Randomized Algorithms
Randomized Algorithms

• Randomized algorithms incorporate random, rather than deterministic, decisions
• Commonly used in situations where no exact and/or fast algorithm is known
• Works for algorithms that behave well on typical data, but poorly in special cases
• Main advantage is that no input can reliably produce worst-case results because the algorithm runs differently each time.
Select

- **Select(L, k)** finds the $k^{th}$ smallest element in L
- **Select(L,1)** find the smallest...
  - Well known $O(n)$ algorithm
    
    ```python
    minv = HUGE
    for v in L:
        if (v < minv):
            minv = v
    ```

- **Select(L, len(L)/2)** find the median...
  - How?
  - median = sorted(L)[len(L)/2] → $O(n \log n)$
- **Can we find medians, or 1st quartiles in O(n)?**
Select Recursion

- **Select**($L$, $k$) finds the $k^{th}$ smallest element in $L$
  - Select an element $m$ from unsorted list $L$ and partition $L$ the array into two smaller lists:
    
    $L_{lo}$ - elements smaller than $m$
    
    and
    
    $L_{hi}$ - elements larger than $m$

- If $\text{len}(L_{lo}) > k$ then
  
  Select($L_{lo}$, $k$)

- else if $k > \text{len}(L_{lo}) + 1$ then
  
  Select($L_{hi}$, $k - (\text{len}(L_{lo}) + 1)$)

- else $m$ is the $k^{th}$ smallest element
Example of Select(L, 5)

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

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

\[
L = \{ 6, 3, 2, 8, 4, 5, 1, 7, 0, 9 \}
\]
Step 2: Split the array into $L_{lo}$ and $L_{hi}$

$L_{lo} = \{3, 2, 4, 5, 1, 0\}$

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

$L_{hi} = \{8, 7, 9\}$
Example of Select($L,5$) (cont’d)

**Step 3:** Recursively call Select on either $L_{lo}$ or $L_{hi}$ until len($L_{lo}$)+1 = k, then return $m$.

len($L_{lo}$) > k = 5 $\rightarrow$ Select({3, 2, 4, 5, 1, 0}, 5)

$m = 3$

$L_{lo} = \{2, 1, 0\}$, $L_{hi} = \{4, 5\}$

k = 5 > len($L_{lo}$) + 1 $\rightarrow$ Select({4, 5}, 5 - 3 - 1)

$m = 4$

$L_{lo} = \{\text{empty}\}$, $L_{hi} = \{5\}$

k = 1 == len($L_{lo}$) + 1 $\rightarrow$ return 4
def select(L, k):
    value = L[0]
    Llo = [t for t in data if t < value]
    Lhi = [t for t in data if t > value]
    below = len(Llo) + 1
    if (k < len(Llo)):
        return select(Llo, k)
    elif (k > below):
        return select(Lhi, k - below)
    else:
        return value
Select with Good Splits

- Runtime depends on our selection of $m$:
  
  - A good selection will split $L$ evenly such that
    
    $$|L_{lo}| = |L_{hi}| = |L|/2$$
    
    - The recurrence relation is:
      $$T(n) = T(n/2)$$
    
    $$n + n/2 + n/4 + n/8 + n/16 + \ldots = 2n \Rightarrow O(n)$$

Same as search for minimum
Select with Bad Splits

However, a poor selection will split L unevenly and in the worst case, all elements will be greater or less than m so that one Sublist is full and the other is empty.

For a poor selection, the recurrence relation is

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

In this case, the runtime is \( O(n^2) \).

Our dilemma:

\( O(n) \) or \( O(n^2) \),

depending on the list… or \( O(n \log n) \) independent of it
Select Analysis (cont’d)

• Select seems risky compared to Sort
• To improve Select, we need to choose \( m \) to give good ‘splits’
• It can be proven that to achieve \( O(n) \) running time, we don’t need a perfect splits, just reasonably good ones.
• In fact, if both subarrays are at least of size \( n/4 \), then running time will be \( O(n) \).
• This implies that half of the choices of \( m \) make good splitters.
A Randomized Approach

• To improve Select, *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.
def randomizedSelect(L, k):
    value = random.choice(L)
    Llo = [t for t in data if t < value]
    Lhi = [t for t in data if t > value]
    below = len(Llo) + 1
    if (k < len(Llo)):
        return randomizedSelect(Llo, k)
    elif (k > below):
        return randomizedSelect(Lhi, k-below)
    else:
        return value
RandomizedSelect Analysis

- Worst case runtime: $O(n^2)$
- Expected runtime: $O(n)$.
- Expected runtime is a good measure of the performance of randomized algorithms, often more informative than worst case runtimes.
- Worst case runtimes are rarely repeated
- RandomizedSelect always returns the correct answer, which offers a way to classify Randomized Algorithms.
Types of Randomized Algorithms

- **Las Vegas Algorithms** – always produce the correct solution (i.e. randomizedSelect)

- **Monte Carlo Algorithms** – do not always return the correct solution.

Of course, Las Vegas Algorithms are always preferred, but they are often hard to come by.
Recall the Motif Finding Problem

Motif Finding Problem: Given a list of $t$ sequences each of length $n$, find the “best” pattern of length $k$ that appears in each of the $t$ sequences.

$k = 8$

$DNA$

$t = 5$

$n = 69$

4/25/16  Comp 555  Spring 2016
A New Motif Finding Approach

• **Motif Finding Problem**: Given a list of $t$ length $n$ sequences, find the best near-matching pattern of length $k$ in each sequence.

• **Previously**: we have solved the Motif Finding Problem using a Branch-and-Bound or a Exhaustive techniques.

• **Now**: **Randomly** select possible locations and find a way to change those locations in an attempt to converge to the hidden motif.
Profiles Revisited

- Let \( s = (s_1, ..., s_t) \) be the starting positions for \( k \)-mers in our \( t \) sequences.
- The substrings corresponding to these starting positions will form:
  - \( t \times k \) alignment matrix
  - \( 4 \times k \) profile matrix*

* Note that we now define the profile matrix in terms of frequency, not counts as before.

\[
P(X|\text{profile}) = 0.6 \times 0.8 \times 0.8 \times 1.0 \times 0.6 \times 0.8 \times 0.6 \times 0.8 = 0.0885
\]
Scoring Strings with a Profile

- Let k-mer \( a = a_1, a_2, a_3, \ldots a_k \)
- \( P(a | P) \) is defined as the probability that an \( k \)-mer \( a \) was created by the Profile distribution \( P \).
- If \( a \) is very similar to the consensus string of \( P \) then \( P(a | P) \) will be high.
- If \( a \) is very different, then \( P(a | P) \) will be low.

\[
Prob(a | P) = \prod_{i=1}^{k} p(a_i, i)
\]
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>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>0</td>
</tr>
</tbody>
</table>

The probability of the consensus string:

\[
\text{Prob}(\text{aaacct}|P) = ???
\]
Scoring Strings with a Profile (cont’d)

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>0</td>
<td></td>
<td>0</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:

\[ \text{Prob(aaacct|P)} = \frac{1}{2} \times \frac{7}{8} \times \frac{3}{8} \times \frac{5}{8} \times \frac{3}{8} \times \frac{7}{8} = .033646 \]
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>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>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 $k$-mer

- Define the $\mathbf{P}$-most probable $k$-mer from a sequence as an $k$-mer in that sequence which has the highest probability of being created from the profile $\mathbf{P}$.

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

Given a sequence = ctataaaccttacatc, find the $k$-mer that best matches the given profile.
P-Most Probable $k$-mer (cont’d)

<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>

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

First try: $\text{ctataaacccttacatc}$

Second try: $\text{ctataaacccttacatc}$

Third try: $\text{ctataaacccttacatc}$

-Continue this process to evaluate every possible 6-mer
# P-Most Probable k-mer (cont’d)

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

| String, Highlighted in Red | Calculations                                      | $\text{prob}(a | P)$ |
|-----------------------------|--------------------------------------------------|----------------------|
| `ctataaaccttacat`          | $\frac{1}{8} \times \frac{1}{8} \times 3 \times \frac{1}{8} \times 0 \times \frac{1}{8} \times 0$ | 0                    |
| `ctataaaaccttacat`         | $\frac{1}{2} \times \frac{7}{8} \times 0 \times 0 \times \frac{1}{8} \times 0$ | 0                    |
| `ctataaaccttacat`          | $\frac{1}{2} \times \frac{1}{8} \times 3 \times \frac{1}{8} \times 0 \times \frac{1}{8} \times 0$ | 0                    |
| `ctataaacccttacat`         | $\frac{1}{8} \times \frac{7}{8} \times 3 \times \frac{1}{8} \times 0 \times 3 \times \frac{1}{8} \times 0$ | 0                    |
| `ctataaacccttacat`         | $\frac{1}{2} \times \frac{7}{8} \times 3 \times \frac{1}{8} \times 5 \times \frac{1}{8} \times 3 \times \frac{1}{8} \times 7 \times \frac{1}{8}$ | 0.0336               |
| `ctataaacccttacat`         | $\frac{1}{2} \times \frac{7}{8} \times 1 \times 2 \times \frac{5}{8} \times 1 \times 4 \times \frac{7}{8}$ | 0.0299               |
| `ctataaacctttacat`         | $\frac{1}{2} \times \frac{0}{8} \times 1 \times 2 \times \frac{0}{8} \times 0 \times 1 \times 4 \times \frac{7}{8}$ | 0                    |
| `ctataaacctttacat`         | $\frac{1}{8} \times 0 \times 0 \times 0 \times 0 \times 1 \times 8 \times 0$ | 0                    |
| `ctataaacctttacat`         | $\frac{1}{8} \times 1 \times 8 \times 0 \times 0 \times 3 \times 8 \times 0$ | 0                    |
| `ctataaacctttacat`         | $\frac{1}{8} \times \frac{1}{8} \times 3 \times \frac{1}{8} \times 5 \times \frac{1}{8} \times 1 \times 8 \times 7 \times \frac{1}{8}$ | 0.0004               |
P-Most Probable 6-mer in the sequence is aaacctt:

<table>
<thead>
<tr>
<th>String, Highlighted in Red</th>
<th>Calculations</th>
<th>( \text{Prob}(a \mid P) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctataaaccttacat</td>
<td>( \frac{1}{8} \times 1/8 \times 3/8 \times 0 \times 1/8 \times 0 )</td>
<td>0</td>
</tr>
<tr>
<td>ctataaacccttacat</td>
<td>( \frac{1}{2} \times 7/8 \times 0 \times 0 \times 1/8 \times 0 )</td>
<td>0</td>
</tr>
<tr>
<td>ctataaacccttacat</td>
<td>( \frac{1}{2} \times 1/8 \times 3/8 \times 0 \times 1/8 \times 0 )</td>
<td>0</td>
</tr>
<tr>
<td>ctataaaccttacat</td>
<td>( 1/8 \times 7/8 \times 3/8 \times 0 \times 3/8 \times 0 )</td>
<td>0</td>
</tr>
<tr>
<td><strong>ctataacctttacat</strong></td>
<td><strong>( 1/2 \times 7/8 \times 3/8 \times 5/8 \times 3/8 \times 7/8 )</strong></td>
<td><strong>0.0336</strong></td>
</tr>
<tr>
<td>ctataaacccttacat</td>
<td>( 1/2 \times 7/8 \times 1/2 \times 5/8 \times 1/4 \times 7/8 )</td>
<td><strong>0.0299</strong></td>
</tr>
<tr>
<td>ctataaacccttacat</td>
<td>( 1/2 \times 0 \times 1/2 \times 0 \times 1/4 \times 0 )</td>
<td>0</td>
</tr>
<tr>
<td>ctataaacccttacat</td>
<td>( 1/8 \times 0 \times 0 \times 0 \times 1/8 \times 0 )</td>
<td>0</td>
</tr>
<tr>
<td>ctataaacccttacat</td>
<td>( 1/8 \times 1/8 \times 0 \times 0 \times 3/8 \times 0 )</td>
<td>0</td>
</tr>
<tr>
<td>ctataaacccttacat</td>
<td>( 1/8 \times 1/8 \times 3/8 \times 5/8 \times 1/8 \times 7/8 )</td>
<td><strong>0.0004</strong></td>
</tr>
</tbody>
</table>
**P-Most Probable k-mer** (cont’d)

\textcolor{red}{aaacct} is the \textcolor{red}{P}-most probable 6-mer in:

\textcolor{red}{ctataaaccttacatc}

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, 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. Pseudo counts (assigning a prior probability based on our best guess).
P-Most Probable $k$-mers in Many Sequences

- Find the $P$-most probable $k$-mer in each of the “t” sequences.

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

ctataaacgttacatc
atacgattcgactg
cagccccagaaccct
cggttataccttacatc
tgcattcaatagctta
tatcctttccactcact
tccttttccactcactc
ctccaaatccttta
ggtcatccttttactcct
**P-Most Probable $k$-mers in Many Sequences**

*(cont’d)*

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

<table>
<thead>
<tr>
<th>A</th>
<th>5/8</th>
<th>5/8</th>
<th>4/8</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<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>

**ctataaacgtttacatc**

**atagcgattcgactg**

**cagcccaagaccct**

**cggtgaacccttacatc**

**tgccattcaatagctta**

**tgcattcttacatc**

**ctccaaatccttttaca**

**tgtcctgtccactcaca**

**ggtctaccttttatcct**

**P-Most Probable $k$-mers give a new profile**
## Comparing New and Old Profiles

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

### Frequency Changes

- **Red**: frequency increased
- **Blue**: frequency decreased

### Frequency Table

<table>
<thead>
<tr>
<th></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>A</td>
<td>5/8</td>
<td>5/8</td>
<td>4/8</td>
<td>0</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>
Random Profile Motif Search

Use P-Most probable \( k \)-mers to adjust start positions until we reach a “best” profile; this is the motif.

1) Select random starting positions.
3) Create a profile \( P \) from the substrings at these starting positions.
4) Find the \( P \)-most probable \( k \)-mer \( a \) in each sequence and change the starting position to the starting position of \( a \).
5) Compute a new profile based on the new starting positions after each iteration and proceed until we cannot increase the score anymore.
6) Repeat the entire process (Steps 1-5) a few times and keep the best answer.
RandomProfileMotifSearch Algorithm

def Profile(seqList, k, start):
    dist = [dict([(base,0.1) for base in "acgt"]) for i in xrange(k)]
    # Count base occurrences in each column
    for t in xrange(len(seqList)):
        for i, base in enumerate(seqList[t][start[t]:start[t]+k]):
            dist[i][base] += 1.0
    # Normalize (divide by total)
    for i in xrange(k):
        total = sum(dist[i].values())
        for base in "acgt":
            dist[i][base] /= total
    # return Distribution
    return dist

def Score(seq, si, k, dist):
    prob = 1.0
    for i, base in enumerate(seq[si:si+k]):
        prob *= dist[i][base]
    return prob
def RandomProfileMotifSearch(seqList, k):
    start = [random.randint(0,len(seqList[t])-k+1) for t in xrange(len(seqList))]
    bestScore = 0.0
    while True:
        distr = Profile(seqList, k, start)
        score = 0.0
        for t in xrange(len(seqList)):
            score += Score(seqList[t], start[t], k, distr)
        if (score <= bestScore):
            break
        bestScore = score
        for t in xrange(len(seqList)):
            newStart, newScore = -1, 0.0
            for i in xrange(len(seqList[t])-k+1):
                score = Score(seqList[t], i, k, distr)
                if (score > newScore):
                    newStart = i
                    newScore = score
            start[t] = newStart
    return score, start
Example

def FindMotif(seqList, k, N):
    highScore = 0.0
    for i in xrange(N):
        score, start = RandomProfileMotifSearch(seqList, k)
        if score > highScore:
            motif = [s for s in start]
            highScore = score
    return highScore, motif

%timeit s, m = FindMotif(seqApprox, 10, 100)
print s
for i, si in enumerate(m):
    print si, seqApprox[i][si:si+10]

1 loops, best of 3: 457 ms per loop
0.297843115489
17 tagatctgaa
47 tggatccgaa
18 tagacccgaa
33 taaatccgaa
21 taggtccaaa
0 tagattcagga
46 cagatccgaa
70 tagatcggta
16 tagatccaaa
65 tcgatccgaa
RandomProfileMotifSearch 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, $O(n)$, with the hope that random starting positions will be close to the optimum solution simply by chance.
- Can we do better than a random guess and then following a greedy path?