Optimizing An Exact Solution to the (l, d)-Planted Motif 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

Maria Clara Isabel D. Sia

2015

Abstract

DNA motif finding is widely recognized as a difficult problem in computational biology and computer science. Because of the usual large search space involved, exact solutions typically require a significant amount of execution time before discovering a motif of length l that occurs in an input set $\{S_1, ..., S_n\}$ of sequences, allowing for at most d substitutions.

This study implements a novel optimization to EMS-GT, a motif search algorithm which operates on a compact bit-based representation of the search space. The optimization takes advantage of distance-related patterns in the search space, in order to speed up the bulk bit-setting operations performed by the algorithm. A Java implementation is shown to be highly competitive against PMS8 and qPMS9, two current state-of-the-art exact algorithms. EMS-GT works extremely well for problems involving short motifs, outperforming both competitors for challenge instances with (l, d) values (9,2), (11,3), (13,4) and (15,5), showing runtime reductions of 76%, 81%, 77% and 37% respectively for these instances, while ranking second to qPMS9 for challenge instance (17,6).

TABLE OF CONTENTS

CHAPTER

CHAPTER I

Introduction

DNA motif finding is widely recognized as a difficult problem in computational biology and computer science. Motifs are sequences that occur repeatedly in DNA and have some biological significance [3]; a motif might be a transcription factor binding site, a promoter element, a splicing site, or a marker useful for classification. There are many variants of motif finding problem in the literature. Some look for a motif that repeatedly occurs in a single sequence. Others look for a motif that occurs over some or all of a set of DNA sequences [4]. One of the latter type is the planted motif problem.

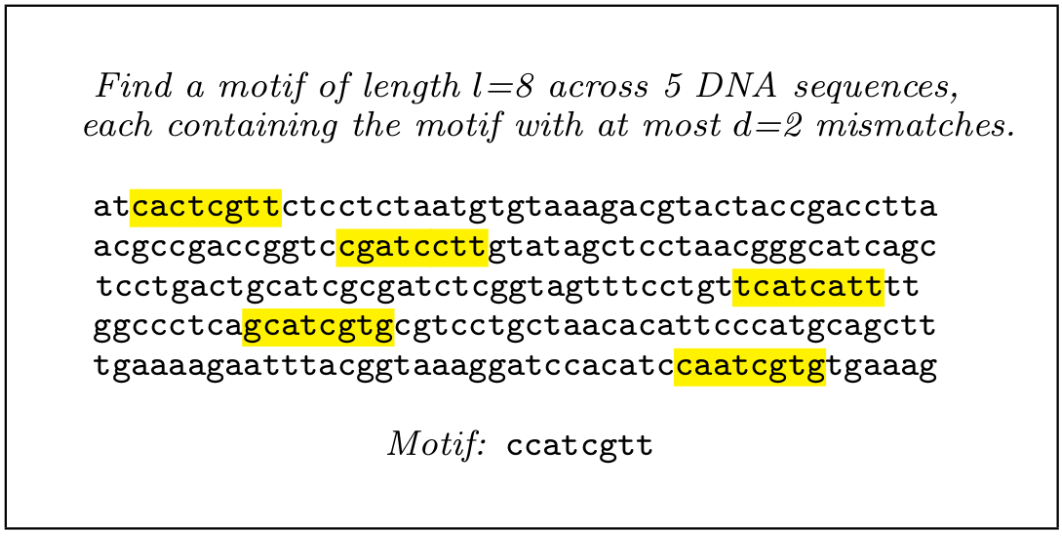

Figure 1.1. Sample instance of the planted motif problem.

The planted motif problem simply asks: "*Given a set of DNA sequences, can we find an unknown motif of length l that appears at different positions in each of the sequences* [13]?" Initially it seems an exhaustive string search will suffice for this problem. However, due to biological mutation, motif occurrences in DNA are allowed to differ from the original motif by up to d characters. This greatly impacts complexity: two distinct variants of a motif—both counting as valid occurrences of the motif—might differ in as many as $2d$ characters! Brute-force solutions quickly become infeasible as values of l and d increase. All of this shows why (l, d)-motifs are sometimes called "subtle" signals in DNA [13], and why finding them is difficult and computationally expensive. In fact, the motif finding problem has already been shown to be NP-complete [11].

This study is concerned with the EMS-GT (*Exact Motif Search - Generate and Test*) algorithm [10], which solves the planted motif problem for any arbitrary instance up to l=17. This study investigates certain Hamming distance-related patterns appearing in EMS-GT's bit-based representation of the motif search space, and uses these to develop a novel optimization for EMS-GT.

1.1 Context of the Study

This section formally defines the planted motif problem. It also defines key terms used throughout this paper in discussing exact motif-search algorithms.

DEFINITION 1. l-mer, Hamming distance, d-neighborhood

An ***l*-mer** is a sequence of length l. Given a sequence S of length $L > l$, the i^{th} l-mer in S starts at the i^{th} position. The **Hamming distance dH** between two l-mers of equal length is the number of characters that differ between them. "Distance" refers to Hamming distance in this paper, unless otherwise stated.

Ex. If l = 5, the second l-mer in `gattaca` is `attac`.

$dH(\texttt{gattaca}, \texttt{cgttaga}) = 3$.

The ***d*-neighborhood of an *l*-mer x** is the set $N(x, d)$ of all d-neighbors of x: all l-mers x' whose Hamming distance from x is at most d, i.e., $dH(x, x') \leq d$. Meanwhile, the ***d*-neighborhood of a sequence S** of length $L > l$ is the set $\mathcal{N}(S, d)$ of all d-neighbors of all the l-mers in S.

Ex. `gatct`, `cctta`, and `aatta` are all in $N(\texttt{gatta}, 2)$.

For l = 5, $\mathcal{N}(\texttt{gattaca}, 2) = N(\texttt{gatta}, 2) \cup N(\texttt{attac}, 2) \cup N(\texttt{ttaca}, 2)$.

DEFINITION 2. (l, d) Planted Motif Problem

We formally define the (l,d) planted motif problem as follows:

Given a set $\mathcal{S} = \{S_1, ...S_t\}$ of n DNA sequences of length L,

find M, the set of sequences (or motifs) of length $l < L$

which have at least one d-neighbor in each sequence in S.

1.2 Objectives of the Study

The main objective of this study is to improve the performance of the EMS-GT algorithm. Specifically, it aims:

1. To develop an optimization for EMS-GT that takes advantage of distance-related patterns in the motif search space.

2. To evaluate the resulting optimization with regard to improvement in runtime and solvable problem instances.

3. To evaluate the optimized version of EMS-GT against state-of-the-art motif search algorithms.

1.3 Research Questions

This study aims to answer the question: How can the performance of the EMS-GT algorithm be improved? Specifically, it aims to answer the following:

1. How can distance-related patterns observed within the motif search space be exploited in an optimization for EMS-GT?

2. What performance improvement does a pattern-based optimization produce, with regard to runtime and solvable problem instances?

3. How does the optimized version of EMS-GT compare with state-of-the-art motif search algorithms?

1.4 Significance of the Study

Motif finding in DNA and other types of nucleotide sequences is an important task in bioinformatics. Genome analysis requires fast, efficient algorithms to identify biological motifs which may be linked to protein synthesis, gene function, or even disease and targets for medical treatment.

Optimizing the EMS-GT algorithm, which was shown to be competitive with the state-of-the-art, results in an improved option for real-world motif finding applications. Furthermore, this study's investigation and insights regarding distance-related patterns within an organized search space may prove applicable to other types of search tasks, pattern-matching tasks, and problems involving Hamming distances.

1.5 Scope and Limitations of the Study

This study is concerned with integrating a novel bit-masking optimization into the existing Java implementation of EMS-GT. The optimized version is benchmarked by its runtime on synthetic datasets for challenging instances of the problem. (Unoptimized EMS-GT had been previously tested for correctness, us-

ing real biological data containing known motifs; re-testing would be redundant since the optimization does not change the algorithm's logic.) Furthermore, the performance of EMS-GT is compared to that of PMS8 and qPMS9 running on a single core. Although both competitor algorithms are also capable of using multiple procesors, parallelization is beyond the current scope of the optimizations explored for EMS-GT.

CHAPTER II

Review of Related Literature

Motif finding is a well-studied problem in computing. Various motif search algorithms have been developed, falling into two categories: *heuristic* and *exact*. This section gives an overview of algorithms of both types, and provides an in-depth description of the exact algorithm EMS-GT.

2.1 Heuristic Algorithms

Heuristic algorithms perform an iterative local search, for instance by repeatedly refining an input sampling or projection until a motif is found.

Gibbs sampling [8] and Expectation Maximization (EM), used in the motif-finding tool MEME [9, 1] both use probabilistic computations to optimize an initial random alignment. (An alignment is simply a vector $(a_1, a_2, ..., a_n)$ of n positions, which predicts that the motif occurs at position a_i in the given sequence S_i.) Gibbs sampling attempts to refine the alignment one position at a time; EM may recompute the entire alignment in a single iteration.

Projection [2] combines a pattern-based approach with EM's probabilistic approach, trying to guess every successive character of a tentative motif and using EM to verify its guesses. GARPS [7] uses a random version of the projec-

tion strategy, in tandem with the iteratively self-correcting Genetic Algorithm (GA), for yet another iterative approach. These are just some of many successful heuristic algorithms for motif finding.

2.2 Exact Algorithms

While they can be efficient, heuristic approaches are non-exhaustive and thus not always guaranteed to find a solution. Exact motif search algorithms, on the other hand, perform an exhaustive search of possible motifs and so always find the planted motif.

WINNOWER [13] and its successor MITRA [6] are exact algorithms that look at pairwise l-mer similarity to find motifs. In a set of DNA sequences, there are numerous pairs of "similar" l-mers, which come from different sequences and have Hamming distances of at most $2d$ from each other (meaning that they could possibly be two d-neighbors of the same l-mer). WINNOWER represents these pairs in a graph, with l-mers as nodes and edges connecting l-mer pairs. It then prunes the graph to identify "cliques" of pairs that indicate a motif. MITRA refines this graph representation into a mismatch tree containing all possible l-mers, organized by prefix. The tree structure allows MITRA to eliminate entire branches at a time, making it significantly faster than WINNOWER at removing the many spurious edges that are not part of any motif clique.

The current state-of-the-art in exact motif search is qPMS9, the most recent in a series [5, 11, 12] of Planted Motif Search algorithms. It performs a sample-driven step, which generates a k-tuple of l-mers from each of k input strings, followed by a pattern-driven step, which generates the common d-neighborhood of the tuple and then checks whether any of the l-mers in this common neighborhood is a motif. To identify neighbors, qPMS9 efficiently traverses the tree of all possible l-mers, using certain pruning criteria explored by predecessors PMSPrune and qPMS7 [5] to quickly discard non-neighbor branches. Sampling in qPMS9 is an improvement on its predecessor PMS8 [11]; in building a k-tuple, qPMS9 intelligently prioritizes l-mers that have fewer matches with the l-mers already selected, such that the common d-neighborhood becomes smaller and thus faster to check through. Finally, both PMS8 and qPMS9 have been implemented to run on multiple processors, allowing them to solve problem instances with (l, d) as large as (50, 21) in a few hours.

The BitBased algorithm [4] uses similar approaches to EMS-GT (see subsection 2.3.1): it also maps l-mers to binary strings, and uses an array of bits to represent the motif search space. The main difference is that BitBased is optimized for parallel computation on multiple cores, requiring specialized GPU hardware (Nvidia Tesla C1060 or S1070). BitBased is able to solve the challenge problem instance (21,8) in 1.1 hours.

2.3 EMS-GT

EMS-GT [10] is an exact motif search algorithm based on the candidate generate-and-test principle. It operates on a compact bit-based representation of the search space, identifying the common d-neighbors of the n given DNA sequences as motifs. The main idea of EMS-GT is to narrow down the search space to a small set of "candidate" motifs based on the first n' sequences, then do a brute-force search for each candidate on the remaining $(n - n')$ sequences to confirm whether or not it is a motif. EMS-GT's approach proceeds in two main steps:

1. *Generate candidates*

   This step takes the intersection of the d-neighborhoods of the first n' sequences $S_1, S_2, ..., S_{n'}$. Every l-mer in the resulting set C is a candidate motif.

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

2. *Test candidates*

   This step simply checks each candidate motif c in C, to determine whether a d-neighbor of c appears in all of the remaining sequences $S_{n'+1}, S_{n'+2}, ..., S_n$. If this is the case, c is accepted as a motif in set M.

   $$M = C \cap \mathcal{N}(S_{n'+1}, d) \cap ... \cap \mathcal{N}(S_n, d). \tag{2.2}$$

Algorithm 2.1 EXACT MOTIF SEARCH - GENERATE AND TEST

Input: set $S = \{S_1, S_2, ..., S_n\}$ of L-length sequences,
motif length l, allowable mismatches d

Output: set M of candidate motifs

```
C ← {}                                              ▷ generate candidates
N(S_1, d) ← {}
for j ← 1 to L − l + 1 do
    x ← j^th l-mer in S_1
    N(S_1, d) ← N(S_1, d) ∪ N(x, d)
end for
C ← N(S_1, d)
for i ← 2 to n' do
    N(S_i, d) ← {}
    for j ← 1 to L − l + 1 do
        x ← j^th l-mer in S_1
        N(S_i, d) ← N(S_i, d) ∪ N(x, d)
    end for
    C ← C ∩ N(S_i, d)
end for
M ← {}                                              ▷ test candidates
for each l-mer u in C do
    isMotif ← true
    for i ← (n' + 1) to n do
        found ← false
        for j ← 1 to L − l + 1 do
            x ← j^th l−mer in S_i
            if dH(x, u) ≤ d then
                found ← true
                break
            end if
        end for
        if !found then
            isMotif ← false
            break
        end if
    end for
    if isMotif then
        M ← M ∪ u
    end if
end for
return M
```

In practice, EMS-GT must perform speedy operations on an array of bits representing the entire motif search space. Subsections 2.3.1 to 2.3.4 discuss the efficiency strategies EMS-GT uses for important tasks such as representing sets in the search space, determining whether l-mers are neighbors, and generating all possible d-neighbors of a given l-mer.

2.3.1 Bit-based set representation and l-mer enumeration

The motif search space consists of the 4^l possible l-mers that can be formed from the nucleic alphabet {`a`, `g`, `c`, `t`}. To efficiently represent sets—such as a d-neighborhood, or a set of candidate motifs—within this space, EMS-GT assigns each of the 4^l l-mers a bit flag in an array, set to 1 if the l-mer is a member of the set and 0 otherwise. Bit flags correspond to l-mers via a simple mapping: EMS-GT maps an l-mer s to a bit flag index x by replacing each character with 2 bits (`a=00`, `c=01`, `g=10`, `t=11`). Note that this mapping scheme enumerates l-mers in strict alphabetical order.

Ex. `tacgt` maps to `1100011011` = 795; thus, its flag is the 795^{th} bit in the array.

2.3.2 Bit-array compression

EMS-GT's implementation compresses the required set-representation array of 4^l bits into an equivalent array of $\frac{4^l}{32}$ 32-bit integers. The x^{th} bit is now found at position (x mod 32) of the integer at array index $\frac{x}{3}$.

Ex. `tacgt` maps to `1100011011` = 795 in decimal.

array index = $\frac{795}{32}$ = 24 , *bit position* = 795 mod 32 = 27;

Thus, the flag for `tacgt` is the 27^{th} bit of the integer at array index 24.

2.3.3 XOR-based Hamming distance computation

The mapping of l-mers to binary numbers is also useful for computing Hamming distances. An exclusive OR (XOR) bitwise operation between the mappings of two l-mers will produce a nonzero pair of bits at every mismatch position; counting these nonzero pairs of bits in the XOR result gives us the Hamming distance. See Algorithm 2.2 for the implementation.

Ex. `tacgt` maps to `1100011011`

`ttcgg` maps to `1111011010`

XOR produces `00`$\underline{\texttt{11}}$`000`$\underline{\texttt{00}}$`1` = 2 mismatches.

2.3.4 Recursive neighborhood generation

To generate a d-neighbor of an l-mer x, we choose $d' \leq d$ positions from 1, 2,..., l-1, l and change the character at each of the d' positions in x. EMS-GT uses a recursive procedure (Algorithm 2.3) to do this, effectively (1) traversing the tree of all d-neighbors and (2) setting the bit flag in the neighborhood array N for each neighbor it encounters. Since we choose up to d positions in the l-mer, and have 3 possible substitute characters at each position, the size of the neighborhood $N(x, d)$ is given by:

$$|N(x,d)| = \sum_{i=0}^{d} \binom{l}{i} 3^i \tag{2.3}$$

Algorithm 2.2 HAMMING DISTANCE COMPUTATION

Input: l-mer mappings u and v
Output: $dH(u, v)$

$dH(u, v) = 0$
$z \leftarrow u^v$
for $i \leftarrow 1$ to l **do**
    if z & 3 != 0 **then**
        $dH(u, v) \leftarrow dH(u, v) + 1$
    end if
    $z \leftarrow z >> 2$
end for
return $dH(u, v)$

Algorithm 2.3 RECURSIVE NEIGHBORHOOD GENERATION

Input: DNA sequence S, motif length l, mismatches d
Output: bit-array $\mathcal{N}$ representing $\mathcal{N}(S, d)$

$\mathcal{N}[lmer] \leftarrow 0, \ \forall\ lmer \in$ search space
for each l-mer x in S **do**
    ADDNEIGHBORS(x, 0, d) ▷ *recursive procedure*
end for

▷ *make d changes in l-mer x, from position s onward*
procedure ADDNEIGHBORS(x, s, d)
    for $i \leftarrow s$ to l **do**
        $\Sigma' \leftarrow \{\mathtt{a}, \mathtt{g}, \mathtt{c}, \mathtt{t}\} - x_i$ ▷ *remove i^{th} character of x*
        for $j \leftarrow 1$ to $|\Sigma'|$ **do**
            $neighbor \leftarrow concatenate(x_{1\ldots i-1}, \Sigma_j, x_{i+1\ldots l})$
            $\mathcal{N}[neighbor] \leftarrow 1$
            if $d > 1$ and $i < l$ **then**
                ADDNEIGHBORS($neighbor$, $i + 1$, $d - 1$)
            end if
        end for
    end for
end procedure
return $\mathcal{N}$

CHAPTER III

Methodology

This section briefly describes the procedure used to evaluate the optimized version of the EMS-GT algorithm.

3.1 Datasets

Synthetic datasets were created using a DNA sequence generator written in Java. Each nucleotide character in a sequence is randomly generated; {a, g, c, t} each have a 25% chance of being selected, independent from other characters in the sequence. The motif is then planted at a random position in the sequence. As prescribed in [13] every dataset contains 20 DNA sequences each 600 bases long, with an (l, d) motif planted exactly once in each sequence.

3.2 Implementation

Thee Java implementation of EMS-GT operates on a compact, bit-based enumerative representation of the motif search space. Since a significant part of runtime is spent locating and setting bits in this bit-based representation, optimizations were explored for the bit-setting portion of the algorithm. Investigation of some Hamming distance-based patterns in the search space led to the development and integration of a bit-masking speed-up technique, which exploits these patterns to set bits in entire blocks.

3.3 Evaluation

The optimized version of EMS-GT was compared to the original EMS-GT algorithm, as well as to the state-of-the-art algorithms PMS8 and qPMS9, by benchmarking their performance on challenging instances of the (l, d) planted motif problem. An (l, d) problem instance is defined to be a challenging instance if d is the largest value for which the expected number of l-length motifs that would occur in the input by random chance does not exceed some limit—typically 500 random motifs [12]. The specific challenge instances used were (9,2), (11,3), (13,4), (15,5), and (17,6), as identified in [12, 5].

CHAPTER IV

Results and Analysis

This section provides a step-by-step derivation of the distance-related patterns that occur within a bit-array representing an l-mer neighborhood in EMS-GT. It describes how an optimization for EMS-GT based on these patterns was implemented. Finally, it quantifies the performance improvement due to this optimization.

4.1 Deriving distance-related patterns in l-mer neighborhoods

1. We can represent the neighborhood $N(x, d)$ of an l-mer x as an array N of 4^l bit flags, set to 1 if the corresponding l-mer is a neighbor and 0 otherwise.

$$N_{x'} = \begin{cases} 1 & \text{if } dH(x, x') \leq d, \\ 0 & \text{otherwise.} \end{cases} \quad \text{for any } l\text{-mer } x'.$$

2. We find that if we divide this bit-array N into consecutive blocks of 4^k flags each, for some block degree k, $0 < k < l$, each block will conform to one of at most $(k + 2)$ possible bit patterns. We exploit this regularity to build N in blocks.

3. Say we wish to generate $N(x, d)$ for some l-mer x. We divide x into its **prefix** $\boldsymbol{y}$ (first $l - k$ characters) and its $\boldsymbol{k}$**-suffix** $\boldsymbol{z}$ (last k characters).

   Ex. For k = 5, x = `acgtacgtacgt` $\rightarrow$ y = `acgtacg` and z = `tacgt`.

   As later explained in steps 7-8, the prefix will decide which of $(k+2)$ patterns is applicable in a particular block in N, while the k-suffix will determine the structure of these $(k + 2)$ patterns.

4. We generate the distribution $D(z)$ of Hamming distances from z to all 4^k possible k-suffixes. We consider this distribution to be "centered" at z.

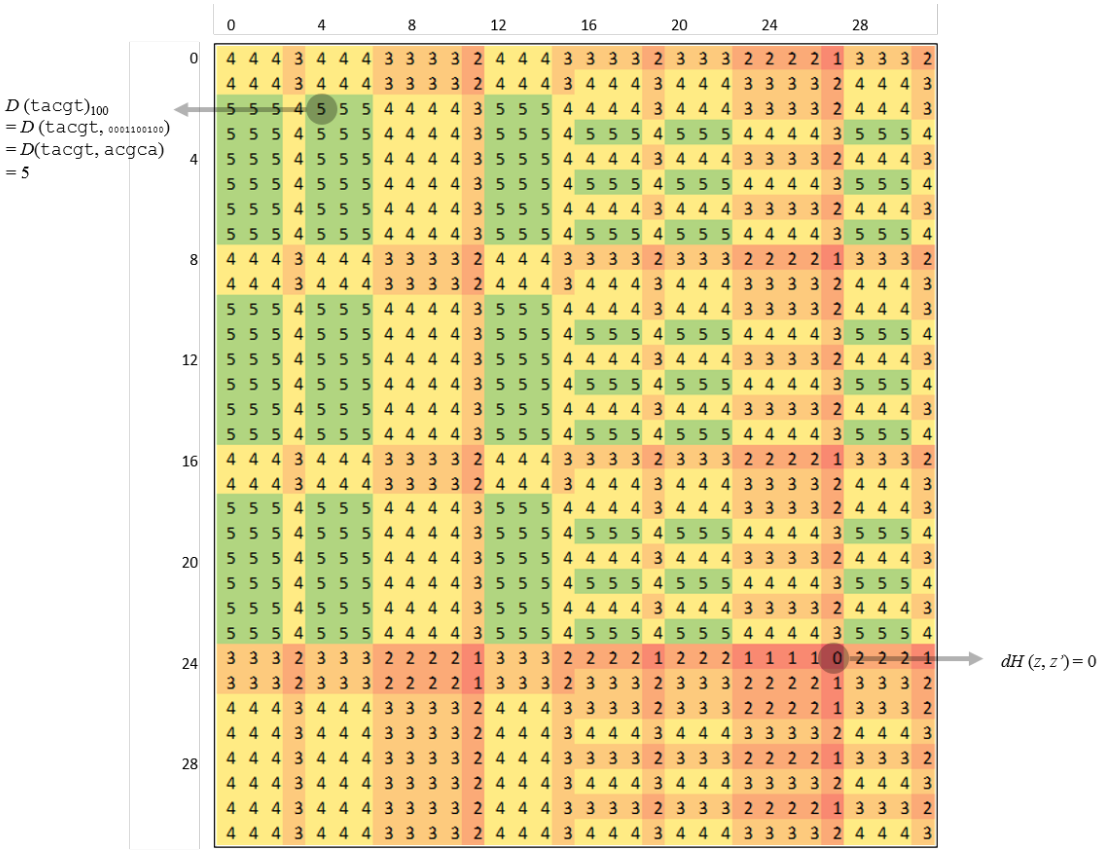

Figure 4.1. Distance distribution from `tacgt` to all $4^5 = 32 \times 32$ k-suffixes, k=5.

The value in the p^{th} cell—found at row $\frac{p}{32}$, column p *mod* 32—of the 32×32 table in Figure 4.1 is the distance $dH(z, z)$ from z = `tacgt` to the k-suffix z' which maps to the binary number p.

Ex. $D(\texttt{tacgt})_{100}$ = $dH(\texttt{tacgt}, \texttt{0b0001100100})$

= $dH(\texttt{tacgt}, \texttt{acgca})$

= 5 , at row $\frac{100}{32} = 3$, column $(100 \; mod \; 32) = 4$

5. Due to the alphabetical enumeration, the 4^k $l-$mers grouped together in a block will all begin with the same $(l-k)$ characters, which we will call the **block prefix y'**. We can compute a single prefix distance $d_{y'}$ for an entire block: this is simply the distance $dH(y, y')$ between x's prefix and the block prefix.

   Ex. For the block containing l-mers $\{$`acgttgcaaaaa` to `acgttgcttttt`$\}$, the prefix distance from z = `acgtacgtacgt` is
   $d_{y'} = dH($`acgtacg`$,$ `acgttgc`$) = 3.$

6. We can infer that the distance between any two l-mers is equal to the sum of the distance between their prefixes and the distance between their k-suffixes ; thus, knowing $d_{y'}$ and $\mathcal{D}(z)$for any l-mer $x' = y'z'$ in the search space, we can compute the distance from x as:

$$dH(x, x') = d_{y'} + \mathcal{D}(z)_{z'} \tag{4.1}$$

7. We can now redefine the criteria for setting a bit in N:

$$N_{x'} = \begin{cases} 1 & \text{if } d_{y'} + \mathcal{D}(z)_{z'} \leq d, \\ 0 & \text{otherwise.} \end{cases} \quad \text{for } x' = y'z'.$$

   From this we see that a bit at position z' within a block with prefix y' will be set if and only if $D(z)_{z'} \leq d - d_{y'}$. The values in $D(z)$ range from 0 to k; therefore, we examine $(k+2)$ cases for the value of $(d - d_{y'})$ with respect to the range (0, ..., k):

Case -1:	$d - d_{y'} < 0$,	no bits in the block are set;
Case 0:	$d - d_{y'} = 0$,	the "center" bit (at position z) is set;
Cases 1 to $k-1$:	$1 \leq d - d_{y'} < k$,	some of the bits are set; and
Case k:	$d - d_{y'} \geq k$,	all bits in the block are set.

8. We see that the $(k+2)$ patterns of blocks in N correspond to the $(k+2)$ cases listed in step 7, and thus the pattern to be used in a certain block is indicated by the value of $(d - d_{y'})$. Meanwhile, the polarity (0 or 1) of a bit in one of these patterns is determined by the value of $D(z)$ at the corresponding position.

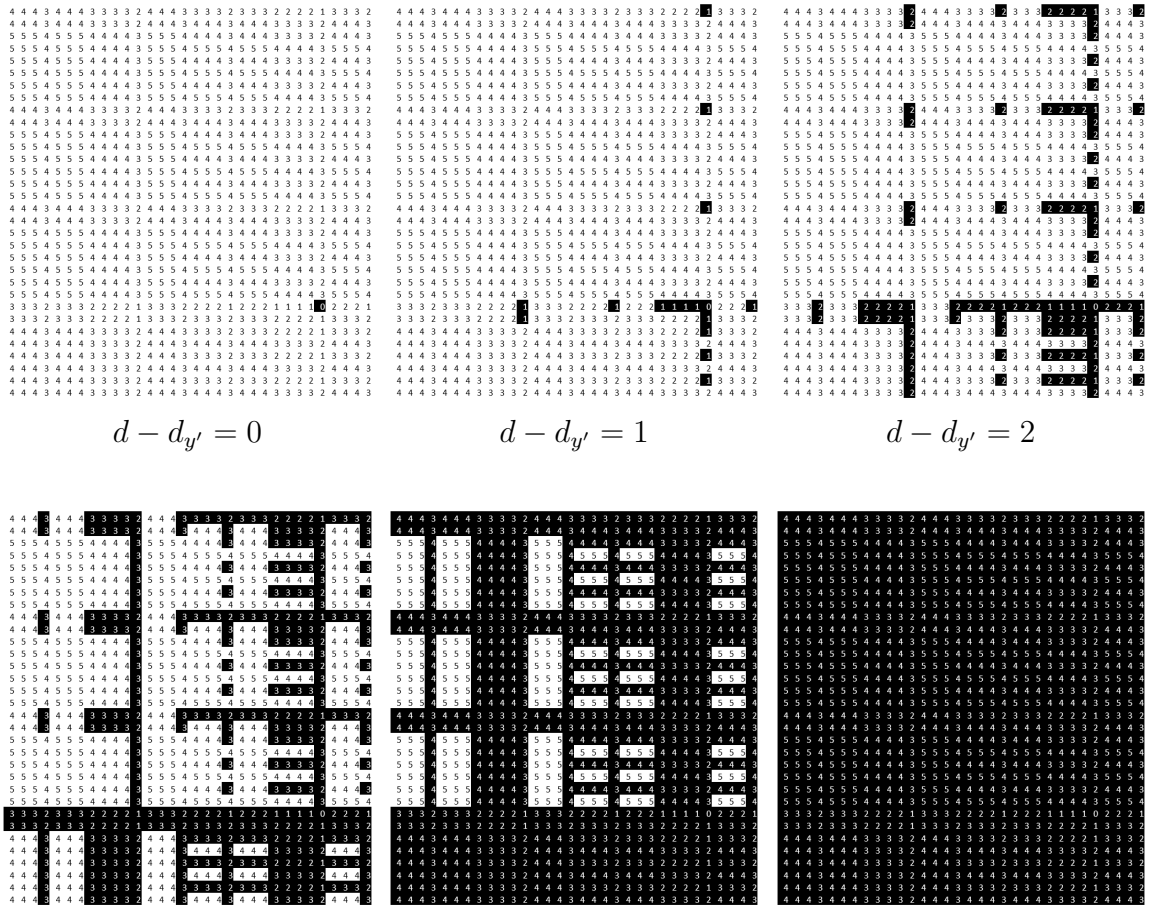

Figure 4.2. Bit patterns followed by blocks in $N(\texttt{tacgt}, 5)$. Black signifies a bit set to 1. There are (5 + 2) = 7 possible patterns; the "empty" pattern (for $d - d_{y'} \leq 0$, in which all bits are 0) is not shown. Compare the structure of these patterns with the coloring of the distribution $D(\texttt{tacgt})$ in Figure 4.1.

4.2 Implementing a pattern-based optimization for EMS-GT

The previous section has shown that the bit-array N, representing the d-neighborhood of an l-mer, can be defined in terms of repeating block patterns. This definition can be used toward a larger goal—constructing the bit-array $\mathcal{N}$ which represents the d-neighborhood $\mathcal{N}(S, d)$ of a sequence S.

To do this, we first initialize the array $\mathcal{N}$ of 4^l bit flags, all set to zero. We select a value for the block degree k, and pre-generate all 4^k possible sets of block patterns (one set for each possible "center" k-suffix). Then, for every l-mer $x = yz$ in S:

1. We retrieve $\mathcal{P}(z) = \{P_1, ..., P_{k-1}\}$, the set of unique block patterns "centered" at x's k-suffix z. For convenience, this set excludes three "trivial" patterns: the empty pattern P_{-1} (all 0's), the full pattern P_k (all 1's), and the pattern P_0 in which only the center bit (the bit at position z) is set.

2. For every d-neighbor y' of x's prefix y, we locate the block—$block(\mathcal{N}, y')$—in $\mathcal{N}$ whose block prefix is y'. We then apply the appropriate pattern to the block, based on the value of $d - d_{y'}$:

   (a) If $d - d_{y'} = 0$, we set the "center" bit, i.e. the bit at position z in $block(\mathcal{N}, y')$.

   (b) If $0 < d - d_{y'} < k$, we mask the pattern $P_{d-d_{y'}}$ onto $block(\mathcal{N}, y')$.

   (c) If $d - d_{y'} \geq k$, we set all bits in $block(\mathcal{N}, y')$.

This entire procedure effectively performs $\mathcal{N}(S, d) \leftarrow \mathcal{N}(S, d) \cup N(x, d)$ for each l-mer x in S, as specified in lines 3-6 and 10-13 of EMS-GT (Algorithm 2.1).

Step 2 requires us to generate all d-neighbors of x's prefix y.We can use our recursive procedure (Algorithm 2.3) to traverse the neighborhood $N(y, d)$; as Table 1 shows,

this will require generating much fewer neighbors than traversing the entire neighborhood of x, $N(x,d)$. Note that, with k=5, $N(y,d)$ is only 10-20% the size of $N(x,d)$!

(l,d)	$N(x,d)$ $\sum_{i=0}^{l}\binom{l}{i}3^i$	$N(y,d)$, k=5 $\sum_{i=0}^{l-k}\binom{l-k}{i}3^i$	% reduction
9,2	351	66	81.2%
11,3	4,983	693	86.1%
13,4	66,378	7,458	88.8%
15,5	853,569	81,921	90.4%
17,6	10,738,203	912,717	91.5%

Table 4.1. Number of individual neighbors generated for $N(x,d)$ vs. $N(y,d)$

This large reduction in recursive neighbor generation explains why, for most (l,d) values, building the neighborhood bit-array in blocks proves significantly faster than the current approach—recursively generating each individual neighbor, then locating and setting its bit flag. This is shown in Table 2, comparing EMS-GT's performance with and without the pattern-based optimization.

(l,d)	Without block-based optimization	With block-based optimization, $k=5$	speedup
9,2	0.06 s	0.11 s	–
11,3	0.22 s	0.20 s	6.7%
13,4	1.98 s	1.04 s	47.5%
15,5	25.06 s	15.51 s	38.1%
17,6	308.61 s	175.85 s	43.0%

Table 4.2. Performance of EMS-GT with vs. without block-masking optimization (runtimes averaged over 20 synthetic datasets for each (l,d) instance).

4.3 Performance of optimized EMS-GT

Finally, the optimized EMS-GT and two competitor algorithms were run on an Intel Xeon, 2.10 GHz processor (single core only). Their performance, averaged over 20 synthetic datasets for each (l, d) challenge instance, is outlined in Table 3:

(l, d)	**PMS8**	**qPMS9**	**EMS-GT**	**% speedup**
9,2	0.74 s	0.47 s	**0.11 s**	76.6%
11,3	1.58 s	1.06 s	**0.20 s**	81.1%
13,4	5.39 s	4.52 s	**1.04 s**	77.0%
15,5	36.45 s	24.63 s	**15.51 s**	37.0%
17,6	3.91 min	**1.96 min**	*2.93 min*	–

Table 4.3. Runtimes of PMS8, qPMS9 and EMS-GT

For every challenge instance except (17,6) EMS-GT outperforms qPMS9; it outperforms PMS8 for (17,6). EMS-GT was run including the block-masking optimization, with the default suffix length of $k = 5$. Observe that our EMS-GT implementation can only solve problem instances where $l \leq 17$. This is because when we reach l=18, the size of the integer array needed to represent the entire search space ($\frac{4^{18}}{32} = \frac{2^{36}}{2^5} = 2^{31}$ integers) begins to exceed the maximum size for Java arrays, which is $(2^{31} - 1)$ elements.

BIBLIOGRAPHY

[1] Timothy L Bailey and Charles Elkan. Unsupervised learning of multiple motifs in biopolymers using expectation maximization. *Machine learning*, 21(1-2):51–80, 1995.

[2] Mathieu Blanchette and Martin Tompa. Discovery of regulatory elements by a computational method for phylogenetic footprinting. *Genome research*, 12(5):739–748, 2002.

[3] Modan K Das and Ho-Kwok Dai. A survey of dna motif finding algorithms. *BMC bioinformatics*, 8(Suppl 7):S21, 2007.

[4] 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.

[5] 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.

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

[7] 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. Fourth International Conference on*, pages 1–6. IEEE, 2009.

[8] 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.

[9] Charles E Lawrence and Andrew A Reilly. An expectation maximization (em) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences. *Proteins: Structure, Function, and Bioinformatics*, 7(1):41–51, 1990.

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

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

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

[13] 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.