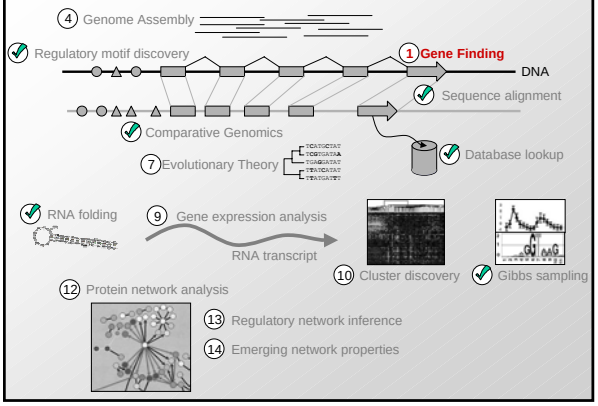
6.096 – Algorithms for Computational Biology – Lecture 7

Gene Finding and HMMs

Lecture 1 - Introduction
Lecture 2 - Hashing and BLAST
Lecture 3 - Combinatorial Motif Finding
Lecture 4 - Statistical Motif Finding
Lecture 5 - Sequence alignment and Dynamic Programming
Lecture 6 - RNA structure and Context Free Grammars
Lecture 7 - Gene finding and Hidden Markov Models

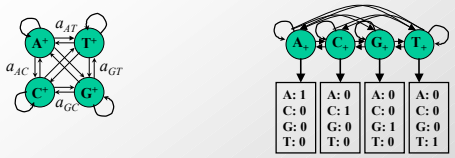
Challenges in Computational Biology



Outline

- Computational model
  - Simple Markov Models
  - Hidden Markov Models
- Working with HMMs
  - Dynamic programming (Viterbi)
  - Expectation maximization (Baum-Welch)
- Gene Finding in practice
  - GENSCAN
  - Performance Evaluation

Markov Chains & Hidden Markov Models



- Markov Chain
  - Q: states
  - p: initial state probabilities
  - A: transition probabilities
- HMM
  - Q: states
  - V: observations
  - p: initial state probabilities
  - A: transition probabilities
  - E: emission probabilities

Markov Chain

Definition: A ***Markov chain*** is a triplet **(*Q, p, A*)**, where:

- ***Q*** is a finite set of states. Each state corresponds to a symbol in the alphabet Σ
- ***p*** is the initial state probabilities.
- ***A*** is the state transition probabilities, denoted by a_{st} for each ***s, t*** in ***Q***.
- For each ***s, t*** in ***Q*** the transition probability is: $a_{st} \equiv P(x_i = t | x_{i-1} = s)$

Output: The output of the model is the set of states at each instant time => the set of states are observable

Property: The probability of each symbol x_i depends only on the value of the preceding symbol x_{i-1} : $P(x_i | x_{i-1}, \ldots, x_1) = P(x_i | x_{i-1})$

Formula: The probability of the sequence:

$$P(x) = P(x_L, x_{L-1}, \ldots, x_1) = P(x_L | x_{L-1}) P(x_{L-1} | x_{L-2}) \ldots P(x_2 | x_1) P(x_1)$$

HMM (Hidden Markov Model)

Definition: An ***HMM*** is a 5-tuple **(*Q, V, p, A, E*)**, where:

- ***Q*** is a finite set of states, |**Q**|=**N**
- **V** is a finite set of observation symbols per state, |**V**|=**M**
- ***p*** is the initial state probabilities.
- ***A*** is the state transition probabilities, denoted by a_{st} for each ***s, t*** in ***Q***.
  - For each ***s, t*** in ***Q*** the transition probability is: $a_{st} \equiv P(x_i = t | x_{i-1} = s)$
- **E** is a probability emission matrix, $e_{sk} \equiv P(v_k \text{ at time } t \mid q_t = s)$

Output: Only emitted symbols are observable by the system but not the underlying random walk between states -> "hidden"

Property: Emissions and transitions are dependent on the current state only and not on the past.

Typical HMM Problems

Annotation Given a model *M* and an observed string *S*, what is the most probable path through *M* generating *S*

Classification Given a model *M* and an observed string *S*, what is the total probability of *S* under *M*

Consensus Given a model *M*, what is the string having the highest probability under *M*

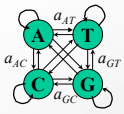
Training Given a set of strings and a model structure, find transition and emission probabilities assigning high probabilities to the strings

Example 1: Finding CpG islands

What are CpG islands?

- Regions of regulatory importance in promoters of many genes
  - Defined by their methylation state (epigenetic information)
- Methylation process in the human genome:
  - Very high chance of methyl-C mutating to T in CpG
    - ➔ CpG dinucleotides are much rarer
  - BUT it is suppressed around the promoters of many genes
    - ➔ CpG dinucleotides are much more frequent than elsewhere
      - Such regions are called **CpG islands**
      - A few hundred to a few thousand bases long
- Problems:
  - Given a short sequence, does it come from a CpG island or not?
  - How to find the CpG islands in a long sequence

Training Markov Chains for CpG islands



- Training Set:
  - set of DNA sequences w/ known CpG islands
- Derive two Markov chain models:
  - **'+' model**: from the CpG islands
  - **'-' model**: from the remainder of sequence
- Transition probabilities for each model:

Probability of C following A

+	A	C	G	T
A	.180	.274	.426	.120
C	.171	.368	.274	.188
G	.161	.339	.375	.125
T	.079	.355	.384	.182

$$a^+_{st} = \frac{c^+_{st}}{\sum_{t'} c^+_{st'}}$$

c^+_{st} is the number of times letter ***t*** followed letter ***s*** inside the CpG islands

$$a^-_{st} = \frac{c^-_{st}}{\sum_{t'} c^-_{st'}}$$

c^-_{st} is the number of times letter ***t*** followed letter ***s*** outside the CpG islands

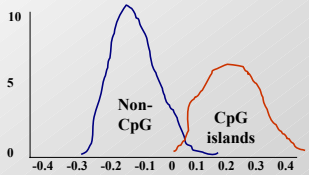
Using Markov Models for CpG classification

Q1: Given a short sequence *x*, does it come from CpG island (**Yes-No** question)

- To use these models for discrimination, calculate the log-odds ratio:

$$S(x) = \log \frac{P(x|\text{model} +)}{P(x|\text{model} -)} = \sum_{i=1}^{L} \log \frac{a^+_{x_{i-1}x_i}}{a^-_{x_{i-1}x_i}}$$

Histogram of log odds scores



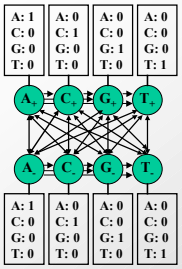
Using Markov Models for CpG classification

Q2: Given a long sequence *x*, how do we find CpG islands in it (**Where** question)

- Calculate the log-odds score for a window of, say, 100 nucleotides around every nucleotide, plot it, and predict CpG islands as ones w/ positive values
- Drawbacks: Window size

Use a hidden state: CpG (+) or non-CpG (-)

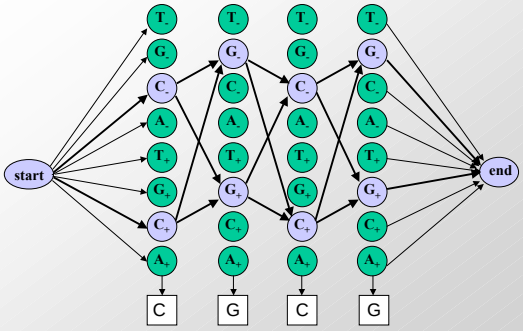
HMM for CpG islands



- Build a single model that combines both Markov chains:
  - **'+' states**: A_+, C_+, G_+, T_+
    - Emit symbols: A, C, G, T in CpG islands
  - **'-' states**: A_-, C_-, G_-, T_-
    - Emit symbols: A, C, G, T in non-islands
- Emission probabilities distinct for the '+' and the '-' states
  - Infer most likely set of states, giving rise to observed emissions

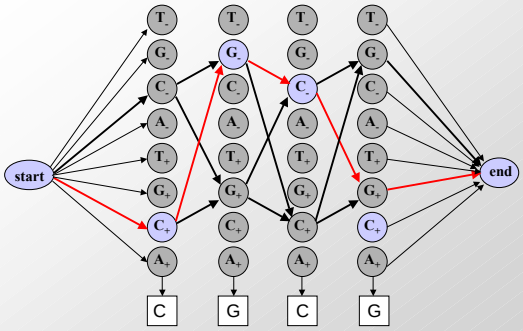
➔ 'Paint' the sequence with + and - states

Finding most likely state path



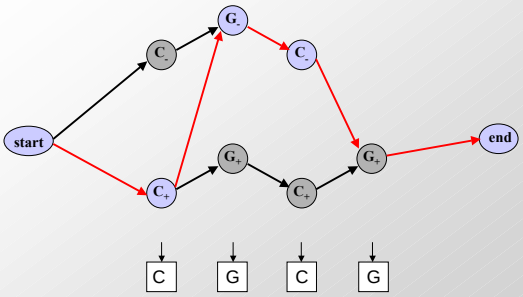
- Given the observed emissions, what was the path?

Probability of given path p & observations x



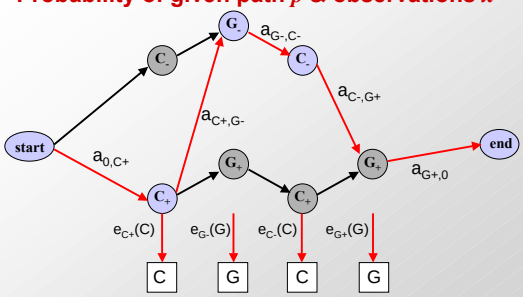
- Known observations: CGCG
- Known sequence path: C+, G-, C-, G+

Probability of given path p & observations x



- Known observations: CGCG
- Known sequence path: C+, G-, C-, G+

Probability of given path p & observations x



- $P(\boldsymbol{p},\boldsymbol{x}) = (a_{0,C+} * 1) * (a_{C+,G-} * 1) * (a_{G-,C-} * 1) * (a_{C-,G+} * 1) * (a_{G+,0})$

But in general, we don't know the path!

The three main questions on HMMs

1. Evaluation

   GIVEN a HMM M, and a sequence x,
   FIND Prob[x | M]

2. Decoding

   GIVEN a HMM M, and a sequence x,
   FIND the sequence π of states that maximizes P[x, π | M]

3. Learning

   GIVEN a HMM M, with unspecified transition/emission probs., and a sequence x,

   FIND parameters $\theta = (e_i(.), a_{ij})$ that maximize P[x | θ]

Problem 1: Decoding

Find the best parse of a sequence

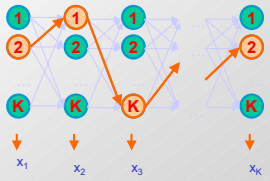
Decoding

GIVEN $x = x_1 x_2 \ldots\ldots x_N$

We want to find $\pi = \pi_1, \ldots\ldots, \pi_N$, such that $P[x, \pi]$ is maximized

$\pi^* = \text{argmax}_\pi P[x, \pi]$

We can use dynamic programming!



Let $V_k(i) = \max_{\{\pi_1,\ldots,\pi_{i-1}\}} P[x_1 \ldots x_{i-1}, \pi_1, \ldots, \pi_{i-1}, x_i, \pi_i = k]$

= Probability of most likely sequence of states ending at state $\pi_i = k$

Decoding – main idea

Given that for all states k, and for a fixed position i,

$V_k(i) = \max_{\{\pi_1,\ldots,\pi_{i-1}\}} P[x_1 \ldots x_{i-1}, \pi_1, \ldots, \pi_{i-1}, x_i, \pi_i = k]$

What is $V_k(i+1)$?

From definition,

$V_l(i+1) = \max_{\{\pi_1,\ldots,\pi_i\}} P[x_1 \ldots x_i, \pi_1, \ldots, \pi_i, x_{i+1}, \pi_{i+1} = l]$

$= \max_{\{\pi_1,\ldots,\pi_i\}} P(x_{i+1}, \pi_{i+1} = l \mid x_1 \ldots x_i, \pi_1, \ldots, \pi_i)\, P[x_1 \ldots x_i, \pi_1, \ldots, \pi_i]$

$= \max_{\{\pi_1,\ldots,\pi_i\}} P(x_{i+1}, \pi_{i+1} = l \mid \pi_i)\, P[x_1 \ldots x_{i-1}, \pi_1, \ldots, \pi_{i-1}, x_i, \pi_i]$

$= \max_k P(x_{i+1}, \pi_{i+1} = l \mid \pi_i = k) \max_{\{\pi_1,\ldots,\pi_{i-1}\}} P[x_1 \ldots x_{i-1}, \pi_1, \ldots, \pi_{i-1}, x_i, \pi_i = k]$

$= e_l(x_{i+1}) \max_k a_{kl} V_k(i)$

The Viterbi Algorithm

Input: $x = x_1 \ldots\ldots x_N$

Initialization:

$V_0(0) = 1$ (0 is the imaginary first position)

$V_k(0) = 0$, for all $k > 0$

Iteration:

$V_j(i) = e_j(x_i) \times \max_k a_{kj} V_k(i-1)$

$\text{Ptr}_j(i) = \text{argmax}_k a_{kj} V_k(i-1)$

Termination:

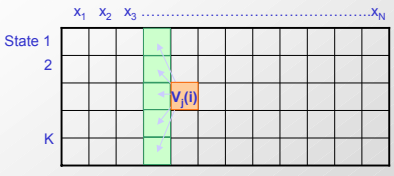
$P(x, \pi^*) = \max_k V_k(N)$

Traceback:

$\pi_N^* = \text{argmax}_k V_k(N)$

$\pi_{i-1}^* = \text{Ptr}_{\pi i}(i)$

The Viterbi Algorithm



Similar to "aligning" a set of states to a sequence

Time:

$O(K^2 N)$

Space:

$O(KN)$

Viterbi Algorithm – a practical detail

Underflows are a significant problem

$P[x_1, \ldots, x_i, \pi_1, \ldots, \pi_i] = a_{0\pi 1} a_{\pi 1 \pi 2} \ldots\ldots a_{\pi i} e_{\pi 1}(x_1) \ldots\ldots e_{\pi i}(x_i)$

These numbers become extremely small – underflow

Solution: Take the logs of all values

$V_l(i) = \log e_k(x_i) + \max_k [V_k(i-1) + \log a_{kl}]$

Example

Let x be a sequence with a portion of ~ 1/6 6's, followed by a portion of ~ ½ 6's…

x = 123456123456…12345 6626364656…1626364656

Then, it is not hard to show that optimal parse is (exercise):

FFF……..F LLL……..L

6 nucleotides "123456" parsed as F, contribute $.95^6 \times (1/6)^6 = 1.6 \times 10^{-5}$
parsed as L, contribute $.95^6 \times (1/2)^1 \times (1/10)^5 = 0.4 \times 10^{-5}$

"162636" parsed as F, contribute $.95^6 \times (1/6)^6 = 1.6 \times 10^{-5}$
parsed as L, contribute $.95^6 \times (1/2)^3 \times (1/10)^3 = 9.0 \times 10^{-5}$

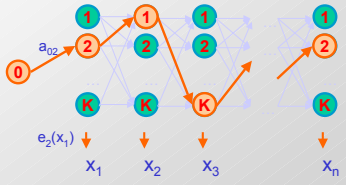
Problem 2: Evaluation

Find the likelihood a sequence is generated by the model

Generating a sequence by the model

Given a HMM, we can generate a sequence of length n as follows:

1. Start at state π_1 according to prob $a_{0\pi 1}$
2. Emit letter x_1 according to prob $e_{\pi 1}(x_1)$
3. Go to state π_2 according to prob $a_{\pi 1 \pi 2}$
4. … until emitting x_n



A couple of questions

Given a sequence x,

- What is the probability that x was generated by the model?
- Given a position i, what is the most likely state that emitted x_i?

Example: the dishonest casino

Say x = 12341623162616364616234161221341

Most likely path: π = FF……F
However: marked letters more likely to be L than unmarked letters

Evaluation

We will develop algorithms that allow us to compute:

P(x)	Probability of x given the model
$P(x_i \ldots x_j)$	Probability of a substring of x given the model
$P(\pi_i = k \mid x)$	Probability that the i^{th} state is k, given x

A more refined measure of which states x may be in

The Forward Algorithm

We want to calculate

P(x) = probability of x, given the HMM

Sum over all possible ways of generating x:

$$P(x) = \Sigma_\pi P(x, \pi) = \Sigma_\pi P(x \mid \pi) P(\pi)$$

To avoid summing over an exponential number of paths π, define

$$f_k(i) = P(x_1 \ldots x_i, \pi_i = k) \text{ (the forward probability)}$$

The Forward Algorithm – derivation

Define the forward probability:

$f_l(i) = P(x_1 \ldots x_i, \pi_i = l)$

$= \Sigma_{\pi 1 \ldots \pi i-1} P(x_1 \ldots x_{i-1}, \pi_1, \ldots, \pi_{i-1}, \pi_i = l)\, e_l(x_i)$

$= \Sigma_k \Sigma_{\pi 1 \ldots \pi i-2} P(x_1 \ldots x_{i-1}, \pi_1, \ldots, \pi_{i-2}, \pi_{i-1} = k)\, a_{kl}\, e_l(x_i)$

$= e_l(x_i) \Sigma_k f_k(i-1)\, a_{kl}$

The Forward Algorithm

We can compute $f_k(i)$ for all k, i, using dynamic programming!

Initialization:

$f_0(0) = 1$

$f_k(0) = 0$, for all $k > 0$

Iteration:

$f_l(i) = e_l(x_i) \Sigma_k f_k(i-1)\, a_{kl}$

Termination:

$P(x) = \Sigma_k f_k(N)\, a_{k0}$

Where, a_{k0} is the probability that the terminating state is k (usually = a_{0k})

Relation between Forward and Viterbi

VITERBI

Initialization:

$V_0(0) = 1$

$V_k(0) = 0$, for all $k > 0$

Iteration:

$V_j(i) = e_j(x_i)\ \max_k V_k(i-1)\, a_{kj}$

Termination:

$P(x, \pi^*) = \max_k V_k(N)$

FORWARD

Initialization:

$f_0(0) = 1$

$f_k(0) = 0$, for all $k > 0$

Iteration:

$f_l(i) = e_l(x_i) \Sigma_k f_k(i-1)\, a_{kl}$

Termination:

$P(x) = \Sigma_k f_k(N)\, a_{k0}$

Motivation for the Backward Algorithm

We want to compute

$P(\pi_i = k \mid x)$,

the probability distribution on the i^{th} position, given x

We start by computing

$P(\pi_i = k, x) = P(x_1 \ldots x_i, \pi_i = k, x_{i+1} \ldots x_N)$

$= P(x_1 \ldots x_i, \pi_i = k)\, P(x_{i+1} \ldots x_N \mid x_1 \ldots x_i, \pi_i = k)$

$= P(x_1 \ldots x_i, \pi_i = k)\, P(x_{i+1} \ldots x_N \mid \pi_i = k)$

Forward, $f_k(i)$ Backward, $b_k(i)$

The Backward Algorithm – derivation

Define the backward probability:

$b_k(i) = P(x_{i+1} \ldots x_N \mid \pi_i = k)$

$= \Sigma_{\pi i+1 \ldots \pi N} P(x_{i+1}, x_{i+2}, \ldots, x_N, \pi_{i+1}, \ldots, \pi_N \mid \pi_i = k)$

$= \Sigma_l \Sigma_{\pi i+1 \ldots \pi N} P(x_{i+1}, x_{i+2}, \ldots, x_N, \pi_{i+1} = l, \pi_{i+2}, \ldots, \pi_N \mid \pi_i = k)$

$= \Sigma_l e_l(x_{i+1})\, a_{kl}\, \Sigma_{\pi i+1 \ldots \pi N} P(x_{i+2}, \ldots, x_N, \pi_{i+2}, \ldots, \pi_N \mid \pi_{i+1} = l)$

$= \Sigma_l e_l(x_{i+1})\, a_{kl}\, b_l(i+1)$

The Backward Algorithm

We can compute $b_k(i)$ for all k, i, using dynamic programming

Initialization:

$b_k(N) = a_{k0}$, for all k

Iteration:

$b_k(i) = \Sigma_l e_l(x_{i+1})\, a_{kl}\, b_l(i+1)$

Termination:

$P(x) = \Sigma_l a_{0l}\, e_l(x_1)\, b_l(1)$

Computational Complexity

What is the running time, and space required, for Forward, and Backward?

Time: $O(K^2N)$
Space: $O(KN)$

Useful implementation technique to avoid underflows

Viterbi: sum of logs
Forward/Backward: rescaling at each position by multiplying by a constant

Posterior Decoding

We can now calculate

$$P(\pi_i = k \mid x) = \frac{f_k(i)\, b_k(i)}{P(x)}$$

Then, we can ask

What is the most likely state at position i of sequence x:

Define $\pi^\wedge$ by Posterior Decoding:

$$\pi^\wedge_i = \text{argmax}_k\, P(\pi_i = k \mid x)$$

Posterior Decoding

- For each state,
  - Posterior Decoding gives us a curve of likelihood of state for each position
  - That is sometimes more informative than Viterbi path π^*
- Posterior Decoding may give an invalid sequence of states
  - Why?

Maximum Weight Trace

- Another approach is to find a sequence of states under some constraint, and maximizing expected accuracy of state assignments
  - $A_j(i) = \max_{k \text{ such that Condition}(k, j)} A_k(i\text{-}1) + P(\pi_i = j \mid x)$
- We will revisit this notion again

Problem 3: Learning

Re-estimate the parameters of the model based on training data

Two learning scenarios

1. Estimation when the "right answer" is known

   Examples:
   GIVEN: a genomic region $x = x_1 \ldots x_{1,000,000}$ where we have good (experimental) annotations of the CpG islands

   GIVEN: the casino player allows us to observe him one evening, as he changes dice and produces 10,000 rolls

2. Estimation when the "right answer" is unknown

   Examples:
   GIVEN: the porcupine genome; we don't know how frequent are the CpG islands there, neither do we know their composition

   GIVEN: 10,000 rolls of the casino player, but we don't see when he changes dice

QUESTION: Update the parameters θ of the model to maximize $P(x|\theta)$

Case 1. When the right answer is known

Given $x = x_1 \ldots x_N$
for which the true $\pi = \pi_1 \ldots \pi_N$ is known,

Define:

A_{kl} = # times $k \to l$ transition occurs in π
$E_k(b)$ = # times state k in π emits b in x

We can show that the maximum likelihood parameters θ are:

$$a_{kl} = \frac{A_{kl}}{\Sigma_i A_{ki}} \qquad e_k(b) = \frac{E_k(b)}{\Sigma_c E_k(c)}$$

Case 1. When the right answer is known

Intuition: When we know the underlying states,
Best estimate is the average frequency of
transitions & emissions that occur in the training data

Drawback:
Given little data, there may be **overfitting**:
$P(x|\theta)$ is maximized, but θ is unreasonable
0 probabilities – VERY BAD

Example:
Given 10 casino rolls, we observe

```
x = 2, 1, 5, 6, 1, 2, 3, 6, 2, 3
π = F, F, F, F, F, F, F, F, F, F
```

Then:
$a_{FF} = 1$; $a_{FL} = 0$
$e_F(1) = e_F(3) = .2$;
$e_F(2) = .3$; $e_F(4) = 0$; $e_F(5) = e_F(6) = .1$

Pseudocounts

Solution for small training sets:

Add pseudocounts

A_{kl} = # times $k \to l$ transition occurs in π + r_{kl}
$E_k(b)$ = # times state k in π emits b in x + $r_k(b)$

r_{kl}, $r_k(b)$ are pseudocounts representing our prior belief

Larger pseudocounts $\Rightarrow$ Strong priof belief

Small pseudocounts ($\varepsilon < 1$): just to avoid 0 probabilities

Pseudocounts

Example: dishonest casino

We will observe player for one day, 500 rolls

Reasonable pseudocounts:

$r_{0F} = r_{0L} = r_{F0} = r_{L0} = 1$;
$r_{FL} = r_{LF} = r_{FF} = r_{LL} = 1$;
$r_F(1) = r_F(2) = \ldots = r_F(6) = 20$ (strong belief fair is fair)
$r_F(1) = r_F(2) = \ldots = r_F(6) = 5$ (wait and see for loaded)

Above #s pretty arbitrary – assigning priors is an art

Case 2. When the right answer is unknown

We don't know the true A_{kl}, $E_k(b)$

Idea:

- We estimate our "best guess" on what A_{kl}, $E_k(b)$ are
- We update the parameters of the model, based on our guess
- We repeat

Case 2. When the right answer is unknown

Starting with our best guess of a model M, parameters θ:

Given $x = x_1 \ldots x_N$
for which the true $\pi = \pi_1 \ldots \pi_N$ is unknown,

We can get to a provably more likely parameter set θ

Principle: EXPECTATION MAXIMIZATION

1. Estimate A_{kl}, $E_k(b)$ in the training data
2. Update θ according to A_{kl}, $E_k(b)$
3. Repeat 1 & 2, until convergence

Estimating new parameters

To estimate A_{kl}:

At each position i of sequence x,

Find probability transition k→l is used:

$P(\pi_i = k, \pi_{i+1} = l \mid x) = [1/P(x)] \times P(\pi_i = k, \pi_{i+1} = l, x_1 \ldots x_N) = Q/P(x)$

where $Q = P(x_1 \ldots x_i, \pi_i = k, \pi_{i+1} = l, x_{i+1} \ldots x_N) =$
$= P(\pi_{i+1} = l, x_{i+1} \ldots x_N \mid \pi_i = k)\, P(x_1 \ldots x_i, \pi_i = k) =$
$= P(\pi_{i+1} = l, x_{i+1} x_{i+2} \ldots x_N \mid \pi_i = k)\, f_k(i) =$
$= P(x_{i+2} \ldots x_N \mid \pi_{i+1} = l)\, P(x_{i+1} \mid \pi_{i+1} = l)\, P(\pi_{i+1} = l \mid \pi_i = k)\, f_k(i) =$
$= b_l(i+1)\, e_l(x_{i+1})\, a_{kl}\, f_k(i)$

So: $$\mathbf{P(\pi_i = k, \pi_{i+1} = l \mid x, \theta) = \frac{f_k(i)\, a_{kl}\, e_l(x_{i+1})\, b_l(i+1)}{P(x \mid \theta)}}$$

Estimating new parameters

So,

$$A_{kl} = \sum_i P(\pi_i = k, \pi_{i+1} = l \mid x, \theta) = \sum_i \frac{f_k(i)\, a_{kl}\, e_l(x_{i+1})\, b_l(i+1)}{P(x \mid \theta)}$$

Similarly,

$$E_k(b) = [1/P(x)] \sum_{\{i \mid x_i = b\}} f_k(i)\, b_k(i)$$

Estimating new parameters

If we have several training sequences, $x^1, \ldots, x^M$, each of length N,

$$A_{kl} = \sum_x \sum_i P(\pi_i = k, \pi_{i+1} = l \mid x, \theta) = \sum_x \sum_i \frac{f_k(i)\, a_{kl}\, e_l(x_{i+1})\, b_l(i+1)}{P(x \mid \theta)}$$

Similarly,

$$E_k(b) = \sum_x (1/P(x)) \sum_{\{i \mid x_i = b\}} f_k(i)\, b_k(i)$$

The Baum-Welch Algorithm

Initialization:

Pick the best-guess for model parameters
(or arbitrary)

Iteration:

1. Forward
2. Backward
3. Calculate A_{kl}, $E_k(b)$
4. Calculate new model parameters a_{kl}, $e_k(b)$
5. Calculate new log-likelihood $P(x \mid \theta)$

GUARANTEED TO BE HIGHER BY EXPECTATION-MAXIMIZATION

Until $P(x \mid \theta)$ does not change much

The Baum-Welch Algorithm – comments

Time Complexity:

iterations × $O(K^2N)$

- Guaranteed to increase the log likelihood of the model

  $P(\theta \mid x) = P(x, \theta) / P(x) = P(x \mid \theta) / (P(x)\, P(\theta))$

- Not guaranteed to find globally best parameters

  Converges to local optimum, depending on initial conditions

- Too many parameters / too large model: Overtraining

Alternative: Viterbi Training

Initialization: Same

Iteration:

1. Perform Viterbi, to find π^*
2. Calculate A_{kl}, $E_k(b)$ according to π^* + pseudocounts
3. Calculate the new parameters a_{kl}, $e_k(b)$

Until convergence

Notes:

- Convergence is guaranteed – Why?
- Does not maximize $P(x \mid \theta)$
- In general, worse performance than Baum-Welch

How to Build an HMM

- General Scheme:
  - Architecture/topology design
  - Learning/Training:
    - Training Datasets
    - Parameter Estimation
  - Recognition/Classification:
    - Testing Datasets
    - Performance Evaluation

Parameter Estimation for HMMs (Case 1)

- Case 1: All the paths/labels in the set of training sequences are known:
  - Use the Maximum Likelihood (ML) estimators for:

$$a_{kl} = \frac{A_{kl}}{\sum_{l'} A_{kl'}} \text{ and } e_{kx} = \frac{E_k(x)}{\sum_{x'} E_k(x')}$$

  - Where A_{kl} and $E_k(x)$ are the number of times each transition or emission is used in training sequences
  - Drawbacks of ML estimators:
    - Vulnerable to overfitting if not enough data
    - Estimations can be undefined if never used in training set (add pseudocounts to reflect a prior biases about probability values)

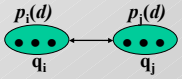
Parameter Estimation for HMMs (Case 2)

- Case 2: The paths/labels in the set of training sequences are UNknown:
  - Use Iterative methods (e.g., Baum-Welch):
    1. Initialize a_{kl} and e_{kx} (e.g., randomly)
    2. Estimate A_{kl} and $E_k(x)$ using current values of a_{kl} and e_{kx}
    3. Derive new values for a_{kl} and e_{kx}
    4. Iterate Steps 2-3 until some stopping criterion is met (e.g., change in the total log-likelihood is small)
  - Drawbacks of Iterative methods:
    - Converge to local optimum
    - Sensitive to initial values of a_{kl} and e_{kx} (Step 1)
    - Convergence problem is getting worse for large HMMs

HMM Architectural/Topology Design

- In general, HMM states and transitions are designed based on the knowledge of the problem under study
- Special Class: Explicit State Duration HMMs:
  - Self-transition state to itself:

    - The probability of staying in the state for d residues:
      $p_i(d \text{ residues}) = (a_{ii})^{d-1}(1-a_{ii})$ – exponentially decaying
    - Exponential state duration density is often inappropriate
      ⇒Need to explicitly model duration density in some form
  - Specified state density:
    - Used in GenScan

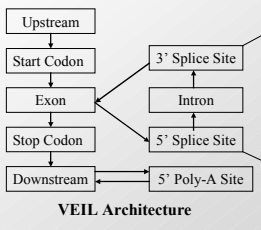


HMM-based Gene Finding

- GENSCAN (Burge 1997)
- FGENESH (Solovyev 1997)
- HMMgene (Krogh 1997)
- GENIE (Kulp 1996)
- GENMARK (Borodovsky & McIninch 1993)
- VEIL (Henderson, Salzberg, & Fasman 1997)

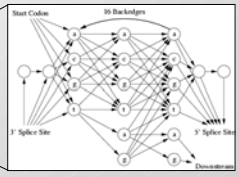
VEIL: Viterbi Exon-Intron Locator

- Contains 9 hidden states or features
- Each state is a complex internal Markovian model of the feature
- Features:
  - Exons, introns, intergenic regions, splice sites, etc.



VEIL Architecture

Exon HMM Model

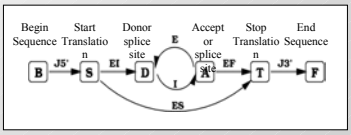


- Enter: start codon or intron (3' Splice Site)
- Exit: 5' Splice site or three stop codons (taa, tag, tga)

Genie

- Uses a generalized HMM (GHMM)
- Edges in model are complete HMMs
- States can be any arbitrary program
- States are actually neural networks specially designed for signal finding

- J5' – 5' UTR
- EI – Initial Exon
- E – Exon, Internal Exon
- I – Intron
- EF – Final Exon
- ES – Single Exon
- J3' – 3'UTR



Genscan Overview

- Developed by Chris Burge (Burge 1997), in the research group of Samuel Karlin, Dept of Mathematics, Stanford Univ.
- Characteristics:
  - Designed to predict complete gene structures
    - Introns and exons, Promoter sites, Polyadenylation signals
  - Incorporates:
    - Descriptions of transcriptional, translational and splicing signal
    - Length distributions (Explicit State Duration HMMs)
    - Compositional features of exons, introns, intergenic, C+G regions
  - Larger predictive scope
    - Deal w/ partial and complete genes
    - Multiple genes separated by intergenic DNA in a seq
    - Consistent sets of genes on either/both DNA strands
- Based on a general probabilistic model of genomic sequences composition and gene structure

Genscan Architecture

- It is based on Generalized HMM (GHMM)
- Model both strands at once
  - Other models: Predict on one strand first, then on the other strand
  - Avoids prediction of overlapping genes on the two strands (rare)
- Each state may output a string of symbols (according to some probability distribution).
- Explicit intron/exon length modeling
- Special sensors for Cap-site and TATA-box
- Advanced splice site sensors

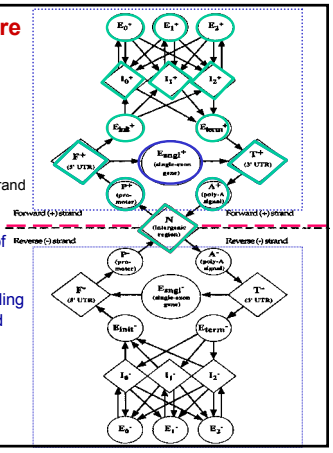
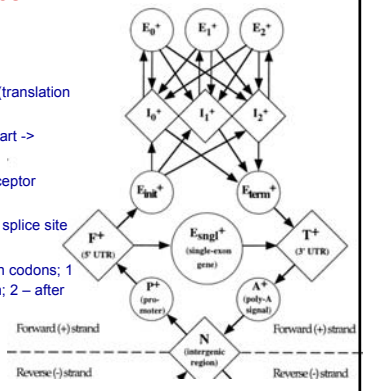


Fig. 3, Burge and Karlin 1997

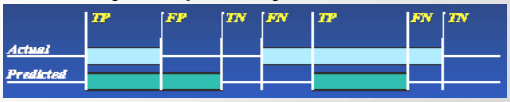
GenScan States

- N - intergenic region
- P - promoter
- F - 5' untranslated region
- E_{sngl} – single exon (intronless) (translation start -> stop codon)
- E_{init} – initial exon (translation start -> donor splice site)
- E_k – phase k internal exon (acceptor splice site -> donor splice site)
- E_{term} – terminal exon (acceptor splice site -> stop codon)
- I_k – phase k intron: 0 – between codons; 1 – after the first base of a codon; 2 – after the second base of a codon



Accuracy Measures

Sensitivity vs. Specificity (adapted from Burset&Guigo 1996)



Predicted \ Actual (Coding / No Coding)	Coding	No Coding
Coding	TP	FP
No Coding	FN	TN

$$Sn = \frac{TP}{TP + FN}$$

$$Sp = \frac{TP}{TP + FP}$$

$$CC = \frac{(TP * TN) - (FN * FP)}{\sqrt{(TP + FN) * (TN + FP) * (TP + FP) * (TN + FN)}}$$

- *Sensitivity (Sn)* — Fraction of actual coding regions that are correctly predicted as coding
- *Specificity (Sp)* — Fraction of the prediction that is actually correct
- *Correlation Coefficient (CC)* — Combined measure of Sensitivity & Specificity Range: -1 (always wrong) → +1 (always right)

Test Datasets

- Sample Tests reported by Literature
  - Test on the set of 570 vertebrate gene seqs (Burset&Guigo 1996) as a standard for comparison of gene finding methods.
  - Test on the set of 195 seqs of human, mouse or rat origin (named HMR195) (Rogic 2001).

Results: Accuracy Statistics

Table: Relative Performance (adapted from Rogic 2001)

	Test By Rogic 2001					
Programs	*# of seq*	*Nucleotide accuracy*			*Exon accuracy*	
		Sn	*Sp*	*CC*	*ESn*	*ESp*
Genscan	195(3)	0.95	0.90	0.91	0.70	0.70
HMMgene	195(5)	0.93	0.93	0.91	0.76	0.77
MZEF	119(8)	0.70	0.73	0.66	0.58	0.59

# of seqs - number of seqs effectively analyzed by each program; in parentheses is the number of seqs where the absence of gene was predicted;

Sn -nucleotide level sensitivity; **Sp** - nucleotide level specificity;

CC - correlation coefficient;

ESn - exon level sensitivity; **ESp** - exon level specificity

Complicating Factors for Comparison

- Gene finders were trained on data that had genes homologous to test seq.
  - Percentage of overlap is varied
- Some gene finders were able to tune their methods for particular data
- Methods continue to be developed

Needed

- Train and test methods on the same data.
- Do cross-validation (10% leave-out)

Why not Perfect?

- **Gene Number**
  usually approximately correct, but may not
- **Organism**
  primarily for human/vertebrate seqs; maybe lower accuracy for non-vertebrates. 'Glimmer' & 'GeneMark' for prokaryotic or yeast seqs
- **Exon and Feature Type**
  Internal exons: predicted more accurately than Initial or Terminal exons;
  Exons: predicted more accurately than Poly-A or Promoter signals
- **Biases in Test Set** (Resulting statistics may not be representative)
  The Burset/Guigó (1996) dataset:
  - Biased toward short genes with relatively simple exon/intron structure

  The Rogic (2001) dataset:
  - DNA seqs: GenBank r-111.0 (04/1999 <- 08/1997);
  - source organism specified;
  - consider genomic seqs containing exactly one gene;
  - seqs>200kb were discarded; mRNA seqs and seqs containing pseudo genes or alternatively spliced genes were excluded.

What We Learned…

- Genes are complex structures which are difficult to predict with the required level of accuracy/confidence
- Different HMM-based approaches have been successfully used to address the gene finding problem:
  - Building an architecture of an HMM is the hardest part, it should be biologically sound & easy to interpret
  - Parameter estimation can be trapped in local optimum
- Viterbi algorithm can be used to find the most probable path/labels
- These approaches are still not perfect