Pattern Discovery in Biosequences
ISMB 2002 tutorial

Stefano Lonardi
University of California, Riverside

Latest version of the slides at http://www.cs.ucr.edu/~stelo/ismb02/

Roadmap

• Why “pattern discovery”
• Basic concepts
• Problem definition
• Classification of patterns
• Complexity results
• Efficient algorithms for pattern discovery
  – Deterministic patterns: Verbumculus
  – Rigid patterns: Teiresias, Winnower, Projection, Weeder
  – Profiles: Gibbs sampling, Meme
• Appendix
Why “pattern discovery”?  

Discovery of regulatory elements  

- **Promoter**: a region of DNA involved in binding of RNA polymerase to initiate transcription  
- **Enhancer**: a region of DNA that increases the utilization of (some) promoters (it can function in either orientation and any location relative to the promoter)  
- **Repressor**: a region of DNA that decreases the utilization of (some) promoters
Transcription

- Different factors are involved in the transcription machinery
  - presence of transcription factors and their binding sites
  - ability of DNA to bend
  - relative location of the binding sites
  - presence of CpG islands ("p" is for phosphate)
  - ...
Transcription factors binding sites

Co-regulated genes

Pattern discovery

Putative binding sites

Basic concepts
Some notations

\( \Sigma : \) alphabet
\( a, b, c, \ldots : \) symbols from \( \Sigma \)
\( x : \) sequence/substring over \( \Sigma \), \( |x| = n \)
\( \{x_1, x_2, \ldots, x_k\} : \) multi-sequence, \( \sum_{i=1}^{k} |x_i| = n \)
\( y \) (or \( w \)) : substring of \( x \), \( |y| = m \)
\( y^i \) : the substring \( \underbrace{yy \cdots y}_{i \text{ times}} \) \( (i \geq 0) \)

Some notations

\( y_{[i]} : \) the \( i \)-th symbol of \( y \) \((1 \leq i \leq m)\)
\( y_{[i,j]} : \) the string \( y_{[i]}y_{[i+1]} \cdots y_{[j]} \) \((1 \leq i \leq j \leq m)\)
\( y_{[1,j]} : \) are the prefixes of \( y \) \((1 \leq j \leq m)\)
\( y_{[j,m]} : \) are the suffixes of \( y \) \((1 \leq j \leq m)\)
Occurrences vs. Colors

Some notations

\( f(y) \) : number of occurrences of \( y \)

\( c(y) \) : number of colors of \( y \)

Depending on the application, \( f(y) \) and/or \( c(y) \) is used. In general, these quantities are called the support of \( y \).
Occurrences: types

For our purposes, any of the above is simply an occurrence.
Keep in mind that in some cases you may have to distinguish them.

Example (DNA)

\[ \begin{align*}
x_1 &= \text{CCACCCCTTTTGTGGGGCTTCTATTTTCAAGG} \\
x_2 &= \text{TTGTTCTTCTGCATGTGCGGCAGG} \\
x_3 &= \text{TTCTAAAAAGGGCGATTATCAGGAAAAAGAAG} \\
x_4 &= \text{GTGTAAATTTGTGTGCTACCTACGTATTTA} \\
\end{align*} \]

- \[ \Sigma = \{A, C, G, T\} \quad |\Sigma| = 4 \quad n = 120 \]
- e.g., \( y = \text{AAAA} \) is a substring of \( x_3 \) and \( x_4 \)
  - \( f(y) = 4 \) (occurrences can overlap)
  - \( c(y) = 2 \)
Bernoulli and Markov models

- Two typical hypothesis about the source (probabilistic models)
- Bernoulli: symbols are generated independently and they are identically distributed (i.i.d. or memoryless)
- Markov: the probability distribution for the “next” symbol depends on the previous $h$ symbols ($h > 0$ is the order of Markov chain)

Example (no. of occurrences)

- We want to describe the number of occurrences $f(y)$ of a pattern $y$ in the text $x$
- Recall $|x| = n$, $|y| = m$
- Let us choose a particular location $i$ in the text ($i < n-m+1$)
Example on no. of occurrences

• Assuming a Bernoulli model for the source that generated \( x \), i.e., symbols are generated i.i.d., the probability that \( y \) occurs at position \( i \) in the text is

\[
P(x[i]=y_1, x[i+1]=y_2, \ldots, x[i+m-1]=y_m) = P(x[i]=y_1) P(x[i+1]=y_2) \ldots P(x[i+m-1]=y_m) = p_{y[1]} p_{y[2]} \ldots p_{y[m]}
\]

• Note that in general we do not know the “true” \( p_a \) and they have to be estimated from the observation \( x \)

Example on no. of occurrences

• Now we have to consider that \( y \) can occur in \( n-m+1 \) positions (from 1 to \( n-m+1 \))

• At each position the probability is \( p_{y[1]} p_{y[2]} \ldots p_{y[m]} \)

• Since we assumed a Bernoulli model, the random variables for each position are independent, then

\[
E[X] = (n-m+1) \prod_{j=1..m} p_{y[j]}
\]
A r.v. for the no. of occurrences

Let $Z_y$ be a r.v. for the number of occurrences of $y$, $p_a$ be the probability of $a \in \Sigma$, and $|y| = m \leq (n+1)/2$, then

- $E(Z_y) = (n-m+1) \prod_{i=1}^{m} p_{y_{(i)}} = (n-m+1) \hat{p}$
- $Var(Z_y) = E(Z_y)(1-\hat{p}) - \hat{p}^2(n-m+1)(n-m) + 2\hat{p}B(y)$

where $B(y) = \sum_{d \in P(y)} (n-m+1-d) \prod_{i=m-d+1}^{m} p_{y_{(i)}}$

and $P(y)$ is the set of period lengths of $y$

A r.v. for the no. of colors

Let $W_y$ be a r.v. for the number of colors of $y$ in $\{x_1, x_2, \ldots, x_k\}$, and $Z_y^i$ be the random variable of occurrences of $y$ in the $i$-th sequence $(1 \leq i \leq k)$, then

- $E(W_y) = k - \sum_{i=1}^{k} P[Z_y^i = 0]$

(because $E(W_y) = P[Z_y^i > 0]$)
“Pattern Discovery”: the problem

Pattern discovery: the problem

- Given a set of sequences $S^+$ and a model of the source for $S^+$
- Find a set of patterns in $S^+$ which have a support that is statistically significant with respect to the probabilistic model
- If we are also given negative examples $S^-$, we must ensure that the patterns do not appear in $S^-$
Pattern discovery problem

Pos & Neg examples

F-
S-
S+
F+

Only Pos examples

F-
S+
S+
F+

Noisy data

Pos & Neg examples

F-
S-
S+
F+

Only Pos examples

F-
S+
S+
F+
Pattern discovery “dimensions”

- Type of learning
  - from positive examples only (unsupervised)
  - from both positive and negative examples (supervised)
  - noisy data
- Type of patterns
  - deterministic, rigid, profiles, …
- Measure of statistical significance
- *A priori* knowledge

Output

- *True positive*: a pattern belonging to the positive training set which has been correctly classified
- *True negative*: a pattern belonging to the negative training set which has been correctly classified
- *False positive*: misclassified as positive
- *False negative*: misclassified as negative
Measuring pattern discovery

• *Complete:* if no true pattern is missed
  – but we may report too many patterns
• *Sound:* if no false pattern is reported
  – but we may miss true positives
• Usually there is a tradeoff between soundness and completeness: if you increase one, you will decrease the other

Measuring the performance

• Information Retrieval measures
  – *Precision* = \( \frac{true\ pos}{true\ pos + false\ pos} \)
    • expresses the proportion of discovered patterns out of the total reported positive
    • also called *sensitivity*

  – *Recall* = \( \frac{true\ pos}{true\ pos + true\ neg} \)
    • expresses the proportion of discovered patterns out of the total of true patterns
A classification of patterns

Types of patterns

• Deterministic patterns

• Rigid patterns
  – Hamming distance

• Flexible patterns
  – Edit distance

• Matrix profiles

✓ A motif is any of these patterns, as long as it is associated with biological relevance
Deterministic Patterns

- **Definition**: Deterministic patterns are strings over the alphabet $\Sigma$
  - e.g., “TATAAA” (TATA-box consensus)
- Discovery algorithms are faster on these types of patterns
- Usually not flexible enough for the needs of molecular biology

Rigid patterns

- **Definition**: Rigid patterns are patterns which allow substitutions/“don’t care” symbols
- Note that the size of the pattern is not allowed to change
Hamming distance

• Definition: Given two strings $y$ and $w$ such that $|y| = |w|$, the Hamming distance $h(w, y)$ is given by the number of mismatches between $y$ and $w$

• Example:
  $y=$GATTACA
  $w=$TATAATA
  $h(w, y) = h(y, w) = 3$

Hamming neighborhood

• Definition: Given a string $y$, all strings at Hamming distance at most $d$ from $y$ are in its $d$-neighborhood

• Fact: The size $N(m, d)$ of the $d$-neighborhood of a string $y$, $|y| = m$, is

$$N(m, d) = \sum_{j=0}^{d} \binom{m}{j} (|\Sigma| - 1)^j \in O\left(m^d |\Sigma|^d\right)$$
Hamming neighborhood

- **Example:**
  \( y = \text{ATA} \) the 1-neighborhood is
  \( \{\text{CTA}, \text{GTA}, \text{TTA}, \text{AAA}, \text{ACA}, \text{AGA}, \text{ATC}, \text{ATG}, \text{ATT}, \text{ATA}\} \)

- This set can be written as a rigid pattern
  \( \{\text{NTA}\mid\text{ANA}\mid\text{ATN}\} \)

Models

- We may be able to observe occurrences of the neighbors of \( y \), but we may never observe an occurrence of \( y \)

- **Definition:** The center of the \( d \)-neighborhood \( y \) is also called the *model*

- **Definition:** We say that a model is *valid* if it has enough support (occurrences/colors)
y is the (unknown) model

$d$ is the number of allowed mismatches

$w_1, w_2, w_3$ belongs to the neighborhood of $y$

---

**Hamming neighborhood**

- **Fact:** Given two strings $w_1$ and $w_2$ in the $d$-neighborhood of the model $y$, then $h(w_1, w_2) \leq 2d$

- The problem of finding $y$ given $w_1, w_2, ...$ is sometimes called *Steiner sequence* problem

- Unfortunately, even if we were able to determine exactly all the $w_i$ in the neighborhood, there is no guarantee to find the unknown model $y$
Example

- Suppose $m=4$, $d=1$ and that we found occurrences of $\{\text{AAAA, TATA, CACA}\}$

- The pairwise Hamming distance is 2 but there is no string at Hamming distance 1 to each of these.

Word match filtering

- Fact: Given two strings $w_1$ and $w_2$ in the $d$-neighborhood of the model $y$, they both contain an occurrence of a word of length at least $|m/(2d+1)|$

- Example: $y = \text{GATTACA}$
  
  $w_1 = \text{GATTTCA}$
  $w_2 = \text{GGTTACA}$

  TT and CA are occurring exactly. In fact $|m/(2d+1)| = |7/3| = 2$
Word match filtering

- **Proof:** there are at least $m-2d$ matching positions, divided into at most $2d+1$ segments (some possibly of length 0) by intervening mismatches. The average length of a segment is therefore $(m-2d)/(2d+1)$. Hence there exists a segment with no mismatches of length at least \[ \lceil (m-2d)/(2d+1) \rceil \]
  \[ = \lfloor m/(2d+1) \rfloor. \]

Flexible patterns

- **Definition:** *Flexible patterns* are patterns which allow substitutions/“don’t care” symbols and variable-length gaps
  - e.g., Prosite \textbf{F}–\textbf{x}(5)–\textbf{G}–\textbf{x}(2,4)–\textbf{G}–*–\textbf{H}
- Note that the length of these pattern is variable
- Very expressive
- Space of all patterns is huge
Edit distance

- **Definition**: the *edit distance* between two strings *y* and *w* is defined as the minimum number of edit operations - insertions, deletions and substitutions - necessary to transform *y* into *w* (matches do not count)

- **Definition**: a sequence of edit operations to transform *y* into *w* is called an *edit script*

Edit distance

- The *edit distance problem* is to compute the edit distance between *y* and *w*, along with an optimal edit script that describes the transformation

- An alternative representation of the edit script is the *alignment*
Example

- Given \( w = \text{GATTACA} \)
  \[ y = \text{TATATA} \]
  
  \[
  \text{GATTACA} \text{«ATTACA»TTACA} \\
  \text{TATACA} \text{«TATATA»} (1 \text{ ins, 2 \text{ del, 1 \text{ sub}}})
  \]

  \[
  \text{GATTACA} \text{«TATTACA} \\
  \text{TATACA} \text{«TATATA»} (0 \text{ ins, 1 \text{ del, 2 \text{ sub}}})
  \]

- Edit distance is 3

Corresponding global alignment

- Given \( w = \text{GATTACA} \)
  \[ y = \text{TATATA} \]

- We can produce the following alignments
  
  \[
  \text{GAT} - \text{TAC} - \text{A} \quad \text{G} - \text{ATTAC} - \text{A} \\
  \text{--TATA} - \text{TA} \quad \text{--TAT} - \text{A} - \text{TA}
  \]

  where “–” represents a space (we cannot have “–” aligned with “–”)
Profiles

• *Position weight matrices, or profiles*, are $|\Sigma| \times m$ matrices containing real numbers in the interval $[0,1]$
  
  – e.g.

  
  \[
  \begin{array}{cccc}
    \text{A} & 0.26 & 0.22 & 0.00 & 1.00 & 0.11 \\
    \text{C} & 0.17 & 0.18 & 0.59 & 0.00 & 0.35 \\
    \text{G} & 0.09 & 0.15 & 0.00 & 0.00 & 0.00 \\
    \text{T} & 0.48 & 0.45 & 0.41 & 0.00 & 0.54 \\
  \end{array}
  \]

  – *consensus*

Profiles as classifiers

• Profiles can be used directly to implement very simple classifiers

• Suppose we have a sample $S+$ of known sites, and a sample $S-$ of non-sites

• Given a new sequence $x$, how do we classify $x$ in $S+$ or in $S-$?
Example: CRP binding sites

\[ S^+ = \{ \text{TTGTGGC, ACGTGAT, CTGTGAC, TTTTGAT, ATGTGAG, ATGAGAC, AAGTGTC, TTGTGAG, TTGTGAG} \} \]

Cyclic AMP receptor protein TFs in E.coli [Stormo & Hartzell, 89]

Training (CRP sites)

- Assume a Bernoulli model for each position

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>T</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.350</td>
<td>0.043</td>
<td>0.000</td>
<td>0.043</td>
</tr>
<tr>
<td></td>
<td>0.170</td>
<td>0.087</td>
<td>0.043</td>
<td>0.043</td>
</tr>
<tr>
<td></td>
<td>0.130</td>
<td>0.000</td>
<td>0.780</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>0.350</td>
<td>0.870</td>
<td>0.170</td>
<td>0.910</td>
</tr>
</tbody>
</table>

- Assume the uniform Bernoulli model for the non-sites \( S^- \), that is \( p_A = 0.25, p_C = 0.25, p_T = 0.25, p_G = 0.25 \) for all the positions
Testing

• Suppose you get $x = \text{GGTGTA}$
  Is $x$ more likely to belong to $S+$ or to $S-$?
  In other words, it is more likely to be generated from the Bernoulli model for $S+$ or from the uniform Bernoulli model (for $S-$)?

• Let’s compute the probability

$$P(x = \text{GGTGTA} \mid S+) = .35*.87*.78*.91*.83*.83*.30 = 0.045$$

$$P(x = \text{GGTGTA} \mid S-) = (.25)^7 = 0.0000061$$

$$LR(x) = P(x \mid S+) / P(x \mid S+)$$

Discovering Deterministic Patterns
Enumerative approach: idea

- Define the search space
- List exhaustively all the patterns in the search space
- Compute the statistical significance for all of them
- Report the patterns with the highest statistical significance

Enumerative approach

- E.g., search space for deterministic patterns of size $m$ is $O(|\Sigma|^m)$
- Can we do better than $|\Sigma|^m$?
Enumerative approach

- The search space for deterministic pattern is already too big
  - e.g., there are \(1,048,576\) possible deterministic patterns of size 10 on the DNA alphabet

- How to prune it?

Detection of Unusual Patterns
Naïve approaches

1) Enumerate and test all patterns composed by \( m \) symbols, for \( 1 \leq m \leq n \)

2) Enumerate and test all patterns which occur in the sequences

\[
n = 1,000,000 \quad |\Sigma| = 4
\]

1) Patterns to be tested \( O(|\Sigma|^n) \)
   in this case \( \propto 4^{1,000,000} \)

2) Patterns to be tested \( O(n^2) \)
   in this case \( \propto 1,000,000^2 \)
Enumerating the $O(n^2)$ patterns

Basic operations on the trie

(“trie” comes from information retrieval)

- **Construction**
- **Traversals**
  - Breadth-first
  - Depth-first
- **Query**
  - Given a pattern of length $m$, we can check if it belongs to the trie in time $O(m)$
**Suffix trie**

- We build a trie with all the suffixes of the text $x$
- Example: if $x = \text{GATTACA}$ we use
  - GATTACA
  - ATTACA
  - TTACA
  - TACA
  - ACA
  - CA
  - A

The suffix trie collects in the internal nodes *all* the substrings of $x$
Suffix trie for “GATTACA$”

The suffix trie collects in the internal nodes all the substrings of x$

Suffix trie

- Construction $O(n^2)$
- Space $O(n^2)$
- Query $O(m)$

- We can do better by removing unary nodes from the tree, and coalescing the edges
- The result is called suffix tree
Suffix tree for “GATTACA$”

The suffix tree collects in the implicit internal nodes all the substrings of $x$

The locus of a string is the node in the tree corresponding to it

The label in the leaves identifies the suffix position (used to find pos all occs)

The number of leaves in the subtree corresponds to the number of occurrences

Space analysis

- Every node is branching
- The number of leaves is $n$
- Therefore the overall number of nodes is at most $2n-1$
- Use two integers (constant space) to identify labels on the arcs
- Therefore the overall size of the tree is $n$
- Note: we assume the standard RAM model that $log n$ bits can be read, written or compared in constant time
**Brute force construction**

Worst case $O(n^2)$  
Average case $O(n \log n)$

**Computing number of occurrences**

**Annotation**  
$Annotate_f(w)$ (suffix tree $T$)  

- for each leaf $u$ of $T$ do
  - let $f(L(u)) = 1$
- visit $T$ in depth-first traversal, for each internal node $u$ do
  - let $f(L(u))$ equal to the sum of $f(\cdot)$ of the children of $u$

Time complexity is $O(n)$
Suffix tree for “abaababaabaababaababa$”

Internal nodes are annotated the count of the number of occurrences.

Suffix links connect the locus of $cw$ to the locus of $w, c \in \Sigma, w \in \Sigma^*$
Suffix links

Suffix links help identifying isomorphic subtrees

All the suffix links (except leaves)
Suffix trees in a nutshell

- Suffix trees can be built in $O(n)$ time and space [Weiner73, McCreight76, Ukkonen95, Farach97]
- Number of occurrences can be computed in $O(n)$ time
- Number of colors can be computed in $O(n)$ time [Hui92, Muthu02]

Which patterns do we count? What do we expect, under the given model?

What is unusual?

How do we count efficiently?

How many patterns can be unusual?

How do we compute statistical parameters efficiently?
Scores based on occurrences

\[ z_1(y) = f(y) - E(Z_y) \]
\[ z_2(y) = \frac{f(y) - E(Z_y)}{\sqrt{E(Z_y)}} \]
\[ z_3(y) = \frac{f(y) - E(Z_y)}{\sqrt{E(Z_y)} \ (1 - \hat{p})} \]
\[ z_4(y) = \frac{f(y) - E(Z_y)}{\sqrt{Var(Z_y)}} \]

where \( Z_y \) is a r.v. for the number of occurrences of \( y \)

Scores based on colors

\[ z_7(y) = c(y) - E(W_y) \]
\[ z_8(y) = \frac{c(y) - E(W_y)}{\sqrt{E(W_y)}} \]

where \( W_y \) is a r.v. for the number of colors of \( y \)
What is “unusual”?

Definition:
Let $y$ be a substring of $x$ and $T \in \mathbb{R}^+$
- If $z(y) > T$, then $y$ is over-represented
- If $z(y) < -T$, then $y$ is under-represented
- If $|z(y)| > T$, then $y$ is unusual

Problem

Given
- Single/multi sequence $x$
- Type of count ($f$ or $c$)
- Score function $z$
- Threshold $T$

Find
- The set of all unusual patterns in $x$ w.r.t. $(f/c,z,T)$
How to choose the threshold

\[
P \left( \frac{f(y) - E(Z_y)}{\sqrt{Var(Z_y)}} > 2 \right) = 0.0456
\]

Computational Problems

- Counting “events” in strings
  - occurrences
  - colors

- Computing expectations, variances, and scores (under the given model)

- Detecting and visualizing unusual patterns
Combinatorial Problem

• A sequence of size $n$ could have $O(n^2)$ unusual patterns

• How to *limit* the set of unusual patterns?

Theorem:
Let $C$ be a set of patterns from text $x$. If $f(y)$ remains constant for all $y$ in $C$, then any score of the type

$$z(y) = \frac{f(y) - E(y)}{N(y)}$$

is monotonically increasing with $|y|$ provided that

- $N(y)$ is monotonically decreasing with $|y|$
- $E(y)/N(y)$ is monotonically decreasing with $|y|$
Theorem:
Score functions
\[ z(y) = f(y) - E(Z_y), \quad z(y) = c(y) - E(W_y), \]
\[ z(y) = \frac{f(y) - E(Z_y)}{\sqrt{E(Z_y)}}, \quad z(y) = \frac{c(y) - E(W_y)}{\sqrt{E(W_y)}}, \]
\[ z(y) = \frac{f(y) - E(Z_y)}{\sqrt{E(Z_y)(1 - \hat{p})}}, \]
are monotonically increasing with \(|y|\), for all \(y\) in class \(C\)

Theorem:
If \(p_{\text{max}} < \min \{1/\sqrt{4|y|}, \sqrt{2} - 1\}\), then
\[ z(y) = \frac{f(y) - E(Z_y)}{\sqrt{\text{Var}(Z_y)}} \]
is monotonically increasing with \(|y|\), for all \(y\) in class \(C\)
abaababaabaababaabaababa

abaababaabaababaabaababa

abaababaabaababaabaababa

abaaababaabaababaabaababa

aa  aa
abaababaabaababaabaababa
abaababaabaababaabaababa
abaabaabaabaabaabaabaabaabaaba
min(C): candidate under-repr

max(C): candidate over-repr
The partition \( \{C_1, C_2, \ldots, C_l\} \) of the set of all substrings of \( x \), has to satisfy the following properties, for all \( 1 \leq i \leq l \),

- \( \min(C_i) \) and \( \max(C_i) \) are unique
- all \( w \) in \( C_i \) belong to some \( (\min(C_i), \max(C_i)) \)-path
- all \( w \) in \( C_i \) have the same count
\[ x = \text{abaababaababaababaababa} \]

\[ \text{a} \]
(13)

\[ \text{aa} \]
aab
aaba
baa
baab
baaba
abaab
abaaba
abaababa
(4)

\[ \text{bab} \]
baa
aba
abaab
abaaba
aababa
baababa
baaba
ababa
bababa
(3)

\[ \text{b} \]
ba
ab
aba
(8)

\[ \text{babaa} \]
babaab
babaaba
ababaaa
ababaab
ababaaba
aababaab
aababaaba
baababa
baababaab
baababaaba
(3)

\[ \text{baabaa} \]
babaab
babaaba
ababaab
ababaaba
aababaaba
aababaaba
baababa
baababaab
baababaaba
(4)

\[ \text{baaba} \]
babaab
babaaba
ababaab
ababaaba
aababaaba
aababaaba
baababa
baababaab
baababaaba
(2)

\[ \text{babaaab} \]
babaab
babaaba
ababaab
ababaaba
aababaaba
aababaaba
baababa
baababaab
baababaaba
(1)
Theorem:
The number of classes is at most $2n$
Computing on the Suffix Tree

- Equivalence classes can be computed in $O(n)$ time (by merging isomorphic sub-trees of the suffix tree [ABL, Recomb02])

- Expectations, variances and scores can be computed in amortized constant time per node [ABLX, JCB00]

Theorem:
The set of over- and under-represented patterns can be detected in $O(n)$ time and space
Conclusions on Verbumculus

• Pros:
  – exhaustive
  – linear time and space

• Cons:
  – limited to deterministic patterns

http://www.cs.ucr.edu/~stelo/Verbumculus
Discovering Rigid Patterns

Complexity results

• Li et al., [STOC 99] proved several important theoretical facts

• Many of the problems in pattern discovery turn out to be NP-hard

• For some there is a polynomial time approximation scheme (PTAS)
Consensus Patterns

• Consensus patterns problem: Given a multisequence \( \{x_1, x_2, \ldots, x_k\} \) each of length \( n \) and an integer \( m \), FIND a string \( y \) of length \( m \) and substring \( t_i \) of length \( m \) from each \( x_i \) such that \( \sum_i h(y, t_i) \) is minimized

• Theorem [Li et al., 99]: The consensus pattern problem is NP-hard

Closest string

• Closest string problem: given a multisequence \( \{x_1, x_2, \ldots, x_k\} \) each of length \( n \), FIND a string \( y \) of length \( n \) and the minimum \( d \) such that \( h(y, x_i) \leq d \), for all \( i \)

• Theorem: The closest string problem is NP-hard
Closest substring

- Closest substring problem: given a multisequence \( \{x_1, x_2, \ldots, x_k\} \) each of length \( n \) and an integer \( m \), FIND a string \( y \) of length \( m \) and the minimum \( d \) such that for each \( i \) there is a substring \( t_i \) of \( x_i \) of length \( m \) satisfying \( h(y, t_i) \leq d \)

- Theorem: The closest substring problem is NP-hard (it is a harder version of Closest string)

NP-hard: what to do?

- Change the problem
  - e.g., “relax” the class of patterns

- Accept the fact that the method may fail to find the optimal patterns
  - Heuristics
  - Randomized algorithms
  - Approximation schemes
Discovering Rigid Patterns

• We report on four recent algorithms

• Teiresias [1998]
• Winnower [2000]
• Projection [2001]
• Weeder [2001]

• (disclaimer: my selection is biased)
Teiresias algorithm

- By Rigoustos and Floratos [Bioinformatics, 1998]

- The worst case running time is exponential, but works reasonably fast on average

- A recent improved algorithm runs in polynomial time by reporting only to irredundant patterns [Parida et al., 2000]

Teiresias patterns

- Teiresias searches for rigid patterns on the alphabet $\Sigma \cup \{.\}$ where “.” is the don’t care symbol

- However, there are some constrains on the density of “.” that can appear in a pattern
\(<L, W>\) patterns

- **Definition:** Given integers \( L \) and \( W \), \( L \leq W \), \( y \) is a \(<L, W>\) pattern if
  - \( y \) is a string over \( \Sigma \cup \{.,\} \)
  - \( y \) starts and ends with a symbol from \( \Sigma \)
  - any substring of \( y \) containing exactly \( L \) symbols from \( \Sigma \) has to be shorter (or equal) to \( W \)

Example of \(<3,5>\) patterns

- **AT..CG..T** is a \(<3,5>\) pattern

- **AT..CG.T.** is not a \(<3,5>\) pattern, because it ends with “.”

- **AT.C.G..T** is not a \(<3,5>\) pattern, because the substring **C.G..T** is 6 characters long
Teiresias

- **Definition:** A pattern $w$ is more *specific* than a pattern $y$, if $w$ can be obtained from $y$ by changing one or more “.” to symbols from $\Sigma$, or by appending any sequence of $\Sigma \cup \{.\}$ to the left or to the right of $y$

- **Example:** given $y = \text{AT.CG.T}$, the following patterns are more specific then $y$:
  - ATCCG.T,
  - CAT.CGCT,
  - AT.CG.T.A,
  - T.AT.CGTT.A

Teiresias

- **Definition:** A pattern $y$ is *maximal* with respect to the sequences $\{x_1,x_2,\ldots,x_k\}$ if there exists no pattern $w$ which is more specific than $y$ and $f(w) = f(y)$

- Given $\{x_1,x_2,\ldots,x_k\}$ and parameters $L,W,K$, Teiresias reports all the *maximal* $<L,W>$ patterns that have at least $K$ colors
Teiresias algorithm

• **Idea:** if $y$ is a $<L,W>$ pattern with at least $K$ colors, then its substrings are also $<L,W>$ patterns with at least $K$ colors

• Therefore, Teiresias assembles the maximal patterns from smaller patterns

• **Definition:** A pattern $y$ is *elementary* if it is a $<L,W>$ pattern containing exactly $L$ symbols from $\Sigma$

Teiresias algorithm

• Teiresias works in two phases
  – Scanning: find all elementary patterns with at least $K$ colors; these become the initial set of patterns
  – Convolution: repeatedly extend the patterns by “gluing” them together

• **Example:** $y = \textbf{AT} . . \textbf{CG.T}$ and $w = \textbf{G.T.A}$ can be merged to obtain $\textbf{AT} . . \textbf{CG.T.A}$
Convolution phase

- For each elementary pattern $y$, try to extend it with all the other elementary patterns
- Any pattern that cannot be extended without losing support can be potentially maximal

Convolution phase

- To speed-up this phase, one wants to avoid the all-against-all comparison
- The authors devise two partial orderings $<_{pf}$ and $<_{sf}$ on the universe of patterns
- Using these orderings to schedule the convolution phase, they guarantee that
  - all patterns are generated
  - a maximal pattern $y$ is generated before any non-maximal pattern subsumed by $y$
Partial ordering $\prec_{pf}$

- **Definition**: determine whether $y \prec_{pf} w$ or $w \prec_{pf} y$ using the following algorithm
  - align $y$ and $w$ such that the leftmost residues are in the same column
  - examine one column after the other (left to right) and stop whenever one column has a residue and the other has a “.”
  - if the residue comes from $y$ then $y \prec_{pf} w$
  - if the residue comes from $w$ then $w \prec_{pf} y$

**Example**

- $y = \text{ASD...F}$
  $w = \text{SE.ERF.DG}$
  $y \prec_{pf} w$

- $y = \text{ASD...F}$
  $w = \text{SE.ERF.DG}$
  $w \prec_{sf} y$
Teiresias algorithm

- Initialize the stack with elementary patterns with support at least $K$
- Order the stack according to $<_{pf}$ and $<_{sf}$
- Repeat
  - Repeat
    - Try to extend the top pattern to the right with all the others in the prefix-wise ordering
    - If a new pattern is formed with have enough support, it becomes the new top
  - Until the top can no longer be extended to the right
  - Do the same for left extension, using the ordering $<_{sf}$
  - Check the top for maximality, if so pop it and report it
- Until stack is empty

Conclusions on Teiresias

- It can be proved that Teiresias correctly reports all $<L,W>$ maximal patterns
- Pros:
  - provably correct
  - fast on average input
- Cons:
  - exponential time complexity
  - limited to $<L,W>$ patterns
Winnower

Pevzner and Sze, UCSD

Winnower

• Invented by Pevzner and Sze [ISMB 2000]
• Initially designed to solve the $(15,4)$-motif challenge
• Planted $(m,d)$-motif problem:
  – The problem is to determine an *unknown* pattern $y$ of length $m$ in a set of $k$ nucleotide sequences, each of length $n$, and each one containing exactly one occurrence of a string $w$ such that $h(y,w)=d$
Winnower

- Pevzner and Sze show that the most popular algorithms (Consensus, GibbsDNA, MEME) fail to solve (most of the times) the $(15,4)$-motif problem $[n=600, k=20]$
- (Note: this comparison is not totally *fair*)
- Why the $(15,4)$-motif problem is difficult?
- Because two strings in the class of the $(15,4)$ unknown pattern may differ by as many as 8 positions out of 15, a rather large number

Winnower

- **Idea**: Search for groups of strings of length $m$ such that any two in a group differ at most by $2d$ positions
- Remember however that this may not be sufficient
Winnower

- How to find groups of patterns such that given any two elements \( w_1 \) and \( w_2 \) in the group, \( h(w_1, w_2) \leq 2d \)?

- One could generate \( \binom{k}{2} \) multiple alignments to find out all pairs of substrings of length \( m \) that have at most \( 2d \) mismatches (Consensus [Hertz & Stormo 1999])

Winnower

- Winnower builds a graph \( G \) in which
  - each vertex corresponds to a distinct string of length \( m \)
  - two vertices are connected by an edge if the Hamming distance between the corresponding strings is at most \( 2d \), and the strings do not come from the same sequence (remember that we are guaranteed that there is only one occurrence of the unknown pattern in each sequence)
Graph for the \((15,4)\)-problem

- They report that for each “signal”-edge there are about 20,000 spurious-edges

- Finding the signal among the noise is a “daunting task”

Winnower

- Winnower searches the graph \(G\) for \textit{cliques}, which are subsets of vertices totally connected

- But the problem of finding large cliques in graphs is \textit{NP}-complete
Multipartite graphs

- **Definition**: A graph $G$ is $n$-partite if its vertices can be partitioned into $n$ sets, such that there is no edge between any two vertices within a set.

- **Fact**: Winnower’s graph is $k$-partite.

---

Example

- Given sequences $\{abde, afcg, hbc, jbc\}$ we look for a $(3,1)$-motif.
Idea

- Each vertex of the clique has to be in a different partition

- We look for cliques that have exactly one vertex in each partition

Extendable cliques

- **Definition**: a vertex $u$ is a *neighbor* of a clique $\{v_1, \ldots, v_s\}$ if $\{v_1, \ldots, v_s, u\}$ is also a clique for $G$, when $s < k$

- **Definition**: a clique is called *extendable* if it has at least one neighbor which has at least one vertex in every part of the $k$-partite graph $G$
Extendable cliques

- **Definition**: A clique with $k$ vertices, each in a different partition is called *maximal*
- Consider a maximal clique and take a subset of $t$ of its vertices: this subset is an extendable clique
- **Idea**: remove edges that do not belong to extendable cliques

---

**Fact**: For any clique of size $k$ there are $\binom{k}{t}$ extendable cliques with $t$ vertices

**Fact**: Any edge belonging to a clique with $k$ vertices is member of at least $\binom{k-2}{t-2}$ extendable cliques of size $t$
Idea

An edge that is not member of at least \( \binom{k-2}{t-2} \)
expandable cliques of size \( t \) cannot be part of a maximal clique and therefore it can be removed.

\[ t=1 \]

- For \( t=1 \), each vertex is a clique
  - it is extendable if it is connected to at least one vertex in each partition
- Delete all edges corresponding to vertices that do not have a neighbor in each partition
- Iterate
Example

Example
\[ t=2 \]

- For \( t=2 \), each pair of vertices \( u, v \) such that there is an edge \((u,v)\) is a clique
  - it is extendable if there is vertex \( z \) in each of the other \( k-2 \) partitions such that \((u,v,z)\) is a cycle of length 3
  - each edge should belong to at least \((k-2 \text{ choose } t-2)=(n-2 \text{ choose } 0)=1\) clique of size 2

\[ t>2 \]

- For \( t=3 \), Winnower removes edges that belong to less than \( k-2 \) extendable cliques of size 3

- For \( t=4 \), Winnower remove edges that belong to less than \((k-2)(k-1)/2\) extendable cliques of size 4

- ...
Remarks on Winnower

• Pros:
  – more effective than Meme, Consensus and GibbsDNA for the (15,4) problem

• Cons:
  – randomized
  – time-complexity can be very high (e.g., for \( t=3 \) is \( O(n^4) \))
  – need to know \( m \) and \( d \) in advance
  – assume exactly one occurrence per sequence

Projection
Random Projection algorithm

- Proposed by Buhler and Tompa [Recomb 2001]
- The algorithm was initially designed to solve the \((m,d)\)-motif planted problem

Analysis on \((m,d)\)-motif problem

Suppose A,C,T,G have probability 1/4. Then the probability that a pattern of size \(m\) occurs at a given position is \(p_{(0)} = (1/4)^m\). If we allow up to one mismatch, the probability becomes

\[
p_{(1)} = p_{(0)} + m(3/4)(1/4)^{m-1}.
\]

If we allow at most two, it becomes

\[
p_{(2)} = p_{(1)} + \frac{m(m-1)}{2}(3/4)^2(1/4)^{m-2}.
\]

In general, if we allow up to \(d\) mismatches, \(p_{(d)} = \sum_{i=0}^{d} \binom{m}{i} \left(\frac{3}{4}\right)^i \left(\frac{1}{4}\right)^{m-i}\).
Analysis on \((m,d)\)-motif problem

If \(Z\) is the r.v. for the number of occurrences, then
\[
P(Z > 0) = 1 - P(Z = 0) = 1 - (1 - p_{(d)})^{n-m+1}
\]
If we have \(k\) sequences, we get that the probability that a particular \(y\) occurs at least once in each sequence is
\[
\left(1 - (1 - p_{(d)})^{n-m+1}\right)^{k}.
\]
Therefore, the expected number of patterns is
\[
E(n, m, k, d) = 4^{m} \left(1 - (1 - p_{(d)})^{n-m+1}\right)^{k}.
\]

Stats of spurious \((m,d)\)-motifs in simulated data \((k=20,n=600)\)

<table>
<thead>
<tr>
<th>(m)</th>
<th>(d)</th>
<th>(E(600,m,20,d))</th>
<th>(E(600,m+1,20,d))</th>
<th>ape</th>
<th>Correct</th>
<th>Spurious</th>
<th>19/20</th>
<th>iter</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>2</td>
<td>1.6 (\times) 10^{-8}</td>
<td>6.1 (\times) 10^{-8}</td>
<td>0.28</td>
<td>11</td>
<td>5</td>
<td>4</td>
<td>1483</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>4.7 (\times) 10^{-7}</td>
<td>3.2 (\times) 10^{-7}</td>
<td>0.026</td>
<td>1</td>
<td>13</td>
<td>6</td>
<td>2443</td>
</tr>
<tr>
<td>13</td>
<td>4</td>
<td>5.2 (\times) 10^{-7}</td>
<td>4.2 (\times) 10^{-7}</td>
<td>0.062</td>
<td>2</td>
<td>15</td>
<td>3</td>
<td>4178</td>
</tr>
<tr>
<td>15</td>
<td>5</td>
<td>2.8 (\times) 10^{-7}</td>
<td>2.3 (\times) 10^{-7}</td>
<td>0.018</td>
<td>0</td>
<td>7</td>
<td>13</td>
<td>6495</td>
</tr>
<tr>
<td>17</td>
<td>6</td>
<td>0.88 (\times) 10^{-8}</td>
<td>7.1 (\times) 10^{-8}</td>
<td>0.022</td>
<td>0</td>
<td>8</td>
<td>12</td>
<td>9272</td>
</tr>
</tbody>
</table>

**Bottom-line:** the \((9,2)\)-, \((11,3)\)-, \((13,4)\)-, \((15,5)\)- and \((17,6)\)-motif problems are probably impossible to solve
Random Projections

- **Idea**: select $t$ random positions and for each substring of length $m$ of the text hash its selected positions into a table

- Hopefully, the cell corresponding to the planted motif will be the one with the highest count

Random Projection algorithm

- Parameters $(m,d), n, k, s$, possibly $i$
- Set $t < m-d$ and $4^t > k(n-m+1)$
- Build a table with all substrings of length $m$
- Repeat $i$ times
  - Select randomly $t$ positions
  - Repeat for all substrings in the table
    - Increase the count of the cell indexed by the $t$ positions
- Select all cells with count $\geq s$
Random Projection algorithm

- We want $t < m-d$ because we want to sample from the “non-varying” positions

- The number of iterations $i$ can be estimated from $m$, $d$ and $t$

Random Projection algorithm

- Since we are hashing $k(n-m+1)$ substrings of size $m$ into $4^t$ buckets, if $4^t > k(n-m+1)$ each bucket will contain on average less than one substring (set $s=1$)

- The constrain is designed to filter out the noise

- The bucket corresponding to the planted motif is expected to contain more motif instances than those produced by a random sequence
Random Projection algorithm

- If the constrain $4^t > k(n-m+1)$ cannot be enforced, the authors suggest to set $t = m-d-1$ and the threshold $s = 2 \left[ k(n-m+1)/4^t \right]$ (twice the average bucket size)

Motif refinement

- The algorithm will try to recover the unknown motif from each cell having at least $s$ elements

- The primary tool for motif refinement is expectation maximization (EM)
Experiments

• Projection can handle the $(15,4)$- $(14,4)$- $(16,5)$- and $(18,6)$-motif problem ($k=20$, $n=600$)

• Winnower fails the $(14,4)$- $(16,5)$- and $(18,6)$-motif problem

Results

<table>
<thead>
<tr>
<th>$m$</th>
<th>$d$</th>
<th>Gibbs</th>
<th>WINNOWER</th>
<th>SP-STAR</th>
<th>PROJECTION</th>
<th>Correct</th>
<th>iter</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>
<td>20</td>
<td>72</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>
<td>20</td>
<td>16</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>
<td>20</td>
<td>259</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>
<td>20</td>
<td>62</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>
<td>19</td>
<td>647</td>
</tr>
<tr>
<td><strong>15</strong></td>
<td><strong>4</strong></td>
<td><strong>0.19</strong></td>
<td><strong>0.92</strong></td>
<td><strong>0.73</strong></td>
<td><strong>0.93</strong></td>
<td><strong>20</strong></td>
<td><strong>172</strong></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>
<td>16</td>
<td>1292</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>
<td>19</td>
<td>378</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>
<td>16</td>
<td>2217</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>
<td>20</td>
<td>711</td>
</tr>
</tbody>
</table>

$k=20$, $n=600$, winnower ($t=2$), projection ($t=7, s=4$, 20 random instances)
Remarks about Projection

- Pros:
  - fast and effective

- Cons:
  - need to know $m$ and $d$ in advance
  - randomized

Weeder
Weeder

- Proposed by Pavesi, Mauri and Pesole [ISMB 2001]

- Draw ideas from PRATT by [Jonassen95, Jonassen97] and [Sagot98]

- It is an exhaustive approach for a particular class of rigid patterns

Exhaustive approach

- Suppose that you want to spell out all possible $(m,d)$ rigid patterns that has at support least $q$

- One way to do it, is to use a (generalized) suffix tree [Sagot 98]
Idea [Sagot 98]

- Any deterministic pattern (substring) \( w \) corresponds to a path in the tree ending in a node \( u \), called the *locus* of \( w \) – the number of leaves in the subtree rooted at \( u \) gives the support
- Any model (rigid pattern) corresponds to a set of paths in the tree ending in nodes \( \{u_1, u_2, \ldots, u_l\} \) – the total number of leaves in the subtrees rooted at \( \{u_1, u_2, \ldots, u_l\} \) gives the support

Example

![Diagram of virtual trie and suffix tree with marked Hamming distance and support values](image)

- For the substring "ATA", the Hamming distance is 2, and the support is 4.
Exhaustive approach [Sagot 98]

- Start with all paths of length $d$ with enough support (they represent valid models)
- At each path-extension keep track of the mismatches and the support
  - if the number of mismatches has not been reached the model will be extended by the symbols in $\Sigma$ (therefore the number of models will be scaled up by a factor $|\Sigma|$)
  - otherwise we are allowed just to follow the arcs
Time complexity [Sagot 98]

- Finding all the models with support=occurrences in a single sequence takes $O(n N(m,d)) = O(n m^d |\Sigma|^d)$
- Finding all the models with support=colors in a multisequence takes $O(n k^2 N(m,d)) = O(n k^2 m^d |\Sigma|^d)$
- Note that the complexity is exponential (with $d$)

Weeder

- Pavesi et al., implemented the algorithm by Sagot but it was running too slow, and they decided to change the class of patterns
- Weeder is designed to find rigid patterns which have an amount of mismatches proportional to their length (the same constrain applies also to all their prefixes)
Example $\varepsilon = 0.25$

Time complexity

- By restricting the number of mismatches to $\varepsilon m$, the time complexity becomes $O(n k [1/\varepsilon]^\varepsilon m |\Sigma|^\varepsilon m)$
The \((15,4)\)-motif challenge … again

- Since the restriction on the density of the mismatches, the authors report that Weeder has probability 0.6 to catch the motif in ONE sequence
- Then, the probability of Weeded to get the motif in all the 20 sequence is almost zero
- On the other hand, running the Sagot’s version is too time-consuming

Idea

- Split the set of sequence into two halves
- Run Weeder on each of the two sets requiring support \(k/4\) (instead of \(k/2\))
- The probability that the \((15,4)\)-motif will be in either subset is 0.98
- The pool of model candidates is then processed with Sagot’s algorithm
Remarks about Weeder

• Pros:
  – Possibly exhaustive (if using Sagot’s algorithm)
  – The relative error rate $\varepsilon$ may be more meaningful than $d$ and allows one not to specify in advance $m$

• Cons:
  – Very slow if run exhaustively - it cannot be considered exhaustive in practice

Discovering Profiles
Discovering Profiles

• If one assumes the unknown profile to have been generated by a sequence of independent r.v.s then the observed frequency of letters in the columns of the profile are the ML estimates of the distributions of the r.v.s

• Unfortunately we do not know the positions of the profile in the multisequence

Gibbs sampler
Gibbs sampling

• Proposed by Lawrence, *et al.*, [Science, 1993]
• Web servers at
  [http://bayesweb.wadsworth.org/gibbs/gibbs.html](http://bayesweb.wadsworth.org/gibbs/gibbs.html) and
  [http://argon.cshl.org/ioschikz/gibbsDNA/](http://argon.cshl.org/ioschikz/gibbsDNA/)
• **Input**: multisequence \( \{x_1, x_2, \ldots, x_k\} \)
  pattern length \( m \)
• **Output**: a matrix profile \( q_{i,b}^m, b \in \Sigma, 1 \leq i \leq m, \) and
  positions \( s_j, 1 \leq j \leq k, \) of the profile in the \( k \)
  sequences

Gibbs sampling

• The algorithm maintains the background distribution \( p_A, \ldots, p_T \) of the symbols not
described by the profiles

• \( P(y) \) is the probability of \( y \) based on the
  background distribution \( p_b, b \in \Sigma \)

• \( Q(y) \) is the probability of \( y \) based on the
  profile \( q_{i,b}^m, 1 \leq i \leq m, b \in \Sigma \)
Gibbs sampling

• Idea: the profile is obtained by locating the positions which maximizes $Q(y)/P(y)$; once the positions are obtained a new, more accurate, version of the profile can be obtained

• Initialize the initial positions $s_j$ randomly

Gibbs sampling

Gibbs sampler iterates 1), 2) until convergence

1) Predictive update step: randomly choose one of the $k$ sequences, say $r$. The matrix profile $q_{i,b}$ and the background frequencies $p_b$ are recomputed from the current positions $s_j$ in all sequences excluding $r$

2) Sampling step: assign a weight $z(y)=Q(y)/P(y)$ to each substring $y$ of length $m$. Select randomly a substring $y$ with probability $z(y)/\sum_y z(y)$, and then update $s_j$
Gibbs sampling

- The more accurate the pattern description in step 1), the more accurate the determination of its position in step 2), and vice versa.
- Once some correct positions have been selected by chance, $q_{i,b}$ begins to reflect, albeit imperfectly, the unknown pattern.
- This process tends to recruit further correct positions which in turn improve the discriminating power of the evolving pattern.

Gibbs sampling

- How to update the matrix profile $q_{i,b}$ and the background frequencies $p_b$?
- We set $q_{i,b} = (f^i(b) + d_b)/(k - 1 + \sum_c d_c)$ where $f^i(b)$ is the number of times we observe symbol $b$ in the position $i$ of the profile (currently placed at position $s_j$), except for sequence $r$ ($d_b$ are pseudo-counts).
- We set the background probabilities $p_b = f(b)/\sum_c f(c)$ for all symbols in positions not covered by the profile.
Phase shift problem

• Suppose that the “strongest” pattern begin, for example, at position 7, 19, 8, 23, ...

• If Gibbs happens to choose $s_1=9, s_2=21$ it will most likely choose $s_3=10$ and $s_4=25$

• The algorithm can get stuck in local maxima, which are the shifted form of the optimal pattern

Phase shift problem

• The problem can be alleviated by adding a step in which the current set of positions are compared with sets of shifted left and right positions, up to a certain number of symbols

• Probability ratios may be calculated for all positions, and a random selection is made with respect to the appropriate weight
Gibbs sampling

• It can be generalized to:

• Find also the length of pattern $m$

• Find a set of matrix profiles, instead of one

Gibbs sampling

• Since Gibbs sampler is an heuristic rather than a rigorous optimization procedure, one cannot guarantee the optimality of the result

• It is a good practice to run the algorithm several times from different random initial positions
Gibbs sampling vs. EM

- Although EM and Gibbs are built on common statistical foundation, the authors claim that Gibbs outperforms EM both in term of time complexity and performance.

- “EM is deterministic and tends to get trapped by local optima which are avoided by Gibbs … HMMs permit arbitrary gaps … have greater flexibility, but suffer the same penalties …”

Expectation Maximization and MEME
Expectation maximization

• EM was designed by Dempster, Laird, Rubin [1977]

• EM is a family of algorithms for maximum likelihood estimation of parameters with “missing data”

EM, when?

• When we want to find the maximum likelihood estimate of the parameters of a model and
  – data is incomplete, or
  – the optimization of the maximum likelihood function is analytically intractable but the likelihood function can be simplified by assuming the existence of additional, missing, parameters value
Expectation maximization

- EM approaches the problem of missing information by iteratively solving a sequence of problems in which expected information is substituted for missing information.

Expectation maximization

- All EM algorithms consist of two steps:
  1) the expectation step (E-step)
  2) the maximization step (M-step)

- The expectation step is with respect to the unknown underlying variables, using the current estimate of the parameters and conditioned upon the observation.
Expectation maximization

- The maximization step provides a new estimate of the parameters

\[ \theta_1 \gg \theta_2 \gg \theta_3 \gg \ldots \gg \theta_t \gg \theta_{t+1} \gg \ldots \]

- The two steps are iterated until convergence

General framework for EM

- Suppose we want to find the parameters \( \theta \) of a model (training)

- We observe \( x \) (training set)

- The probability of \( x \) under \( \theta \) is also determined by the missing data \( y \)
Incomplete data model

An occurrence of \((x,y)\) implies an occurrence of \(x\), however only \(x\) can be observed. This observation reveals the subset \(\{(x,y), for\ all\ y\}\)

Expectation maximization

- **Example**: For HMMs, \(x\) is the sequence we want to learn from, \(\theta\) is the transition and emission probabilities, \(y\) is the path through the model
- **Example**: In the case of Random Projections, \(x\) are the subsequences corresponding to a cell with count higher than the threshold, \(\theta\) are the parameters of a representation of the \((m,d)\) pattern, \(y\) are all the missing positions
MEME

- Proposed by Bailey and Elkan [Machine Learning J., 1995]
- “Multiple EM for Motif Elicitation” (MEME) is an improved version of the expectation maximization approach by Lawrence and Reilly [Proteins, 1990] (see appendix)
- Designed to discover profiles (no gaps)
- Server at [http://meme.sdsc.edu/meme/](http://meme.sdsc.edu/meme/)

MEME

- There are three main differences w.r.t. Lawrence *et al.*:
  1) the initial profiles are not chosen randomly, but they are substrings which actually occur in the sequences
  2) the assumption that there is only one occurrence of the motif is dropped
  3) once a profile has been found, it is reported, and the iterative process continues
Using substring as starting points

• **Idea**: substrings actually occurring in sequence are better starting points than random choices
• Each substring is converted into a profile
• Assigning 1.0 to the occurring symbol and 0.0 to the others is a bad choice, because EM cannot move from this
• The authors arbitrarily assign probability 0.5 to the symbol and 0.5/3 for the other three

Using substring as starting points

• It would be too expensive to run EM until convergence from each substring

• It turns out that this is not necessary

• EM converges very quickly from profiles obtained from substrings, and the best starting point can be found running only one iteration
MEME algorithm

• Repeat
  – For each substring $y$ in $\{x_1, x_2, \ldots, x_k\}$ do
    • Run one EM iteration with profile computed from $y$
    • Choose the profile $q$ with highest likelihood
    • Run EM until convergence starting from $q$
    • Report the profile $q$
    • Erase the occurrences of $q$ from dataset
  • Until max number of iterations is reached

Dealing with multiple occurrences

• MEME allows to drop the “one-per-sequence” assumption
• The basic idea is to require the user to supply an estimated number of occurrences of the unknown profile and use that to normalize the estimation process of the EM algorithm
• The authors claim that the exact value of the number of occurrences is not critical
Finding multiple profiles

- MEME does not stop after finding the most likely profile
- Once a profile is found and reported, it is “probabilistically erased” by changing some position-dependent weight
- The process continues until a number of predetermined motifs have been found
  - (see appendix for mega-prior heuristic)

THE END

Latest version of the slides at [http://www.cs.ucr.edu/~stelo/ismb02/](http://www.cs.ucr.edu/~stelo/ismb02/)