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

Maria Clara Isabel Sia

October 17, 2015

Introduction

DNA motif finding

- motifs: repeated sub-sequences in DNA that have some biological significance
- DNA motif finding: search 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 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`

Given: set of DNA sequences $\mathcal{S} = \{S_1, ...S_n\}$, motif length l, allowable mismatches d

Find: motif M occurring with at most d mismatches in each of the sequences in $\mathcal{S}$.

There are two main types of motif search algorithms:

- heuristic algorithms perform an iterative local search, ex. repeatedly refining an input sampling or projection until a motif is found
- exact algorithms perform an exhaustive search of possible motifs, i.e.

Introduction

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

- *l*-mer
- Hamming distance $dH(x_1, x_2)$
- *d*-neighborhood $N(x, d)$ of *l*-mer x
- *d*-neighborhood $\mathcal{N}(S, d)$ of sequence S

Introduction

l-mers, Hamming distances, and d-neighborhoods

- l-mer
  - sequence of length l

  $S_1 =$ `atcactcgttctcctctaatgtgtaaagacgtactaccgacctta`

- Hamming distance $dH(x_1, x_2)$
- d-neighborhood $N(x, d)$ of l-mer x
- d-neighborhood $\mathcal{N}(S, d)$ of sequence S

Introduction

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

- l-mer
- Hamming distance $dH(x_1, x_2)$
  - number of mismatches between l-mers x_1 and x_2

$$x_1 = \texttt{cgatcctt}$$
$$x_2 = \texttt{ccatcgtt}$$

- d-neighborhood $N(x, d)$ of l-mer x
- d-neighborhood $\mathcal{N}(S, d)$ of sequence S

Introduction

l-mers, Hamming distances, and d-neighborhoods

- l-mer
- Hamming distance $dH(x_1, x_2)$
- d-neighborhood $N(x, d)$ of l-mer x
  - set of all l-mers having at most d mismatches with x

  N(ccatcgtt, 2)

  = { ccatcgtt,

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

  ...1 mismatch

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

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

  ...2 mismatches

  }
- d-neighborhood $\mathcal{N}(S, d)$ of sequence S

Introduction

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

- *l*-mer
- Hamming distance $dH(x_1, x_2)$
- *d*-neighborhood $N(x, d)$ of *l*-mer x
- *d*-neighborhood $\mathcal{N}(S, d)$ of sequence S
  - set of all *d*-neighbors of all *l*-mers in S

$S =$ `acgccgattacatccgatccttgtatagctcctaacgggcatcac`

$\mathcal{N}(S, 2) = N$(acgccgat, 2) $\cup$ N(cgccgatt, 2) $\cup$ ... $\cup$ N(ggcatcac, 2)

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 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 `tcctgactgcatcgcgatctcggtagtttcctgttcatcattttt`

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

- *l*-mer enumeration scheme

  EMS-GT maps an l-mer to a $2l$-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 $\Sigma = \{\texttt{a}, \texttt{c}, \texttt{g}, \texttt{t}\}$. To represent sets in this space, EMS-GT assigns each l-mer a bit flag, indexed by mapping:

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

- 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.

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

	27
23	00000011100001000100100000110011
24	00110111110000000000111000000011100
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 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,

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

EMS-GT

Key observations

include graph here

- 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.
- we know that EMS-GT's main bit-array enumerates l-mers in strict alphabetical order; can we use this for greater efficiency?

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 EMS-GT 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).

Results

Key observation

A bit-array N_x representing an l-mer neighborhood
can be divided into consecutive blocks of 4^k bits each.

Each block will conform to one of at most $(k + 2)$ patterns.

Let $k = 5$.

Results

Deriving block patterns in an l-mer neighborhood

- Let bit-array N_x represent the neighborhood of l-mer x
- We show Hamming distances d_H are additive
- We redefine N_x with the additive property
- We examine $(k+2)$ cases for $d - d_H(y, y')$

Results

Deriving block patterns in an l-mer neighborhood

- Let bit-array N_x represent the neighborhood of l-mer x
  A bit x' is set in N_x if and only if x' is a d-neighbor of x.

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

- We show Hamming distances d_H are additive
- We redefine N_x with the additive property
- We examine $(k+2)$ cases for $d - d_H(y, y')$

Results

Deriving block patterns in an l-mer neighborhood

- Let bit-array N_x represent the neighborhood of l-mer x
- We show Hamming distances d_H are additive
  We can divide an l-mer x into a prefix y and a suffix z, where $|z| = k$:

$$d_H(x, x') = d_H(y, y') + d_H(z, z').$$

$$\begin{array}{lcl} x & = & \texttt{acgtacg} \quad \texttt{tacgt} \\ x' & = & \texttt{acctatg} \quad \texttt{taggt} \end{array}$$

- We redefine N_x with the additive property
- We examine $(k+2)$ cases for $d - d_H(y, y')$

Results

Deriving block patterns in an l-mer neighborhood

- Let bit-array N_x represent the neighborhood of l-mer x
- We show Hamming distances d_H are additive
- We redefine N_x with the additive property

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

The condition for setting a bit in N_x is now

$$d_H(y, y') + d_H(z, z') \leq d$$
$$\text{or, } d_H(z, z') \leq d - d_H(y, y')$$

- We examine $(k + 2)$ cases for $d - d_H(y, y')$

Results

Deriving block patterns in an l-mer neighborhood

- Let bit-array N_x represent the neighborhood of l-mer x
- We show Hamming distances d_H are additive
- We redefine N_x with the additive property
- We examine $(k+2)$ cases for $d - d_H(y, y')$
  We set bit $x' = y'z'$ in N_x iff $d_H(z, z') \leq d - d_H(y, y')$.
  The value of $dH(z, z')$ ranges from 0 to $|z| = k$, therefore

if	$d - d_H(y, y') < 0$	no bits are set	Case -1
if	$0 \leq d - d_H(y, y') < k$	some bits are set	Cases 0-(k-1)
if	$k \leq d - d_H(y, y')$	all bits are set	Case k

Results

Applying block patterns in an l-mer neighborhood

- l-mers in $N_{(x,d)}$ are organized into blocks

- Do: pre-generate distribution of suffix distances $dH(z, z')$
- Do: compute all prefixes of distance $dH(y, y') \leq d$

Results

Applying block patterns in an l-mer neighborhood

- l-mers in $N_{(x,d)}$ are organized into blocks
  The strict alphabetical enumeration scheme
- Do: pre-generate distribution of suffix distances $dH(z, z')$
- Do: compute all prefixes of distance $dH(y, y') \leq d$

Results

Applying block patterns in an l-mer neighborhood

- l-mers in $N_{(x,d)}$ are organized into blocks
- Do: pre-generate distribution of suffix distances $dH(z, z')$
- Do: compute all prefixes of distance $dH(y, y') \leq d$

Results

Applying block patterns in an l-mer neighborhood

- l-mers in $N_{(x,d)}$ are organized into blocks

- Do: pre-generate distribution of suffix distances $dH(z, z')$
- Do: compute all prefixes of distance $dH(y, y') \leq d$

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	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%

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

Results

Performance against PMS8 and qPMS9

(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	–

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

Conclusions