Randomized Algorithms for Motif Finding

Outline
- Randomized Algorithms
- Greedy Profile Motif Search
- Gibbs Sampling

Randomized Algorithms
- Randomized algorithms make random rather than deterministic decisions.
- Commonly used when no exact and fast algorithm is known.

Two Types of Randomized Algorithms
- Las Vegas Algorithms – always produce the correct solution
- Monte Carlo Algorithms – not guaranteed to return the correct solution
  • Examples?

Motif finding: a probabilistic approach
- Motif Finding Problem: Given a list of \( t \) sequences each of length \( n \), find the "best" pattern of length \( l \) that appears in each of the \( t \) sequences.
- A probabilistic approach: randomly select possible starting locations.

PWMs Revisited
- Let \( s=(s_1,\ldots,s_t) \) be the set of starting positions for \( l \)-mers in our \( t \) sequences.
- The substrings corresponding to these starting positions will form:
  - \( t \times l \) alignment matrix
  - \( 4 \times l \) PWM, \( P \).

*We make a special note that the PWM will be defined in terms of the frequencies, and not counts of letters.
Scoring a sequence with a PWM

- **Prob(a|P):** the probability that an l-mer a was generated by the PWM P:
  \[ \text{Prob}(a|P) = \prod_{i=1}^{n} p_{a_i} \]

- If a is very similar to the consensus string of P, then \( \text{Prob}(a|P) \) will be high.
- If a is very different, then \( \text{Prob}(a|P) \) will be low.

**Scoring a sequence with a PWM (cont)**

Given a PWM:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>T</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1/2</td>
<td>7/8</td>
<td>3/8</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>1/8</td>
<td>0</td>
<td>1/2</td>
<td>3/8</td>
</tr>
<tr>
<td>T</td>
<td>1/8</td>
<td>1/8</td>
<td>0</td>
<td>1/4</td>
</tr>
<tr>
<td>G</td>
<td>1/4</td>
<td>0</td>
<td>1/8</td>
<td>3/8</td>
</tr>
</tbody>
</table>

The probability of the consensus string:

\[ \text{Prob}(a|aaccct) = ???? \]

**Scoring a sequence with a PWM (cont)**

Given a PWM:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>T</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1/2</td>
<td>7/8</td>
<td>3/8</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>1/8</td>
<td>0</td>
<td>1/2</td>
<td>3/8</td>
</tr>
<tr>
<td>T</td>
<td>1/8</td>
<td>1/8</td>
<td>0</td>
<td>1/4</td>
</tr>
<tr>
<td>G</td>
<td>1/4</td>
<td>0</td>
<td>1/8</td>
<td>3/8</td>
</tr>
</tbody>
</table>

The probability of the consensus string:

\[ \text{Prob}(a|aaccct) = 1/2 x 7/8 x 3/8 x 5/8 x 3/8 x 7/8 = .033646 \]

Probability of a different string:

\[ \text{Prob}(a|atacag) = 1/2 x 1/8 x 3/8 x 5/8 x 1/8 x 1/8 = .001602 \]

**P-Most Probable l-mer**

- Define the P-most probable l-mer from a sequence as an l-mer in that sequence which has the highest probability of being created from the PWM P.

\[ \mathbf{P} = \begin{array}{cccc}
A & 1/2 & 7/8 & 3/8 \\
C & 1/8 & 0 & 1/2 \\
T & 1/8 & 1/8 & 0 \\
G & 1/4 & 0 & 1/8 \\
\end{array} \]

Given the sequence ctataacccttacatc, find the P-most probable l-mer

**P-Most Probable l-mer (cont)**

Find the \( \text{Prob}(a|P) \) of every possible 6-mer:
- First try: ctataacccttacatc
- Second try: ctataacccttacatc
- Third try: ctataacccttacatc

Slide a window of size 6, evaluating every 6-mer in the sequence
**P-Most Probable l-mer (cont’d)**

Compute \( \text{prob}(a|P) \) for every possible 6-mer:

| String, Highlighted in Red | Calculations | prob(a|P) |
|---------------------------|--------------|----------|
| ctataaaacctttacatc        | 1/8 x 1/8 x 3/8 x 0 x 1/8 x 0 | 0       |
| ctataaaacctttacatc        | 1/2 x 7/8 x 0 x 0 x 1/8 x 0 | 0       |
| ctataaaacctttacatc        | 1/2 x 7/8 x 0 x 0 x 1/8 x 0 | 0       |
| ctataaaacctttacatc        | 1/8 x 7/8 x 0 x 0 x 1/8 x 0 | 0       |
| ctataaaacctttacatc        | 1/2 x 7/8 x 0 x 0 x 1/8 x 0 | 0       |
| ctataaaacctttacatc        | 1/2 x 7/8 x 0 x 0 x 1/8 x 0 | 0       |
| ctataaaacctttacatc        | 1/8 x 7/8 x 0 x 0 x 1/8 x 0 | 0       |
| ctataaaacctttacatc        | 1/2 x 7/8 x 0 x 0 x 1/8 x 0 | 0       |
| ctataaaacctttacatc        | 1/8 x 7/8 x 0 x 0 x 1/8 x 0 | 0       |

**P-Most Probable l-mer (cont’d)**

The P-Most Probable 6-mer in the sequence is aaacct:

\[ \text{prob}(\text{aaacct}|P) = 0.0336 \]

because \( \text{Prob}(\text{aaacct}|P) = 0.0336 \) is greater than the \( \text{Prob}(a|P) \) of any other 6-mer in the sequence.

**Dealing with Zeroes**

- In our toy example \( \text{prob}(a|P) = 0 \) in many cases. In practice, we will have more sequences so the number of elements in the PWM with a frequency of zero will be smaller.
- To avoid entries with \( \text{prob}(a|P) = 0 \), one usually replaces zero with a small number so that a zero does not make the entire probability of a sequence zero.

**P-Most Probable l-mers in Many Sequences**

- Find the P-most probable l-mer in each of the sequences.

**P** =

```
A 12 7/8 3/8 0 1/8 0
C 1/8 0 1/2 5/8 3/8 0
G 1/8 0 1/8 0 1/4 7/8
T 1/8 1/8 0 0 1/4 7/8

1 1 1 1 1 1 1
2 1 1 1 1 1 1
3 1 1 1 1 1 1
4 1 1 1 1 1 1
5 1 1 1 1 1 1
6 1 1 1 1 1 1
7 1 1 1 1 1 1
8 1 1 1 1 1 1
9 1 1 1 1 1 1
```

**P-Most Probable l-mers in Many Sequences (cont)**

**P** =

```
cttaaaacgttcatc
atagcgaattgactg
cagcccgagaacctt
tgctatcttacatctg
tacctttccactctac
tccaaatctttatatca
ggtcatcttttatcatc
```

**P-Most Probable l-mers form a new PWM**
Comparing New and Old PWMs

Red – frequency increased, Blue – frequency decreased

Greedy PWM Motif Search

1) Select random starting positions.
2) Create a PWM $P$ from the substrings at these starting positions.
3) Find the $P$-most probable $l$-mer $a$ in each sequence and change the starting position to the starting position of $a$.
4) Compute a new PWM based on the new starting positions after each iteration and proceed until we cannot increase the score anymore.

GreedyPWMsearch Algorithm

GreedyPWMsearch(DNA, $t$, $n$, $l$)
Randomly select starting positions $s$=(s$_1$,...,s$_t$) from DNA
bestScore $\leftarrow$ 0
while Score($s$, DNA) > bestScore
  Form PWM $P$ from $s$
  bestScore $\leftarrow$ Score($s$, DNA)
  for $i$ $\leftarrow$ 1 to $t$
    Find a $P$-most probable $l$-mer $a$ from the $i$th sequence $s_i$,
    starting position of $a$
  return bestScore

Analysis of GreedyPWMsearch

Random starting positions: little chance that our guess will be close to an optimal motif
Unlikely that the random starting positions will lead us to the correct solution.
Therefore need to run the algorithm many times with the hope that random starting positions will lead to the optimal solution by chance.

Gibbs Sampling

We can improve the algorithm by introducing Gibbs Sampling, a probabilistic method that discards one $l$-mer at each iteration and replaces it with a new one.
Gibbs Sampling chooses a new $l$-mers at random according to prob(a|P)
How Gibbs Sampling Works

1) Randomly choose starting positions \( s = (s_1, ..., s_t) \) and form the set of \( l \)-mers associated with these starting positions.
2) Randomly choose one of the \( t \) sequences.
3) Create a PWM \( P \) from the other \( t-1 \) sequences.
4) For each position in the removed sequence, calculate the probability that the \( l \)-mer starting at that position was generated by \( P \).
5) Choose a new starting position for the removed sequence at random based on the probabilities calculated in step 4.
6) Repeat steps 2-5 until there is no improvement.

Gibbs Sampling: an Example

Input:
\( t = 5 \) sequences, motif length \( l = 8 \)

1. GTAAACAATATTATAGC
2. AAAATTTACCTGAAGG
3. CCGTACTGCAAGCCTGG
4. TGAATACACGACGTCCCA
5. TACTTAACCCCTGTCAAA

Gibbs Sampling: an Example

1) Randomly choose starting positions, \( s = (s_1, s_2, s_3, s_4, s_5) \) in the 5 sequences:

\( s_1 \): GTAAACAATATTATAGC
\( s_2 \): AAAATTTACCTGAAGG
\( s_3 \): CCGTACTGCAAGCCTGG
\( s_4 \): TGAATACACGACGTCCCA
\( s_5 \): TACTTAACCCCTGTCAAA

Gibbs Sampling: an Example

2) Choose one of the sequences at random:

\( s_1 \): GTAAACAATATTATAGC
\( s_2 \): AAAATTTACCTGAAGG
\( s_3 \): CCGTACTGCAAGCCTGG
\( s_4 \): TGAATACACGACGTCCCA
\( s_5 \): TACTTAACCCCTGTCAAA

Gibbs Sampling: an Example

3) Create PWM \( P \) from \( l \)-mers in remaining 4 sequences:

<table>
<thead>
<tr>
<th>1</th>
<th>A</th>
<th>A</th>
<th>T</th>
<th>A</th>
<th>T</th>
<th>T</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>T</td>
<td>C</td>
<td>A</td>
<td>A</td>
<td>G</td>
<td>C</td>
<td>G</td>
</tr>
<tr>
<td>4</td>
<td>C</td>
<td>T</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>G</td>
<td>C</td>
</tr>
<tr>
<td>5</td>
<td>T</td>
<td>A</td>
<td>C</td>
<td>T</td>
<td>A</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>A</td>
<td>1/4</td>
<td>2/4</td>
<td>1/4</td>
<td>3/4</td>
<td>1/4</td>
<td>1/4</td>
<td>2/4</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>1/4</td>
<td>1/4</td>
<td>0</td>
<td>0</td>
<td>2/4</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>2/4</td>
<td>1/4</td>
<td>1/4</td>
<td>2/4</td>
<td>1/4</td>
<td>1/4</td>
<td>1/4</td>
</tr>
<tr>
<td>G</td>
<td>1/4</td>
<td>0</td>
<td>0</td>
<td>1/4</td>
<td>0</td>
<td>3/4</td>
<td>0</td>
</tr>
</tbody>
</table>

Consensus string: T A A A T C G A
Gibbs Sampling: an Example

4 Calculate prob(a|P) for every 8-mer in the removed sequence:

| 8-mer                         | prob(a|P)  |
|-------------------------------|-----------|
| AAAATTTACCTTAGAAGG            | 0.000732  |
| AAAATTTACCTTAGAAGG            | 0.000122  |
| AAAATTTACCTTAGAAGG            | 0         |
| AAAATTTACCTTAGAAGG            | 0         |
| AAAATTTACCTTAGAAGG            | 0         |
| AAAATTTACCTTAGAAGG            | 0         |
| AAAATTTACCTTAGAAGG            | 0.000183  |
| AAAATTTACCTTAGAAGG            | 0         |
| AAAATTTACCTTAGAAGG            | 0         |

5 Convert prob(a|P) into a probability distribution for the starting positions, and choose a new starting position according to this distribution:

Starting Position 1: prob(AAAATTTA|P) = .000732
Starting Position 2: prob(AAAATTTA|P) = .000122
Starting Position 8: prob(ACCTTAGA|P) = .000183
P(selecting starting position 1) = .706
P(selecting starting position 2) = .118
P(selecting starting position 8) = .176

Gibbs Sampler in Practice

- Gibbs sampling needs to be modified when applied to samples with unequal distributions of nucleotides.
- Gibbs sampling often converges to locally optimal motifs rather than globally optimal motifs.
- Needs to be run with many randomly chosen seeds to achieve good results.

MEME

- (Multiple EM for Motif Elicitation)
  Bailey & Elkan, 1995
- Very general iterative method based on Expectation Maximization
- Guaranteed to find locally optimal solutions
- Effective in practice
- Generalized a 1990 algorithm by Lawrence and Reilly that was limited to motifs that occur once in a sequence

Hidden Information

Hidden variables that describe the position of the motif:

\[ Z = \{Z_j\} \]

where

\[ Z_j = \begin{cases} 
1 & \text{if motif instance starts at position } j \text{ of } X \\
0 & \text{otherwise}
\end{cases} \]

- The idea of MEME: search for a probabilistic model that most likely generated X and Z
Model Parameters

- Motif parameters: $4 \times l$ matrix $\theta = (\theta_{ri})$
  - $r \in \{A,C,G,T\}$
  - $1 \leq i \leq l$
  - $\theta_{ri} = P(\text{residue } r \text{ in position } i \text{ of motif})$
- Background distribution:
  - $\theta_{r0} = P(\text{residue } r \text{ in random non-motif position})$

Expectation Maximization

- Goal: Find $\theta, Z$ that maximize $P(X, Z | \theta)$
- Use Expectation Maximization to find a local maximum
- At iteration $t$:
  - **E-step**: $Z^{(t)} = E(Z | X, \theta^{(t)})$
  - **M-step**: Find $\theta^{(t+1)}$ that maximizes $P(X, Z^{(t)} | \theta^{(t+1)})$

E-step

$$Z_{ij}^{(t)} = \frac{P(X | \text{motif model}) \lambda}{P(X | \text{motif model}) \lambda + P(X | \text{background model})(1 - \lambda)}$$

$\lambda$ is the probability that a random position belongs to the motif $X_i$.

M-step

- If $Z_{ij}^{(t)} \in \{0,1\}$:
  - Calculate PWM $\theta_1, \theta_2, ..., \theta_l$ from motif instances and $\theta_{r0}$ from frequency outside of motif instances.
  - But $Z_{ij}^{(t)} \in [0,1]$, so weight these frequencies by the appropriate values of $Z_{ij}^{(t)}$

Random Projections

- **Objective**: find good initialization for algorithms such as Gibbs sampling and MEME
- **Guiding principle**: Some instances of a motif agree on a subset of positions, i.e. those positions are not affected by mutations in most instances of the motif.
- **Idea**: search for a "good" subset of positions (projection)

Projections

- Choose $k$ positions in string of length $l$.
- Concatenate nucleotides at chosen $k$ positions to form $k$-mer.
- This is a projection of an $l$-dimensional space onto $k$-dimensional subspace.

Example projection:

**ATGGCA**

Projection = (2, 4, 5, 7, 11, 12, 13)
**Random Projections Algorithm**

- Select \( k \) out of \( l \) positions uniformly at random.
- For each \( l \)-tuple in input sequences, hash into bucket based on letters at \( k \) selected positions.
- Recover motif from enriched bucket that contain many \( l \)-tuples.

**Example**

- \( l = 7 \) (motif size), \( k = 4 \) (projection size)
- Choose projection \((1,2,5,7)\)

**Hashing and Buckets**

- Hash function \( h(x) \) obtained from \( k \) positions of projection.
- Buckets are labeled by values of \( h(x) \).
- **Enriched buckets** contain more than \( s \) \( l \)-mers, for some parameter \( s \).

**Motif Refinement**

- How do we recover the motif from the sequences in the enriched buckets?
- Use information in other \( l-k \) positions as starting point for a search scheme such as Gibbs sampling.

**Synergy between Random Projections and Gibbs Sampler**

- Random Projection is a procedure for finding good starting points: every enriched bucket is a potential starting point.
- Feeding these starting points into existing algorithms (like Gibbs sampler) provides good local search in vicinity of every starting point.
- These algorithms work particularly well for "good" starting points.
Building PWMs from Buckets

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>T</th>
<th>G</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>0</td>
<td>25</td>
<td>50</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>25</td>
<td>50</td>
<td>0</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>0</td>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>25</td>
</tr>
</tbody>
</table>

PWM P
Gibbs sampler
Refined motif P*

Motif Refinement

- For each bucket \( h \) containing more than \( s \) sequences, form PWM \( P(h) \)
- Use Gibbs sampler algorithm with starting point \( P(h) \) to obtain refined PWM \( P^* \)

Random Projections Algorithm: A Single Iteration

- Choose a random \( k \)-projection.
- Hash each \( l \)-mer \( x \) in input sequence into bucket labeled by \( h(x) \)
- From each enriched bucket (a bucket with more than \( s \) sequences), form a PWM \( P \) and refine it using Gibbs sampling

Choosing Projection Size

- Projection size \( k \)
  - choose \( k \) small enough so that several motif instances hash to the same bucket.
  - choose \( k \) large enough to avoid contamination by spurious \( l \)-mers

How Many Iterations?

- Motif bucket: bucket with hash value \( h(M) \), where \( M \) is the motif.
- Choose number of iterations, \( m \), such that
  
  \[
  P(\text{Motif bucket contains at least } s \text{ sequences in at least one of } m \text{ iterations}) = 0.95
  \]
- Probability is readily computable since iterations form a sequence of independent Bernoulli trials

Other strategies for motif finding
An alternative search strategy

- Until now: search for the motif by exploring the space of starting positions
- Alternative: search motif space directly - search for the consensus sequence

How to search motif space?

Start from random sample strings

Search small neighborhoods

Exhaustive local search

A lot of work, most of it unnecessary

Best Neighbor

Branch from the seed strings
Find "best" neighbor
Don't consider branches that are not as good as best score so far

Scoring a Motif

- PatternBranching uses the total distance score to define the quality of a motif:
  
  \[ d(M, S_i) = \min \{ d(M, A) : A \text{ is an } l\text{-mer in } S_i \} \]

  where \( d(X, Y) \) is the hamming distance

- The total distance of \( A \) from the sample \( S = \{ S_1, \ldots, S_n \} \):
  
  \[ d(A, S) = \sum d(M, S_i) \]

  Sequence neighborhoods:

  \( \text{Neighbors}(M) \): the set of \( k \)-mers which differ from \( M \) in exactly \( l \) positions.

  \( \text{BestNeighbor}(A) \): the \( k \)-mer \( B \) in \( \text{Neighbors}(A) \) with lowest total distance \( d(B, S) \).
PatternBranching

- Starting from each 1-mer \( A_0 \) in our sequences iteratively apply bestNeighbor to the neighborhood of the current 1-mer:

\[
A_0 \rightarrow A_1 \rightarrow \cdots \rightarrow A_k
\]


Brute force search revisited

- Using sophisticated data structures such as suffix trees it is possible to efficiently explore all possible motifs

Discriminative motif finding

- Want to find motifs that are over-represented in a set of sequences in comparison to a set of background sequences.

Some Motif Finding Programs

- **CONSENSUS**
  Hertz, Stormo (1989)
- **GibbsDNA**
  Lawrence et al (1993)
- **MEME**
  Bailey, Elkan (1995)
- **RandomProjections**
  Buhler, Tompa (2002)
- **MITRA**
  Eskin, Pevzner (2002)
- **Pattern Branching**
- **Weeder**
  Pavesi et al. (2004)
- **DME**
  Smith et al. (2005)