11 Motif Finding

This exposition is based on the following sources, which are all recommended reading:


11.1 Motivation

Micro-array experiments indicate that sets of genes are regulated by common “transcription factors (TFs)”. These attach to the DNA upstream of the coding sequence, at certain binding sites. Such a site displays a short motif of DNA that is specific to a given type of TF.

To find such motifs, one considers a collection of genes that are believed to be coregulated:

![Diagram of motif finding](image)

The motif for a given TF is not 100% conserved, but may vary from one gene to another. The computational problem is to determine such a motif by analyzing a set of sequences that contain instances of the motif.

11.2 Planted Motif Problem

We formalize the problem as follows:

**Planted (l, d)-Motif Problem**: Suppose there is a fixed but unknown nucleotide sequence $M$ (the motif) of length $l$. The problem is to determine $M$, given $t$ sequences each of length $n$, and each containing a planted variant of $M$. More precisely, each such planted variant is a substring that is $M$ with exactly $d$ point-substitutions.

To inspire research in this area, Pevzner and Sze formulated the following:
Challenge Problem: Find a \((15, 4)\)-motif in \(t = 20\) sequences of length 600.

These are typical values for finding TF binding sites in coregulated gene promoter regions in yeast.

11.3 Brute-Force Algorithm

One brute-force approach to solving this problem is the following:

For each sequence \(s_i\), consider all \(n - l + 1\) contained \(l\)-mers. For each such choice of \(t\) selected \(l\)-mers, compute the consensus sequence \(C\) and the total distance of all \(t\) selected \(l\)-mers to \(C\). Return the sequence \(C\) with the smallest total distance. The run time of this is \(O(n^t)\).

Another brute-force approach is:

For all \(4^l\) possible \(l\)-mers \(M\): compute the total distance of \(M\) to all \(t\) sequences. Return the \(l\)-mer \(M\) with the smallest total distance. The run time of this is \(O(4^lnt)\).

In both cases, the algorithm is too slow.

11.4 When is the Problem Solvable?

Consider the expected number of \((l, d)\)-motifs in the problem. For simplicity, assume that the background sequences are i.i.d.

Then the probability that a given \(l\)-mer \(C\) occurs with up to \(d\) substitutions at a given position of a random sequence is:

\[
p(l, d) = \sum_{i=0}^{d} \binom{l}{i} \left( \frac{3}{4} \right)^i \left( \frac{1}{4} \right)^{l-i}
\]

Then the expected number of length \(l\) motifs that occur with up to \(d\) substitutions at least once in each of the \(t\) random length \(n\) sequences is:

\[
E(l, d, t, n) \approx 4^l \left(1 - (1 - p(l, d))^{n-t+1}t\right).
\]

The above formulas are only an estimate since they do not model overlapping motifs, and the assumption of i.i.d. background distribution is usually incorrect. Nevertheless, the formula gives a good estimate of the solvability of the respective problem.

For example, by this estimate, 20 random sequences of length 600 are expected to contain more than one \((9, 2)\)-motif by chance, whereas the chances of finding a random \((10, 2)\)-motif are less than \(10^{-7}\).

So, the \((9, 2)\) problem is impossible to solve, because “random motifs” are as likely as the planted motif. However, for the \((10, 2)\) the probability of a random motif occurring is very small.

11.5 Motif Finding Algorithms

We will discuss a number of different algorithms that address motif finding. These are all heuristics, and aren’t guaranteed to solve the problem:

- Gibbs Sampling
- The Projection Algorithm
- Pattern Branching
- Profile Branching
Gibbs sampling and Projection are methods that search in the space of starting positions. Pattern Branching and Profile Branching are examples of methods that search in the space of possible motifs.

11.6 Gibbs Sampling

Gibbs sampling is a well-known method for finding motifs in DNA sequences (Lawrence et al. 1993). Given \( t \) sequences \( s_1, \ldots, s_t \), each of length \( n \), and an integer \( l \), the goal is to find an \( l \)-mer in each of the sequences such that the “similarity” between these \( l \)-mers is maximized.

Let \((a_1, \ldots, a_t)\) be a list of \( l \)-mers contained in \( s_1, \ldots, s_t \). These form a \( t \times l \) alignment matrix.

Let \( X(a) = (x_{ij}) \) denote the corresponding \( 4 \times l \) profile, where \( x_{ij} \) denotes the frequency with which we observe nucleotide \( i \) at position \( j \). Usually, we add pseudo counts to ensure that \( X \) does not contain any zeros (Laplace correction).

11.7 Greedy Profile Search

For a given profile \( X \) and an arbitrary \( l \)-mer \( a \), consider

\[
P(a \mid X) = \prod_{i=1}^{l} x_{a_i},
\]

the probability that \( a \) was generated by \( X \). Any \( l \)-mer that is similar to the consensus string of \( X \) will have a “high” probability, while dissimilar ones will have “low” probabilities.

For example, consider \( X \) given by:

\[
\begin{array}{ccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
A & .33 & .60 & .08 & 0 & 0 & .49 & .71 & .06 & .15 \\
C & .37 & .13 & .04 & 0 & 0 & .03 & .07 & .05 & .19 \\
G & .18 & .14 & .81 & 1 & 0 & .45 & .12 & .84 & .20 \\
T & .12 & .13 & .07 & 0 & 1 & .03 & .09 & .05 & .46 \\
\end{array}
\]

We obtain:

\[
P(\text{CAGGTAAGT} \mid X) = 0.02417294365920 \text{ and } P(\text{TCCGTCCCA} \mid X) = 0.00000000982800.
\]

So, given a profile \( X \), we can evaluate the probability of every \( l \)-mer \( a \) in a sequence \( s \) to find the \( X \)-most probable \( l \)-mer in \( a \), defined as

\[
a = \arg \max P(a \mid X).
\]

This motivates a simple greedy heuristic, greedy profile search:

Given sequences \( s_1, \ldots, s_t \) of length \( n \), randomly select one \( l \)-mer \( a_i \) for each sequence \( s_i \) and construct an initial profile \( X \). For each sequence \( s_i \), determine the \( X \)-most probable \( l \)-mer \( a_i' \). Set \( X \) equal to the profile obtained from \( a_1', \ldots, a_t' \) and repeat.

This naive approach starts with a random seed profile and then attempts to improve on it using a greedy strategy.

Does it work well? No.

The number of possible seeds is huge and thus any randomly chosen seed will rarely be close to the optimum. Even if we run it many times, this approach does not work well.

In each iteration, the greedy profile search method can change any or all \( t \) of the profile \( l \)-mers and thus will jump around in the search space.
Gibbs sampling is similar in that it starts with a random seed profile, and the key idea is that it is then only allowed to change one l-mer per iteration.

11.8 Gibbs Sampling Algorithm

Gibbs sampling operates as follows:

- Randomly select an l-mer \( a_i \) in each input sequence \( s_i \).
- Randomly select one input sequence \( s_h \).
- Build a \( 4 \times l \) profile \( X \) from \( a_1, \ldots, a_{h-1}, a_{h+1}, \ldots, a_t \).
- Compute background frequencies \( Q \) from input sequences \( s_1, \ldots, s_{h-1}, s_{h+1}, \ldots, s_t \).
- For each l-mer \( a \in s_h \), compute \( w(a) = \frac{P(a|X)}{P(a|Q)} \).
- Set \( a_h = a \), for some \( a \in s_h \) chosen randomly with probability \( \frac{w(a)}{\sum_{a' \in s_h} w(a')} \).
- Repeat until “converged”

Gibbs sampling is a well known method that often works well in practice. However, it has difficulties finding subtle motifs. Also, its performance degrades if the input sequences are skewed, that is, if some nucleotides occur much more often than others. The algorithm may be attracted to low complexity regions like AAAAAAAA . . .

To address this problem, the algorithm can be modified to use “relative entropies” rather than frequencies. Another modification is the use of “phase shifts”: The algorithm can get trapped in local minima that are shifted up or down a few positions from the strongest pattern. To address this, every in \( M^{th} \) iteration the algorithm tries shifting some \( a_i \) up or down a few positions.

11.9 The Projection Algorithm

To address the \((l, d)\)-Planted Motif Problem, the key idea of this method is to choose \( k \) of \( l \) positions at random, then to use the \( k \) selected positions of each l-mer \( x \) as a hash function \( h(x) \). When a sufficient number of l-mers hash to the same bucket, it is likely to be enriched for the planted motif \( M \):

\[
\begin{align*}
S_1 & \quad x \times \bullet o \\
& \quad m_1 \\
S_2 & \quad x \times o x o \\
& \quad m_2 \\
S_3 & \quad x \circ o x x \\
& \quad m_3 \\
S_4 & \quad x \times x o x o \\
& \quad m_4
\end{align*}
\]

(Here, for each instance \( m_i \) of the planted motif \( M \), \( x \)’s mark the \( d = 3 \) substitutions and \( o \)’s mark the \( k = 2 \) positions used in hashing.)
Like many probabilistic algorithms, the Projection algorithm performs a number of independent trials of a basic iteration.

In each such trial, it chooses a random projection $h$ and hashes each $l$-mer $x$ in the input sequences to its bucket $h(x)$.

Any hash bucket with sufficiently many entries is explored as a source of the planted motif, using a series of refinement steps, as described below.

### 11.10 Random Projections

Choose $k$ of the $l$ positions at random, without replacement. For an $l$-mer $x$, the hash function $h(x)$ is obtained by concatenating the selected $k$ residues of $x$. Viewing $x$ as a point in $l$-dimensional Hamming space, $h(x)$ is the projection of $x$ onto a $k$-dimensional subspace.

If $M$ is the (unknown) motif, then we call the bucket with hash value $h(M)$ the planted bucket.

The key idea is that, if $k < l - d$, then there is a good chance that some the $t$ planted instances of $M$ will be hashed to the planted bucket, namely all planted instances for which the $k$ hash positions and $d$ substituted positions are disjoint.

So, there is a good chance that the planted bucket will be enriched for the planted motif, and will contain more entries then an average bucket.

### 11.11 Example

Given the sequences

\[
\begin{align*}
\{ s_1: cagtaat \\ s_2: ggaacctt \\ s_3: aagcaca \} 
\end{align*}
\]

and the (unknown) (3, 1)-motif $M = \text{aaa}$, hashing with $k = 2$ using the first 2 of $l = 3$ positions produces the following hash table:

<table>
<thead>
<tr>
<th>$h(x)$ positions</th>
<th>$h(x)$ positions</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1,5), (2,3), (3,1)</td>
<td>(2,2)</td>
</tr>
<tr>
<td>(2,4), (3,5)</td>
<td>(3,3)</td>
</tr>
<tr>
<td>(1,2), (3,2)</td>
<td>(2,1)</td>
</tr>
<tr>
<td>(1,6)</td>
<td>(1,2)</td>
</tr>
<tr>
<td>(1,1), (3,4), (3,6)</td>
<td>(1,4)</td>
</tr>
<tr>
<td>(2,5)</td>
<td>tc</td>
</tr>
<tr>
<td>cg</td>
<td>tg</td>
</tr>
<tr>
<td>ct</td>
<td>tt (2,6)</td>
</tr>
</tbody>
</table>

The motif $M$ is planted at positions (1, 5), (2, 3) and (3, 1) and in this example, all three instances hash to the planted bucket $h(M) = \text{aa}$.

### 11.12 Finding the Planted Bucket

Obviously, the algorithm does not know which bucket is the planted bucket.

So, it attempts to recover the motif from every bucket that contains at least $s$ elements, where $s$ is a threshold that is set so as to identify buckets that look as if they may be the planted bucket.

In other words, the first part of the Projection algorithm is a heuristic for finding promising sets of $l$-mers in the sequence. It must be followed by a refinement step that attempts to generate a motif from each such set.
11.13 Choosing the Parameters

The algorithm has three main parameters:

- the projection size $k$,
- the bucket (inspection) threshold $s$, and
- the number of independent trails $m$.

In the following, we will discuss how to choose each of these parameters.

**Projection size:** Ideally, the algorithm should hash a significant number of instances of the motif into the planted bucket, while avoiding contamination of the planted bucket by random background $l$-mers.

To minimize the contamination of the planted bucket, we must choose $k$ large enough. What size must we choose $k$ so that the average bucket will contain less than 1 random $l$-mer?

Since we are hashing $t(n-l+1)$ $l$-mers into $4^k$ buckets, if we choose $k$ such that

$$4^k > t(n-l+1),$$

then the average bucket will contain less than one random $l$-mer.

For example, in the Challenge (15, 4)-Problem, with $t = 20$ and $n = 600$, we must choose $k$ to satisfy:

$$k < l - d = 15 - 4 = 11 \quad \text{and} \quad k > \frac{\log(t(n-l+1))}{\log(4)} = \frac{\log(20(600 - 15 + 1))}{\log(4)} \approx 5.38.$$

**Bucket threshold:** In the Challenge Problem, a bucket size of $s = 3$ or 4 is practical, as we should not expect too many instances to hash to the same bucket in a reasonable number of trails.

If the total amount of sequence is very large, then it may be that one cannot choose $k$ to satisfy both $k < l - d$ and $4^k > t(n-l+1)$. In this case, set $k = l - d - 1$, as large as possible, and set the bucket threshold $s$ to twice the average bucket size $t(n-l+1)/4^k$.

**Number of independent trails:** We want to choose $m$ so that the probability is at least $q = 0.95$ that the planted bucket contains $s$ or more planted motif instances in at least one of the $m$ trails.

Let $\hat{p}(l,d,k)$ be the probability that a given planted motif instance hashes to the planted bucket, that is:

$$\hat{p}(l,d,k) = \frac{(l-d)}{\binom{l}{k}}.$$

Then the probability that fewer than $s$ planted instances hash to the planted bucket in a given trail is $B_t,\hat{p}(l,d,k)(s)$.

Here, $B_t,p(s)$ is the probability that there are fewer than $s$ successes in $t$ independent Bernoulli trails, each trial having probability $p$ of success.

If the algorithm is run for $m$ trails, the probability that $s$ or more planted instances hash to the planted bucket in at least one trail is:

$$1 - \left(B_t,\hat{p}(l,d,k)(s)\right)^m \geq q.$$

To satisfy this equation, choose:

$$m = \left\lceil \frac{\log(1-q)}{\log(B_t,\hat{p}(l,d,k)(s))} \right\rceil.$$

Using this criterion for $m$, the choices for $k$ and $s$ above require at most thousands of trails, and usually many fewer, to produce a bucket containing sufficiently many instances of the planted motif.
11.14 Motif Refinement

The main loop of the Projection algorithm finds a set of buckets of size $\geq s$. In the refinement step, each such bucket is explored in an attempt to recover the planted motif.

The idea is that, if the current bucket is the planted bucket, then we have already found $k$ of the planted motif residues. These, together with the remaining $l - k$ residues, should provide a strong signal that makes it easy to obtain the motif in only a few iterations of refinement.

We will process each bucket of size $\geq s$ to obtain a candidate motif. Each of these candidates will be “refined” and the best refinement will be returned as the final solution.

Candidate motifs are refined using the expectation maximization (EM) algorithm (which we will discuss in detail later). This is based on the following probabilistic model:

- An instance of some length-$l$ motif occurs exactly once per input sequence.
- Instances are generated from a $4 \times l$ weight matrix model $W$, whose $(i, j)$th entry gives the probability that base $i$ occurs in position $j$ of an instance, independent of its other positions.
- The remaining $n - l$ residues in each sequence are chosen randomly and independently according to some background distribution.

Let $S$ be a set of $t$ input sequences, and let $P$ be the background distribution. EM-based refinement seeks a weight matrix model $W^*$ that maximizes the likelihood ratio

$$\frac{\Pr(S \mid W^*, P)}{\Pr(S \mid P)},$$

that is, a motif model that explains the input sequences much better than $P$ alone.

The position at which the motif occurs in each sequence is not fixed a priori, making the computation of $W^*$ difficult, because $\Pr(S \mid W^*, P)$ must be summed over all possible locations of the instances.

To address this, the EM algorithm uses an iterative calculation that, given an initial guess $W_0$ at the motif model, converges linearly to a locally maximum-likelihood model in the neighborhood of $W_0$.

An initial guess $W_h$ for a bucket $h$ is formed as follows: set $W_h(i, j)$ to the frequency of base $i$ among the $j$th positions of all $l$-mers in $h$.

This guess forms a centroid for $h$, in the sense that positions that are well conserved in $h$ are strongly biased in $W_h$, while poorly conserved positions are less biased. To avoid zero entries in $W_h$, add a Laplace correction of $b_i$, to $W_h(i, j)$, where $b_i$ is the background probability of residue $i$ in the input.

Once we have used the EM algorithm to obtain a refinement $W^*_h$ of $W_h$, the final step is to identify the planted motif from $W^*_h$.

To do so, we select from each input sequence the $l$-mer $x$ with the largest likelihood ratio:

$$\frac{\Pr(x \mid W^*_h)}{\Pr(x \mid P)}.$$

The resulting multiset $T$ of $l$-mer represents the motif in the input that is most consistent with $W^*_h$.

Let $C_T$ be the consensus of $T$, and let $s(T)$ be the number of elements of $T$ whose Hamming distance to $C_T$ is $\leq d$. The algorithm returns the sequence $C_T$ that minimizes $s(T)$, over all considered buckets $h$ and over all trials.

11.15 Summary of Projection Algorithm

Algorithm Projection
Input: sequences \( s_1, \ldots, s_t \), parameters \( k, s \) and \( m \)
Output: best guess motif

for \( i = 1 \) to \( m \) do
    choose \( k \) different positions \( I_k \subset \{1, 2, \ldots, l\} \)
    for each \( l \)-mer \( x \in s_1, \ldots, s_t \) do
        compute hash value \( h_{I_k}(x) \)
        Store \( x \) in hash bucket
    for each bucket with \( \geq s \) elements do
        refine bucket using EM algorithm
return consensus pattern of best refined bucket

11.16 Performance

The following table gives an overview of the performance of PROJECTION compared to other motif finders on the \((l, d)\)-motif problem. The measure is the average performance defined as \( \frac{|K \cap P|}{|K \cup P|} \) where \( K \) is the set of the \( lt \) residue positions of the planted motif instances, and \( P \) is the corresponding set of positions predicted by the algorithm.

<table>
<thead>
<tr>
<th>( l )</th>
<th>( d )</th>
<th>Gibbs</th>
<th>WINNOWER</th>
<th>SP-STAR</th>
<th>PROJECTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>2</td>
<td>0.20</td>
<td>0.78</td>
<td>0.56</td>
<td>0.82</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>0.68</td>
<td>0.90</td>
<td>0.84</td>
<td>0.91</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>0.03</td>
<td>0.75</td>
<td>0.33</td>
<td>0.81</td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td>0.60</td>
<td>0.92</td>
<td>0.92</td>
<td>0.92</td>
</tr>
<tr>
<td>14</td>
<td>4</td>
<td>0.02</td>
<td>0.02</td>
<td>0.20</td>
<td>0.77</td>
</tr>
<tr>
<td>15</td>
<td>4</td>
<td>0.19</td>
<td>0.92</td>
<td>0.73</td>
<td>0.93</td>
</tr>
<tr>
<td>16</td>
<td>5</td>
<td>0.02</td>
<td>0.03</td>
<td>0.04</td>
<td>0.70</td>
</tr>
<tr>
<td>17</td>
<td>5</td>
<td>0.28</td>
<td>0.03</td>
<td>0.69</td>
<td>0.93</td>
</tr>
<tr>
<td>18</td>
<td>6</td>
<td>0.03</td>
<td>0.03</td>
<td>0.03</td>
<td>0.74</td>
</tr>
<tr>
<td>19</td>
<td>6</td>
<td>0.05</td>
<td>0.03</td>
<td>0.40</td>
<td>0.96</td>
</tr>
</tbody>
</table>

11.17 Pattern Branching

let \( M \) be an unknown motif of length \( l \), and let \( A_0 \) be an occurrence of \( M \) in the sample with exactly \( k \) substitutions.

Given \( A_0 \), how do we determine \( M \)? Since the Hamming distance \( d(M, A_0) = k \), we have \( M \in D_{=k}(A_0) \), defined as the set of patterns of distance exactly \( k \) from \( A_0 \).

We could look at all \( \binom{l}{k} \) \( 3^k \) elements of \( D_{=k}(A_0) \) and score each pattern as a guess of \( M \). However, as this must be applied to all sample strings \( A_0 \) of length \( l \), it would be too slow.

The idea of the Pattern Branching algorithm is to construct a path of patterns

\[ A_0 \rightarrow A_1 \rightarrow \ldots \rightarrow A_k, \]

in each step, moving to the “best neighbor” in \( D_{=1}(A_i) \). The pattern \( A_k \) is scored as a guess for \( M \).

Given a pattern \( A \) of length \( l \), two questions must be addressed:

- How do we score \( A \)?
- How do we determine the “best neighbor” of \( A \)?

First, we score \( A \) using its total distance from the sample. For each sequence \( s_i \) in the sample \( S = \{s_1, \ldots, s_t\} \), let

\[ d(A, s_i) = \min\{d(A, P) \mid P \in s_i\}, \]
where $P$ denotes an $l$-mer contained in $s_i$.

Then the total distance of $A$ from the sample is

$$d(a, S) = \sum_{s_i \in S} d(A, s_i).$$

Second, we define a best neighbor of $A$ to be any pattern $B \in D_{=1}(A)$ with lowest total distance $d(B, S)$.

The resulting algorithm is very straight-forward:

**Algorithm** Pattern Branching

Input: Sequences $S$, motif length $l$, number of substitutions $k$

Output: best guess motif $M$

Init: $M \leftarrow$ arbitrary motif pattern

for each $l$-mer $A_0 \in S$ do

for $j \leftarrow 0$ to $k$ do

if $d(A_j, S) < d(M, S)$ then $M \leftarrow A_j$

$A_{j+1} \leftarrow \text{BestNeighbor}(A_j)$

Output $M$

To conduct a more thorough search of $D_{=k}(A_0)$, one can keep a set $A$ of $r$ patterns at each iteration instead of a single pattern, defining BestNeighbors($A$) to be the set of $r$ patterns $B \in D_{=1}(A)$ with lowest total distance $d(B, S)$.

Letting $A_0 = \{A_0\}$, we thus have $|A_0| = 1$ and $|A_j| = r$ for $j > 0$.

The algorithm returns the motif that has the smallest total distance to all input strings.

### 11.18 Profile Branching

The Profile Branching algorithm is similar to the Pattern Branching algorithm. However, the search is in the space of motif profiles, instead of motif patterns. The algorithm is obtained from the Pattern Branching algorithm by making the following changes:

1. convert each sample string $A_0$ to a profile $X(A_0)$,
2. generalize the scoring method to score profiles,
3. modify the branching method to apply to profiles, and
4. use the top-scoring profile found as a seed for the EM algorithm.

How to convert an initial sample string $A_0$ into a profile $X(A_0)$?

Let $A_0 = a_1 \ldots a_l$ be an $l$-mer of nucleotides. Then $X(A_0)$ is defined as the $4 \times l$ profile ($x_{vw}$) which in column $w$ has probability

$$x_{vw} = \begin{cases} \frac{1}{2} & \text{if } v = a_w, \\ \frac{1}{6} & \text{else.} \end{cases}$$

For example, for $A_0 = \text{ACGA}$ we obtain:

\[
\begin{array}{cccc}
A & 1 & 2 & 3 & 4 \\
\hline
A & \frac{1}{2} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \\
C & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \\
G & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \\
T & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \\
\end{array}
\]
The total distance score for patterns is replaced by an entropy score for profiles:

Let $X = (x_{vw})$ be a profile and $P = p_1 \ldots p_l$ a pattern. The log probability of sampling $P$ from $X$ is given by:

$$e(X, P) = \sum_{w=1}^{l} \log(x_{p_w w}).$$

For each sequence $s_i \in S = \{s_1, \ldots, s_t\}$, let

$$e(X, s_i) = \max\{e(x, P) | P \in s_i\}.$$  

The *entropy score* of $X$ is

$$e(X, S) = \sum_{s_i \in S} e(X, s).$$

This value describes how well $X$ matches its best occurrence in each sequence of the input.

For patterns, $D_{-1}(A)$ is a natural choice for candidates for branching.

For profiles, we define $D_{-1}(X)$ to be the set of profiles obtained from $X$ by *amplifying* a single nucleotide in a single position $w$ of $X$ to create a new profile $\bar{X} = (\bar{x}_{vw})$.

In more detail, this is done in such a way that the relative entropy of $X$ and $\bar{X}$ (a kind of distance between the two distributions) equals an implicit parameter $\rho$, which controls the similarity of the old and new profile.

The relative entropy is defined as:

$$RE(\bar{X}, X) = \sum_{v} \bar{x}_{vw} \log(\bar{x}_{vw}/x_{vw}).$$

For example, given nucleotide probabilities

$$\left(\frac{1}{2}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}\right),$$

by amplifying the second position we get:

$$(0.27, 0.55, 0.09, 0.09).$$

At a given position $w$, a nucleotide $v$ is only eligible for amplification, if $x_{vw} < 0.5$.

We define the *best neighbor* of a profile $X$ to be the profile $Y \in D_{-1}(X)$ with highest entropy $e(Y, S)$.

The Profile Branching algorithm proceeds as follows. For each $l$-mer $A_0$ in the sample $S$, let $X_0 = X(A_0)$ and construct a path of profiles:

$$X_0 \rightarrow X_1 \rightarrow \ldots \rightarrow X_k,$$

by iteratively applying the best neighbor calculation for profiles.

After branching for $k$ iterations for each $l$-mer $A_0$ in the input sample, the EM algorithm is run to convergence on the top-scoring profile found.

The algorithm is as follows:

**Algorithm** Profile Branching

Input: Sequences $S$, motif length $l$, number of substitutions $k$

Output: best guess motif profile $X$

Init: $X^* \leftarrow$ arbitrary motif profile

for each $l$-mer $A_0 \in S$ do

$X_0 \leftarrow X(A_0)$
for \( j \leftarrow 0 \) to \( k \) do
if \( e(X_j, S) < e(X^*, S) \) then \( X^* \leftarrow X_j \)
\( X_{j+1} \leftarrow \text{BestNeighbor}(X_j) \)

Run EM algorithm with \( X^* \) as seed and return result

This algorithm runs about 5 times slower than the Pattern Branching algorithm.

The Pattern Branching algorithm clearly outperforms the Profile Branching algorithm on Challenge-like problems. However, pattern-based algorithms have difficulty finding motifs with many degenerate positions.

### 11.19 The EM Algorithm

Both the Projection algorithm and the Profile Branching algorithm use the EM algorithm to refine a given motif. We now discuss the EM algorithm in general terms.

Suppose we are given a probability density function \( p(x \mid \Theta) \) that depends on some parameters \( \Theta \).

Suppose we are given measurements \( X = \{x_1, \ldots, x_N\} \). The goal of maximum likelihood estimation is to find parameters \( \Theta \) that maximize:

\[
p(X \mid \Theta) = \prod_{x_i} p(x_i \mid \Theta) =: \mathcal{L}(\Theta \mid X),
\]

that is, to find

\[
\Theta^* = \arg \max_{\Theta} \mathcal{L}(\Theta \mid X).
\]

**Expectation maximization (EM)** is a general technique for finding the maximum likelihood estimate of the parameters of an underlying distribution from a given dataset, when the data is incomplete or has missing values.

Assume that \( X \) is observed data that is generated by some distribution. Let us call \( X \) the *incomplete-data*. Assume that a *complete* data set \( Z = (X, Y) \) exists and has the joint density function

\[
p(z \mid \Theta) = p(x, y \mid \Theta)p(x \mid \Theta).
\]

We define the *complete-data likelihood* function as:

\[
\mathcal{L}(\Theta \mid Z) = \mathcal{L}(\Theta \mid X, Y) = p(X, Y \mid \Theta).
\]

This is a random variable, as \( Y \) is unknown, random and assumed to be governed by some underlying distribution.

Thus, we can think of this likelihood as a function of \( Y \):

\[
\mathcal{L}(\Theta \mid X, Y) = h_{X, \Theta}(Y),
\]

where \( X \) and \( \Theta \) are constant and \( Y \) is a random variable.

The **EM algorithm** proceeds in two steps.

**E-step**: Find the expected value of the complete-data log-likelihood \( p(X, Y \mid \Theta) \) with respect to the unknown data \( Y \) and the current parameter estimates. That is, define:

\[
Q(\Theta, \Theta^{(i-1)}) = \mathbb{E}[\log p(X, Y \mid \Theta) \mid X, \Theta^{(i-1)}],
\]

where \( \Theta^{(i-1)} \) are the current parameter estimates and \( \Theta \) are the new parameters that we will optimize to increase \( Q \).
Note that $\mathcal{X}$ and $\Theta^{(i-1)}$ are constants, $\mathcal{Y}$ is a random variable governed by $f(y \mid \mathcal{X}, \Theta^{(i-1)})$ and $\Theta$ is a normal variable that we seek to adjust.

The equation above can be rewritten as:

$$E[\log p(\mathcal{X}, \mathcal{Y} \mid \Theta) \mid \mathcal{X}, \Theta^{(i-1)}] = \int_{y \in \mathcal{Y}} \log p(\mathcal{X}, y \mid \Theta) f(y \mid \mathcal{X}, \Theta^{(i-1)}) dy.$$ 

Here we integrate over all possible values of $y$. This is a deterministic function that could be maximized if desired.

**M-Step:** Maximize the expectation that we computed in the first step. That is, find:

$$\Theta^{(i)} = \arg \max_{\Theta} Q(\Theta, \Theta^{(i-1)}).$$

The two steps are repeated as necessary. The algorithm is guaranteed to converge to a local maximum.