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

Maria Clara Isabel Sia

October 21, 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 atcactcgttctcctctaatgtgtaaagacgtactaccgaccttа
S_2 acgccgaccggtccgatccttgtatagctcctaacgggcatcagc
S_3 tcctgactgcatcgcgatctcggtagtttcctgttcatcatttt
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

  $S_1 =$ atcactcgttctcctctaatgtgtaaagacgtactaccgaccttta

- 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

$$x_1 = \texttt{cgatcctt} \qquad d_H(x_1, x_2) = 2$$
$$x_2 = \texttt{ccatcgtt}$$

- 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 $d_H(x, x') \leq d$

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

  $= \{$ ccatcgtt,

  acatcgtt,gcatcgtt,tcatcgtt,caatcgtt,cgatcgtt,ctatcgtt,

  ...all l-mers with 1 mismatch

  aaatcgtt,agatcgtt,atatcgtt,gaatcgtt,ggatcgtt,gtatcgtt,

  taatcgtt,tgatcgtt,ttatcgtt,acctcgtt,acgtcgtt,acttcgtt,

  ...all l-mers with 2 mismatches

  $\}$

EMS-GT

Nabos, 2014

- 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 $S_1, S_2, ..., S_{n'}$.

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

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

EMS-GT

Generate-and-test approach

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

S_1 atcactcgttctcctctaatgtgtaaagacgtactaccgaccttа

S_2 acgccgaccggtccgatccttgtatagctcctaacgggcatcagc

S_3 tcctgactgcatcgcgatctcggtagtttcctgttcatcattttt

S_4 ggccctcagcatcgtgcgtcctgctaacacattcccatgcagctt

S_5 tgaaaagaatttacggtaaaggatccacatccaatcgtgtgaaag

EMS-GT

Bit-based representation of the search space

- The search space contains all 4^l possible l-mers that can be formed with $\Sigma = \{\mathtt{a}, \mathtt{c}, \mathtt{g}, \mathtt{t}\}$.
- To represent sets in this space, EMS-GT assign 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

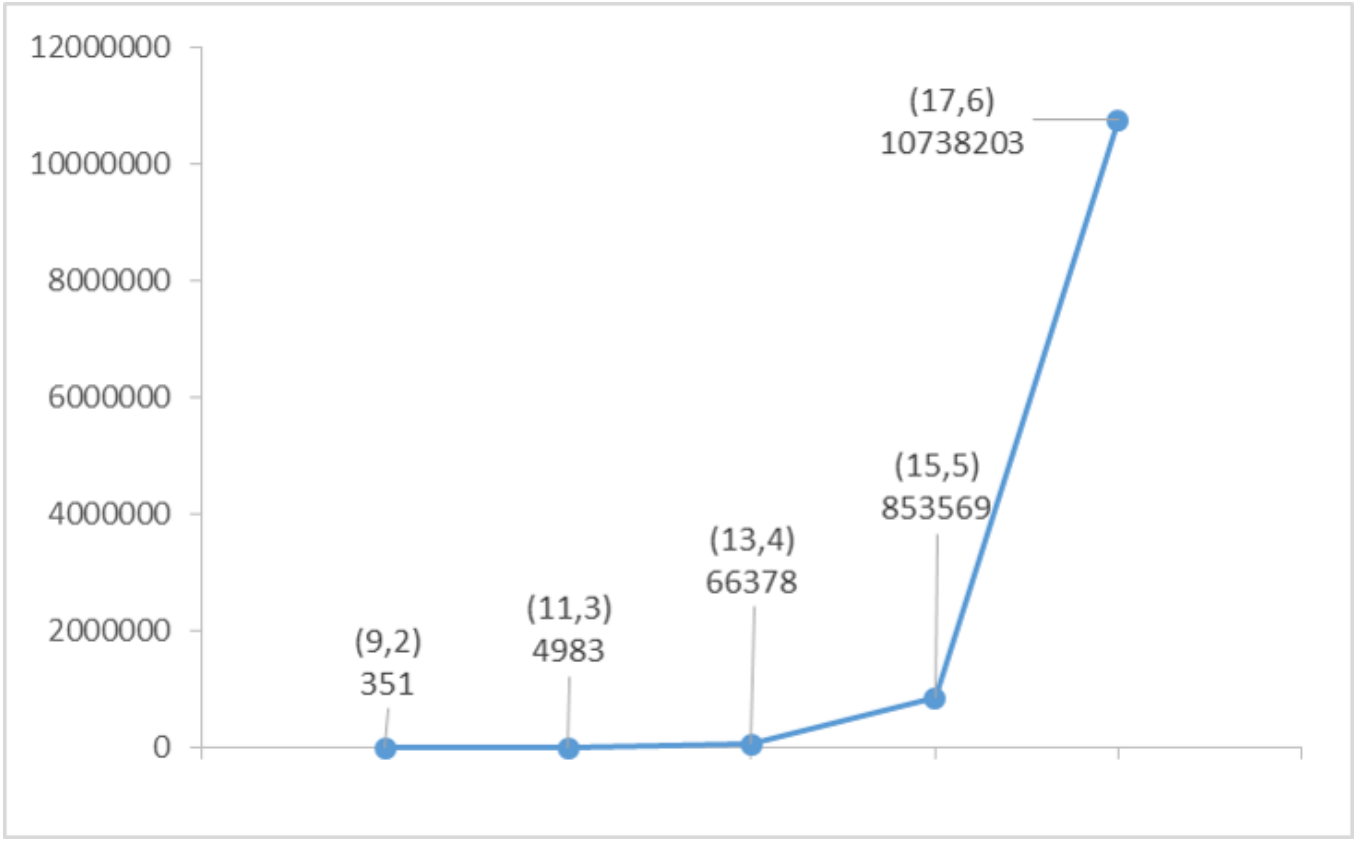
- $N($ `acgt`, 1 $)$: $4^l = 256$, $\frac{4^l}{32} = 8$

  emphasize alphabetical order, maybe show how l-mer corresponds to binary num

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

EMS-GT

Bit-based representation of the search space



- 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

- The bit-array N_x representing the neighborhood of l-mer x can be partitioned into blocks of 4^k bits each.
- l-mers in a block all begin with the same $(l - k)$ characters; all the blocks conform to one of $(k + 2)$ patterns.

Speedup technique

Key observation

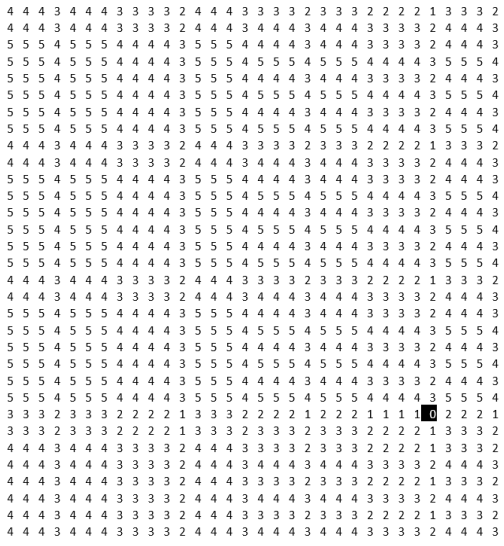
- The bit-array N_x representing the neighborhood of l-mer x can be partitioned into blocks of 4^k bits each.
- l-mers in a block all begin with the same $(l - k)$ characters; all the blocks conform to one of $(k + 2)$ patterns.

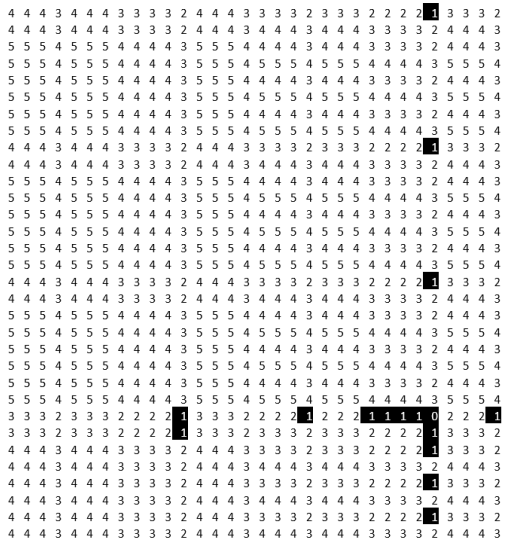
Ex. $N($ `acgtacgtacgt`, 5 $)$, k $= 5$

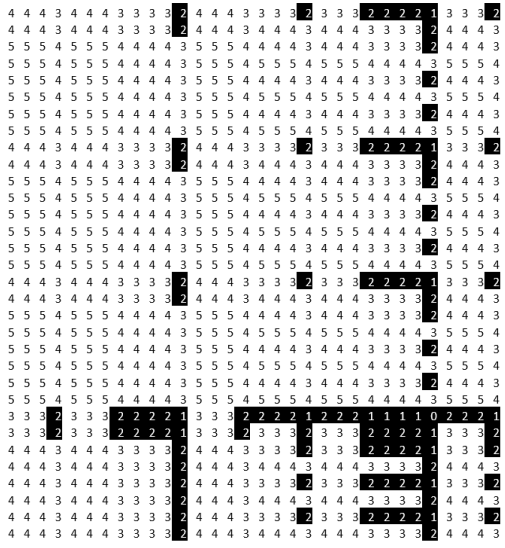
block size $= 4^5 = 32 \times 32$

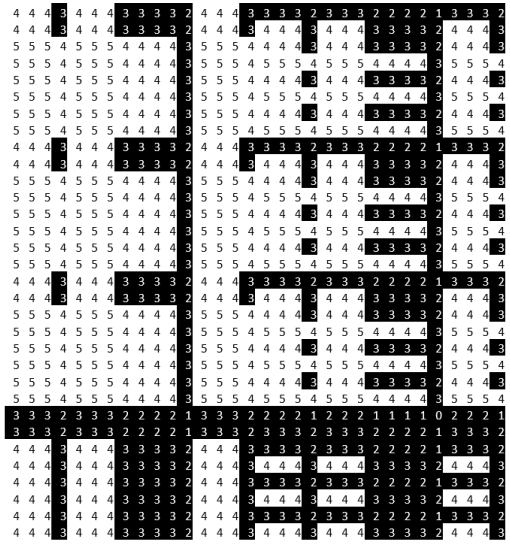
Speedup technique

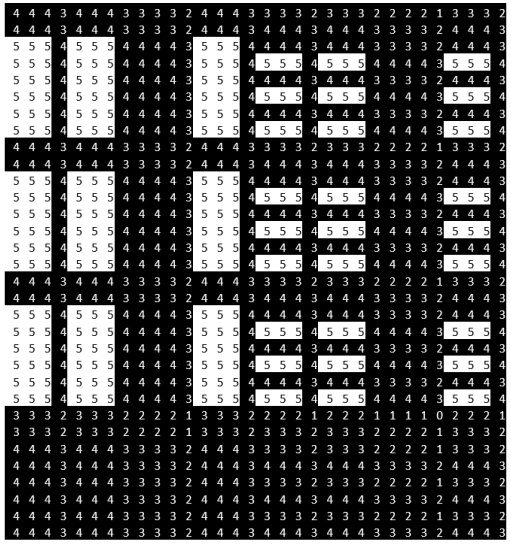
Block patterns in the neighborhood of acgtacgtacgt

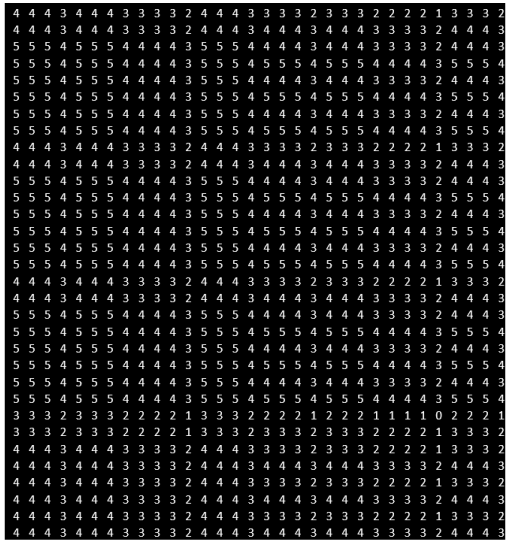












Speedup technique

Key observation

Crucial frame here!

"Think of splitting the allowable mismatches d between prefix and suffix"

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	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
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
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
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	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	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

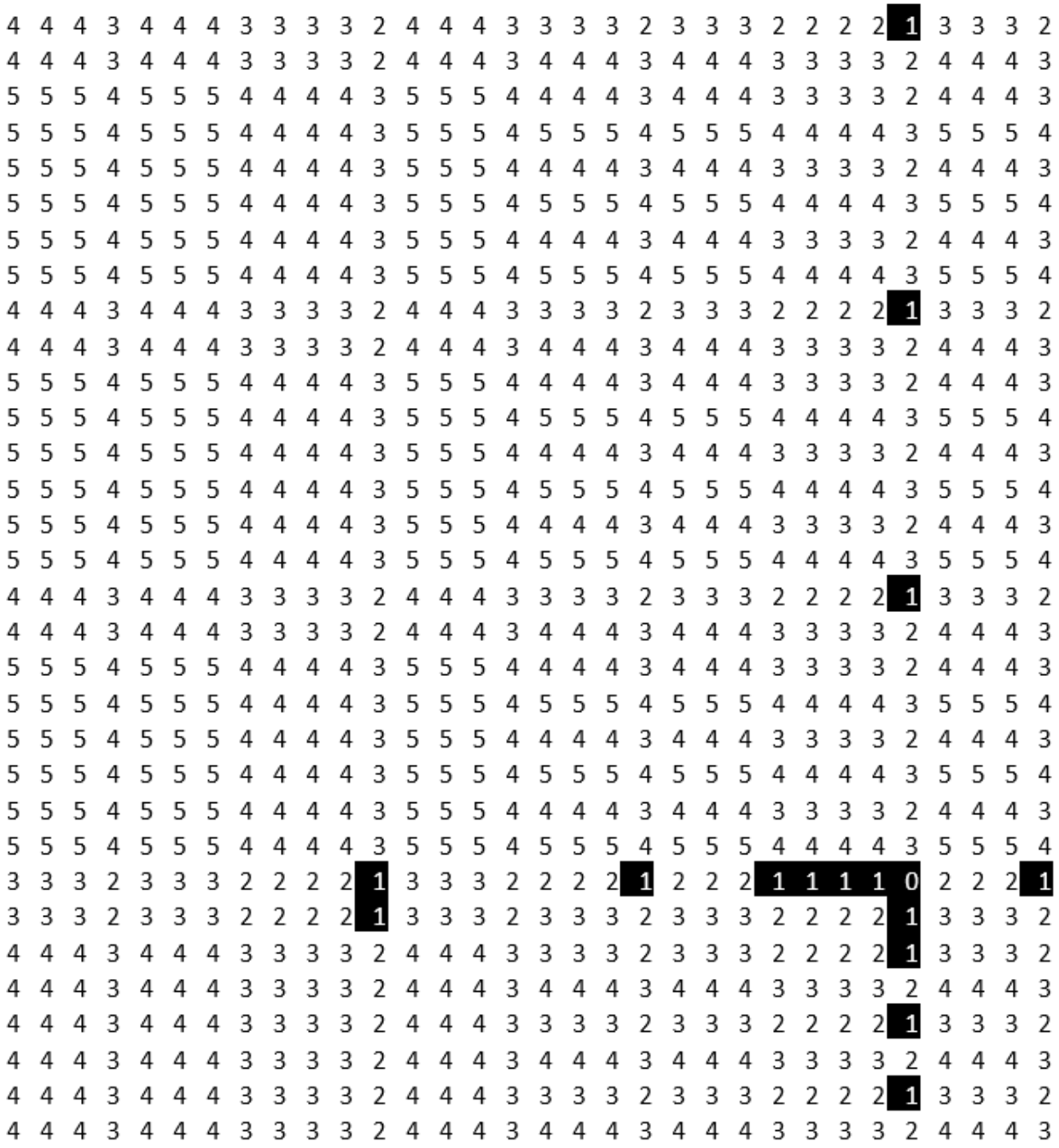
← ttttt

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	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
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
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
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	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	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

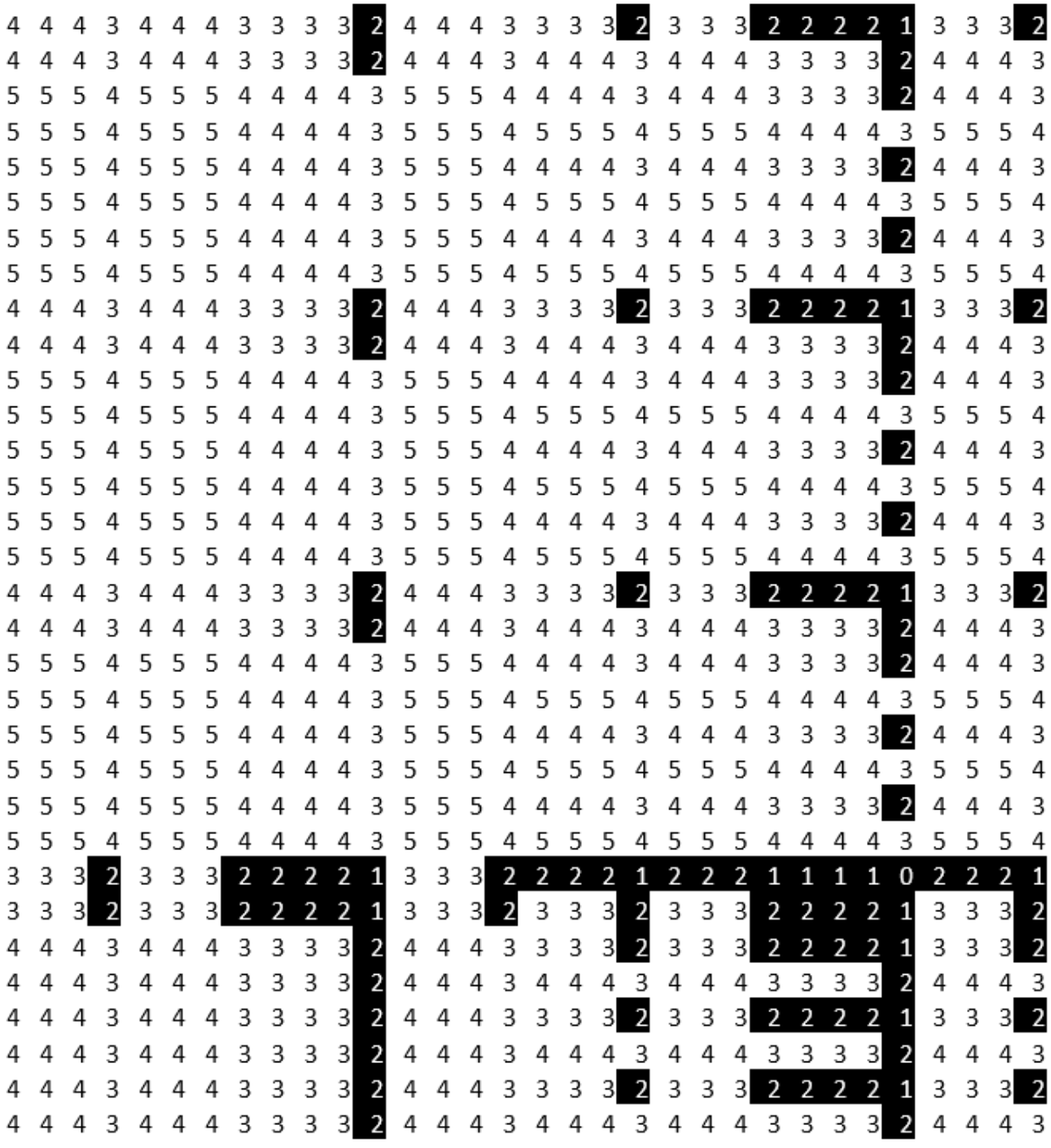
← ttttt

aaaaa →



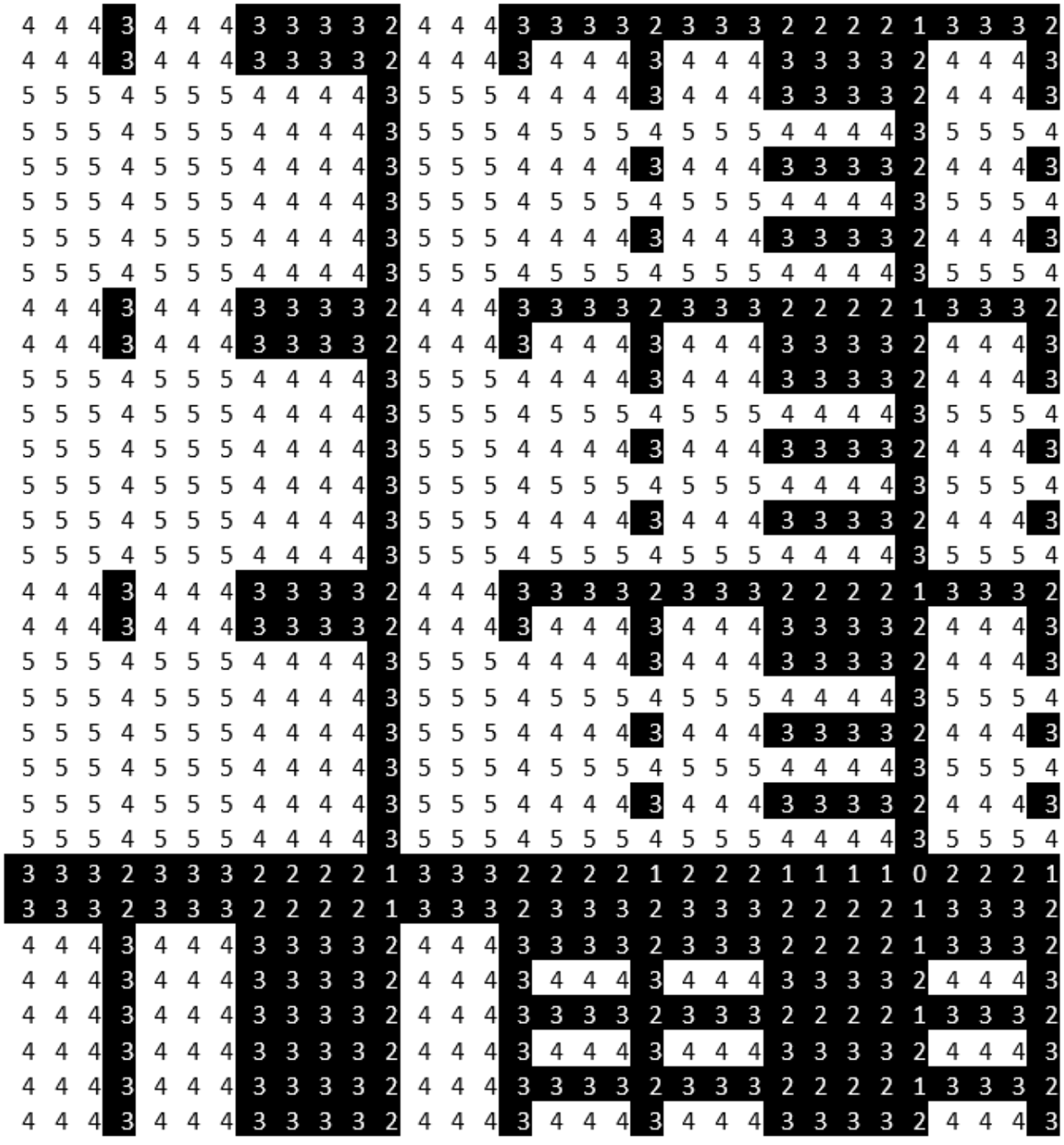
← ttttt

aaaaa →



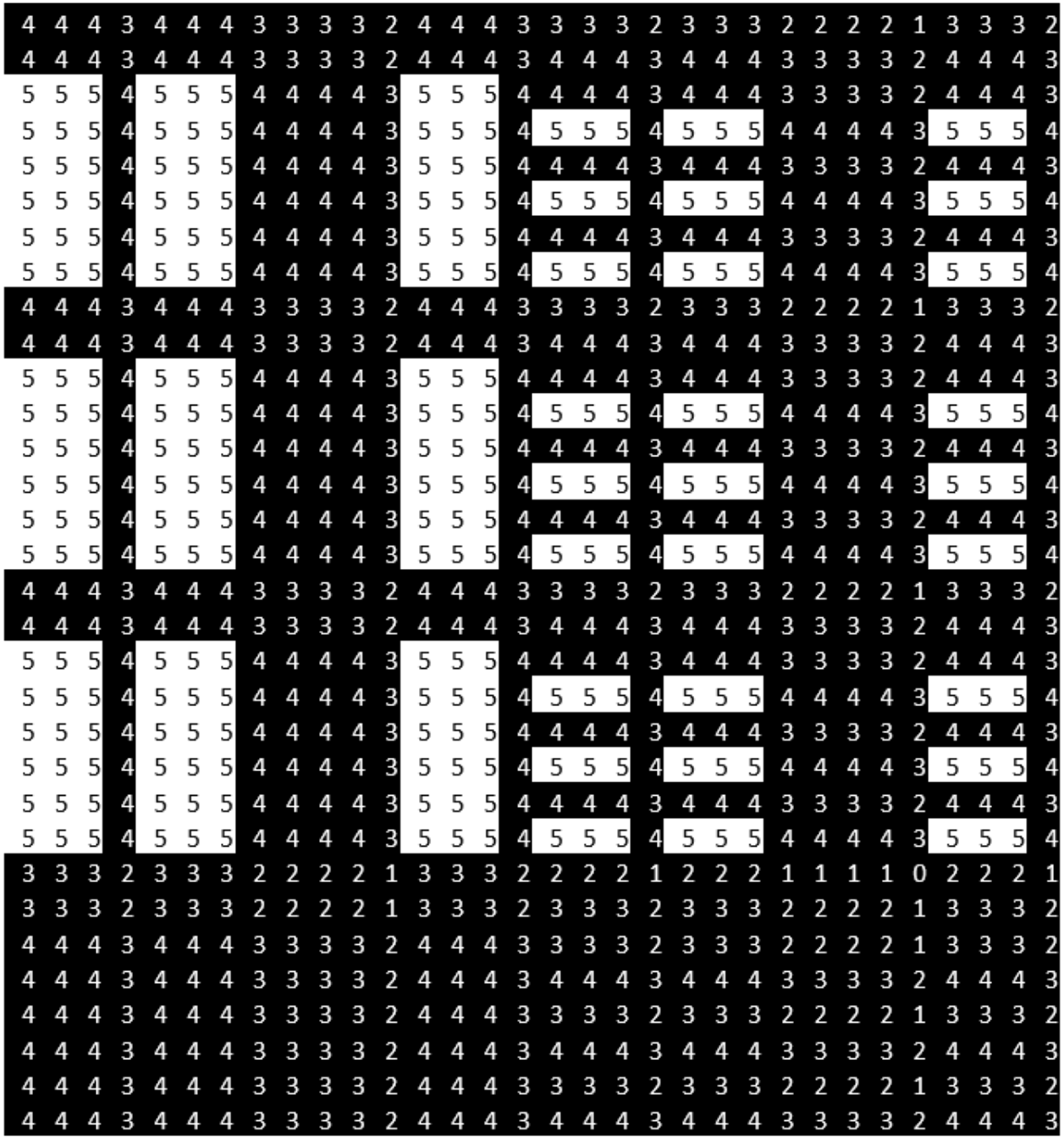
← ttttt

aaaaa $\rightarrow$



$\leftarrow$ ttttt

aaaaa →



← 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	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
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
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
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	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	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

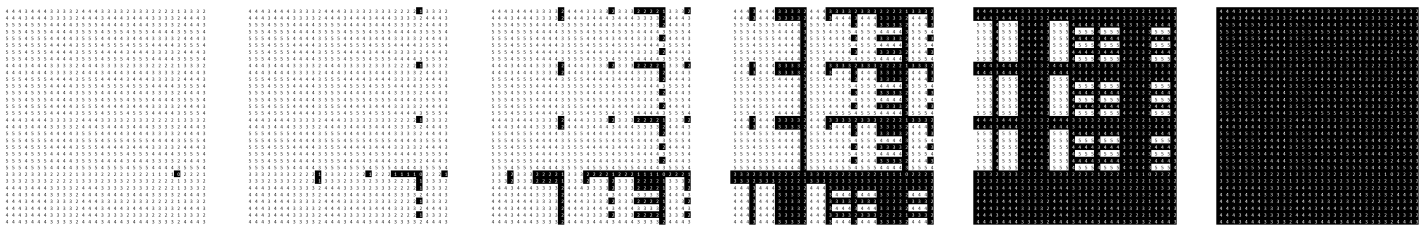
← ttttt

Speedup technique

Generate and apply patterns

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

1. From z, generate P, the set of non-empty block patterns.

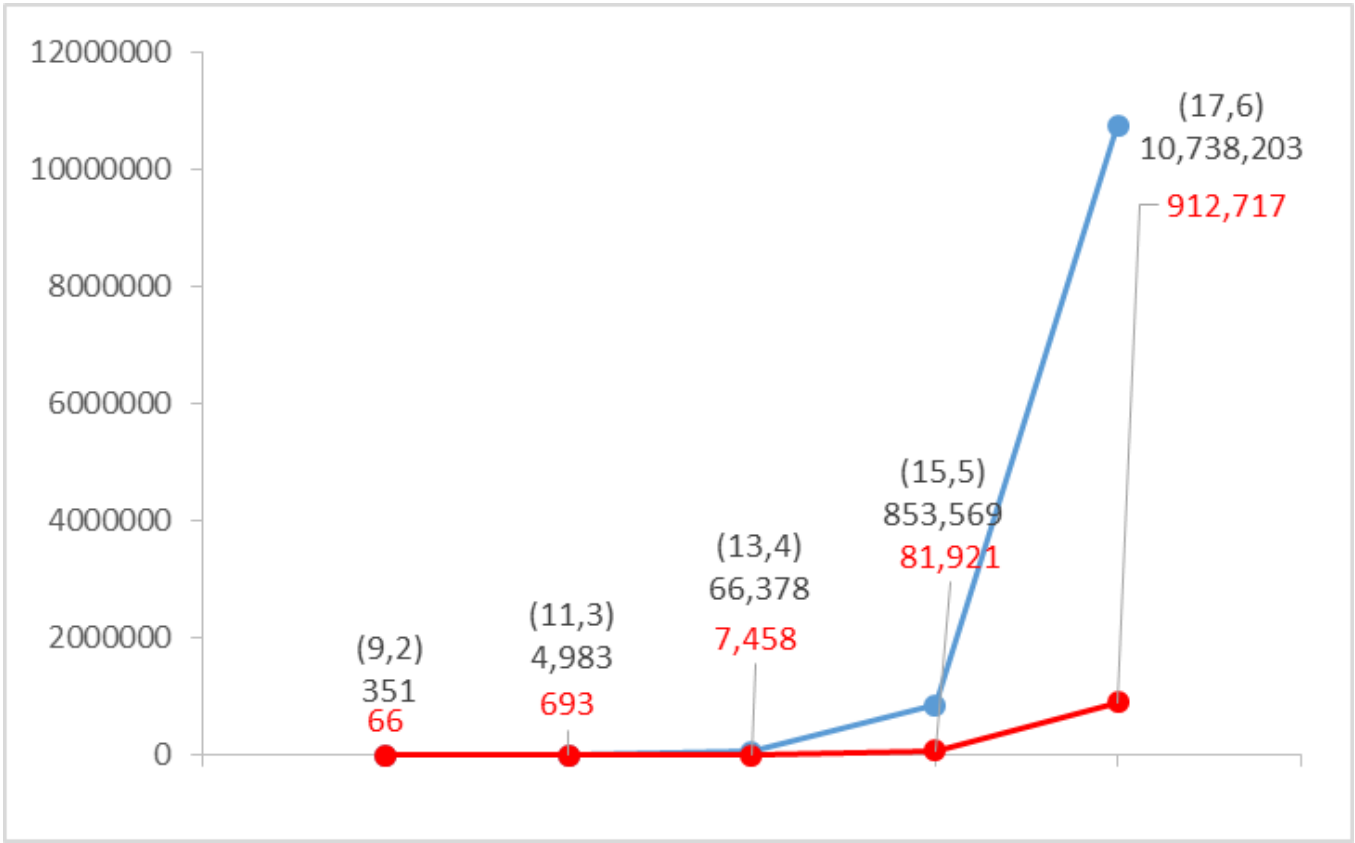


$P[0]$ $P[1]$ $P[2]$ $P[3]$ $P[4]$ $P[5]$

2. From y, recursively generate each d-neighbor y' of y, find the block in N_x that has y' as its prefix, and apply $P[\ d - d_H(y, y')\ ]$ to that block.

Results

Performance improvement with speedup technique



- We still generate d-neighbors recursively, but for a shorter sequence y', of length $l - k$; this graph shows that, for k=5, neighborhood size is reduced by a factor of 10.

Results

Performance improvement with speedup technique

(l, d)	**Without speedup**	**With speedup, k=5**	**% reduction**
(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%

Average performance for 20 synthetic datasets per (l,d) instance

Results

Performance against PMS8 and qPMS9

(l, d)	PMS8	qPMS9	EMS-GT	% reduction
(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	–

Average performance for 20 synthetic datasets per (l,d) instance

Conclusions

- speedup technique
- runtime improvement
- comparison with state-of-the-art