Markov Chains and Hidden Markov Models

COMP 571 - Spring 2015

Luay Nakhleh, Rice University

Markov Chains and Hidden Markov Models

* Modeling the statistical properties of biological sequences and distinguishing regions based on these models
* For the alignment problem, they provide a probabilistic framework for aligning sequences

Example: CpG Islands

* Regions that are rich in CG dinucleotide
* Promoter and "start" regions of many genes are characterized by high frequency of CG dinucleotides (in fact, more C and G nucleotides in general)

CpG Islands: Two Questions

* Q1: Given a short sequence, does it come from a CpG island?
* Q2: Given a long sequence, how would we find the CpG islands in it?

CpG Islands

* Answer to Q1:
  * Given sequence x and probabilistic model M of CpG islands, compute p=P(x|M)
* If p is "significant", then x comes from a CpG island; otherwise, x does not come from a CpG island

CpG Islands

* Answer to Q1:
  * Given sequence x, probabilistic model M_1 of CpG islands, and probabilistic model M_2 of non-CpG islands, compute $p_1 = P(x|M_1)$ and $p_2 = P(x|M_2)$
* If $p_1 > p_2$, then x comes from a CpG island
* If $p_1 < p_2$, then x does not come from a CpG island

CpG Islands

* Answer to Q2:
  * As before, use the models M_1 and M_2, calculate the scores for a window of, say, 100 nucleotides around every nucleotide in the sequence
  * Not satisfactory
  * A more satisfactory approach is to build a single model for the entire sequence that incorporates both Markov chains

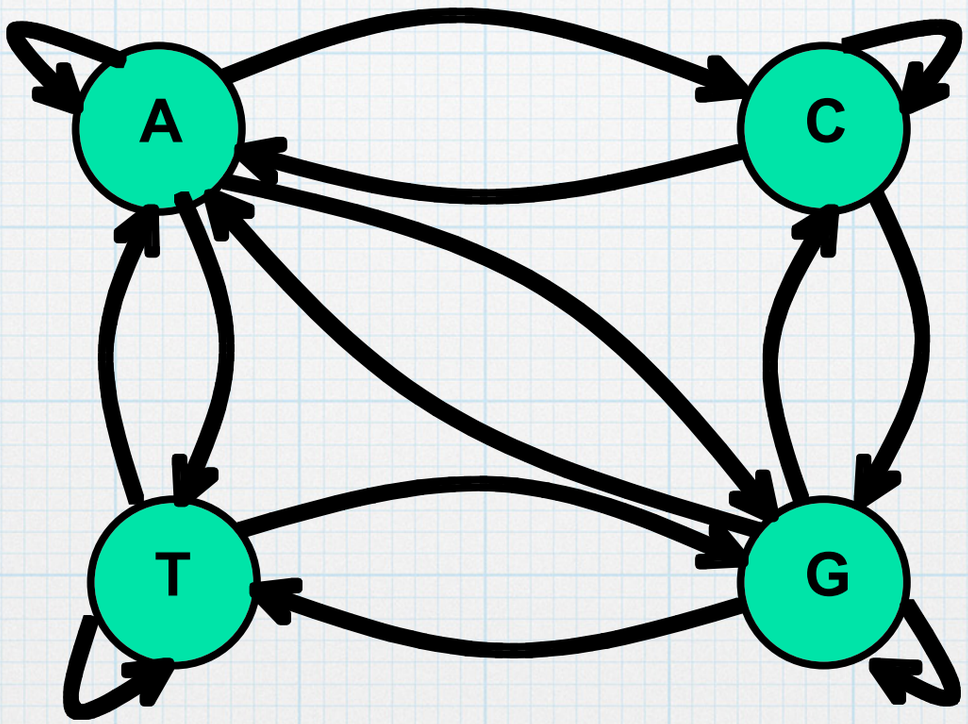
Difference Between the Two Solutions

* Solution to Q1:
  * One "state" for each nucleotide, since we have only one region
    * 1-1 correspondence between "state" and "nucleotide"
* Solution to Q2:
  * Two "states" for each nucleotide (one for the nucleotide in a CpG island, and another for the same nucleotide in a non-CpG island)
    * No 1-1 correspondence between "state" and "nucleotide"

Markov Chains vs. HMMs

* When we have a 1-1 correspondence between alphabet letters and states, we have a Markov chain
* When such a correspondence does not hold, we only know the letters (observed data), and the states are "hidden"; hence, we have a hidden Markov model, or HMM

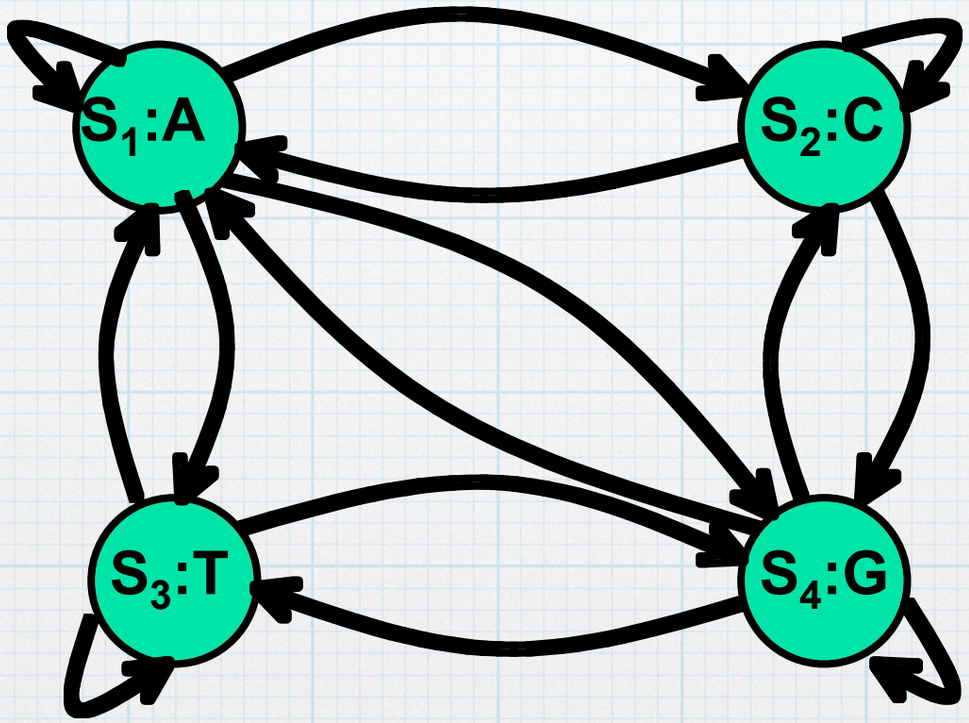
Markov Chains



Associated with each edge is a transition probability

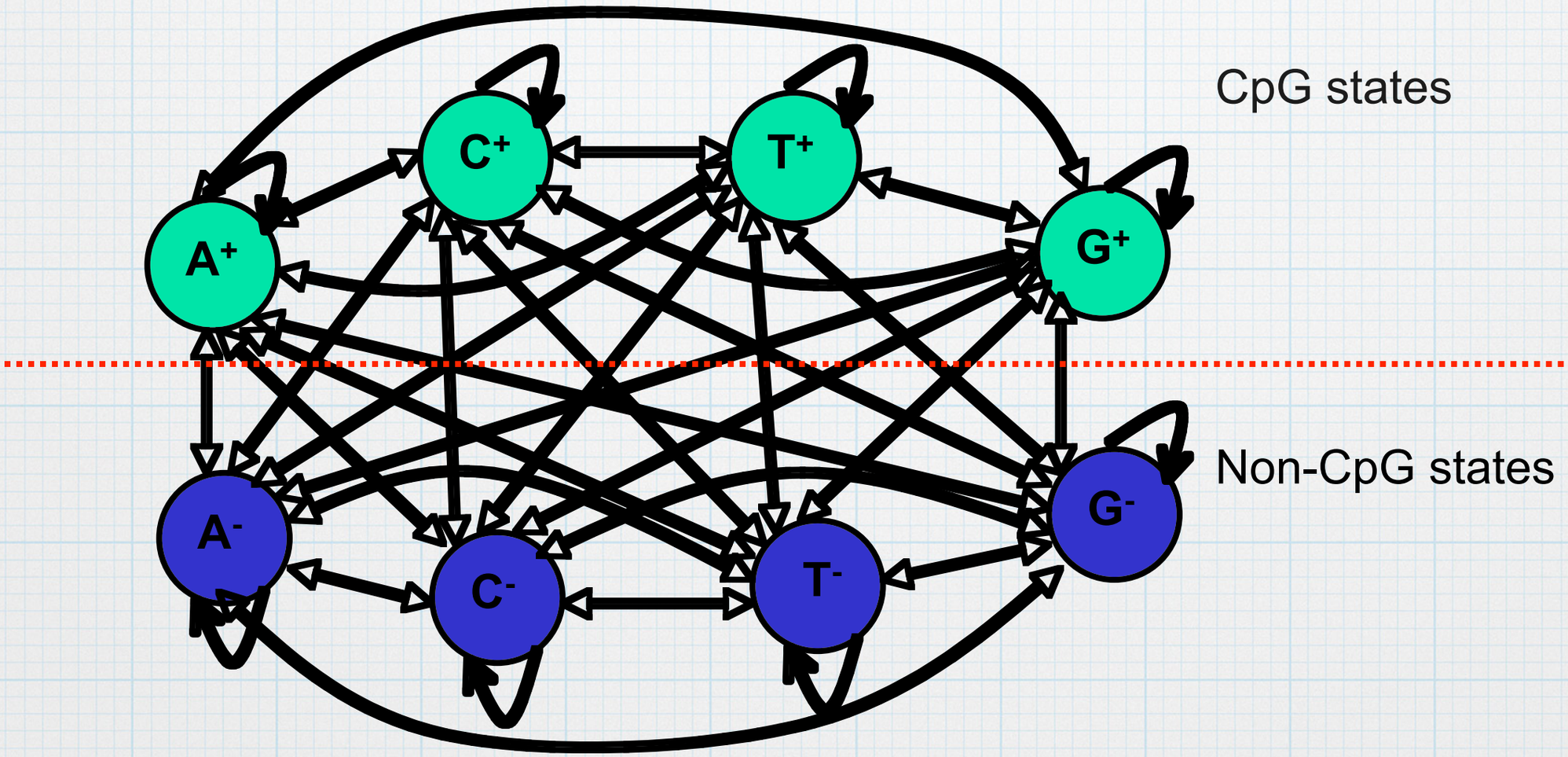
Markov Chains: The 1-1 Correspondence

Sequence: GAGCGCGTAC



States: $S_4S_1S_4S_2S_4S_2S_4S_3S_1S_2$

HMMs: No 1-1 Correspondence (2 States Per Nucleotide)



What's Hidden?

* We can "see" the nucleotide sequence
* We cannot see the sequence of states, or path, that generated the nucleotide sequence
* Hence, the state sequence (path) that generated the data is hidden

Markov Chains and HMMs

* In Markov chains and hidden Markov models, the probability of being in a state depends solely on the previous state
* Dependence on more than the previous state necessitates higher order Markov models

Sequence Annotation Using Markov Chains

* The annotation is straightforward: given the input sequence, we have a unique annotation (mapping between sequence letters and model states)
* The outcome is the probability of the sequence given the model

Sequence Annotation Using HMMs

* For every input sequence, there are many possible annotations (paths in the HMM)
* Annotation corresponds to finding the best mapping between sequence letters and model states (i.e., the path of highest probability that corresponds to the input sequence)

Markov Chains: Formal Definition

* A set Q of states
* Transition probabilities
  * $a_{st}=P(x_i=t|x_{i-1}=s)$: probability of state t given that the previous state was s
* In this model, the probability of sequence $x=x_1x_2...x_L$ is

$$\mathbf{P}(x) = \mathbf{P}(x_L|x_{L-1})\mathbf{P}(x_{L-1}|x_{L-2})\cdots\mathbf{P}(x_2|x_1)\mathbf{P}(x_1) = \mathbf{P}(x_1)\prod_{i=2}^{L} a_{x_{i-1}x_i}$$

Markov Chains: Formal Definition

* Usually, two states "start" and "end" are added to the Markov chain to model the beginning and end of sequences, respectively
* Adding these two states, the model defines a probability distribution on all possible sequences (of any length)

HMMs: Formal Definition

* A set Q of states
* An alphabet Σ
* Transition probability a_{st} for every two states s and t
* Emission probability $e_k(b)$ for every letter b and state k (the probability of emitting letter b in state k)

HMMs: Sequences and Paths

* Due to the lack of a 1-1 correspondence, we need to distinguish between the sequence of letters (e.g., DNA sequences) and the sequence of states (path)

* For every sequence (of letters) there are many paths for generating it, each occurring with its probability

* We use x to denote a (DNA) sequence, and π to denote a (state) path

HMMs: The Model Probabilities

* Transition probability $a_{k\ell} = \mathbf{P}(\pi_i = \ell | \pi_{i-1} = k)$
* Emission probability $e_k(b) = \mathbf{P}(x_i = b | \pi_i = k)$

HMMs: The Sequence Probabilities

* The joint probability of an observed sequence and a path is

$$\mathbf{P}(x, \pi) = a_{0\pi_1} \prod_{i=1}^{L} e_{\pi_i}(x_i) a_{\pi_i \pi_{i+1}}$$

* The probability of a sequence is

$$\mathbf{P}(x) = \sum_{\pi} \mathbf{P}(x, \pi)$$

HMMs: The Parsing Problem

* Find the most probable state path that generates a given a sequence

$$\pi^* = \text{argmax}_\pi \mathbf{P}(x, \pi)$$

HMMs: The Posterior Decoding Problem

* Compute "confidence" for the states on a path

$$\mathbf{P}(\pi_i = k|x)$$

HMMs: The Parameter Estimation Problem

* Compute the transition and emission probabilities of an HMM (from a given training data set)

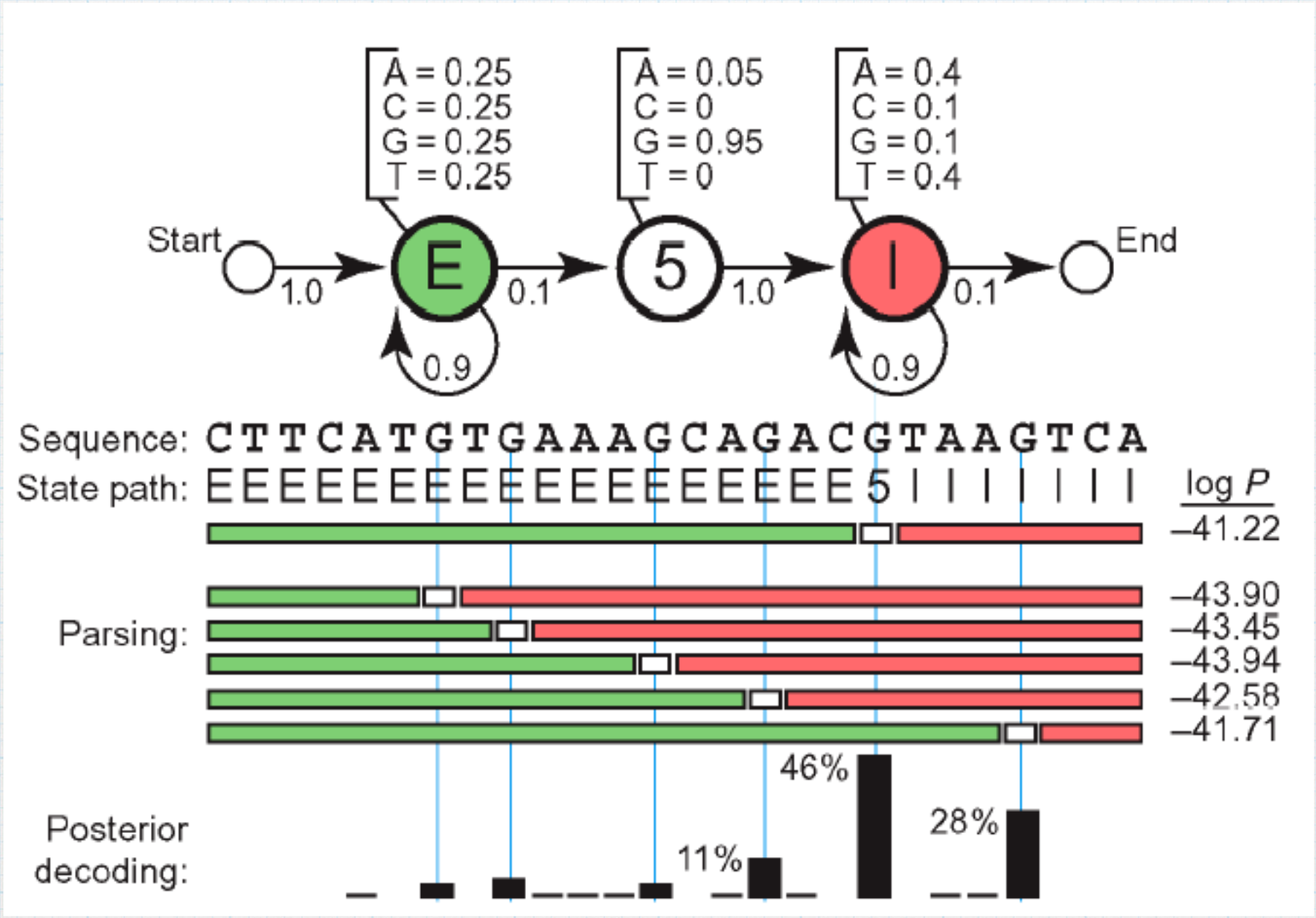
A Toy Example: 5' Splice Site Recognition

* From "What is a hidden Markov model?", by Sean R. Eddy
* 5' splice site indicates the "switch" from an exon to an intron

A Toy Example: 5' Splice Site Recognition

* Assumptions
  * Uniform base composition on average in exons
  * Introns are A/T rich (40% A/T, 10% G/C)
  * The 5' splice site consensus nucleotide is almost always a G (say, 95% G and 5% A)

A Toy Example: 5' Splice Site Recognition



HMMs: A DP Algorithm for the Parsing Problem

* Let $v_k(i)$ denote the probability of the most probable path ending in state k with observation x_i
* The DP structure:

$$v_\ell(i+1) = e_\ell(x_{i+1}) \max_k (v_k(i) a_{k\ell})$$

The Viterbi Algrorithm

Initialization

$$v_0(0) = 1, \quad v_k(0) = 0 \; \forall k > 0$$

Recursion

$$v_\ell(i) = e_\ell(x_i) \max(v_k(i-1)a_{k\ell}) \qquad i = 1 \dots L$$
$$ptr_i(\ell) = \text{argmax}_k(v_k(i-1)a_{k\ell})$$

Termination

$$\mathbf{P}(x, \pi^*) = \max_k(v_k(L)a_{k0})$$
$$\pi_L^* = \text{argmax}_k(v_k(L)a_{k0})$$

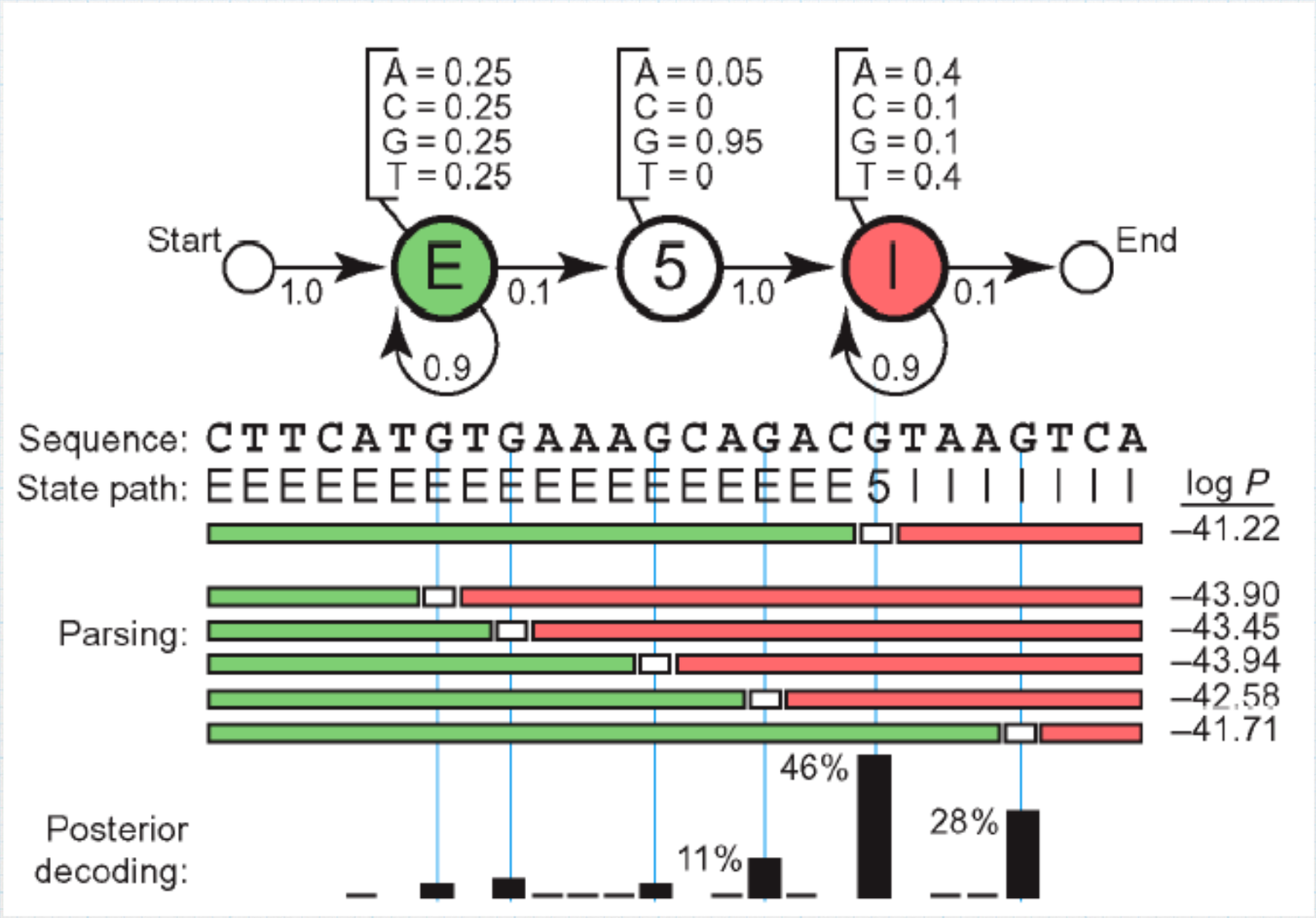
Traceback

$$\pi_{i-1}^* = ptr_i(\pi_i^*) \qquad i = 1 \dots L$$

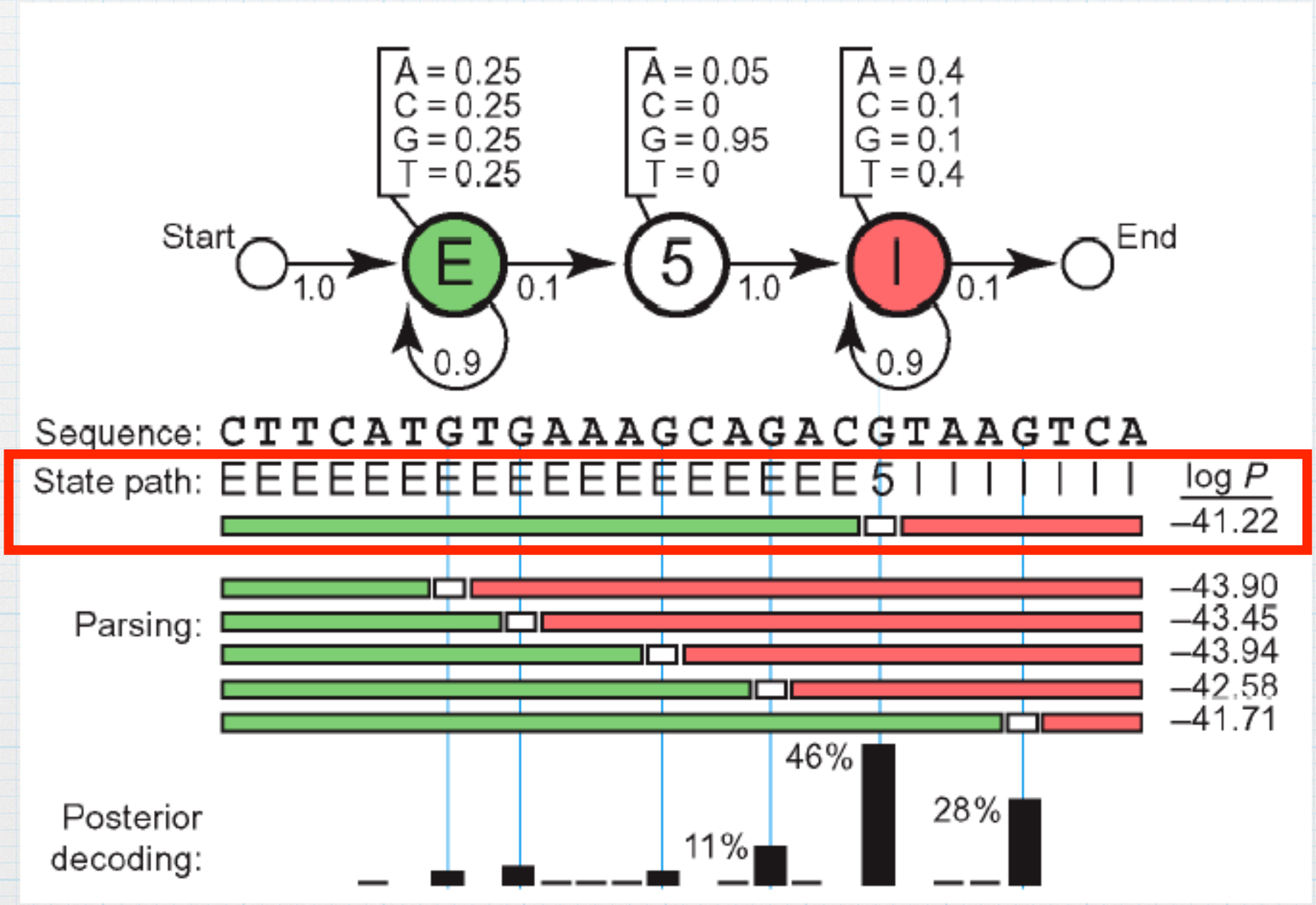
The Viterbi Algrorithm

* Usually, the algorithm is implemented to work with logarithms of probabilities so that the multiplication turns into addition
* The algorithm takes $O(Lq^2)$ time and $O(Lq)$ space, where L is the sequence length and q is the number of states

A Toy Example: 5' Splice Site Recognition



A Toy Example: 5' Splice Site Recognition



Other Values of Interest

* The probability of a sequence, P(x)
* Posterior decoding: $\mathbf{P}(\pi_i = k|x)$
* Efficient DP algorithms for both using the forward and backward algorithms

The Forward Algorithm

* $f_k(i)$: the probability of the observed sequence up to and including x_i, requiring that $\pi_i=k$
* In other words, $f_k(i)=P(x_1,\ldots,x_i, \pi_i=k)$
* The structure of the DP algorithm:

$$f_\ell(i+1) = e_\ell(x_{i+1}) \sum_k f_k(i) a_{k\ell}$$

The Forward Algorithm

* **Initialization:** $f_0(0) = 1, \quad f_k(0) = 0 \; \forall k > 0$
* **Recursion:** $f_\ell(i) = e_\ell(x_i) \sum_k f_k(i-1) a_{k\ell} \quad i = 1 \ldots L$
* **Termination:** $\mathbf{P}(x) = \sum_k f_k(L) a_{k0}$

The Backward Algorithm

* $b_k(i)$: the probability of the last observed L-i letters, requiring that $\pi_i=k$
* In other words, $b_k(i)=P(x_L,\ldots,x_{i+1} | \pi_i=k)$
* The structure of the DP algorithm:

$$b_\ell(i) = \sum_k a_{\ell k} e_k(x_{i+1}) b_k(i+1)$$

The Backward Algorithm

* **Initialization:** $b_k(L) = a_{k0} \quad \forall k$
* **Recursion:** $b_\ell(i) = \sum_k a_{\ell k} e_\ell(x_{i+1}) b_k(i+1) \quad i = L-1, \ldots, 1$
* **Termination:** $\mathbf{P}(x) = \sum_k a_{0k} e_k(x_1) b_k(1)$

The Posterior Probability

$$\begin{aligned} f_k(i)b_k(i) &= \mathbf{P}(x, \pi_i = k) \\ &= \mathbf{P}(\pi_i = k|x)\mathbf{P}(x) \end{aligned}$$

$$\Rightarrow \mathbf{P}(\pi_i = k|x) = \frac{f_k(i)b_k(i)}{\mathbf{P}(x)}$$

The Probability of a Sequence

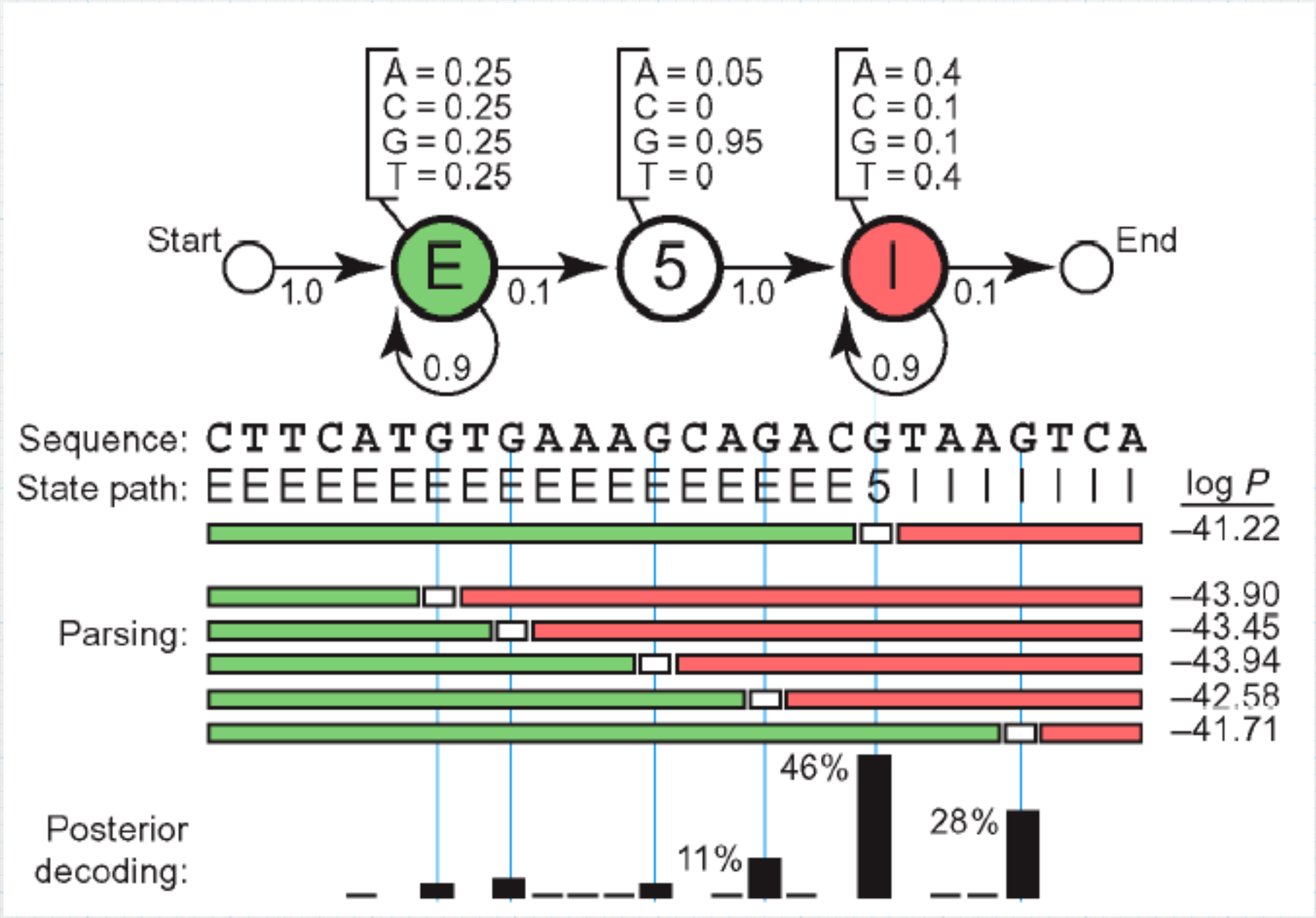
$$\mathbf{P}(x) = \sum_k f_k(L) a_{k0}$$

$$\mathbf{P}(x) = \sum_k a_{0k} e_k(x_1) b_k(1)$$

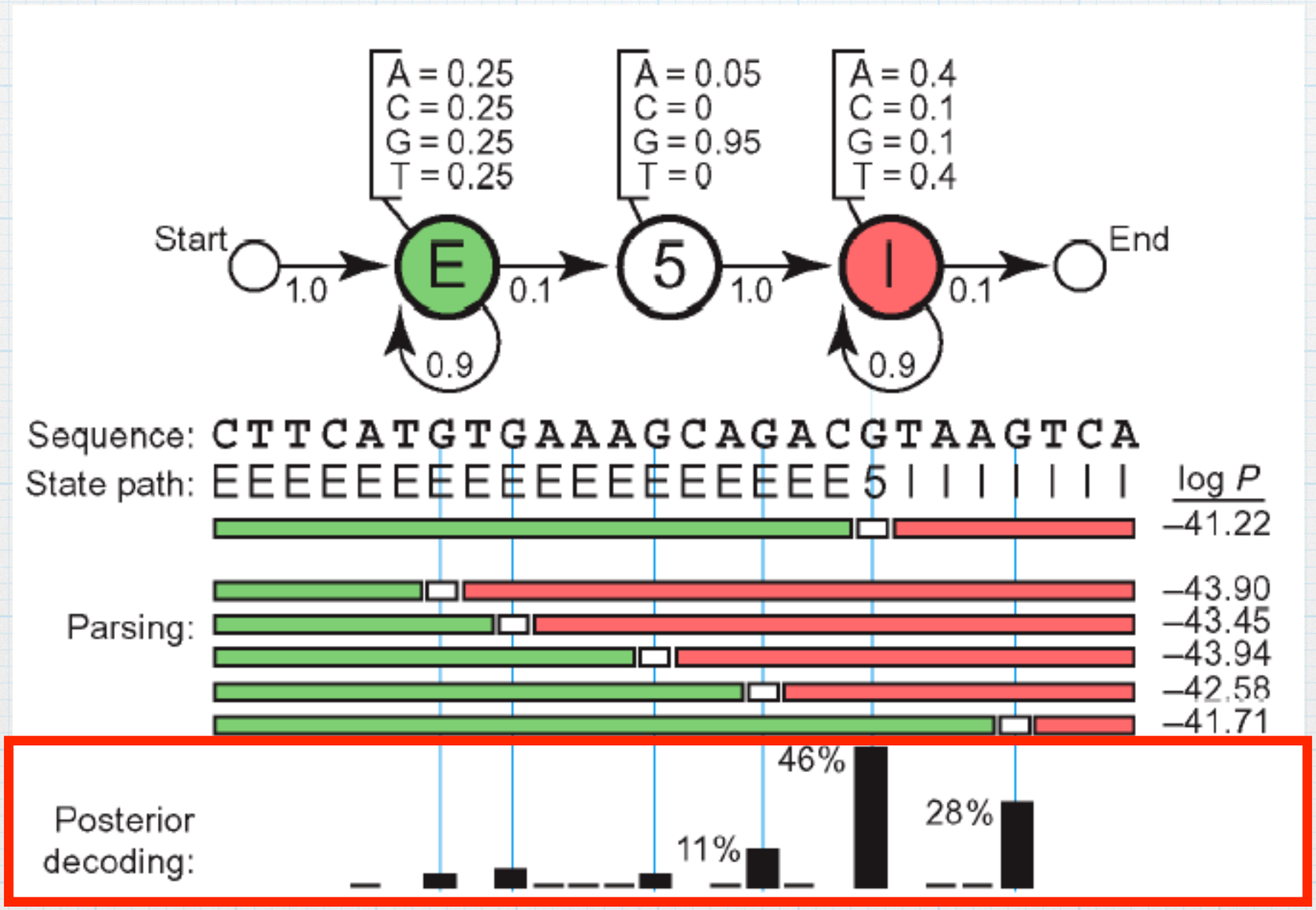
Computational Requirements of the Algorithms

* Each of the algorithms takes $O(Lq^2)$ time and $O(Lq)$ space, where L is the sequence length and q is the number of states

A Toy Example: 5' Splice Site Recognition



A Toy Example: 5' Splice Site Recognition



Applications of Posterior Decoding (1)

* Find the sequence π' of states where $\pi'_i = \text{argmax}_k \mathbf{P}(\pi_i = k|x)$
* This is a more appropriate path when we are interested in the state assignment at a particular point i (however, this sequence of states may not be a legitimate path!)

Applications of Posterior Decoding (2)

* Assume function g(k) is defined on the set of states
* We can consider $G(i|x) = \sum_k \mathbf{P}(\pi_i = k|x)g(k)$
* For example, for the CpG island problem, setting g(k)=1 for the "+" states, and g(k)=0 for the "-" states, G(i|x) is precisely the posterior probability according to the model that base i is in a CpG island

Parameter Estimation for HMMs

* Two components:
  * the probabilities (emission and transition): there is a well-developed theory
  * the structure (states): more of an "art"
* We'll focus on estimating the probabilities

Estimating HMM Emission and Transition Probabilities

* Given the structure of an HMM, and a set of training sequences, we'd want to estimate the probabilities from the training data set

* There are two cases
  * The training sequences are already annotated (i.e., the state sequences are known)
  * The training sequences are not annotated (i.e., the state sequences are not known)

Estimating HMM Probabilities: Known State Sequences

* Given a training data set, count the number of times each particular transition or emission is used; denote these by A_{kl} and $E_k(b)$

* Then

(1)
$$a_{k\ell} = \frac{A_{k\ell}}{\sum_{\ell'} A_{k\ell'}} \qquad e_k(b) = \frac{E_k(b)}{\sum_{b'} E_k(b')}$$

Estimating HMM Probabilities: Unknown State Sequences

* The Baum-Welch algorithm, which is an expectation-maximization (EM) algorithm
* Informally, the algorithm first estimates the A_{kl} and $E_k(b)$ by considering probable paths for the training sequences using the current values of a_{kl} and $e_k(b)$
* Then, new values of the as and es are derived using the equations on the previous slide
* This process is iterated until some stopping criterion is reached

The Baum-Welch Algorithm

* It is possible to show that the overall log likelihood of the model is increased by the iteration, and hence that the process will converge to a local maximum

* Unfortunately, there are usually many local maxima, and which one you end up with depends strongly on the starting values of the parameters

* The problem of local maxima is particularly severe when estimating large HMMs

The Baum-Welch Algorithm

* More formally, the Baum-Welch algorithm calculates A_{kl} and $E_k(b)$ as the expected number of times each transition or emission is used, given the training sequences
* To do this, it uses the forward and backward values

The Baum-Welch Algorithm

* The probability that a_{kl} is used at position i in sequence x is

$$\mathbf{P}(\pi_i = k, \pi_{i+1} = \ell | x, \theta) = \frac{f_k(i) a_{k\ell} e_\ell(x_{i+1}) b_\ell(i+1)}{\mathbf{P}(x)}$$

The Baum-Welch Algorithm

* From this we derive the expected number of times that a_{kl} is used by summing over all positions and over all training data sets

(2) $$A_{k\ell} = \sum_j \frac{1}{\mathbf{P}(x^j)} \sum_i f_k^j(i) a_{k\ell} e_\ell(x_{i+1}^j) b_\ell^j(i+1)$$

(f^j and b^j are the forward and backward values for sequence x^j)

The Baum-Welch Algorithm

* Similarly, we can find the expected number of times that letter b appears in state k

(3) $$E_k(b) = \sum_j \frac{1}{\mathbf{P}(x^j)} \sum_{\{i|x_i^j=b\}} f_k^j(i) b_k^j(i)$$

The Baum-Welch Algorithm

* Having calculated these expectations, the new model parameters are calculated just as before, using (1)
* We can iterate using the new values of the parameters to obtain new values of the As and Es as before, but in this case we are converging in a continuous-values space, and so will never in fact reach the maximum
* It is therefore necessary to set a convergence criterion, typically stopping when the change in total log likelihood is sufficiently small

The Baum-Welch Algorithm

1. Initialization: Pick arbitrary model parameters (θ)
2. Recurrence:
    1. Set all the A and E variables to their pseudocount values r (or to zero)
    2. For each sequence j=1,...,n
        1. Calculate $f_k(i)$ for sequence j using the forward algorithm
        2. Calculate $b_k(i)$ for sequence j using the backward algorithm
        3. Add the contribution of sequence j to A (2) and E (3)
    3. Calculate the new model parameters using (1)
    4. Calculate the new log likelihood of the model
3. Termination: Stop if the change in the log likelihood is less than some predefined threshold or the maximum number of iterations is exceeded

Viterbi Training

* An alternative to the Baum-Welch algorithm is frequently used, which is called Viterbi training
* In this approach, the most probable paths for the training sequences are derived using the Viterbi algorithm, and these are used in the re-estimation process
* Again, the process is iterated when the new parameter values are obtained
* In this case, the algorithm converges precisely, because the assignment of paths is a discrete process, and we can continue until none of the paths change
* At this point the parameter estimates will not change either, because they are determined completely by the paths

Viterbi Training

* Unlike Baum-Welch, this procedure does not maximize the true likelihood (the probability of the sequences, given the parameters)
* Instead, it finds the value of θ that maximizes the contribution to the likelihood $P(x^1,\ldots,x^n|\theta,\pi^*(x^1),\ldots,\pi^*(x^n))$ from the most probable paths for all the sequences
* This is a probable reason for why Viterbi training performs less well in general than Baum-Welch
* However, it is widely used, and it can be argued that when the primary use of the HMM is to produce decodings via Viterbi alignments, then it is good to train using them