative algorithm for the PMS problem. Planted Motif Search is formally defined in [22] as *Input are t sequences of length n each. Input also are two integers l and d. The problem is to find a motif (i.e., a sequence) M of length l. It is given that each input sequence contains a variant of M . The variants of interest are sequences that are at a hamming distance of d from M.* In solving the PMS problem, traditional string matching won't be efficient since these biological motifs are not typically exact but are subject to mutations. As a matter of fact, PMS has already been proven to be NP-hard, which means that it is very unlikely to

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

In this study we introduce EMS-GT2 an improved version of the EMS-GT algorithm that implements a series of speedup techniques that use the block processing, introduced in [24], 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 in the alphabet Σ. In this study, Σ = {a, c, g, t} since we are dealing with DNA sequences. Formally, an l-mer is an element of the set Σ^l.

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

DEFINITION 2. Hamming distance d_H

The **Hamming distance d_H** between two l-mers, of equal length, is equal to the number of positions where the l-mers have mismatches.

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

the underlined characters represents the mismatch positions.

DEFINITION 3. d-neighbor

An l-mer x is considered a ***d*-neighbor** of another l-mer y if the Hamming distance between the two is at most d.

Ex. Given two 7-mers, $\underline{\texttt{a}}\texttt{tta}\underline{\texttt{g}}\texttt{ct}$ and $\underline{\texttt{g}}\texttt{tta}\underline{\texttt{c}}\texttt{ct}$ are d-neighbors,

where $d \geq 2$.

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

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

Ex. $\texttt{ccgga}$, $\texttt{ccaaa}$, and $\texttt{gctta}$ are all in $N(\texttt{cctta}, 2)$ where $l = 5$.

DEFINITION 5. d-neighborhood of a sequence S

The ***d*-neighborhood of a sequence S** is the set $\boldsymbol{\mathcal{N}(S, d)}$ of all d-neighbors of all l-mers in sequence S.

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

where $l = 5$.

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

Given a set of n sequences, where each sequence is of length m, an l and d integer values, 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 sequence exactly once at a random position.

1.2 Objectives of the study

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

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

1.3 Research questions

This study aims to answer the following questions:

- How can we further improve the runtime performance of EMS-GT algorithm?

- What is the runtime performance of the proposed EMS-GT2 on the synthetic datasets?
- What is the performance of the proposed EMS-GT2 compared to its predecessor EMS-GT and to qPMS9 algorithm?

1.4 Significance of the study

Motif signifies important information that is use 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.

Motif usually signifies important information that are useful in the field of Computational Biology.

EMS-GT algorithm is competitive and able to beat current 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 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 proved

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 introduces a memory contstraint when l is sufficiently large, each new speedup technique was evaluated using all synthetic datasets of (l, d)-challenge instance where $l < 18$. Finally, EMS-GT2 doesn't have the capability to run on multiple processors and, thus, we evaluate the algorithms in a single-processor execution.

CHAPTER II

Review of Related Literature

2.1 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. Approximate algorithms, although it is 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 iteratively refine the alignment until a certain criteria has met. MEME [1] is a tool for motif finding that implements Expected Maximization. Other approximate algorithms that use local search are GARP [14], GibbsDNA [16] and Random Projection [3, 14]. WINNOWER [19] reduces the PMS problem into 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 [21] and CONSENSUS [13];

Although it is not 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 or l-mers 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.

Rajasekaran et. al. proposed a series of exact algorithms for the (l,d)-motif search problem. The algorithm PMS1 [22] 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 [17] 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 has a common neighbor. Each l-mer that belongs to the common neighborhood of the tuple T will be checked if it appears in the remaining $n - k$ sequences. The current state-of-the-art algorithm qPMS9 [18] 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.

Many exact algorithms solve the PMS problem using suffix trees and other related data structures. RISO [4], RISOTTO [20], SPELLER [23] 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 are similar in how our algorithm works are the Voting algorithm and Bit-based algorithm. Voting algorithm [5] maintains a hash table that tracks the number the occurrence of every possible l-mer and makes sure that every l-mer is only counted once in each sequence. An l-mer is considered a motif if its total occurrences is equal to the total number of sequences in the dataset. Bit-based algorithm [6] generates the neighborhood of each sequence and intersects it to get the set of motif. Unlike PMS1, the Bit-based algorithm maps every l-mer to its corresponding integer value. It uses an array of size Σ^l to represent the neighborhood of a sequence and uses the integer representation of an l-mer to flag if its a member of the array. It generates the neighborhood of all sequences and merges it using the logical operator AND. The resulting array represents the set of motif.

CHAPTER III

Methodology

This section describes how the speedup-techniques was explored and implemented. Each algorithm was evaluated using the $(9, 2)$, $(11, 3)$, $(13, 4)$, $(15, 5)$ and $(17, 6)$ challenge 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) [18].

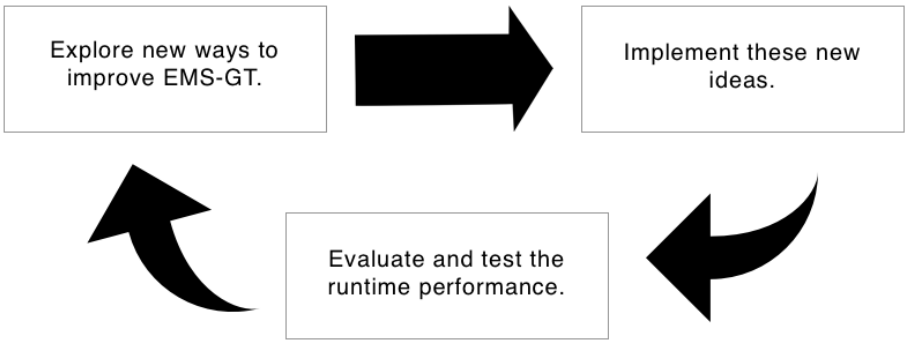

Figure 3.1. Speedup Technique Development Cycle.

The study aims to improve the algorithm by pre-computation of values and exploring other usage of the block-processing technique. The speedup techniques were developed using the development cycle shown in Figure 1.

3.1 Improving the EMS-GT algorithm

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

A previous study [24] improved the neighborhood generation by setting the neighborhood array by blocks of bits 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.

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. Lastly, we explored a pruning strategy in the neighborhood generation of a sequence.

3.2 Parameter Fine Tuning

The EMS-GT algorithm defines an integer value n' ($1 < n' < n$) 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 [24] 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 $\mathcal{C}$ to be evaluated if they are 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 a sufficiently large candidate motifs set for them to take effect. We run an experimentation that records the average runtime of the algorithm with the speedup techniques over 5 tests and having different values for n'. The values for n' ranges from 5 to 10 in this experiment.

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

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

$\boldsymbol{Sequence}$	$(9, 2)$	$(11, 3)$	$(13, 4)$	$(15, 5)$	$(17, 6)$
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	**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 speedup over different n' values.

Additionally, the block flags speedup technique also defines another parameter n'' where $1 < n'' < n'$. This parameter represents the sequence number where the algorithm will start using the block flags. We also run an experimentation to assess the best value for the parameter n''. Table 3.2 shows the runtime performance of EMS-GT with the block flags speedup technique using different n'' values. The experiments show that the ideal value for the n'' parameter is 6.

3.3 Evaluation

We first assess different combinations of speedup techniques for EMS-GT. The combination with the fastest runtime performance will become the proposed EMS-GT2. The different combinations are composed of the following: (1) EMS-

GT with pruning technique, (2) EMS-GT with block flags, (3) EMS-GT with block flags and improved Hamming distance computation, (4) EMS-GT with faster candidate motif elimination, (5) EMS-GT with faster candidate motif elimination and improved Hamming distance computation and (6) EMS-GT with block flags, faster candidate motif elimination and improved Hamming distance computation.

We compared EMS-GT2 to its predecessor EMS-GT and the algorithm qPMS9. We used the challenging (l, d) instances defined in [18]. The instances used in the evaluation are the following: (9, 2), (11, 3), (13, 4), (15, 5) and (17, 6).

3.4 Datasets

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

3.4.1 Synthetic Datasets

Algorithms that solve PMS [19, 17, 18] use a dataset containing 20 string sequences where each nucleotide is in $\Sigma = \{a, c, g, t\}$. Each string sequence is 600 base pairs (bp) long and each nucleotide is randomly generated with equal chance of being selected. A motif is then generated and for each string sequence in the dataset, a d-neighbor is planted at a random position. In this study, a

generator was run to produce a dataset of PMS problem instances with this configuration, and this dataset was then used to evaluate the algorithms. Furthermore, a converter program was used to translate the dataset into FASTA format in order to execute qPMS9.

3.4.2 Real Datasets

The EMS-GT2 algorithm was also evaluated using real data sets that were previously used in the earlier implementation of EMS-GT. The real data sets are the sets of promoter sequences of yeast (*Saccharomyces cervisiae*) [26] and sets of orthologous sequences of different gene families of eukaryotes.

CHAPTER IV

Results and Analysis

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

4.1 Pruning strategy in Neighborhood Generation of a Sequence

EMS-GT generates the d-neighborhood of a sequence by collecting all the d-neighbors of every l-mer in the sequence. It is easy to see that generating the neighborhood of an l-mer identical to the already processed l-mers would be a waste of time.

4.2 Block boolean flags strategy in Generate phase

The EMS-GT algorithm follows the pattern-driven approach where it exhaustively tests the 4^l bit-array of possible motifs. It quickly filters the 4^l bit-array in the Generate phase by intersecting the d-neighborhood of the first n' sequences. To be specific, EMS-GT maintains two 4^l bit-array during the Generate phase. The first array holds the remaining candidate motifs and the second array holds the d-neighborhood of the current sequence. For every sequence in the first n'

sequences, we generate its d-neighborhood by blocks and interects it with the candidate motifs array, the resulting set will be 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 masks in this part of the neighbordhood 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 strategy that tracks these empty blocks in the candidate motifs array.

4.2.1 Maintaining block boolean flags for the candidate motifs array

We divide the candidate motifs array 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 that array. Figure 1 illustrates the way of grouping these 4^k l-mers and its corresponding block boolean flags.

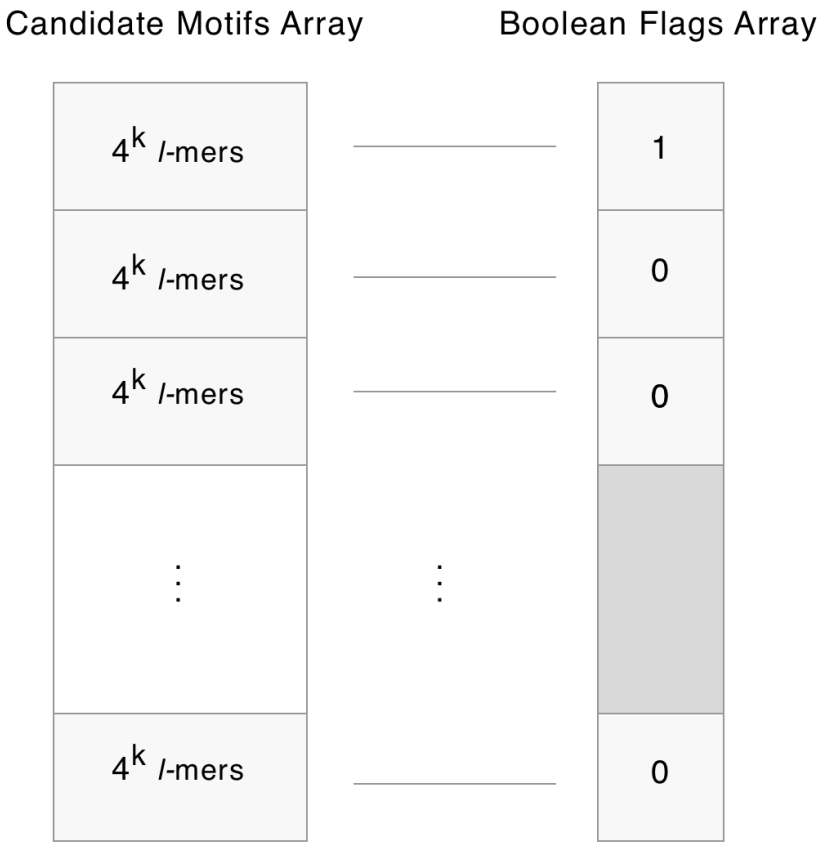

Figure 4.1. Assigning boolean flags for every 4^k l-mers block in the candidate motifs array.

4.2.2 Usage of block boolean flags strategy

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. Otherwise, it will only add up to the runtime of the implementation. We introduced a new parameter n'', where

$1 < n'' < n'$, that holds the sequence number where we will start using the block boolean flags strategy. Experiments have shown that it is best to use $n'' = 6$ in this study.

In d-neighborhood generation of a sequence, each block bits setting has a block start value and the block pattern to apply. Block start value represents a row in the neighborhood array where it will start applying its corresponding block pattern.

We use the block start value to check if the block is empty in the candidate motifs array. Specifically, we are considering two scenarios based on the block start value. If the block start value is divisilbe

Given our approach in maintaining the boolean flags, checking of block boolean flags considers if the block start value is the same with the row start value of the block boolean flag is representing or not.

4.3 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 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 1 illustrates a part of the search space's representation.

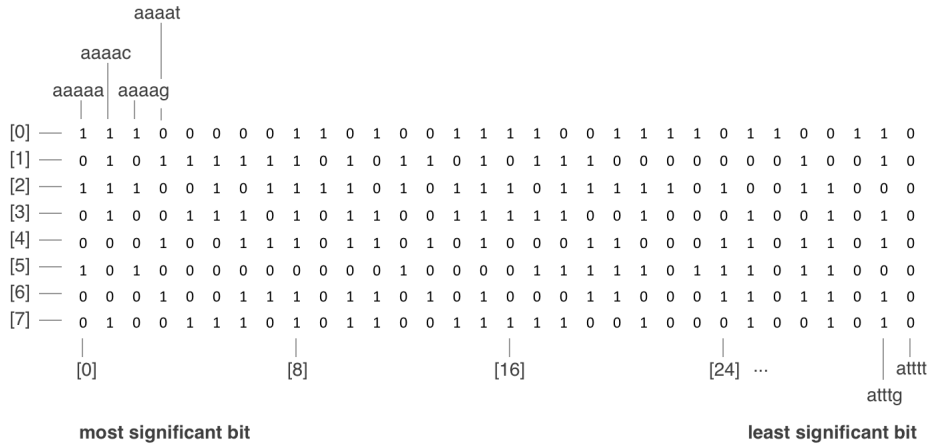

Figure 4.2. First 8 rows of the 4^5 search space with random flag values.

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 2

shows an example of partitioned search space showing blocks of l-mers sharing a common prefix string. In our implementation we use $k = 5$ and every block has a height of 32.

We process the testing of candidate motifs now by blocks. If candidate motifs x and y are within a block and x has been eliminated as a candidate motif in sequence S_i $(n' \leq i \leq n)$, we can filter out l-mers $z \in S_i$ where $d_H(x, z) > d+k$. We collect the remaining l-mers in S_i and use it for testing the remaining candidate motifs in the block along with the other l-mers in the remaining sequences in $S'_n, S'_n + 1, ..., S_n - S_i$. The theorem below formalized 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 and $d_H(x, y) \leq k$. Let d be the number of allowed mutations in the problem. Let z be another l-mer. If $d_H(x, z) > (d + k)$ then $d_H(y, z) > d$ and $z \notin N(y, d)$*

Proof of Theorem 1. Using proof by contradiction, suppose that $d_H(y, z) \leq d$. We know that $d_H(x, z) < d_H(y, z) + d_H(x, y)$ from the triangle inequality. We can write this equation into $d_H(x, z) < d + k$ thus a contradiction to the previous assumption that $d_H(x, z) > (d + k)$. □

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.

4.3.1 Pre-computation of Mismatch Values

The EMS-GT2 uses the hamming distance computation heavily on the Test phase. To compute the hamming distance of two binary represented l-mers, the algorithm uses the boolean operator XOR between the two l-mers and results an integer with nonzero pair of bits at every mismatch position. Counting these pair of bits results to the hamming distance value. The total number of pairs of bits is equal to the length of the motif and is also equal to the total number of comparisons it has to do in the hamming distance computation.

Instead of repeatedly counting this nonzero pairs of bits everytime we compute the hamming distance, using a lookup table of nonzero pair of bits count for all possible integer values will help save computational time. Although, pre-computing these nonzero pair counts for all possible 32 bit values will introduce an overhead computation problem. A more efficient approach is to pre-compute only up to b number of bits where $b < 32$ and b is an even number. Then we count for the hamming distance by looking up the nonzero counts b number of bits at a time as shown in Algorithm 1. In our experimentation the maximum bits that

we are considering to represent l-mers is 34 bits (for $(17, 6)$-instance). Given this, we pre-compute up to 18 bit (2^9) values only and use the lookup table twice for the computation of hamming distance.

4.4 Performance of EMS-GT with speedup techniques

The EMS-GT2, EMS-GT and qPMS9 was evaluated 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 $l \leq 17$. Table 4.1 shows the runtime results between EMS-GT2 vs EMS-GT while table 4.2 shows the runtime results between EMS-GT2 vs the state-of-the-art algorithm qPMS9.

Table 4.1. 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 efficient 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 for 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 in-

stances and failed to improve the overall runtime of the implementation.

Table 4.2. EMS-GT2 and qPMS9 runtime evaluation.

(l, d)	**EMS-GT2**	**qPMS9**	**% 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%

Previous implementations of EMS-GT2 failed to beat qPMS9 in $(17, 6)$-challenge instance. EMS-GT2 not only improved the runtime of the implementation but also succeed in beating the qPMS9 in this challenge instance. The implementation of EMS-GT now is 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 memory constraint, studies shown that the typical length of motifs is around 10 base pairs (bp) [25]

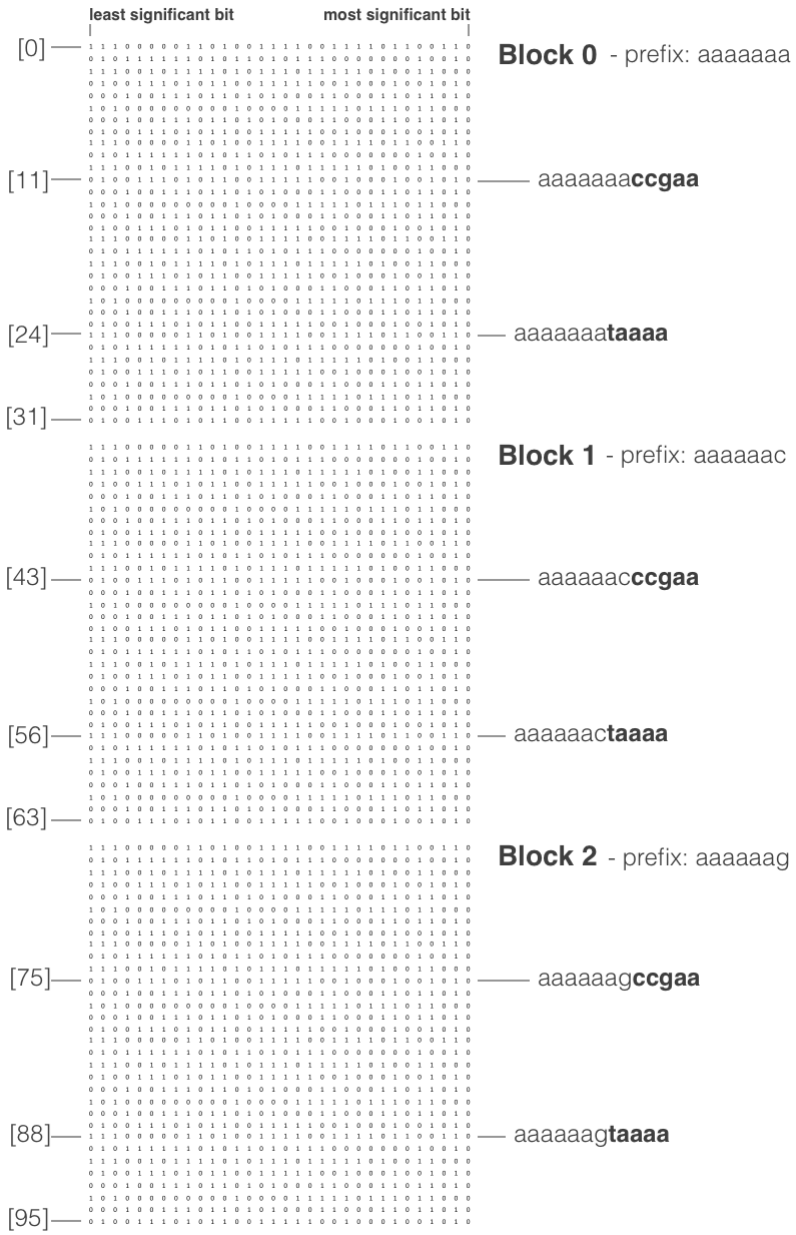

Figure 4.3. Illustration of the block partitioning of a 4^{12} search space. There are 4^5 l-mers in each block and has a height of 32. The illustration shows the first 3 blocks only.

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** $d_H(u, v)$

CHAPTER V

Conclusions

5.1 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 in block by filtering out l-mers using a property of the search space array that we have discovered. 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 $l \leq 17$.

BIBLIOGRAPHY

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[19] 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.

[20] 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.

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

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

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

[24] 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.

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

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