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

Maria Clara Isabel Sia

October 20, 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
- motif finding is 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	`atcactcgttctcctctaatgtgtaaagacgtactaccgacctta`
S_2	`acgccgaccggtccgatccttgtatagctcctaacgggcatcagc`
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 (i.e. probability-based sampling, random projection) perform an iterative local search which is efficient, but not guaranteed to find all motifs
- exact methods (i.e. combinatorial search, search-space 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

  $S_1 =$ `atcactcgttctcctctaatgtgtaaagacgtactaccgacctta`

- 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(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 the (l,d) planted motif problem for any arbitrary instance with $l \leq 17$
- efficiently operates on a compact, bit-based representation of the motif 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 `atcactcgttctcctctaatgtgtaaagacgtactaccgacctta`

S_2 `acgccgaccggtccgatccttgtatagctcctaacgggcatcagc`

S_3 `tcctgactgcatcgcgatctcggtagtttcctgttcatcatttt`

S_4 `ggccctcagcatcgtgcgtcctgctaacacattcccatgcagctt`

S_5 `tgaaaagaatttacggtaaaggatccacatccaatcgtgtgaaag`

EMS-GT

Bit-based efficiency strategies

- l-mer enumeration scheme
- Bit-based representation of sets
- Bit-array compression
- Recursive neighborhood generation

EMS-GT

Bit-based efficiency strategies

- *l*-mer enumeration scheme
  EMS-GT maps an *l*-mer to a 2*l*-bit binary number by replacing each character with two bits (a=00, c=01, g=10, t=11).

  aaaaa aaaac aaaag ..., tacgt tacta ...

  0000000000, 0000000001, 0000000010, ..., 1100011011, 1100011100, ...

  $\hookrightarrow$ 0 $\hookrightarrow$ 1 $\hookrightarrow$ 2 $\hookrightarrow$ 795 $\hookrightarrow$ 796

- Bit-based representation of sets
- Bit-array compression
- Recursive neighborhood generation

EMS-GT

Bit-based efficiency strategies

- l-mer enumeration scheme
- Bit-based representation of sets
  The motif search space includes all 4^l l-mers that can be formed with $\Sigma = \{\mathtt{a}, \mathtt{c}, \mathtt{g}, \mathtt{t}\}$. To represent sets in this space, EMS-GT assigns each l-mer a bit flag:

$$Flags[\ 795\ ] = \begin{cases} 1 & \text{if } \mathtt{tacgt} \text{ is a member of the set,} \\ 0 & \text{otherwise.} \end{cases}$$

- Bit-array compression
- Recursive neighborhood generation

EMS-GT

Bit-based efficiency strategies

- l-mer enumeration scheme
- Bit-based representation of sets
- Bit-array compression
  EMS-GT stores 4^l bit flags as an array of $\frac{4^l}{32}$ 32-bit integers.

  Ex. the flag for `tacgt` is in int index $\frac{795}{32} = 24$, at bit $(795 \ mod \ 32) = 27$.

	27
23	00000011100001000100100000110011
24	00110111100000000011100000011100
25	11111001011001000011111100000011

- Recursive neighborhood generation

EMS-GT

Bit-based efficiency strategies

- *l*-mer enumeration scheme
- Bit-based representation of sets
- Bit-array compression
- Recursive neighborhood generation

  To generate a d-neighborhood, EMS-GT recursively generates each d-neighbor, then finds and sets its bit flag.

  Generating a d-neighbor changes up to d characters in x, with 3 alternatives per change (ex. `c` $\rightarrow$ `a`, `g` or `t`); thus,

  l-mer x will have $\displaystyle\sum_{i=0}^{d} \binom{l}{i} 3^i$ possible d-neighbors.

EMS-GT

Key observations

(l, d)	Neighborhood size
9,2	351
11,3	4,983
13,4	66,378
15,5	853,569
17,6	10,738,203

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

Methods

Research objectives

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.

Methods

Work summary

To fulfill these objectives, we:

- investigated repeating block patterns in EMS-GT's bit-based representation of an l-mer neighborhood;
- designed a more efficient bit-setting procedure that sets bits according to these block patterns; and
- measured EMS-GT's performance on synthetic data for "challenging" (l,d): (9,2), (11,3), (13,4), (15,5) and (17,6).

Speedup technique

Key observation

A 4^l-bit array N_x representing an l-mer neighborhood

can be divided into consecutive 4^k-bit blocks,

and each block will conform to one of $(k+2)$ patterns.

Speedup technique

Key observation

A 4^l-bit array N_x representing an l-mer neighborhood

can be divided into consecutive 4^k-bit blocks,

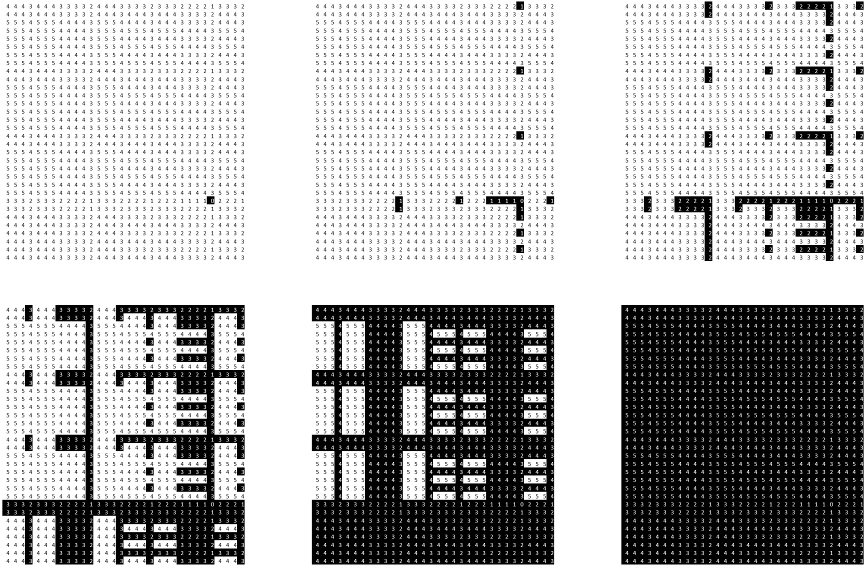
and each block will conform to one of $(k+2)$ patterns.

Ex. $x = \text{acgtacgtacgt}$, $d = 6$, $\text{k} = 5$

Block size $= 4^5 = 1024 = 32 \times 32$

Speedup technique

Block patterns in the neighborhood of acgtacgtacgt



Speedup technique

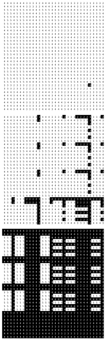
In terms of prefix and suffix

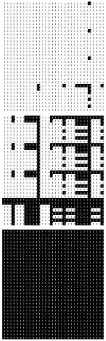
acgtacg

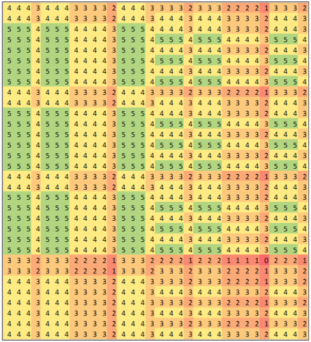
x's prefix y of length $(l-k)$ determines which patterns apply to which blocks.

tacgt

x's suffix z of length k determines the structure of the $(k+2)$ patterns.







Speedup technique

In terms of prefix and suffix

- A 4^l-bit array N_x representing an l-mer neighborhood
- can be divided into consecutive 4^k-bit blocks,
- and each block will conform to one of $(k+2)$ patterns.

Speedup technique

In terms of prefix and suffix

- A 4^l-bit array N_x representing an l-mer neighborhood

  The distance between two l-mers is the sum of the distances between their prefixes and between their suffixes; therefore,

$$N_x[\, x' \,] = \begin{cases} 1 & \text{if } d_H(y, y') + d_H(z, z') \leq d, \\ 0 & \text{otherwise.} \end{cases} \quad \text{for } x' = y'z'.$$

  This means we set a bit iff $d_H(z, z') \leq d - d_H(y, y')$.

- can be divided into consecutive 4^k-bit blocks,
- and each block will conform to one of $(k + 2)$ patterns.

Speedup technique

In terms of prefix and suffix

- A 4^l-bit array N_x representing an l-mer neighborhood
- can be divided into consecutive 4^k-bit blocks,
  Blocks in N_x for x=acgtacgtacgt, k=5:

Block 0:	bit flags for `aaaaaaaaaaa` - `aaaaaattttt`
Block 1:	bit flags for `aaaaaacaaaaa` - `aaaaaacttttt`
...	...
Block 1,734:	bit flags for `acgtacgaaaaa` - `acgtacgttttt`
...	...
Block 16,833:	bit flags for `ttttttgaaaaa` - `ttttttgttttt`
Block 16,834:	bit flags for `tttttttaaaaa` - `tttttttttttt`

  The l-mers in each block all have the same prefix.
  Blocks all follow the same sequence of suffixes (aaaaa - ttttt).

- and each block will conform to one of $(k + 2)$ patterns.

Speedup technique

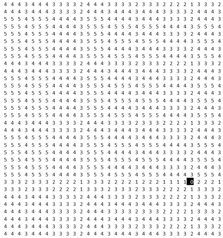
In terms of prefix and suffix

- A 4^l-bit array N_x representing an l-mer neighborhood
- can be divided into consecutive 4^k-bit blocks,
- and each block will conform to one of $(k+2)$ patterns.
  We set a bit iff $d_H(z, z') \leq d - d_H(y, y')$; $0 \leq d_H(z, z') \leq k$, therefore there are $(k+2)$ ways to set bits in a block:

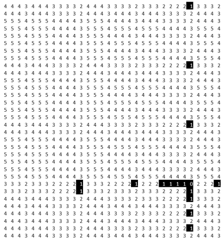
For blocks where:	we set bits for:
$d - d_H(y, y') < 0$	no suffixes
$d - d_H(y, y') = 0$	suffix z only
$d - d_H(y, y') = 1$	suffixes with up to 1 mismatch with z
$d - d_H(y, y') = 2$	suffixes with up to 2 mismatches with z
...	...
$d - d_H(y, y') = k\text{-}1$	suffixes with up to k-1 mismatches with z
$d - d_H(y, y') = k$	suffixes with up to k mismatches with z (all suffixes)

Speedup technique

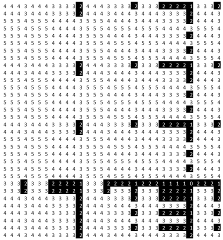
Block patterns in the neighborhood of acgtacgtacgt



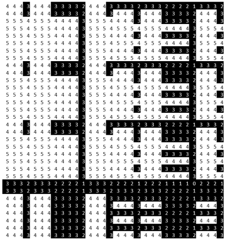
$d_H(y, prefix) = d$



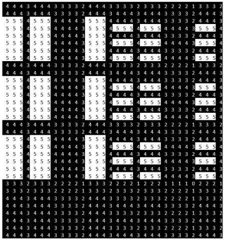
$d_H(y, prefix) = d - 1$



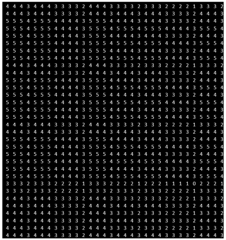
$d_H(y, prefix) = d - 2$



$d_H(y, prefix) = d - 3$



$d_H(y, prefix) = d - 4$



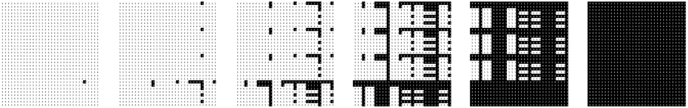
$d_H(y, prefix) = d - 5$

Speedup technique

Generate and apply patterns

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

1. From z, generate P, the set of $(k+1)$ 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', find the block that has y' as its prefix, and apply $P[\ d - d_H(y, y')\ ]$.

Results

Performance improvement with speedup technique

(l, d)	**Without speedup** $\lvert N(x,d)\rvert$	**With speedup, k=5** $\lvert N(y,d)\rvert$	**% 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%

Reduction in neighborhood size without vs. with speedup

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