

# Improving an Exact Solution to the $(l, d)$ Planted Motif Problem

Maria Clara Isabel Sia

October 23, 2015

# Introduction

## DNA motif finding

- ▶ **motifs** are repeated sub-sequences in DNA that have some biological significance
- ▶ **DNA motif finding** searches for motifs over a set of DNA sequences, allowing for mismatches due to mutation
- ▶ known as a difficult problem in computational biology and CS (proven **NP-complete**)

# Introduction

## The $(l,d)$ planted motif problem

*Find a motif of length  $l=8$  across these 5 DNA sequences, each containing the motif with at most  $d=2$  mismatches.*

$S_1$  atcactcgttctcctctaattgtgttaaagacgtactaccgacctta  
 $S_2$  acgccgaccgggtcgcgaccttgtatagctcctaacgggcatcagc  
 $S_3$  tcctgactgcatcgcgatctcggtagtttcctgttcatcattttt  
 $S_4$  ggccctcagcatcgtgcgtcctgctaacacattcccatgcagctt  
 $S_5$  tgaaaagaatttacggtaaaggatccacatccaatcgtgtgaaag

*Planted motif: ccatcgtt*

# Introduction

## Solutions to the $(l,d)$ planted motif problem)

There are two types of methods used by motif search algorithms:

- ▶ **heuristic methods** (ex. probabilistic sampling, projection)  
perform an iterative local search which is efficient, but not guaranteed to find all motifs
- ▶ **exact methods** (ex. combinatorial search, tree pruning)  
perform an exhaustive search which will find all possible motifs, at the cost of time/space efficiency

# Introduction

$l$ -mers, Hamming distances, and  $d$ -neighborhoods

- ▶  $l$ -mer
- ▶ Hamming distance  $d_H$
- ▶  $d$ -neighbor

# Introduction

$l$ -mers, Hamming distances, and  $d$ -neighborhoods

- ▶  $l$ -mer

- sequence of length  $l$

$S_1 = \text{atcactcgttctcctctaattgtgtaaagacgtactaccgacctta}$

- ▶ Hamming distance  $d_H$

- ▶  $d$ -neighbor

# Introduction

$l$ -mers, Hamming distances, and  $d$ -neighborhoods

►  $l$ -mer

► Hamming distance  $d_H$

- number of mismatches between  $l$ -mers  $x_1$  and  $x_2$

$x_1 = \text{c} \textcolor{red}{g} \text{atc} \textcolor{red}{c} \text{tt}$

$x_2 = \text{c} \textcolor{red}{c} \text{atc} \textcolor{red}{g} \text{tt}$

$$d_H(x_1, x_2) = 2$$

►  $d$ -neighbor

# Introduction

## *l*-mers, Hamming distances, and *d*-neighborhoods

► *l*-mer

► Hamming distance  $d_H$

► *d*-neighbor

- two *l*-mers  $x$  and  $x'$  are *d*-neighbors if  $d_H(x, x') \leq d$

$N(\text{ccatcgtt}, 2) \rightarrow d\text{-neighborhood of ccatcgtt, } d=2$

$= \{ \text{ccatcgtt},$

$\text{acatcgtt}, \text{gcatcgtt}, \text{tcacgtt}, \text{caatcgtt}, \text{cgatcgtt}, \text{ctatcgtt},$

...all *l*-mers with 1 mismatch

$\text{aaatcgtt}, \text{agatcgtt}, \text{atatcgtt}, \text{gaatcgtt}, \text{ggatcgtt}, \text{gtatcgtt},$

$\text{taatcgtt}, \text{tgatcgtt}, \text{ttatcgtt}, \text{acctcgtt}, \text{acgtcgtt}, \text{acttcgtt},$

...all *l*-mers with 2 mismatches

$\}$



# Introduction

$l$ -mers, Hamming distances, and  $d$ -neighborhoods

- ▶  $l$ -mer
- ▶ Hamming distance  $d_H$
- ▶  $d$ -neighbor

# EMS-GT

Nabos, 2015

- ▶ an exact motif search (EMS) algorithm based on the candidate generate-and-test (GT) principle
- ▶ solves any  $(l,d)$  planted motif problem instance,  $l \leq 17$
- ▶ operates on a bit-based representation of the search space

# EMS-GT

## Generate-and-test approach

EMS-GT proceeds in two steps:

1. **Generate the set  $C$  of candidate motifs:** find the common  $d$ -neighbors of the first  $n'$  sequences  $S_1, S_2, \dots, S_{n'}$ .

$$C = \mathcal{N}(S_1, d) \cap \mathcal{N}(S_2, d) \cap \dots \cap \mathcal{N}(S_{n'}, d), \quad n' \leq n$$

2. **Test every candidate  $c \in C$ :** if a  $d$ -neighbor of  $c$  appears in each of the remaining sequences  $S_{n'+1}, S_{n'+2}, \dots, S_n$ , accept  $c$  as a motif.

# EMS-GT

## Generate-and-test approach

$$(l, d) = (8, 2)$$

$S_1$  atcactcgtttctcctctaattgtgttaaagacgtactaccgacctta

$S_2$  acgccgaccgggtccgatccttgtatagctcctaacgggcatcagc

$S_3$  tcctgactgcatcgcgatctcggtagtttcctgttcatcattttt

$S_4$  ggccctcagcatcgtgcgtcctgctaacacattcccatgcagctt

$S_5$  tgaaaagaattttacggtaaaggatccacatccaatcgtgtgaaag

# EMS-GT

## Bit-based representation of the search space

- ▶ The search space contains all  $4^l$  possible  $l$ -mers that can be formed with  $\Sigma = \{a, c, g, t\}$ .
- ▶ To represent sets in this space, EMS-GT assigns each of the  $4^l$   $l$ -mers a bit flag, which is 1 if the  $l$ -mer is a member of the set, 0 otherwise.
- ▶ For efficiency, EMS-GT stores the  $4^l$  bits as  $\frac{4^l}{32}$  32-bit integers.

# EMS-GT

## Bit-based representation of the search space

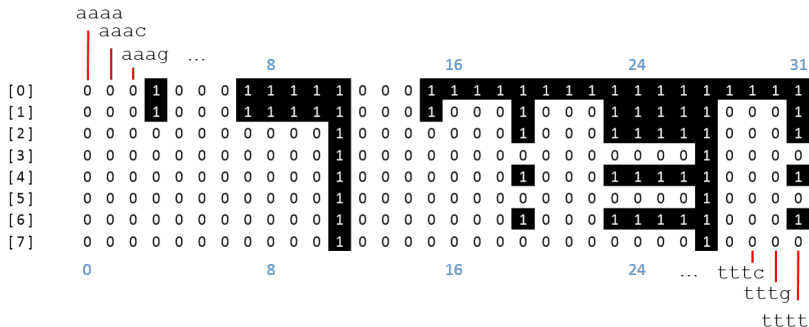
- $N(\text{acgt}, 1)$ :  $4^l = 256$ ,  $\frac{4^l}{32} = 8$

	0	8	16	24	31
[0]	0 0 0 1 0 0 0	1 1 1 1 1	0 0 0	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	
[1]	0 0 0 1 0 0 0	1 1 1 1 1	0 0 0	1 0 0 0 1 0 0 0 1 1 1 1 1 1 0 0 0 1	
[2]	0 0 0 0 0 0 0	0 0 0 0 1	0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 1 1 1 0 0 0 1		
[3]	0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0		
[4]	0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 1 1 1 0 0 0 1		
[5]	0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0		
[6]	0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 1 1 1 0 0 0 1		
[7]	0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0		
	0	8	16	24	31

# EMS-GT

## Bit-based representation of the search space

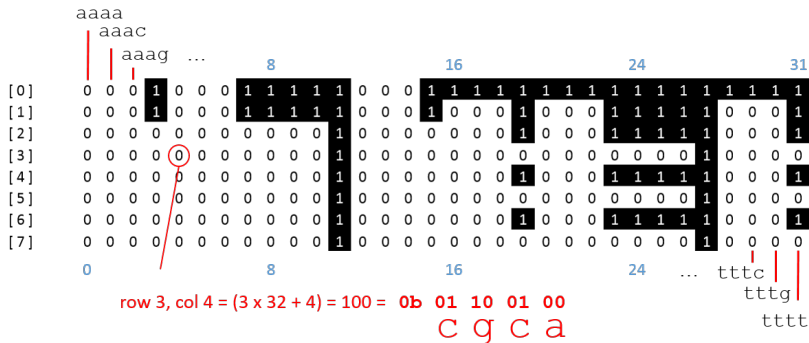
- $N(\text{acgt}, 1)$ :  $4^l = 256$ ,  $\frac{4^l}{32} = 8$



# EMS-GT

## Bit-based representation of the search space

- $N(\text{acgt}, 1)$ :  $4^l = 256$ ,  $\frac{4^l}{32} = 8$

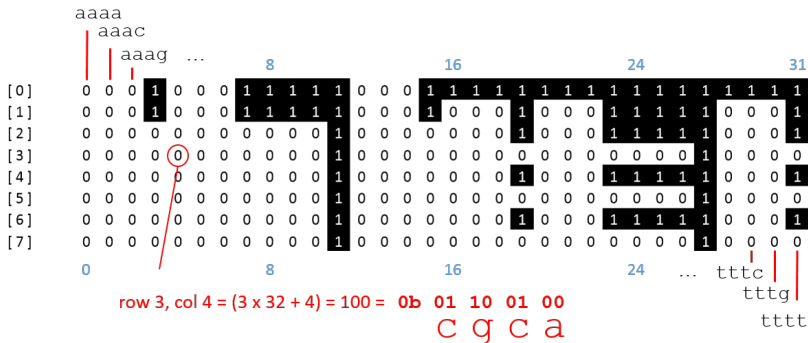




# EMS-GT

## Bit-based representation of the search space

- $N(\text{acgt}, 1)$ :  $4^l = 256$ ,  $\frac{4^l}{32} = 8$

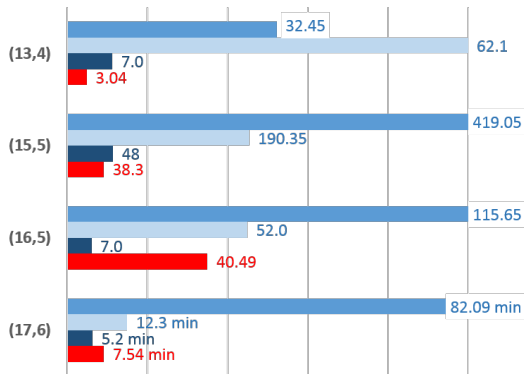


- EMS-GT generates a neighborhood bit-array by generating each individual neighbor, then finding and setting its bit flag.

# EMS-GT

## Performance comparison

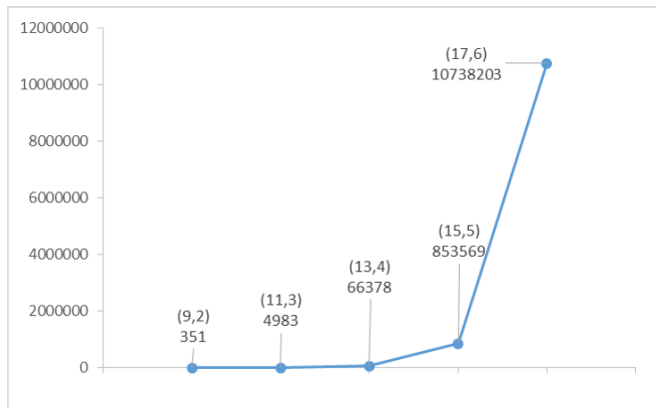
- ▶ EMS-GT is competitive against exact motif search algorithms PMSprune, qPMS7 and PMS8; however, for larger  $(l,d)$  values, it no longer outperforms the state-of-the-art.



Runtime in seconds, averaged over 20 synthetic datasets per  $(l,d)$

# EMS-GT

Neighborhood sizes for challenging  $(l,d)$



- $l$ -mer neighborhoods grow very quickly with  $(l,d)$ , meaning that EMS-GT must spend more time locating and setting bits.

# Research objectives

## Improving EMS-GT

The main objectives of this research are:

1. To **develop a speedup technique** for EMS-GT that takes advantage of distance-related patterns in the search space;
2. To **evaluate** the speedup technique with regard to **improvement in runtime**; and
3. To **evaluate** the improved version of the EMS-GT algorithm **against state-of-the-art** motif search algorithms.

# Speedup technique

Key observation

# Speedup technique

## Key observation

- ▶ If a bit-array  $N_x$  representing the neighborhood of  $l$ -mer  $x$  is partitioned into blocks of  $4^k$  bits each,

# Speedup technique

## Key observation

- ▶ If a bit-array  $N_x$  representing the neighborhood of  $l$ -mer  $x$  is partitioned into blocks of  $4^k$  bits each,

$$\text{Ex. } N(\text{acgtacgtacgt}, 5), k = 5$$
$$\text{block size} = 4^5 = 1024 = 32 \times 32$$

# Speedup technique

## Key observation

- ▶ If a bit-array  $N_x$  representing the neighborhood of  $l$ -mer  $x$  is partitioned into blocks of  $4^k$  bits each,

$$\text{Ex. } N(\text{acgtacgtacgt}, 5), k = 5$$
$$\text{block size} = 4^5 = 1024 = 32 \times 32$$

- ▶ the  $4^k$   $l$ -mers represented in a block all begin with the same prefix (first  $l - k$  characters), and will differ only in the  $k$ -suffix (last  $k$  characters);



# Speedup technique

## Key observation

- ▶ If a bit-array  $N_x$  representing the neighborhood of  $l$ -mer  $x$  is partitioned into blocks of  $4^k$  bits each,

$$\text{Ex. } N(\text{acgtacgtacgt}, 5), k = 5$$
$$\text{block size} = 4^5 = 1024 = 32 \times 32$$

- ▶ the  $4^k$   $l$ -mers represented in a block all begin with the same prefix (first  $l - k$  characters), and will differ only in the  $k$ -suffix (last  $k$  characters);
- ▶ and each block conforms to one of  $(k + 2)$  patterns.

Block patterns in the  $d$ -neighborhood of acgtacgtacgt,  $d=5$ ,  $k=5$

[illegible]

```

4 4 4 4 4 4 4 3 3 2 2 4 4 3 3 3 2 2 2 1 3 3 3 2
4 4 4 4 4 4 3 3 3 2 2 4 4 3 4 4 3 4 3 3 3 2 4 4 3
5 5 5 5 5 5 4 4 4 3 3 5 5 4 4 4 3 4 4 3 3 3 2 4 4 3
5 5 5 5 5 5 4 4 4 3 3 5 5 4 4 4 3 4 4 3 3 3 2 4 4 3
5 5 5 5 5 5 4 4 4 3 3 5 5 4 4 4 3 4 4 3 3 3 2 4 4 3
5 5 5 5 5 5 4 4 4 3 3 5 5 5 5 5 5 5 4 4 4 4 3 5 5 4
5 5 5 5 5 5 4 4 4 3 3 5 5 5 5 5 5 5 4 4 4 4 3 5 5 4
4 4 4 4 4 4 3 3 3 2 2 4 4 3 3 3 2 2 2 1 3 3 3 2
5 5 5 5 5 5 4 4 4 3 3 5 5 4 4 4 3 4 4 3 3 3 2 4 4 3
5 5 5 5 5 5 4 4 4 3 3 5 5 5 5 5 5 5 4 4 4 4 3 5 5 4
5 5 5 5 5 5 4 4 4 3 3 5 5 5 5 5 5 5 4 4 4 4 3 5 5 4
5 5 5 5 5 5 4 4 4 3 3 5 5 5 5 5 5 5 4 4 4 4 3 5 5 4
5 5 5 5 5 5 4 4 4 3 3 5 5 5 5 5 5 5 4 4 4 4 3 5 5 4
4 4 4 4 4 4 3 3 3 2 2 4 4 3 3 3 2 2 2 1 3 3 3 2
4 4 4 4 4 4 3 3 3 2 2 4 4 3 4 4 3 4 3 3 3 2 4 4 3
5 5 5 5 5 5 4 4 4 3 3 5 5 4 4 4 3 4 4 3 3 3 2 4 4 3
5 5 5 5 5 5 4 4 4 3 3 5 5 4 4 4 3 4 4 3 3 3 2 4 4 3
5 5 5 5 5 5 4 4 4 3 3 5 5 5 5 5 5 5 4 4 4 4 3 5 5 4
5 5 5 5 5 5 4 4 4 3 3 5 5 5 5 5 5 5 4 4 4 4 3 5 5 4
3 3 3 3 3 2 2 2 1 3 3 3 2 2 2 1 3 3 3 2 2 2 1 3 3 3 2
4 4 4 4 4 4 3 3 3 2 2 4 4 3 3 3 2 2 2 1 3 3 3 2
4 4 4 4 4 4 3 3 3 2 2 4 4 3 4 4 3 4 3 3 3 2 4 4 3
4 4 4 4 4 4 3 3 3 2 2 4 4 3 4 4 3 4 3 3 3 2 4 4 3
4 4 4 4 4 4 3 3 3 2 2 4 4 3 4 4 3 4 3 3 3 2 4 4 3
4 4 4 4 4 4 3 3 3 2 2 4 4 3 4 4 3 4 3 3 3 2 4 4 3

```

Figure 1 displays a 10x10 grid of 100 small images, each showing a handwritten digit. The digits are arranged in a 10x10 grid, with each row containing 10 digits. The digits are generated by a generative model, and the sequence shows a smooth transition from 0 to 9. The digits are arranged in a 10x10 grid, with each row containing 10 digits. The digits are generated by a generative model, and the sequence shows a smooth transition from 0 to 9.

Figure 1 displays a 4x4 grid of 16 small images, each showing a different combination of the four basic patterns (A, B, C, D) from the MNIST dataset. The patterns are: A (top-left), B (top-right), C (bottom-left), and D (bottom-right). The grid shows various combinations of these patterns, including single patterns, pairs, and more complex arrangements.

Figure 1 displays a 10x10 grid of 100 grayscale images. Each image shows a handwritten digit from the MNIST dataset, arranged in a 10x10 pattern. The digits are arranged in a 10x10 grid, with each digit being a different color (e.g., red, green, blue, yellow, etc.). The digits are arranged in a 10x10 grid, with each digit being a different color (e.g., red, green, blue, yellow, etc.). The digits are arranged in a 10x10 grid, with each digit being a different color (e.g., red, green, blue, yellow, etc.).

[illegible]

# Speedup technique

## Key observation

- ▶ In a  $d$ -neighbor of  $x$ , the  $d$  allowable mismatches from  $x$  are distributed between the prefix and the  $k$ -suffix.

$x = \text{acgtacg tacgt}$

$x' = \text{acg}\textcolor{red}{aaaa} \textcolor{red}{tccga}$

- ▶ If a block's prefix already has  $p$  mismatches from  $x$ 's prefix, then for an  $l$ -mer in the block to be a neighbor of  $x$ , it must have a suffix with at most  $d - p$  mismatches from  $x$ 's suffix.

# Speedup technique

## Key observation

- ▶ The  $(k + 2)$  block patterns thus correspond to  $k + 2$  cases for the  $p$  mismatches between  $x$ 's prefix and a block's prefix:

prefix mismatches	suffix mismatches
more than $d$	—
$d$	0
$d - 1$	0 or 1
$d - 2$	0, 1, or 2
...	...
$d - (k-1)$	0, 1, 2, ..., or $(k-1)$
$d - k$ or less	0, 1, 2, ..., $(k-1)$ , or $k$

aaaaa →

[illegible]

← ttttt

aaaaa →

4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	3	3	2	3	3	3	2	2	2	2	1	3	3	3	2		
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	4	4	4	3	4	4	4	3	3	3	2	4	4	4	3		
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	4	4	4	3	4	4	4	3	3	3	2	4	4	4	3		
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	4	4	4	4	3	5	5	4		
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3		
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	4	4	4	3	5	5	5	4		
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	4	4	3	4	4	4	3	4	3	3	3	2	4	4	4	3	
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	3	3	3	2	3	3	3	2	2	2	2	1	3	3	3	2	
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	4	5	5	4	4	4	3	5	5	4
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	4	5	5	4	4	4	3	5	5	4
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	4	5	5	4	4	4	3	5	5	4
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	4	5	5	4	4	4	3	5	5	4
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	3	3	3	2	3	3	3	2	2	2	2	1	3	3	3	2	
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	4	5	5	4	4	4	3	5	5	4
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	4	5	5	4	4	4	3	5	5	4
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	4	5	5	4	4	4	3	5	5	4
3	3	3	2	3	3	3	2	2	2	1	3	3	3	2	2	2	2	1	2	2	2	1	1	1	1	0	2	2	2	1		
3	3	3	2	3	3	3	2	2	2	1	3	3	3	2	3	3	3	2	3	3	3	2	2	2	2	2	2	1	3	3	3	2
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	3	3	3	2	3	3	3	2	2	2	2	1	3	3	3	2	
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	3	3	3	2	3	3	3	2	2	2	2	1	3	3	3	2	
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	3	3	3	2	3	3	3	2	2	2	2	1	3	3	3	2	
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	3	3	3	2	3	3	3	2	2	2	2	1	3	3	3	2	
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	3	3	3	2	3	3	3	2	2	2	2	1	3	3	3	2	
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	

← ttttt

aaaaa →

4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	3	3	3	2	2	2	1	3	3	3	2						
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3					
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	4	4	4	3	3	3	3	2	4	4	4	3					
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	4	4	4	3	5	5	5	4		
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	4	4	4	3	4	4	4	3	3	3	2	4	4	4	3		
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	4	4	4	4	3	5	5	5		
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	4	4	4	3	4	4	4	3	3	3	2	4	4	4	3		
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	3	3	2	3	3	3	2	2	2	1	3	3	3	2			
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	4	4	4	3	4	4	4	3	3	3	2	4	4	4	3		
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	4	4	4	3	4	4	4	3	3	3	2	4	4	4	3		
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	4	4	4	3	4	4	4	3	3	3	2	4	4	4	3		
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	4	4	4	3	4	4	4	3	3	3	2	4	4	4	3		
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	3	3	3	2	3	3	3	2	2	2	1	3	3	3	2		
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	4	4	4	3	4	4	4	3	3	3	2	4	4	4	3		
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	4	4	4	3	4	4	4	3	3	3	2	4	4	4	3		
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	4	4	4	3	5	5	5	4	4	4	4	3	5	5	5	4	
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	4	4	4	3	4	4	4	3	3	3	2	4	4	4	3		
5	5	5	4	5	5	5	4	4	4	4	3	5	5</																			

← ttttt

aaaaa →

4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	3	3	2	3	3	3	2	2	2	2	1	3	3	3	2				
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	4	4	4	3	4	4	4	3	3	3	2	4	4	4	3				
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	4	4	4	3	4	4	4	3	3	3	2	4	4	4	3				
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4			
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3				
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	4	4	4	3	5	5	5	4				
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	4	4	3	4	4	4	3	4	3	3	3	2	4	4	4	3			
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4			
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	3	3	3	2	3	3	3	2	2	2	2	1	3	3	3	2			
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3			
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3			
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	4	5	5	4	4	4	4	3	5	5	5	4
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3			
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	4	5	5	4	4	4	4	3	5	5	5	4
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	4	5	5	4	4	4	4	3	5	5	5	4
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	4	4	4	3	4	4	4	3	4	3	3	3	2	4	4	4	3		
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	4	5	5	4	4	4	4	3	5	5	5	4
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	4	4	4	3	4	4	4	3	4	4	3	3	3	2	4	4	4	3	
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	4	5	5	4	4	4	4	3	5	5	5	4
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	4	4	4	3	4	4	4	3	4	4	3	3	3	2	4	4	4	3	
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	4	5	5	4	4	4	4	3	5	5	5	4
3	3	3	2	3	3	3	2	2	2	2	1	3	3	3	2	2	2	2	1	2	2	2	2	1	1	1	1	0	2	2	2	2	1	
3	3	3	2	3	3	3	2	2	2	2	1	3	3	3	2	3	3	3	2	3	3	3	2	2	2	2	2	1	3	3	3	2		
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	3	3	3	2	3	3	3	2	2	2	2	2	1	3	3	3	2		
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3			
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	3	3	3	2	3	3	3	2	2	2	2	2	1	3	3	3	2		
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3			
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	3	3	3	2	3	3	3	2	2	2	2	2	1	3	3	3	2		
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	3	3	3	2	3	3	3	2	2	2	2	2	1	3	3	3	2		
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	3	3	3	2	3	3	3	2	2	2	2	2	1	3	3	3	2		
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3			

← ttttt



aaaaa  $\rightarrow$

4	4	4	3	4	4	4	3	3	3	2	4	4	4	3	3	3	2	3	3	3	2	2	2	2	1	3	3	3	2	
4	4	4	3	4	4	4	3	3	3	2	4	4	4	3	4	4	3	4	4	3	3	3	2	4	4	4	3			
5	5	5	4	5	5	5	4	4	4	3	5	5	5	4	4	4	3	4	4	4	3	3	3	2	4	4	4	3		
5	5	5	4	5	5	5	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	5	4	4	4	3	5	5	5	4
5	5	5	4	5	5	5	4	4	4	3	5	5	5	4	4	4	3	4	4	4	3	3	3	2	4	4	4	3		
5	5	5	4	5	5	5	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	5	4	4	4	3	5	5	5	4
5	5	5	4	5	5	5	4	4	4	3	5	5	5	4	4	4	3	4	4	4	3	3	3	2	4	4	4	3		
5	5	5	4	5	5	5	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	5	4	4	4	3	5	5	5	4
4	4	4	3	4	4	4	3	3	3	2	4	4	4	3	3	3	2	3	3	3	2	2	2	2	1	3	3	3	2	
4	4	4	3	4	4	4	3	3	3	2	4	4	4	3	3	3	2	3	3	3	2	2	2	2	1	3	3	3	2	
5	5	5	4	5	5	5	4	4	4	3	5	5	5	4	4	4	3	4	4	4	3	3	3	2	4	4	4	3		
5	5	5	4	5	5	5	4	4	4	3	5	5	5	4	4	4	3	4	4	4	3	3	3	2	4	4	4	3		
5	5	5	4	5	5	5	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	5	4	4	4	3	5	5	5	4
5	5	5	4	5	5	5	4	4	4	3	5	5	5	4	4	4	3	4	4	4	3	3	3	2	4	4	4	3		
5	5	5	4	5	5	5	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	5	4	4	4	3	5	5	5	4
5	5	5	4	5	5	5	4	4	4	3	5	5	5	4	4	4	3	4	4	4	3	3	3	2	4	4	4	3		
5	5	5	4	5	5	5	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	5	4	4	4	3	5	5	5	4
4	4	4	3	4	4	4	3	3	3	2	4	4	4	3	3	3	2	3	3	3	2	2	2	2	2	1	3	3	3	2
4	4	4	3	4	4	4	3	3	3	2	4	4	4	3	4	4	3	4	4	4	3	3	3	2	4	4	4	3		
5	5	5	4	5	5	5	4	4	4	3	5	5	5	4	4	4	3	4	4	4	3	3	3	2	4	4	4	3		
5	5	5	4	5	5	5	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	5	4	4	4	3	5	5	5	4
5	5	5	4	5	5	5	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	5	4	4	4	3	5	5	5	4
5	5	5	4	5	5	5	4	4</																						

← ttttt

aaaaa →

```

4 4 4 3 4 4 4 3 3 3 3 2 4 4 4 3 3 3 3 2 3 3 3 2 2 2 2 1 3 3 3 2
4 4 4 3 4 4 4 3 3 3 3 2 4 4 4 3 4 4 4 3 3 3 3 2 4 4 4 3
5 5 5 4 5 5 5 4 4 4 4 3 5 5 5 4 4 4 4 3 4 4 4 3 3 3 2 4 4 4 3
5 5 5 4 5 5 5 4 4 4 4 3 5 5 5 4 5 5 5 4 5 5 5 4 4 4 4 3 5 5 5 4
5 5 5 4 5 5 5 4 4 4 4 3 5 5 5 4 4 4 4 3 4 4 4 3 3 3 3 2 4 4 4 3
5 5 5 4 5 5 5 4 4 4 4 3 5 5 5 4 5 5 5 4 5 5 5 4 4 4 4 3 5 5 5 4
5 5 5 4 5 5 5 4 4 4 4 3 5 5 5 4 4 4 4 3 4 4 4 3 3 3 3 2 4 4 4 3
5 5 5 4 5 5 5 4 4 4 4 3 5 5 5 4 5 5 5 4 5 5 5 4 4 4 4 3 5 5 5 4
5 5 5 4 5 5 5 4 4 4 4 3 5 5 5 4 5 5 5 4 5 5 5 4 4 4 4 3 5 5 5 4
4 4 4 3 4 4 4 3 3 3 3 2 4 4 4 3 3 3 3 2 3 3 3 2 2 2 2 1 3 3 3 2
4 4 4 3 4 4 4 3 3 3 3 2 4 4 4 3 4 4 4 3 4 4 4 3 3 3 3 2 4 4 4 3
5 5 5 4 5 5 5 4 4 4 4 3 5 5 5 4 4 4 4 3 4 4 4 3 3 3 3 2 4 4 4 3
5 5 5 4 5 5 5 4 4 4 4 3 5 5 5 4 5 5 5 4 5 5 5 4 4 4 4 3 5 5 5 4
5 5 5 4 5 5 5 4 4 4 4 3 5 5 5 4 4 4 4 3 4 4 4 3 3 3 3 2 4 4 4 3
5 5 5 4 5 5 5 4 4 4 4 3 5 5 5 4 5 5 5 4 5 5 5 4 4 4 4 3 5 5 5 4
5 5 5 4 5 5 5 4 4 4 4 3 5 5 5 4 4 4 4 3 4 4 4 3 3 3 3 2 4 4 4 3
5 5 5 4 5 5 5 4 4 4 4 3 5 5 5 4 5 5 5 4 5 5 5 4 4 4 4 3 5 5 5 4
5 5 5 4 5 5 5 4 4 4 4 3 5 5 5 4 4 4 4 3 4 4 4 3 3 3 3 2 4 4 4 3
5 5 5 4 5 5 5 4 4 4 4 3 5 5 5 4 5 5 5 4 5 5 5 4 4 4 4 3 5 5 5 4
3 3 3 2 3 3 3 2 2 2 2 1 3 3 3 2 2 2 2 1 2 2 2 1 1 1 0 2 2 2 1
3 3 3 2 3 3 3 2 2 2 2 1 3 3 3 2 3 3 3 2 3 3 3 2 2 2 2 1 2 3 3 2
4 4 4 3 4 4 4 3 3 3 3 2 4 4 4 3 3 3 3 2 3 3 3 2 2 2 2 1 3 3 3 2
4 4 4 3 4 4 4 3 3 3 3 2 4 4 4 3 4 4 4 3 4 4 4 3 3 3 3 2 4 4 4 3
4 4 4 3 4 4 4 3 3 3 3 2 4 4 4 3 3 3 3 2 3 3 3 2 2 2 2 1 3 3 3 2
4 4 4 3 4 4 4 3 3 3 3 2 4 4 4 3 4 4 4 3 4 4 4 3 3 3 3 2 4 4 4 3
4 4 4 3 4 4 4 3 3 3 3 2 4 4 4 3 3 3 3 2 3 3 3 2 2 2 2 1 3 3 3 2
4 4 4 3 4 4 4 3 3 3 3 2 4 4 4 3 4 4 4 3 4 4 4 3 3 3 3 2 4 4 4 3

```

← ttttt

aaaaa →

4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	3	3	3	2	3	3	3	2	2	2	2	1	3	3	3	2
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	4	4	4	3	4	4	4	3	3	3	2	4	4	4	3	
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	4	4	4	3	4	4	4	3	3	3	2	4	4	4	3	
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	4	4	4	3	5	5	5	4	
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	4	4	4	3	4	4	4	3	3	3	2	4	4	4	3	
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	4	4	4	3	5	5	5	4	
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	4	4	4	3	4	4	4	3	3	3	2	4	4	4	3	
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	4	4	4	3	5	5	5	4	
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	3	3	3	2	3	3	3	2	2	2	2	1	3	3	3	2
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	4	4	4	3	4	4	4	3	3	3	2	4	4	4	3	
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	4	4	4	3	4	4	4	3	3	3	2	4	4	4	3	
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	4	4	4	3	4	4	4	3	3	3	2	4	4	4	3	
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	4	4	4	3	5	5	5	4	
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	4	4	4	3	4	4	4	3	3	3	2	4	4	4	3	
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	4	4	4	3	5	5	5	4	
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	3	3	3	2	3	3	3	2	2	2	2	1	3	3	3	2
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	4	4	4	3	4	4	4	3	3	3	2	4	4	4	3	
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	4	4	4	3	4	4	4	3	3	3	2	4	4	4	3	
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	4	4	4	3	5	5	5	4	
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	4	4	4	3	4	4	4	3	3	3	2	4	4	4	3	
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	4	4	4	3	4	4	4	3	3	3	2	4	4	4	3	
5	5	5	4	5	5	5	4	4	4	4	3	5	5	5	4	5	5	5	4	5	5	5	4	4	4	3	5	5	5	4	
3	3	3	2	3	3	3	2	2	2	2	1	3	3	3	2	2	2	2	1	2	2	2	1	1	1	0	2	2	2	1	
3	3	3	2	3	3	3	2	2	2	2	1	3	3	3	2	3	3	3	2	3	3	3	2	2	2	2	1	3	3	3	2
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	3	3	3	2	3	3	3	2	2	2	2	1	3	3	3	2
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	4	4	4	3	4	4	4	3	3	3	2	4	4	4	3	
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	3	3	3	2	3	3	3	2	2	2	2	1	3	3	3	2
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	4	4	4	3	4	4	4	3	3	3	2	4	4	4	3	
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	3	3	3	2	3	3	3	2	2	2	2	1	3	3	3	2
4	4	4	3	4	4	4	3	3	3	3	2	4	4	4	3	4	4	4	3	4	4	4	3	3	3	2	4	4	4	3	

← ttttt

aaaaa →

[illegible]

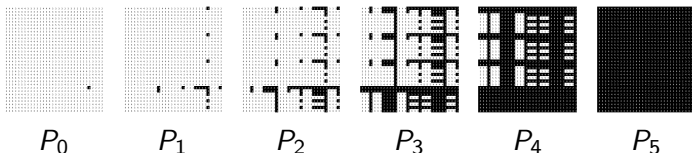
← ttttt

# Speedup technique

## Generate and apply patterns

To generate  $N_x$  for  $x = yz$ , we perform two steps:

1. From  $x$ 's suffix  $z$ , generate  $P$ , the set of block patterns.

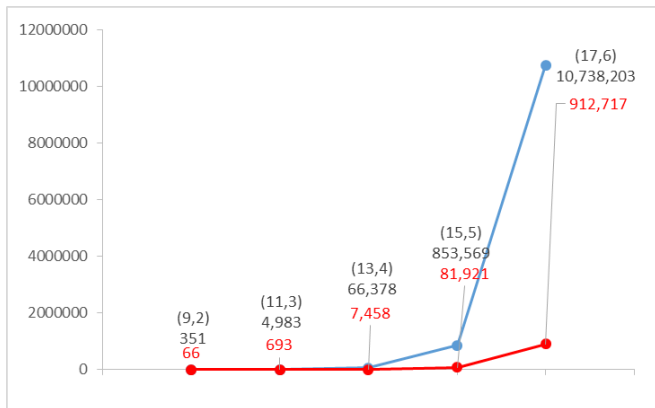


2. From  $x$ 's prefix  $y$ , recursively generate each  $d$ -neighbor  $y'$ , and apply  $P_{(d-d_H(y,y'))}$  to the block whose prefix is  $y'$ .

	acgtaca	$P_4$	acgtaga	$P_3$
acgtacg	atttaca	$P_2$	atttaga	$P_1$
	attcaga	$P_0$		

# Results

## Reduction in recursive neighbor generation



- We still generate individual  $d$ -neighbors, but for  $x$ 's prefix  $y$ ; if we set  $k=5$ ,  $y$ 's neighborhood is only 10-20% the size of  $x$ 's.

# Results

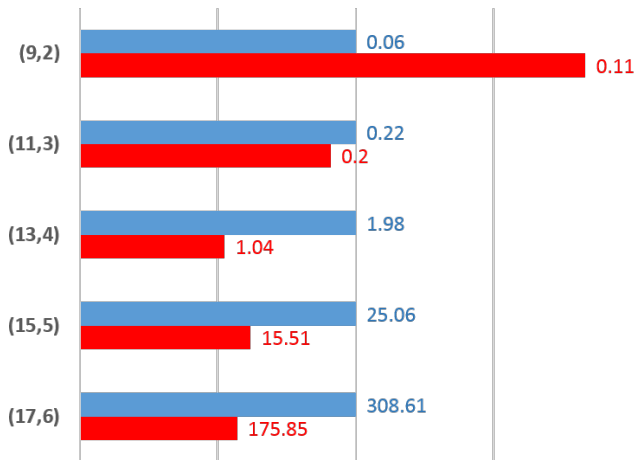
## Runtime comparison of PMS8, qPMS9, and EMS-GT

(l, d)	PMS8	qPMS9	EMS-GT	EMS-GT with speedup
9,2	0.74 s	0.47 s	<b>0.06 s</b>	0.11 s
11,3	1.58 s	1.06 s	0.22 s	<b>0.20 s</b>
13,4	5.39 s	4.52 s	1.98 s	<b>1.04 s</b>
15,5	36.45 s	24.63 s	25.06 s	<b>15.51 s</b>
17,6	3.91 min	<b>1.96 min</b>	5.14 min	<i>2.93 min</i>

Average runtime over 20 synthetic datasets per  $(l, d)$   
on Intel Xeon 2.10 GHz processor (single core execution).

# Results

EMS-GT **without** vs. **with** the speedup technique

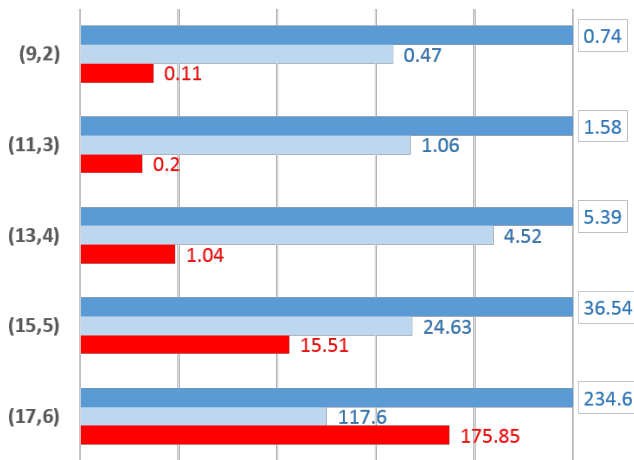


Runtime in seconds, averaged over 20 synthetic datasets per  $(l, d)$



# Results

**PMS8** vs. **qPMS9** vs. **EMS-GT with speedup technique**



Runtime in seconds, averaged over 20 synthetic datasets per  $(l,d)$

# Conclusions

- ▶ We introduced a novel speedup technique for the EMS-GT algorithm, which takes advantage of suffix-related patterns in EMS-GT's bit-based representation of the motif search space.

# Conclusions

- ▶ Our speedup technique **reduces EMS-GT's runtime for most instances**; however, for some  $(l, d)$  values our technique increases runtime due to overhead.

$(l, d)$	(9,2)	(11,3)	(13,4)	(15,5)	(17,6)
<b>Runtime reduction</b>	–	6.7%	47.5%	38.1%	43.0%

# Conclusions

- ▶ EMS-GT with the speedup technique is highly competitive with the state-of-the-art: for all tested  $(l,d)$ , it is faster than PMS8, and faster than qPMS9 except for (17,6).

$(l, d)$	(9,2)	(11,3)	(13,4)	(15,5)	(17,6)
RR from qPMS9	76.6%	81.1%	77.0%	37.0%	–
RR from PMS8	85.1%	87.3%	80.7%	57.6%	25.0%