Randomized Algorithms
and
Motif Finding
Outline

- Randomized QuickSort
- Randomized Algorithms
- Greedy Profile Motif Search
- Gibbs Sampler
- Random Projections
Randomized algorithms make random rather than deterministic decisions.

The main advantage is that no input can reliably produce worst-case results because the algorithm runs differently each time.

These algorithms are commonly used in situations where no exact and fast algorithm is known.
Introduction to QuickSort

- QuickSort is a simple and efficient approach to sorting.
- **Sorting Problem:**
  
  **Input:** unsorted array \( c \) of \( n \) elements
  
  **Output:** sorted array \( c \)
Introduction to QuickSort

- QuickSort is a simple and efficient approach to sorting

- **Sorting Problem:**
  - Input: unsorted array $c$ of $n$ elements
  - Output: sorted array $c$

- Select an element $m$ from unsorted array $c$ and divide the array into two subarrays:
  - $c_{small}$ - elements smaller than $m$ and
  - $c_{large}$ - elements larger than $m$

- Recursively sort the subarrays and combine them together in sorted array $c_{sorted}$
Example of QuickSort

Given an array: \( c = \{6, 3, 2, 8, 4, 5, 1, 7, 0, 9\} \)

**Step 1**: Choose the first element as \( m \)

\[ c = \{6, 3, 2, 8, 4, 5, 1, 7, 0, 9\} \]

↑

Our Selection
Example of QuickSort

Step 2: Split the array into $c_{small}$ and $c_{large}$

$C_{small} = \{3, 2, 4, 5, 1, 0\}$

$C = \{6, 3, 2, 8, 4, 5, 1, 7, 0, 9\}$

$c_{large} = \{8, 7, 9\}$
**Example of QuickSort**

**Step 3:** Recursively do the same thing to \( c_{\text{small}} \) and \( c_{\text{large}} \) until each subarray has only one element or is empty.

\[
\begin{align*}
\text{\( c_{\text{small}} = \{3, 2, 4, 5, 1, 0\} \)} & \quad \text{\( m = 3 \)} \\
\text{\{2, 1, 0\}} & \quad \text{\{4, 5\}} \\
\text{\( m = 2 \)} & \\
\text{\{1, 0\}} & \quad \text{\{empty\}} \\
\text{\( m = 1 \)} & \\
\text{\{0\}} & \quad \text{\{empty\}} \\
\text{\( c_{\text{large}} = \{8, 7, 9\} \)} & \quad \text{\( m = 8 \)} \\
\text{\{7\}} & \quad \text{\{9\}}
\end{align*}
\]
Step 4: Combine the two arrays with $m$ working back out of the recursion and as we build together the sorted array.

$C_{small} = \{0, 1, 2, 3, 4, 5\}$

$C_{large} = \{7, 8, 9\}$
Example of QuickSort

Finally we can assemble $c_{small}$ and $c_{large}$ with our original choice of $m$, creating the sorted array.

\[ c_{small} = \{ 0, 1, 2, 3, 4, 5 \} \quad m = 6 \quad c_{large} = \{ 7, 8, 9 \} \]

\[ c_{sorted} = \{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 \} \]
The QuickSort Algorithm

1. QuickSort(c)
2. if c consists of a single element
   return c
3. \( m \leftarrow c_1 \)
4. Determine the set of elements \( c_{small} \) smaller than \( m \)
5. Determine the set of elements \( c_{large} \) larger than \( m \)
6. QuickSort(\( c_{small} \))
7. QuickSort(\( c_{large} \))
8. Combine \( c_{small} \), \( m \), and \( c_{large} \) into a single array, \( c_{sorted} \)
9. return \( c_{sorted} \)
QuickSort Analysis: Optimistic Outlook

Runtime is based on our selection of \( m \):

- A good selection will split \( c \) evenly such that \( |c_{small}| = |c_{large}| \), then the runtime is \( O(n \cdot \log n) \).
- For a good selection, the recurrence relation is:

\[
T(n) = 2T\left(\frac{n}{2}\right) + \text{const} \cdot n \rightarrow \text{MergeSort}
\]

The time it takes to sort two smaller arrays of size \( n/2 \)

Time it takes to split the array into 2 parts where \( \text{const} \) is a positive constant
QuickSort Analysis: Optimistic Outlook

Runtime is based on our selection of \( m \):

- A good selection will split \( c \) evenly such that \( |c_{small}| = |c_{large}| \), then the runtime is \( O(n \cdot \log n) \).
- For a good selection, the recurrence relation is:

\[
T(n) = 2T\left(\frac{n}{2}\right) + \text{const} \cdot n \rightarrow \text{MergeSort}
\]
QuickSort Analysis: Pessimistic Outlook

However, a poor selection will split c unevenly and in the worst case, all elements will be greater or less than m so that one subarray is full and the other is empty. In this case, the runtime is $O(n^2)$.

- For a poor selection, the recurrence relation is:

\[
T(n) = T(n - 1) + \text{const} \cdot n \rightarrow \text{SelectionSort}
\]

The time it takes to sort one array containing $n-1$ elements

Time it takes to split the array into 2 parts where \text{const} is a positive constant
QuickSort Analysis: Pessimistic Outlook

However, a poor selection will split c unevenly and in the worst case, all elements will be greater or less than m so that one subarray is full and the other is empty. In this case, the runtime is $O(n^2)$.

- For a poor selection, the recurrence relation is:

\[
T(n) = T(n - 1) + \text{const} \cdot n \rightarrow \text{SelectionSort}
\]

\[
T(1) = 1
\]

\[
T(n) = n + (n - 1) + T(n - 2)
\]

\[
= n + (n - 1) + (n - 2) + \ldots + 2 + T(1)
\]

\[
\rightarrow O(n^2)
\]
QuickSort Analysis

- QuickSort seems like an inefficient MergeSort.
- To improve QuickSort, we need to choose \( m \) to be a good “splitter”.
- It can be proven that to achieve \( O(n \cdot \log n) \) running time, we don’t need a perfect split, just reasonably good one. In fact, if both subarrays are at least of size \( \frac{n}{4} \), then running time will be \( O(n \cdot \log n) \).
QuickSort Analysis

Diagram: QuickSort partitioning with indices 1, n/4, n/2, 3n/4, and n.
QuickSort Analysis

1 \quad n/4 \quad n/2 \quad 3n/4 \quad n
QuickSort Analysis
QuickSort Analysis

This implies that half of the choices of $m$ make good splitters.
A Randomized Approach

- To improve QuickSort, randomly select m.
- Since half of the elements will be good splitters, if we choose m at random we will get a 50% chance that m will be a good choice.
- This approach will make sure that no matter what input is received, the expected running time is small.
The RandomizedQuickSort Algorithm

1. RandomizedQuickSort(c)
2. if c consists of a single element
   return c
4. Choose element m uniformly at random from c
5. Determine the set of elements \(c_{\text{small}}\) smaller than m
6. Determine the set of elements \(c_{\text{large}}\) larger than m
7. RandomizedQuickSort(\(c_{\text{small}}\))
8. RandomizedQuickSort(\(c_{\text{large}}\))
9. Combine \(c_{\text{small}}\), m, and \(c_{\text{large}}\) into a single array, \(c_{\text{sorted}}\)
10. return \(c_{\text{sorted}}\)

*Lines Highlighted in red indicate the differences between QuickSort and RandomizedQuickSort*
RandomizedQuickSort Analysis

- Worst case runtime: $O(n^2)$
- Expected runtime: $O(n \cdot \log n)$.
- Expected runtime is a good measure of the performance of randomized algorithms, often more informative than worst case runtimes.
- RandomizedQuickSort will always return the correct answer, which offers a way to classify Randomized Algorithms.
Two Types of Randomized Algorithms

- **Las Vegas Algorithms** - always produce the correct solution (ie. RandomizedQuickSort)
- **Monte Carlo Algorithms** - do not always return the correct solution.

Las Vegas Algorithms are always preferred, but they are often hard to come by.
The Motif Finding Problem

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.
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.

- **Previously:** we solved the Motif Finding Problem using a Branch and Bound (actually, Brute Force at first and then bounded with the tree operations) or a Greedy technique.
- **Now:** randomly select possible locations and find a way to greedily change those locations until we have converged to the hidden motif.
Profiles Revisited

- Let $s=(s_1,...,s_t)$ 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 and
  - $4 \times l$ profile matrix* $P$.

*We make a special note that the profile matrix will be defined in terms of the relative frequency of letters, and not as the count of letters.
Scoring Strings with a Profile

- $\text{Prob}(a|P)$ is defined as the probability that an l-mer $a$ was created by the Profile $P$.
- 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.

$$\text{Prob}(a|P) = \prod_{i=1}^{n} p_{a_i}$$
Scoring Strings with a Profile

Given a profile: \( P = \)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>A</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>C</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1/2</td>
<td>7/8</td>
<td>3/8</td>
<td>0</td>
<td>1/8</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>1/8</td>
<td>0</td>
<td>1/2</td>
<td>5/8</td>
<td>3/8</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>1/8</td>
<td>1/8</td>
<td>0</td>
<td>0</td>
<td>1/4</td>
<td>7/8</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>1/4</td>
<td>0</td>
<td>1/8</td>
<td>3/8</td>
<td>1/4</td>
<td>1/8</td>
<td></td>
</tr>
</tbody>
</table>

The probability of the consensus string:
\[ \text{Prob}(aaacct|P) = ? \]
Scoring Strings with a Profile

Given a profile: \( P = \)

\[
\begin{array}{ccccccc}
A & 1/2 & 7/8 & 3/8 & 0 & 1/8 & 0 \\
C & 1/8 & 0 & 1/2 & 5/8 & 3/8 & 0 \\
T & 1/8 & 1/8 & 0 & 0 & 1/4 & 7/8 \\
G & 1/4 & 0 & 1/8 & 3/8 & 1/4 & 1/8 \\
\end{array}
\]

Cons.  A  A  A  A  C  C  T

The probability of the consensus string:

\[
Prob(aaacct|P) = \frac{1}{2} \times \frac{7}{8} \times \frac{3}{8} \times \frac{5}{8} \times \frac{3}{8} \times \frac{7}{8} = 0.033646
\]
The Randomized QuickSort Algorithm

Given a profile: $P =$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>7/8</th>
<th>3/8</th>
<th>0</th>
<th>1/8</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1/2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>1/8</td>
<td>0</td>
<td>1/2</td>
<td>5/8</td>
<td>3/8</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>1/8</td>
<td>1/8</td>
<td>0</td>
<td>0</td>
<td>1/4</td>
<td>7/8</td>
</tr>
<tr>
<td>G</td>
<td>1/4</td>
<td>0</td>
<td>1/8</td>
<td>3/8</td>
<td>1/4</td>
<td>1/8</td>
</tr>
</tbody>
</table>

The probability of the consensus string:

$$\text{Prob}(\text{aaacct}|P) = \frac{1}{2} \times \frac{7}{8} \times \frac{3}{8} \times \frac{5}{8} \times \frac{3}{8} \times \frac{7}{8} = 0.033646$$

Probability of a different string:

$$\text{Prob}(\text{atacag}|P) = \frac{1}{2} \times \frac{1}{8} \times \frac{3}{8} \times \frac{5}{8} \times \frac{1}{8} \times \frac{1}{8} = 0.001602$$
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 profile P.

\[
P = \begin{bmatrix}
A & 1/2 & 7/8 & 3/8 & 0 & 1/8 & 0 \\
C & 1/8 & 0 & 1/2 & 5/8 & 3/8 & 0 \\
T & 1/8 & 1/8 & 0 & 0 & 1/4 & 7/8 \\
G & 1/4 & 0 & 1/8 & 3/8 & 1/4 & 1/8
\end{bmatrix}
\]

Given a sequence = ctataaacctttacatc, find the P-most probable l-mer
P-Most Probable l-mer

Find the $\text{Prob}(a|P)$ of every possible 6-mer:

1. First try: `ctataa a c c t t a c a t c`
2. Second try: `ctataa a a c c t t a c a t c`
3. Third try: `ctataaac c t t a c a t c`
4. Continue this process to evaluate every possible 6-mer
P-Most Probable l-mer

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

| String, Highlighted in Red | Calculations                                      | \( \text{prob}(a|P) \) |
|---------------------------|---------------------------------------------------|------------------------|
| ctataaacctttacat          | \( \frac{1}{8} \times \frac{1}{8} \times \frac{3}{8} \times \frac{0}{8} \times \frac{1}{8} \times \frac{0}{8} \) | 0                      |
| ctataaacctttacat          | \( \frac{1}{2} \times \frac{7}{8} \times \frac{0}{8} \times \frac{0}{8} \times \frac{1}{8} \times \frac{0}{8} \) | 0                      |
| ctataaacctttacat          | \( \frac{1}{2} \times \frac{1}{8} \times \frac{3}{8} \times \frac{0}{8} \times \frac{1}{8} \times \frac{0}{8} \) | 0                      |
| ctataaacctttacat          | \( \frac{1}{8} \times \frac{7}{8} \times \frac{3}{8} \times \frac{0}{8} \times \frac{3}{8} \times \frac{0}{8} \) | 0                      |
| ctataaacctttacat          | \( \frac{1}{2} \times \frac{7}{8} \times \frac{3}{8} \times \frac{5}{8} \times \frac{3}{8} \times \frac{7}{8} \) | 0.0336                 |
| ctataaacctttacat          | \( \frac{1}{2} \times \frac{7}{8} \times \frac{1}{2} \times \frac{5}{8} \times \frac{1}{4} \times \frac{7}{8} \) | 0.0299                 |
| ctataaacctttacat          | \( \frac{1}{2} \times \frac{0}{8} \times \frac{1}{2} \times \frac{0}{8} \times \frac{1}{4} \times \frac{0}{8} \) | 0                      |
| ctataaacctttacat          | \( \frac{1}{8} \times \frac{0}{8} \times \frac{0}{8} \times \frac{0}{8} \times \frac{1}{8} \times \frac{0}{8} \) | 0                      |
| ctataaacctttacat          | \( \frac{1}{8} \times \frac{1}{8} \times \frac{0}{8} \times \frac{3}{8} \times \frac{0}{8} \times \frac{3}{8} \times \frac{7}{8} \) | 0                      |
| ctataaacctttacat          | \( \frac{1}{8} \times \frac{1}{8} \times \frac{3}{8} \times \frac{5}{8} \times \frac{1}{8} \times \frac{7}{8} \) | 0.0004                 |
P-Most Probable l-mer

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

| String, Highlighted in Red | Calculations | \( \text{Prob}(a|P) \) |
|----------------------------|--------------|------------------------|
| ctataaaccttacat            | 1/8 x 1/8 x 3/8 x 0 x 1/8 x 0 | 0                      |
| ctataaaccttacat            | 1/2 x 7/8 x 0 x 0 x 1/8 x 0   | 0                      |
| ctataaaccttacat            | 1/2 x 1/8 x 3/8 x 0 x 1/8 x 0 | 0                      |
| ctataaaccttacat            | 1/8 x 7/8 x 3/8 x 0 x 3/8 x 0 | 0                      |
| **ctataaacctttacat**       | **1/2 x 7/8 x 3/8 x 5/8 x 3/8 x 7/8** | **.0336**             |
| ctataaaccttacat            | 1/2 x 7/8 x 1/2 x 5/8 x 1/4 x 7/8 | .0299                  |
| ctataaaccttacat            | 1/2 x 0 x 1/2 x 0 1/4 x 0    | 0                      |
| ctataaaccttacat            | 1/8 x 0 x 0 x 0 x 0 x 1/8 x 0 | 0                      |
| ctataaaccttacat            | 1/8 x 1/8 x 0 x 0 x 3/8 x 0   | 0                      |
| ctataaaccttacat            | 1/8 x 1/8 x 3/8 x 5/8 x 1/8 x 7/8 | .0004                  |
aaacct is the P-most probable 6-mer in:

catat\text{aaacct}tacatc

because $\text{Prob}(\text{aaacct}|P) = .0336$ 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, there will be enough sequences so that the number of elements in the profile with a frequency of zero is small.

- To avoid many entries with $\text{prob}(a|P) = 0$, there exist techniques to equate zero to a very small number so that one zero does not make the entire probability of a string zero (we will not address these techniques here).
## P-Most Probable l-mers in Many Sequences

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

|     | ctataa | acgtta | acatc | atagcg | gattc | gcgac | tgcatt | cagccc | cagaca | caccct | cggtg | aacctt | acatc | tgtcct | cgttc | aatagc | ttaac | tttac | cttac | ttcctt | tttatcct |
|-----|--------|--------|-------|--------|-------|-------|--------|--------|--------|--------|--------|-------|-------|--------|--------|--------|-------|-------|-------|--------|________|
| A   | 1/2    | 7/8    | 3/8   | 0      | 1/8   | 0     |        |        |        |        |        |       |       |        |        |        |       |       |       |        |________|
| C   | 1/8    | 0      | 1/2   | 5/8    | 3/8   | 0     |        |        |        |        |        |       |       |        |        |        |       |       |       |        |________|
| T   | 1/8    | 1/8    | 0     | 0      | 1/4   | 7/8   |        |        |        |        |        |       |       |        |        |        |       |       |       |        |________|
| G   | 1/4    | 0      | 1/8   | 3/8    | 1/4   | 1/8   |        |        |        |        |        |       |       |        |        |        |       |       |       |        |________|
P-Most Probable l-mers in Many Sequences

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>c</th>
<th>g</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>t</td>
<td>a</td>
<td>g</td>
<td>c</td>
<td>g</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>a</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>t</td>
</tr>
<tr>
<td>3</td>
<td>g</td>
<td>a</td>
<td>a</td>
<td>c</td>
<td>c</td>
<td>t</td>
</tr>
<tr>
<td>4</td>
<td>a</td>
<td>t</td>
<td>a</td>
<td>g</td>
<td>c</td>
<td>t</td>
</tr>
<tr>
<td>5</td>
<td>g</td>
<td>a</td>
<td>c</td>
<td>c</td>
<td>t</td>
<td>g</td>
</tr>
<tr>
<td>6</td>
<td>a</td>
<td>t</td>
<td>c</td>
<td>c</td>
<td>t</td>
<td>t</td>
</tr>
<tr>
<td>7</td>
<td>t</td>
<td>a</td>
<td>c</td>
<td>c</td>
<td>t</td>
<td>t</td>
</tr>
<tr>
<td>8</td>
<td>A</td>
<td>5/8</td>
<td>5/8</td>
<td>4/8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>4/8</td>
<td>6/8</td>
<td>4/8</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>1/8</td>
<td>3/8</td>
<td>0</td>
<td>0</td>
<td>3/8</td>
<td>6/8</td>
</tr>
<tr>
<td>G</td>
<td>2/8</td>
<td>0</td>
<td>0</td>
<td>2/8</td>
<td>1/8</td>
<td>2/8</td>
</tr>
</tbody>
</table>

P-MostProbable l-mers form a new profile:

ctataaacggttacatc
atatagcgaattcgactg
cagcccagaaaccct
cggtgaaccttacatc
tgcattcaatatgctta
tgtcctgtccactcac
tctccaatccttttacaca
ggtctaccttttatcct
### Comparing New and Old Profiles

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>c</th>
<th>g</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>c</td>
<td>g</td>
<td>t</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>t</td>
<td>a</td>
<td>g</td>
<td>c</td>
<td>g</td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>a</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>t</td>
</tr>
<tr>
<td>4</td>
<td>g</td>
<td>a</td>
<td>a</td>
<td>c</td>
<td>c</td>
<td>t</td>
</tr>
<tr>
<td>5</td>
<td>a</td>
<td>t</td>
<td>a</td>
<td>g</td>
<td>c</td>
<td>t</td>
</tr>
<tr>
<td>6</td>
<td>g</td>
<td>a</td>
<td>c</td>
<td>c</td>
<td>t</td>
<td>g</td>
</tr>
<tr>
<td>7</td>
<td>a</td>
<td>t</td>
<td>c</td>
<td>c</td>
<td>t</td>
<td>t</td>
</tr>
<tr>
<td>8</td>
<td>t</td>
<td>a</td>
<td>c</td>
<td>c</td>
<td>t</td>
<td>t</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>1/2</th>
<th>7/8</th>
<th>3/8</th>
<th>0</th>
<th>1/8</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>0</td>
<td>1/8</td>
<td>0</td>
<td>1/2</td>
<td>5/8</td>
<td>3/8</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>1/8</td>
<td>3/8</td>
<td>0</td>
<td>0</td>
<td>3/8</td>
<td>6/8</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>2/8</td>
<td>0</td>
<td>0</td>
<td>2/8</td>
<td>1/8</td>
<td>2/8</td>
<td></td>
</tr>
</tbody>
</table>

**Red** - frequency increased, **Blue** - frequency decreased
Greedy Profile Motif Search

Use P-Most probable l-mers to adjust start positions until we reach a “best" profile; this is the motif.

1. Select random starting positions.
2. Create a profile 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 profile based on the new starting positions after each iteration and proceed until we cannot increase the score anymore.
GreedyProfileMotifSearch Algorithm

1. GreedyProfileMotifSearch(DNA, t, n, l)
2. Randomly select starting positions $s=(s_1,...,s_t)$ from DNA
3. bestScore ← 0
4. while Score($s$, DNA) > bestScore
5. Form profile $P$ from $s$
6. bestScore ← Score($s$, DNA)
7. for $i ← 1$ to $t$
8. Find a $P$-most probable $l$-mer $a$ from the $i^{th}$ sequence
9. $s_i ←$ starting position of $a$
10. return bestScore
GreedyProfileMotifSearch Analysis

- Since we choose starting positions randomly, there is little chance that our guess will be close to an optimal motif, meaning it will take a very long time to find the optimal motif.
- It is unlikely that the random starting positions will lead us to the correct solution at all.
- **In practice:** this algorithm is run many times with the hope that random starting positions will be close to the optimum solution simply by chance.
Gibbs Sampling

- GreedyProfileMotifSearch is probably not the best way to find motifs.
- However, we can improve the algorithm by introducing Gibbs Sampling, an iterative procedure that discards one l-mer after each iteration and replaces it with a new one.
- Gibbs Sampling proceeds more slowly and chooses a new l-mer at random increasing the odds that it will converge to the correct solution.
How Gibbs Sampling Works

1. Randomly choose starting positions:
   \[ s = (s_1, \ldots, s_t) \] and form the set of l-mers associated with these starting positions.

2. Randomly choose one of the t sequences.

3. Create a profile \( P \) 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. GTAAACAATATTTATAGC
2. AAAATTTACCTCGCAAGG
3. CCGTACTGTCAAGCGTGG
4. TGAGTAAACGACGTCCCA
5. TACTTAACACCCTGTCAA
Gibbs Sampling: an Example

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

   $s_1 = 7 \quad \text{GTAAACAAATATTATAAGC}$
   $s_2 = 11 \quad \text{AAAATTACCTTAGAGG}$
   $s_3 = 9 \quad \text{CCGTACTGTCAAGCGTGG}$
   $s_4 = 4 \quad \text{TGAGTAAACGACGTCCCA}$
   $s_5 = 1 \quad \text{TACTTAACACCCTGTCAGA}$
Gibbs Sampling: an Example

2 Choose one of the sequences at random:
   **Sequence 2:** AAAATTTACCTTAGAAGG

   \[ s_1 = 7 \quad \text{GTAAACAATATTTTAGC} \]
   \[ s_2 = 11 \quad \text{AAAATTTACCTTAGAAGG} \]
   \[ s_3 = 9 \quad \text{CCGTACTGTCAAGCGTGG} \]
   \[ s_4 = 4 \quad \text{TGAGTAAACGACGTCCCA} \]
   \[ s_5 = 1 \quad \text{TACTTAACCCTGTCAAA} \]
2 Choose one of the sequences at random:

**Sequence 2:** AAAATTTACCTTAGAAGG

\[ s_1 = 7 \] GTAAAC\color{green}{AATATTTTA\color{black}{TAGC}}

\[ s_3 = 9 \] CCGTACT\color{green}{G}TCAAGCGTGG

\[ s_4 = 4 \] TGAGTAAACGACGT\color{green}{CGACTCCC}A

\[ s_5 = 1 \] TACTTAAC\color{green}{CACCCTGTTCAA}
### Gibbs Sampling: an Example

3. Create profile $P$ from $l$-mers in remaining 4 sequences:

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

<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>1/4</td>
<td>2/4</td>
<td>2/4</td>
<td>3/4</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>1/4</td>
<td>1/4</td>
<td>0</td>
</tr>
<tr>
<td>T</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>0</td>
</tr>
</tbody>
</table>

**Consensus String:**

|   | T | A | A | A | T | C | G | A |
4. Calculate the prob(a|P) for every possible 8-mer in the removed sequence:

| Sequence                  | prob(a|P) |
|---------------------------|----------|
| AAAATTTACCTTAGAAGGG       | 0.000732 |
| AAAATTTACCTTAGAAGGG       | 0.000122 |
| AAAATTTACCTTAGAAGGG       | 0        |
| AAAATTTACCTTAGAAGGG       | 0        |
| AAAATTTACCTTAGAAGGG       | 0        |
| AAAATTTACCTTGAAGGG        | 0        |
| AAAATTTACCTTGAAGGG        | 0.000183 |
| AAAATTTACCTTGAAGGG        | 0        |
| AAAATTTACCTTAGAAGGG       | 0        |
| AAAATTTACCTTGAAGGG        | 0        |
Gibbs Sampling: an Example

5 Create a distribution of probabilities of l-mers \( \text{prob}(a|P) \), and randomly select a new starting position based on this distribution.

(a) To create this distribution, divide each probability \( \text{prob}(a|P) \) by the lowest probability:

Starting Position 1: \( \text{prob}(\text{AAAATTTA}|P) = \frac{0.000732}{0.000122} = 6 \)

Starting Position 2: \( \text{prob}(\text{AAATTTAC}|P) = \frac{0.000122}{0.000122} = 1 \)

Starting Position 8: \( \text{prob}(\text{ACCTTAGA}|P) = \frac{0.000183}{0.000122} = 1.5 \)

Ratio=6:1:1.5
5 Create a distribution of probabilities of l-mers \( \text{prob}(a|P) \), and randomly select a new starting position based on this distribution.

(b) Define probabilities of starting positions according to computed ratios:

\[
\text{Probability (Selecting Starting Position 1)}: \quad \frac{6}{6 + 1 + 1.5} = 0.706 \\
\text{Probability (Selecting Starting Position 2)}: \quad \frac{1}{6 + 1 + 1.5} = 0.118 \\
\text{Probability (Selecting Starting Position 8)}: \quad \frac{1.5}{6 + 1 + 1.5} = 0.176
\]
Gibbs Sampling: an Example

5 Create a distribution of probabilities of l-mers $\text{prob}(a|P)$, and randomly select a new starting position based on this distribution.

(c) Select the start position according to computed ratios:

\[
\begin{align*}
\text{P(Selecting Starting Position 1)} & = 0.706 \\
\text{P(Selecting Starting Position 2)} & = 0.118 \\
\text{P(Selecting Starting Position 8)} & = 0.176
\end{align*}
\]
Gibbs Sampling: an Example

Assume we select the substring with the highest probability → then we are left with the following new substrings and starting positions.

\[
\begin{align*}
  s_1 &= 7 \quad \text{GTAAACAATATTTATAGC} \\
  s_2 &= 1 \quad \text{AAAATTTCCTTAAGG} \\
  s_3 &= 9 \quad \text{CCGTACTGTCAAGCGTGG} \\
  s_4 &= 4 \quad \text{TGAGTAACGAACGTCCCA} \\
  s_5 &= 1 \quad \text{TACTTTAACCCCTGTC} \\
\end{align*}
\]
We iterate the procedure again with the above starting positions until we cannot improve the score any more.
Gibbs Sampler in Practice

- Gibbs sampling needs to be modified when applied to samples with unequal distributions of nucleotides (relative entropy approach).
- 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.
Another Randomized Approach

- **Random Projection Algorithm** is a different way to solve the Motif Finding Problem.
- If an $l$-long pattern is “implanted" in DNA without mutations, then motif finding is easier → count the all $l$-patterns in all sequences; the most common one is the implanted motif
- **But:** mutations!
- **Guiding principle:** Some instances of a motif agree on a subset of positions → use these constant positions to find the motif
- However, it is unclear how to find these “non-mutated" positions.
  - Constant positions do not form a contiguous string
  - Different instances of pattern may differ in different positions (2 and 6, 3 and 8, etc..)
Another Randomized Approach

- To bypass the effect of mutations within a motif, we randomly select a subset of positions in the pattern creating a projection of the pattern.
- Search for that projection in a hope that the selected positions are not affected by mutations in most instances of the motif.
Projections

- Choose k positions in string of length l.
- Concatenate nucleotides at chosen k positions to form k-tuple.
- This can be viewed as a projection of l-dimensional space onto k-dimensional subspace.

\[ l = 15 \]

\[ \text{Projection} \]

\[ k = 7 \]

\[ \text{Projection} = (2, 4, 5, 7, 11, 12, 13) \]
Random Projections Algorithm

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

Input sequence: 

\[ \ldots TCAAT\textcolor{red}{TGCA}CCT\ldots \]

Bucket TGCT
Random Projections Algorithm

- Some projections will fail to detect motifs but if we try many of them, the probability that one of the buckets fills in is increasing.
- In the example below, the bucket **CC*AC is "bad" while the bucket AT**G*C is "good"

```plaintext
...ccATCCGACca...
...ttATGAGGctc...
...ctATAAGTCgc...
...tcATGTGAACac...
```

ATGCAGTC

(7,2) motif
Example

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

![Diagram showing the motif and buckets](image-url)
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-tuples, for some parameter $s$. 

![Diagram](image_url)
Motif Refinement

- How do we recover the motif from the sequences in the enriched buckets?
- \( k \) nucleotides are from hash value of bucket.
- Use information in other \( l-k \) positions as starting point for local refinement scheme, e.g. Gibbs sampler.

![Diagram showing motif refinement process](image)
Building Profiles from Buckets

\[
\begin{array}{ccccccc}
\text{ATCCGAC} & A & 1 & 0 & 0.25 & 0.50 & 0 & 0.50 \\
\text{ATGAGGC} & C & 0 & 0 & 0.25 & 0.25 & 0 & 0 \\
\text{ATAAGTC} & G & 0 & 0 & 0.50 & 0 & 1 & 0.25 \\
\text{ATGTGAC} & T & 0 & 1 & 0 & 0.25 & 0 & 0.25 \\
\end{array}
\]

Profile P

Gibbs sampler

Refined profile \( P^* \)
Synergy between Random Projection 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.