Stochastic Context-Free-Grammars

Challenges in Computational Biology

DNA

Genome Assembly

Gene Finding

Sequence alignment

Evolutionary Theory

Database lookup

RNA folding

Gene expression analysis

RNA transcript

Protein network analysis

Regulatory network inference

Emerging network properties

The world before DNA or Protein

RNA World

• RNA can be protein-like
  – Ribozymes can catalyze enzymatic reactions by RNA secondary fold
  – Small RNAs can play structural roles within the cell
  – Small RNAs play versatile roles in gene regulatory

• RNA can be DNA-like
  – Made of digital information, can transfer to progeny by complementarity
  – Viruses with RNA genomes (single/double stranded)
  – RNA can catalyze RNA replication

• RNA world is possible
  – Proteins are more efficient (larger alphabet)
  – DNA is more stable (double helix, less flexible)

RNA structure

Comparative methods for RNA structure prediction
Multiple alignment and RNA folding
Given K homologous aligned RNA sequences:

Human: aagacuucggaucuggcgacaccc
Mouse: uacacuucggaugacaccaaagug
Worm: aggucuucggcacgggcaccauuc
Fly: ccaacuucggauuuugcuaccaua
Yeast: aagccuucggagcgggcguaacuc

If i\textsuperscript{th} and j\textsuperscript{th} positions are always base paired and covary, then they are likely to be paired.

Mutual information

\[ M_{ij} = \sum_{a,b} \frac{f_{ab}(i,j)}{f_a(i)f_b(j)} \log_2 \frac{f_{ab}(i,j)}{f_a(i)f_b(j)} \]

Where \( f_{ab}(i,j) \) is the number of times the pair a, b are in positions i, j.

Given a multiple alignment, can infer structure that maximizes the sum of mutual information, by DP.

In practice:
1. Get multiple alignment
2. Find covarying bases – deduce structure
3. Improve multiple alignment (by hand)
4. Go to 2

A manual EM process!!

Results for tRNA

Matrix of co-variations in tRNA molecule

Context Free Grammars
(review)

A Context Free Grammar

<table>
<thead>
<tr>
<th>Grammar Rule</th>
<th>Nonterminals:</th>
<th>Terminals:</th>
</tr>
</thead>
<tbody>
<tr>
<td>S \rightarrow AB</td>
<td>S, A, B</td>
<td>a, b, c, d</td>
</tr>
<tr>
<td>A \rightarrow aAc</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>B \rightarrow bBd</td>
<td>b</td>
<td></td>
</tr>
</tbody>
</table>

Derivation:

S \rightarrow AB \rightarrow aAcB \rightarrow aaaaacccB \rightarrow aaaaacccbBd \rightarrow ... 

Produces all strings \( a^i cb^n d^j \), for i, j \geq 0

Example: modeling a stem loop

S \rightarrow a W_1 u \nW_1 \rightarrow c W_2 g \nW_2 \rightarrow g W_3 c \nW_3 \rightarrow g L c \nL \rightarrow agucg

What if the stem loop can have other letters in place of the ones shown?
Example: modeling a stem loop

\[ S \rightarrow a W_1 u \mid g W_1 u \]
\[ W_1 \rightarrow c W_2 g \]
\[ W_2 \rightarrow g W_3 c \mid a L u \]
\[ L \rightarrow agucg \mid aLc \mid cLu \mid uu \mid aaa \mid \ldots \]

More general: Any 4-long stem, 3-5-long loop:

\[ S \rightarrow aW_1u \mid gW_1u \mid gW_1c \mid cW_1g \mid uW_1g \mid uW_1a \]
\[ W_1 \rightarrow aW_2u \mid gW_2u \mid gW_2c \mid cW_2g \mid uW_2g \mid uW_2a \]
\[ W_2 \rightarrow aW_3u \mid gW_3u \mid gW_3c \mid cW_3g \mid uW_3g \mid uW_3a \]
\[ L \rightarrow al \mid cl \mid gl \mid ul \mid uu \mid aaa \mid \ldots \]

Alignment scores for parses

We can define each rule \( X \rightarrow s \), where \( s \) is a string, to have a score.

**Example:**

\( W \rightarrow a W' u: 3 \) (forms 3 hydrogen bonds)
\( W \rightarrow g W' c: 2 \) (forms 2 hydrogen bonds)
\( W \rightarrow g W' u: 1 \) (forms 1 hydrogen bond)
\( W \rightarrow x W' z -1, \) when \((x, z)\) is not an \(a/u, g/c, g/u\) pair

**Questions:**
- How do we best align a CFG to a sequence? (DP)
- How do we set the parameters? (Stochastic CFGs)

Reformulating the Nussinov Algorithm

**Initialization:**

\[
F(i, j+1) = 0; \quad \text{for } i = 2 \text{ to } N
\]
\[
F(i, j) = 0; \quad \text{for } i = 1 \text{ to } N
\]

**Iteration:**

\[
F(i, j) = \max_{1 \leq k < j} F(i, k) + F(k+1, j)
\]

**Termination:**

Best structure is given by \( F(1, N) \)

Stochastic Context Free Grammars

Define the following grammar, with scores:

\[
S \rightarrow a S u : 3 \mid u S a : 3
\]
\[
g S c : 2 \mid c S g : 2
\]
\[
g S u : 1 \mid u S g : 1
\]

\[
S S : 0
\]

\[
S \rightarrow a \mid c \mid g \mid u
\]

Note: \( \varepsilon \) is the "\(\)" string

Then, the Nussinov algorithm finds the optimal parse of a string with this grammar
Stochastic Context Free Grammars

- In an analogy to HMMs, we can assign probabilities to transitions:
- Given grammar

\[ X_i \rightarrow s_{i1} | \ldots | s_{in} \]

\[ \ldots \]

\[ X_m \rightarrow s_{m1} | \ldots | s_{mn} \]

- Can assign probability to each rule, s.t.

\[ P(X_i \rightarrow s_{i1}) + \ldots + P(X_i \rightarrow s_{in}) = 1 \]

Scoring

- We can find best folding
- E.g.,

\[ P(S \rightarrow u Sa) = 0.2 \]

\[ P(S \rightarrow c Sa) = 0.02 \]

\[ P(S \rightarrow S a) = 0.01 \]

\[ \ldots \]

Decoding: the CYK algorithm

- Given: \( x = x_1 \ldots x_N \) and a SCFG \( G \),

- Goal: find the most likely parsing of \( x \) according to \( G \)

- Dynamic programming variable:

\[ \gamma(i, j, V) = \text{probability that } x_i \ldots x_j \text{ can be generated from nonterminal } V \]

The CYK algorithm (Cocke-Younger-Kasami)

Initialization:
- For \( i = 1 \) to \( N \), any nonterminal \( V \),

\[ \gamma(i, i, V) = P(V \rightarrow x_i) \]

Iteration:
- For \( i = 1 \) to \( N-1 \)
- For \( j = i+1 \) to \( N \)
- For any nonterminal \( V \),

\[ \gamma(i, j, V) = \max_{X,Y} \max_{k<i} \gamma(i,k,X) \cdot \gamma(k+1,j,Y) \cdot P(V \rightarrow XY) \]

Before:

\[ \gamma(i, j, V) = \max_{X,Y} \max_{k<i} \gamma(i,k,X) \cdot \gamma(k+1,j,Y) \]

Or:

\[ \log \gamma(i, j, V) = \max_{X,Y} \max_{k<i} \log \gamma(i,k,X) + \log \gamma(k+1,j,Y) + \log P(V \rightarrow XY) \]

As in Nussinov’s algorithm!

Computational Problems

- Calculate an optimal alignment of a sequence and a SCFG

  (DECODING)

- Calculate Prob[ sequence | grammar ]

  (EVALUATION)

- Given a set of sequences, estimate parameters of a SCFG

  (LEARNING)

Normal Forms for CFGs

Chomsky Normal Form:

\[ X \rightarrow YZ \]

\[ X \rightarrow a \]

All productions are either to 2 nonterminals, or to 1 terminal

Theorem (technical)

Every CFG has an equivalent one in Chomsky Normal Form

(That is, the grammar in normal form produces exactly the same set of strings)
Example of converting a CFG to C.N.F.

\[ S \rightarrow ABC \]
\[ A \rightarrow Aa \mid a \]
\[ B \rightarrow Bb \mid b \]
\[ C \rightarrow CAc \mid c \]

Converting:

\[ S \rightarrow AS' \]
\[ S' \rightarrow BC \]
\[ A \rightarrow AA \mid a \]
\[ B \rightarrow BB \mid b \]
\[ C \rightarrow DC' \mid c \]
\[ C' \rightarrow c \]
\[ D \rightarrow CA \]

Another example

\[ S \rightarrow ABC \]
\[ A \rightarrow C \mid aA \]
\[ B \rightarrow bB \mid b \]
\[ C \rightarrow cCd \mid c \]

Converting:

\[ S \rightarrow AS' \]
\[ S' \rightarrow BC \]
\[ A \rightarrow C'C'' \mid c \mid A'A \]
\[ A' \rightarrow a \]
\[ B \rightarrow B'B \mid b \]
\[ B' \rightarrow b \]
\[ C \rightarrow C'C'' \mid c \]
\[ C' \rightarrow c \]
\[ C'' \rightarrow CD \]
\[ D \rightarrow d \]

Decoding: the CYK algorithm

Given \( x = x_1 \ldots x_N \), and a SCFG \( G \),
Find the most likely parse of \( x \)
(the most likely alignment of \( G \) to \( x \))

Dynamic programming variable:
\[ \gamma(i, j, V): \text{likelihood of the most likely parse of } x_i \ldots x_j, \text{ rooted at nonterminal } V \]

Then,
\[ \gamma(1, N, S): \text{likelihood of the most likely parse of } x \text{ by the grammar} \]

The CYK algorithm (Cocke-Younger-Kasami)

Initialization:
For \( i = 1 \) to \( N \), any nonterminal \( V \),
\[ \gamma(i, i, V) = \log P(V \rightarrow x_i) \]

Iteration:
For \( j = i \) to \( N-1 \)
For any nonterminal \( V \),
\[ \gamma(i, j, V) = \max_x \max_{x'} \left( \gamma(i, k, X) + \gamma(k+1, j, Y) + \log P(V \rightarrow XY) \right) \]

Termination:
\[ \log P(x \mid \theta, \pi^*) = \gamma(1, N, S) \]

Where \( \pi^* \) is the optimal parse tree (if traced back appropriately from above)

A SCFG for predicting RNA structure

\[ S \rightarrow aS \mid cS \mid gS \mid uS \mid \varepsilon \]
\[ \rightarrow Sa \mid Sc \mid Sg \mid Su \]
\[ \rightarrow aSu \mid cSg \mid gSu \mid uSg \mid gSc \mid uSa \]
\[ \rightarrow SS \]

• Adjust the probability parameters to reflect bond strength etc
• No distinction between non-paired bases, bulges, loops
• Can modify to model these events
  – L: loop nonterminal
  – H: hairpin nonterminal
  – B: bulge nonterminal
  – etc
**CYK for RNA folding**

**Initialization:**
\[ \gamma(i, i-1) = \log P(\varepsilon) \]

**Iteration:**
For \( i = 1 \) to \( N \)
  For \( j = i \) to \( N \)
  \[ \gamma(i+1, j-1) + \log P(x_i S x_j) \]
  \[ \gamma(i, j-1) + \log P(S x_j) \]
  \[ \gamma(i, j) = \max \left\{ \gamma(i+1, j) + \log P(x_i S) \right\} \]
  \[ \max_{i < k < j} \left\{ \gamma(i, k) + \gamma(k+1, j) + \log P(S S) \right\} \]

**Evaluation**

Recall HMMs:
- **Forward:** \( f_l(i) = P(x_1 \ldots x_i, \pi_i = l) \)
- **Backward:** \( b_k(i) = P(x_{i+1} \ldots x_N | \pi_i = k) \)

Then,
\[ P(x) = \sum_l a_0 l \cdot e_l(x_1) \cdot b_1(1) \]

Analogue in SCFGs:
- **Inside:** \( a(i, j, V) = P(x_i \ldots x_j \text{ is generated by nonterminal } V) \)
- **Outside:** \( b(i, j, V) = P(x, \text{ excluding } x_i \ldots x_j \text{ is generated by } S \text{ and the excluded part is rooted at } V) \)

**The Inside Algorithm**

To compute
\[ a(i, j, V) = P(x_i \ldots x_j \text{ produced by } V) \]

\[ a(i, j, V) = \sum_X \sum_Y \sum_k a(i, k, X) \cdot a(k+1, j, Y) \cdot P(V \rightarrow XY) \]

**Algorithm: Inside**

**Initialization:**
For \( i = 1 \) to \( N \), \( V \) a nonterminal,
\[ a(i, i, V) = P(V \rightarrow x_i) \]

**Iteration:**
For \( i = 1 \) to \( N-1 \)
  For \( j = i+1 \) to \( N \)
    For \( V \) a nonterminal
    \[ a(i, j, V) = \sum_X \sum_Y \sum_k a(k, i-1, X) \cdot b(k, j, Y) \cdot P(Y \rightarrow X V) + \]
    \[ \sum_X \sum_Y \sum_k a(k+1, j, X) \cdot b(i, k, Y) \cdot P(Y \rightarrow V X) \]

**Termination:**
\[ P(x | \theta) = a(1, N, S) \]

**The Outside Algorithm**

\[ b(i, j, V) = \text{Prob}(x_i \ldots x_k, x_{i+1} \ldots x_j \text{ where the "gap" is rooted at } V) \]

Given that \( V \) is the right-hand-side nonterminal of a production,
\[ b(i, j, V) = \sum_X \sum_Y \sum_k a(i, k, X) \cdot b(k, j, Y) \cdot P(Y \rightarrow XV) \]

**Algorithm: Outside**

**Initialization:**
\[ b(1, N, S) = 1 \]
  For any other \( V \), \( b(1, N, V) = 0 \)

**Iteration:**
For \( i = 1 \) to \( N-1 \)
  For \( j = N \) down to \( i \)
    For \( V \) a nonterminal
    \[ b(i, j, V) = \sum_X \sum_Y \sum_k a(k, i-1, X) \cdot b(k, j, Y) \cdot P(Y \rightarrow XV) + \]
    \[ \sum_X \sum_Y \sum_k a(k+1, j, X) \cdot b(i, k, Y) \cdot P(Y \rightarrow VX) \]

**Termination:**
It is true for any \( i \), that:
\[ P(x | \theta) = \sum b(i, i, X) \cdot P(X \rightarrow x) \]
Learning for SCFGs

We can now estimate
\[ c(V) = \text{expected number of times } V \text{ is used in the parse of } x_1 \ldots x_N \]
\[ c(V) = \frac{1}{P(x | \theta)} \sum_{i \leq j \leq N} a(i, j, V) b(i, j, v) \]

\[ c(V \rightarrow XY) = \frac{1}{P(x | \theta)} \sum_{i \leq k \leq N} \sum_{i < j} b(i, j, V) \sum_{i \leq k < j} a(i, k, X) a(k+1, j, Y) P(V \rightarrow XY) \]

Learning for SCFGs

Then, we can re-estimate the parameters with EM, by:

\[ c(V \rightarrow a) = \frac{\sum_{i \leq j \leq N} a(i, j, V) b(i, j, V)}{c(V)} \]

\[ P_{\text{new}}(V \rightarrow a) = \frac{\sum_{i \leq j \leq N} a(i, j, V) b(i, j, V)}{c(V \rightarrow a) \sum_{i \leq j \leq N} a(i, j, V)} \]

Summary: SCFG and HMM algorithms

<table>
<thead>
<tr>
<th>GOAL</th>
<th>HMM algorithm</th>
<th>SCFG algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal parse</td>
<td>Viterbi</td>
<td>CYK</td>
</tr>
<tr>
<td>Estimation</td>
<td>Forward</td>
<td>Inside</td>
</tr>
<tr>
<td></td>
<td>Backward</td>
<td>Outside</td>
</tr>
<tr>
<td>Learning</td>
<td>EM: FwBck</td>
<td>EM: In/Outs</td>
</tr>
<tr>
<td>Memory Complexity</td>
<td>O(N K)</td>
<td>O(N*K)</td>
</tr>
<tr>
<td>Time Complexity</td>
<td>O(N K)</td>
<td>O(N*K^2)</td>
</tr>
</tbody>
</table>

Where K: # of states in the HMM
# of nonterminals in the SCFG