Randomized Algorithms and Motif Finding
Outline

1. Randomized QuickSort
2. Randomized Algorithms
3. Greedy Profile Motif Search
4. Gibbs Sampler
5. Random Projections
Outline - CHANGES

• Greedy Profile Motif Search – Animate “scoring a string with a profile” – Animate “P=most probable l-mer”
• Gibbs Sampler – Describe relative entropies
Section 1: Randomized QuickSort
Randomized Algorithms

- **Randomized Algorithm**: Makes 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.
  1. Select an element $m$ from unsorted array $c$ and divide the array into two subarrays:
     
     $c_{small} = \text{elements smaller than } m$
     
     $c_{large} = \text{elements larger than } m$

  2. Recursively sort the subarrays and combine them together in sorted array $c_{sorted}$
QuickSort: Example

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

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

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

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

• **Step 2:** Split the array into \( c_{\text{small}} \) and \( c_{\text{large}} \) based on \( m \).

\[
c_{\text{small}} = \{ \} \quad c_{\text{large}} = \{ \}
\]

\( c = \{ 6, 3, 2, 8, 4, 5, 1, 7, 0, 9 \} \)
QuickSort: Example

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

• **Step 2:** Split the array into \( c_{\text{small}} \) and \( c_{\text{large}} \) based on \( m \).

\[
\begin{align*}
\text{c}_{\text{small}} &= \{ 3 \} \\
\text{c}_{\text{large}} &= \{ \} \\
\end{align*}
\]

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

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

• **Step 2:** Split the array into \( c_{\text{small}} \) and \( c_{\text{large}} \) based on \( m \).

\( c_{\text{small}} = \{ 3, 2 \} \)  \hspace{1cm}  c_{\text{large}} = \{ \} \)

\( c = \{ 6, 3, 2, 8, 4, 5, 1, 7, 0, 9 \} \)
QuickSort: Example

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

• **Step 2:** Split the array into \( c_{\text{small}} \) and \( c_{\text{large}} \) based on \( m \).

\[
c_{\text{small}} = \{ 3, 2 \} \quad \quad \quad \quad \quad \quad c_{\text{large}} = \{ 8 \}
\]

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

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

• **Step 2:** Split the array into $c_{\text{small}}$ and $c_{\text{large}}$ based on $m$.

\[
c_{\text{small}} = \{ 3, 2, 4 \} \quad c_{\text{large}} = \{ 8 \}
\]

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

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

• **Step 2:** Split the array into $c_{\text{small}}$ and $c_{\text{large}}$ based on $m$.

  $c_{\text{small}} = \{ 3, 2, 4, 5 \}$  

  $c_{\text{large}} = \{ 8 \}$

  $c = \{ 6, 3, 2, 8, 4, 5, 1, 7, 0, 9 \}$
QuickSort: Example

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

• **Step 2:** Split the array into \( c_{\text{small}} \) and \( c_{\text{large}} \) based on \( m \).

\[
\begin{align*}
\quad c_{\text{small}} &= \{ 3, 2, 4, 5, 1 \} \\
\quad c_{\text{large}} &= \{ 8 \}
\end{align*}
\]

\( c = \{ 6, 3, 2, 8, 4, 5, 1, 7, 0, 9 \} \)
QuickSort: Example

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

• **Step 2:** Split the array into \( c_{\text{small}} \) and \( c_{\text{large}} \) based on \( m \).

\[
c_{\text{small}} = \{ 3, 2, 4, 5, 1 \} \quad \text{and} \quad c_{\text{large}} = \{ 8, 7 \}
\]

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

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

• **Step 2:** Split the array into \( c_{\text{small}} \) and \( c_{\text{large}} \) based on \( m \).

\[
\begin{align*}
c_{\text{small}} & = \{ 3, 2, 4, 5, 1, 0 \} \\
c_{\text{large}} & = \{ 8, 7 \}
\end{align*}
\]

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

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

- **Step 2:** Split the array into \( c_{\text{small}} \) and \( c_{\text{large}} \) based on \( m \).

\[
\begin{align*}
    c_{\text{small}} &= \{ 3, 2, 4, 5, 1, 0 \} \\
    c_{\text{large}} &= \{ 8, 7, 9 \}
\end{align*}
\]

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

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

• **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.

\[
\begin{align*}
c_{\text{small}} &= \{ 3, 2, 4, 5, 1, 0 \} \\
m &= 3 \\
&\quad \{ 2, 1, 0 \} \quad \{ 4, 5 \} \\
m &= 2 \\
&\quad \{ 1, 0 \} \quad \{ \text{empty} \} \\
m &= 1 \\
&\quad \{ 0 \} \quad \{ \text{empty} \}
\end{align*}
\]

\[
\begin{align*}
c_{\text{large}} &= \{ 8, 7, 9 \} \\
m &= 8 \\
&\quad \{ 7 \} \quad \{ 9 \} \\
m &= 4 \\
&\quad \{ 5 \} \\
m &= 1 \\
&\quad \{ \text{empty} \}
\end{align*}
\]
QuickSort: Example

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

- **Step 4:** Combine the two arrays with \( m \) working back out of the recursion as we build together the sorted array.

\[
\begin{align*}
&\{ 0 \} \quad 1 \quad \{ \text{empty} \} \\
&\uparrow \quad \downarrow \\
&\{ 0, 1 \} \quad 2 \quad \{ \text{empty} \} \\
&\uparrow \quad \downarrow \\
&\{ 0, 1, 2 \} \quad 3 \quad \{ 4, 5 \} \\
&\uparrow \quad \downarrow \\
&c_{\text{small}} = \{ 0, 1, 2, 3, 4, 5 \} \\
&\uparrow \quad \downarrow \\
&\{ 7 \} \quad 8 \quad \{ 9 \} \\
&\uparrow \quad \downarrow \\
&c_{\text{large}} = \{ 7, 8, 9 \}
\end{align*}
\]
QuickSort: Example

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

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

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

1. **QuickSort**(c)
2. if c consists of a single element
3. return c
4. \( m \leftarrow c_1 \)
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. **QuickSort**(c_{small})
8. **QuickSort**(c_{large})
9. Combine \( c_{\text{small}} \), \( m \), and \( c_{\text{large}} \) into a single array, \( c_{\text{sorted}} \)
10. return \( c_{\text{sorted}} \)
QuickSort Analysis: Optimistic Outlook

• Runtime is based on our selection of $m$:
  • A good selection will split $c$ evenly so that $|c_{\text{small}}| = |c_{\text{large}}|$.
  • For a sequence of good selections, the recurrence relation is:
    \[
    T(n) = 2T\left(\frac{n}{2}\right) + \text{constant} \cdot n
    \]

    The time it takes to sort two smaller arrays of size $n/2$

    Time it takes to split the array into 2 parts

• In this case, the solution of the recurrence gives a runtime of $O(n \log n)$. 
QuickSort Analysis: Pessimistic Outlook

• However, a poor selection of \( m \) 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.

• For a sequence of poor selection, the recurrence relation is:

\[
T(n) = T(n - 1) + \text{constant} \cdot n
\]

The time it takes to sort one array containing \( n-1 \) elements

Time it takes to split the array into 2 parts where const is a positive constant

• In this case, the solution of the recurrence gives runtime \( O(n^2) \).
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 $O(n \log n)$ running time, we don’t need a perfect split, just a reasonably good one. In fact, if both subarrays are at least of size $n/4$, then the running time will be $O(n \log n)$.

• This implies that half of the choices of $m$ make good splitters.
Section 2: Randomized Algorithms
A Randomized Approach to QuickSort

• To improve QuickSort, \textit{randomly} select \( m \).

• Since half of the elements will be good splitters, if we choose \( m \) at random we will have 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
3. return c
4. Choose element m uniformly at random from c
5. Determine the set of elements c_{small} smaller than m
6. Determine the set of elements c_{large} larger than m
7. RandomizedQuickSort(c_{small})
8. RandomizedQuickSort(c_{large})
9. Combine c_{small}, m, and c_{large} into a single array, c_{sorted}
10. return c_{sorted}

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

- **Worst case runtime:** $O(m^2)$

- **Expected Runtime:** $O(m \log m)$.

- Expected runtime is a good measure of the performance of randomized algorithms; it is often more informative than worst case runtimes.

- RandomizedQuickSort will always return the correct answer, which offers us a way to classify Randomized Algorithms.
Two Types of Randomized Algorithms

1. **Las Vegas Algorithm**: Always produces the correct solution (i.e., RandomizedQuickSort)

2. **Monte Carlo Algorithm**: Does not always return the correct solution.

   • Good Las Vegas Algorithms are always preferred, but they are often hard to come by.
Section 3: Greedy Profile Motif Search
A New Motif Finding 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.

• **Previously**: We solved the Motif Finding Problem using an Exhaustive Search 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
  - $4 \times l$ profile matrix $P$.
- We make a special note that the profile matrix will be defined in terms of the frequency of letters, and not as the count of letters.
Pr\((a \mid P)\) 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 \(Pr(a \mid P)\) will be high.

- If \(a\) is very different, then \(Pr(a \mid P)\) will be low.

- Formula for \(Pr(a \mid P)\):
  \[
  Pr(a \mid P) = \prod_{i=1}^{n} P_{a_{i}, i}
  \]
Scoring Strings with a Profile

- Given a profile: $P =$

<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>5/8</td>
</tr>
<tr>
<td>T</td>
<td>1/8</td>
<td>1/8</td>
<td>0</td>
<td>0</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:
  - $Pr(\text{AAACCT} \mid P) = ???$
Scoring Strings with a Profile

• Given a profile: \( \mathbf{P} = \)

<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>5/8</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:
  • \( \Pr(\text{AAACCT} \mid \mathbf{P}) = 1/2 \times 7/8 \times 3/8 \times 5/8 \times 3/8 \times 7/8 = 0.033646 \)
Scoring Strings with a Profile

• Given a profile: \( P = \)

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></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>
</tr>
<tr>
<td>C</td>
<td>1/8</td>
<td>0</td>
<td>1/2</td>
<td>5/8</td>
<td>3/8</td>
</tr>
<tr>
<td>T</td>
<td>1/8</td>
<td>1/8</td>
<td>0</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>
<td>1/4</td>
</tr>
</tbody>
</table>

• The probability of the consensus string:
  • \( \Pr(\text{AAACCT} \mid P) = 1/2 \times 7/8 \times 3/8 \times 5/8 \times 3/8 \times 7/8 = 0.033646 \)

• The probability of a different string:
  • \( \Pr(\text{ATACAG} \mid P) = 1/2 \times 1/8 \times 3/8 \times 5/8 \times 1/8 \times 1/8 = 0.001602 \)
**P-Most Probable l-mer**

- Define the $P$-most probable $l$-mer from a sequence as the $l$-mer contained in that sequence which has the highest probability of being generated by the profile $P$.

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

- **Example**: Given a sequence = CTATAAACCTTACATC, find the P-most probable $l$-mer.
**P-Most Probable l-mer**

- Find $\Pr(a \mid P)$ of every possible 6-mer:

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

- First Try: **C T A T A A A C C T A C A T C**
- Second Try: **C T A T A A A A C C T T A C A T C**
- Third Try: **C T A T A A A A C C T T A C A T C**

- Continue this process to evaluate every 6-mer.
**P-Most Probable l-mer**

- Compute $\Pr(a \mid P)$ for every possible 6-mer:

<table>
<thead>
<tr>
<th>String, Highlighted in Red</th>
<th>Calculations</th>
<th>$\text{prob}(a \mid P)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTATAAAACCTTACAT</td>
<td>$\frac{1}{8} \times \frac{1}{8} \times \frac{3}{8} \times 0 \times \frac{1}{8} \times 0$</td>
<td>0</td>
</tr>
<tr>
<td>CTATAAACCTTACAT</td>
<td>$\frac{1}{2} \times \frac{7}{8} \times 0 \times 0 \times \frac{1}{8} \times 0$</td>
<td>0</td>
</tr>
<tr>
<td>CTATAAACCTTACAT</td>
<td>$\frac{1}{2} \times \frac{1}{8} \times \frac{3}{8} \times 0 \times \frac{1}{8} \times 0$</td>
<td>0</td>
</tr>
<tr>
<td>CTATAAAACCTTACAT</td>
<td>$\frac{1}{8} \times \frac{7}{8} \times \frac{3}{8} \times 0 \times \frac{3}{8} \times 0$</td>
<td>0</td>
</tr>
<tr>
<td>CTATATAACCTTACAT</td>
<td>$\frac{1}{2} \times \frac{7}{8} \times \frac{3}{8} \times 5/8 \times \frac{3}{8} \times 7/8$</td>
<td>0.0336</td>
</tr>
<tr>
<td>CTATAAAACCTTACAT</td>
<td>$\frac{1}{2} \times \frac{7}{8} \times \frac{1}{2} \times 5/8 \times \frac{1}{4} \times 7/8$</td>
<td>0.0299</td>
</tr>
<tr>
<td>CTATAAACCTTTACAT</td>
<td>$\frac{1}{2} \times 0 \times 1/2 \times 0 \times 1/4 \times 0$</td>
<td>0</td>
</tr>
<tr>
<td>CTATAAACCTTACAT</td>
<td>$\frac{1}{8} \times 0 \times 0 \times 0 \times \frac{1}{8} \times 0$</td>
<td>0</td>
</tr>
<tr>
<td>CTATAAACCTTACAT</td>
<td>$\frac{1}{8} \times 1/8 \times 0 \times 0 \times 3/8 \times 0$</td>
<td>0</td>
</tr>
<tr>
<td>CTATAAACCTTTACAT</td>
<td>$\frac{1}{8} \times 1/8 \times 3/8 \times 5/8 \times 1/8 \times 7/8$</td>
<td>0.0004</td>
</tr>
</tbody>
</table>
**P-Most Probable l-mer**

- The *P*-Most Probable 6-mer in the sequence is thus AAACCT:

| String, Highlighted in Red | Calculations                                      | $\text{Prob}(a|P)$ |
|----------------------------|---------------------------------------------------|--------------------|
| CTATAAACCTTTACAT          | $1/8 \times 1/8 \times 3/8 \times 0 \times 1/8 \times 0$ | 0                  |
| CTATAAACCTTTACAT          | $1/2 \times 7/8 \times 0 \times 0 \times 1/8 \times 0$ | 0                  |
| CTATAAACCTTTACAT          | $1/2 \times 1/8 \times 3/8 \times 0 \times 1/8 \times 0$ | 0                  |
| CTATAAACCTTTACAT          | $1/8 \times 7/8 \times 3/8 \times 0 \times 3/8 \times 0$ | 0                  |
| **CTATAAACCTTTACAT**      | $1/2 \times 7/8 \times 3/8 \times 5/8 \times 3/8 \times 7/8$ | **.0336**          |
| CTATAAACCTTTACAT          | $1/2 \times 7/8 \times 1/2 \times 5/8 \times 1/4 \times 7/8$ | **.0299**          |
| CTATAAACCTTTACAT          | $1/2 \times 0 \times 1/2 \times 0 \times 1/4 \times 0$ | 0                  |
| CTATAAACCTTTACAT          | $1/8 \times 0 \times 0 \times 0 \times 1/8 \times 0$ | 0                  |
| CTATAAACCTTTACAT          | $1/8 \times 1/8 \times 0 \times 0 \times 3/8 \times 0$ | 0                  |
| CTATAAACCTTTACAT          | $1/8 \times 1/8 \times 3/8 \times 5/8 \times 1/8 \times 7/8$ | **.0004**          |
Dealing with Zeroes

• In our toy example $\Pr(a | P) = 0$ 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 $\Pr(a | P) = 0$, there exist techniques to equate zero to a very small number so that having one zero in the profile matrix 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.

<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>5/8</td>
</tr>
<tr>
<td>T</td>
<td>1/8</td>
<td>1/8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>G</td>
<td>1/4</td>
<td>0</td>
<td>1/8</td>
<td>3/8</td>
</tr>
</tbody>
</table>

CTATAAACGTTACATC
ATAGCGATTTCGACTG
CAGCCCCAGAACCCT
CGGTATACCTTACATC
TGCATTCAAATAGCTTA
TATCCTTTCACACTCAC
CTCCAAATCCTTTTACA
GGTCATCCTTTTATCCT
## $P$-Most Probable $l$-mers in Many Sequences

- The $P$-Most Probable $l$-mers form a new profile.

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

CTATAAACGT	TACATC
ATAGCGATTACGACTG
CAGCCCAAGAACCCT
CGGTGAAACCT	TACATC
TGCATTCAATAGCTTA
TGTCTTGTCACCTCAGC
CTCCAAATCTTTTACAC
GGTCTACCTTTATCCTC
Comparing New and Old Profiles

- **Red** = frequency increased, **Blue** – frequency decreased

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

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></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>
<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>
<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>
<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>
<td></td>
</tr>
</tbody>
</table>
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 \( s = (s_1, \ldots, s_t) \) from \( DNA \)
3. \( bestScore \leftarrow 0 \)
4. while \( Score(s, DNA) > bestScore \)
5. Form profile \( P \) from \( s \)
6. \( bestScore \leftarrow Score(s, DNA) \)
7. for \( i \leftarrow 1 \) to \( t \)
8. Find a \( P \)-most probable \( l \)-mer \( a \) from the \( i^{th} \) sequence
9. \( s_i \leftarrow \) 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 actually unlikely that the random starting positions will lead us to the correct solution at all.

• Therefore this is a *Monte Carlo* algorithm.

• In practice, this algorithm is run many times with the hope that random starting positions will be close to the optimal solution simply by chance.
Section 4: Gibbs Sampler
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 new $l$-mers at random, increasing the odds that it will converge to the correct solution.
Gibbs Sampling Algorithm

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: Example

- **Input**: $t = 5$ sequences, motif length $l = 8$
  1. GTAAACAATATTTATAGC
  2. AAAATTTACCTCGCAAGG
  3. CCGTACTGTCAAGCGTGG
  4. TGAGTAAACGACGTCCCA
  5. TACTTAACACCCTGTCAA
Gibbs Sampling: Example

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

\[ s_1 = 7 \quad \text{GTAAAC}\text{AATATTATAGC} \]
\[ s_2 = 11 \quad \text{AAAATTTACC}\text{TTAGAAGG} \]
\[ s_3 = 9 \quad \text{CCGTACTGTCAAGCGTGG} \]
\[ s_4 = 4 \quad \text{TGAGTAAACGACGTCCCA} \]
\[ s_5 = 1 \quad \text{TACTTTAAAC}\text{ACCCTGTC} \]
Gibbs Sampling: Example

2. Choose one of the sequences at random

\[ \begin{align*}
  s_1 &= 7 \quad \text{GTAAAC} \text{AATATT} \text{TATAGC} \\
  s_2 &= 11 \quad \text{AAAATTT} \text{ACCCTTA} \text{GAAGG} \\
  s_3 &= 9 \quad \text{CCGTACTG} \text{TCAAGCGT} \text{GG} \\
  s_4 &= 4 \quad \text{TGA} \text{GTAAACG} \text{ACGT} \text{CCCA} \\
  s_5 &= 1 \quad \text{TACTTAAC} \text{ACCCTGT} \text{C} \text{AA} \\
\end{align*} \]
Gibbs Sampling: Example

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

\[
\begin{align*}
    s_1 &= 7 & \text{GTAAACAATATTTATAGC} \\
    s_2 &= 11 & \text{AAAATTTACCTTAGAAGG} \\
    s_3 &= 9 & \text{CCGTACTGTCGAAGCGTG} \\
    s_4 &= 4 & \text{TGAATAACGACGTCCC} \\
    s_5 &= 1 & \text{TACTTTAACCCTGTCAA}
\end{align*}
\]
3. Create profile $P$ from $l$-mers in remaining 4 sequences:

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

|   | A | C | T | |   | |   | |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|
| A | 1/4 | 2/4 | 2/4 | 3/4 | 1/4 | 1/4 | 1/4 | 2/4 |
| C | 0 | 1/4 | 1/4 | 0 | 0 | 2/4 | 0 | 1/4 |
| T | 2/4 | 1/4 | 1/4 | 1/4 | 2/4 | 1/4 | 1/4 | 1/4 |
| G | 1/4 | 0 | 0 | 0 | 1/4 | 0 | 3/4 | 0 |
| Consensus String | T | A | A | A | T | C | G | A |
4. Calculate $\Pr(a \mid P)$ for every possible 8-mer in the removed sequence:

<table>
<thead>
<tr>
<th>Strings Highlighted in Red</th>
<th>$\Pr(a \mid P)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAAATTTACCTTAGAAGG</td>
<td>.000732</td>
</tr>
<tr>
<td>AAAATTTACCTTAGAAGG</td>
<td>.000122</td>
</tr>
<tr>
<td>AAAATTTACCTTAGAAGG</td>
<td>0</td>
</tr>
<tr>
<td>AAAATTTACCTTAGAAGG</td>
<td>0</td>
</tr>
<tr>
<td>AAAATTTACCTTAGAAGG</td>
<td>0</td>
</tr>
<tr>
<td>AAAATTTACCTTAGAAGG</td>
<td>0</td>
</tr>
<tr>
<td>AAAATTTACCTTAGAAGG</td>
<td>0</td>
</tr>
<tr>
<td>AAAATTTACCTTAGAAGG</td>
<td>0</td>
</tr>
<tr>
<td>AAAATTTACCTTAGAAGG</td>
<td>.000183</td>
</tr>
<tr>
<td>AAAATTTACCTTAGAAGG</td>
<td>0</td>
</tr>
<tr>
<td>AAAATTTACCTTAGAAGG</td>
<td>0</td>
</tr>
<tr>
<td>AAAATTTACCTTAGAAGG</td>
<td>0</td>
</tr>
<tr>
<td>AAAATTTACCTTAGAAGG</td>
<td>0</td>
</tr>
</tbody>
</table>
Gibbs Sampling: Example

5. Create a distribution of probabilities of $l$-mers $\Pr(a \mid P)$, and randomly select a new starting position based on this distribution.

- To create this distribution, divide each probability $\Pr(a \mid P)$ by the lowest probability:

Starting Position 1: $\Pr(\text{AAAATTTA} \mid P) / 0.000122 = 0.000732 / 0.000122 = 6$
Starting Position 2: $\Pr(\text{AAATTTAC} \mid P) / 0.000122 = 0.000122 / 0.000122 = 1$
Starting Position 8: $\Pr(\text{ACCTTAGA} \mid P) / 0.000122 = 0.000183 / 0.000122 = 1.5$

- Ratio = 6 : 1 : 1.5
Turning Ratios into Probabilities

• Define probabilities of starting positions according to the computed ratios.

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

• Select the start position probabilistically based on these ratios.
Gibbs Sampling: Example

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

\[
\begin{align*}
  s_1 &= 7 & \text{GTAAACAAATATTTATAGC} \\
  s_2 &= 1 & \text{AAAATTTACCCTCGCAAGG} \\
  s_3 &= 9 & \text{CCGTACTGTCAAGCGTGG} \\
  s_4 &= 5 & \text{TGAGTAATCGACGTCCCA} \\
  s_5 &= 1 & \text{TACTTCACACCTGTCAA}
\end{align*}
\]
Gibbs Sampling: Example

6. We iterate the procedure again with the above starting positions until we cannot improve the score.
Gibbs Sampling 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.
Section 5: Random Projections
Another Randomized Approach

• The **Random Projection Algorithm** is an alternative way to solve the Motif Finding Problem.

• **Guiding Principle:** Some instances of a motif agree on a subset of positions.
  • However, it is unclear how to find these “non-mutated” positions.

• 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.

• We then search for the projection in a hope that the selected positions are not affected by mutations in most instances of the motif.
Projections: Formation

- Choose \( k \) positions in a string of length \( l \).

- Concatenate nucleotides at the chosen \( k \) positions to form a \( k \)-tuple.

- This can be viewed as a projection of \( l \)-dimensional space onto \( k \)-dimensional subspace.

- 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 contains many $l$-tuples.
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 **GC*AC is “bad” while the bucket AT**G*C is “good”

...ccATCCGACca...
...ttATGAGGCntc...
...ctATAAAGTCgc...
...tcATGTGAACac...

ATGCGTCA

(7,2) motif
Random Projections Algorithm: Example

- $l = 7$ (motif size), $k = 4$ (projection size)

- Projection: $(1,2,5,7)$

\[ \ldots \text{TAGAC} \quad \boxed{\text{ATCCGAC}} \quad \boxed{\text{TTGCCCTTAC}} \quad \text{TAC} \ldots \]

Buckets

- ATGC
- GCTC
Hashing and Buckets

- Hash function $h(x)$ is obtained from $k$ positions of projection.

- Buckets are labeled by values of $h(x)$.

- **Enriched Buckets**: Contain more than $s$ $l$-tuples, for some decided upon parameter $s$. 
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.

ATCCGAC

ATGAGGC

ATAAGTC

ATGCGAC

Local refinement algorithm

ATGCGAC

Candidate motif

ATGC
Synergy Between Random Projection and Gibbs

• 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 a good local search in vicinity of every starting point.

• These algorithms work particularly well for “good” starting points.
Building Profiles from Buckets

Profile $P$

- ATCCGAC
- ATGAGGC
- ATAAGTC
- ATGTGAC
- ATGC

Refined profile $P^*$

Gibbs sampler
Motif Refinement

• For each bucket $h$ containing more than $s$ sequences, form profile $P(h)$.

• Use Gibbs sampler algorithm with starting point $P(h)$ to obtain refined profile $P^*$. 
Random Projection 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 (e.g., a bucket with more than $s$ sequences), form profile $\mathbf{P}$ and perform Gibbs sampler motif refinement.

- Candidate motif is best found by selecting the best motif among refinements of all enriched buckets.
Choosing Projection Size

• 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:

$$4^k \gg t(n - l + 1)$$
How Many Iterations?

- **Planted Bucket**: Bucket with hash value $h(M)$, where $M$ is the motif.

- Choose $m = \text{number of iterations}$, such that $\Pr(\text{planted bucket contains at least } s \text{ sequences in at least one of } m \text{ iterations}) = 0.95$

- This probability is readily computable since iterations form a sequence of independent *Bernoulli trials*. 

**Expectation Maximization (EM)**

- $S = x(1), \ldots, x(t)$ : set of input sequences

- **Given**: A probabilistic motif model $W(\Theta)$ depending on unknown parameters $\Theta$, and a background probability distribution $P$.

- Find value $\Theta_{\text{max}}$ that maximizes the likelihood ratio:
  $$
  \frac{Pr(S \mid W(\Theta_{\text{max}}), P)}{Pr(S \mid P)}
  $$

- EM is local optimization scheme. Requires starting value $\Theta_0$. 

EM Motif Refinement

• For each input sequence \( x(i) \), return \( l \)-tuple \( y(i) \) which maximizes likelihood ratio:
  
  • \( T = \{ y(1), y(2), \ldots, y(t) \} \)
  
  • \( C(T) = \) consensus string

\[
\frac{\Pr(y(i) \mid W(\Theta^*_h))}{\Pr(y(i) \mid P)}
\]