CSE8803/CX4803 Machine Learning in Computational Biology

Lecture 5: Gene/Motif finding using HMMs II
Profile HMMs

Xiuwei Zhang

School of Computational Science and Engineering

Course logistics

- HW1:
  - Deadline extended to Tuesday, 2/1, no grace period.
  - This Wed's OHs are moved to next week Mon. and Tue.
- Paper presentations:
  - Submit your team preference by Friday 1/28
    (no grace period)

Hidden Markov Models (HMM)

V = alphabet of symbols, |V|=M

S = set of states, |S|=N

A = an |S| x |S| matrix where entry (i,j) is the probability of moving from state i to state j.

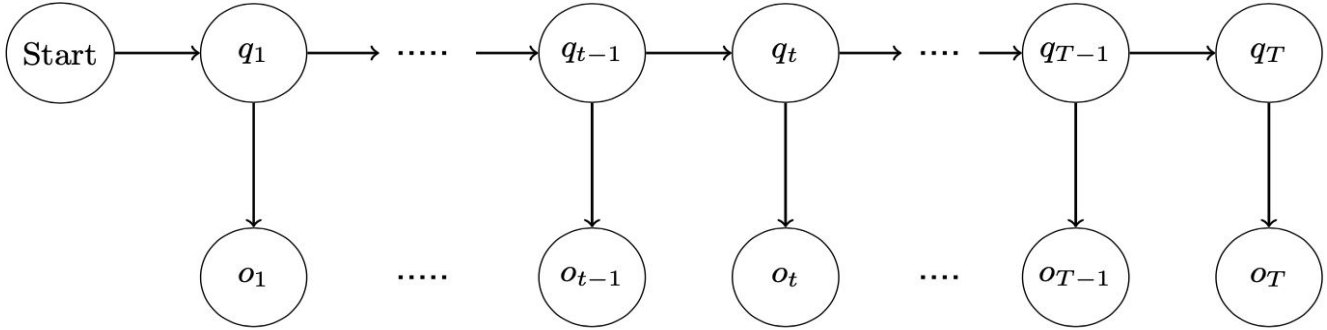
B = a |S| x |V| matrix, where entry (i,k) is the probabili of emitting v_k when in state s_i.

$$\lambda = (\pi, A, B)$$

$$\pi = \{\pi_i\}$$

$$A = \{a_{ij}\}$$

$$B = \{b_i(v_k)\}$$



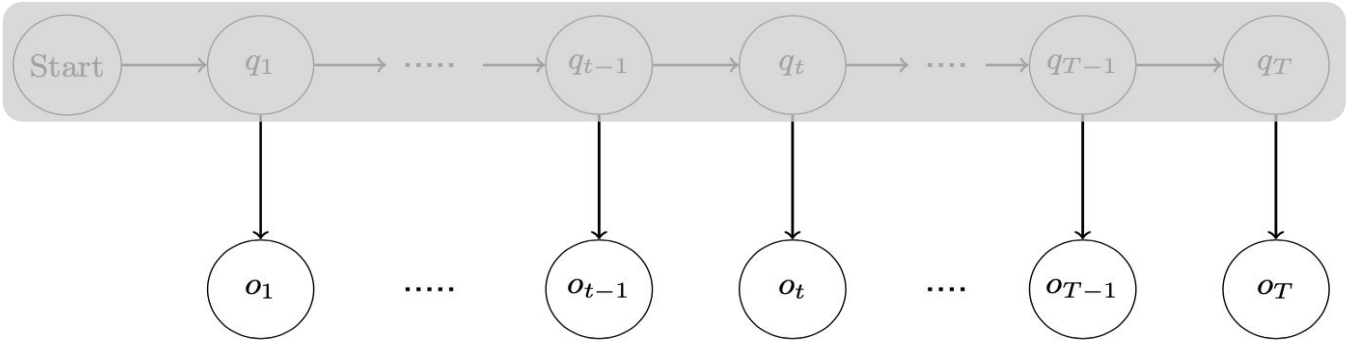
Q: $q_1, q_2, \ldots, q_T$
O: $o_1, o_2, \ldots, o_T$

Hidden Markov Models

Main algorithms with an HMM

- Viterbi algorithm → decoding/detection/matching problem
- Forward algorithm → scoring problem
- Backward algorithm → scoring problem
- Forward-backward / Baum-Welch algorithm → training problem

The Viterbi Algorithm to Find Best Path

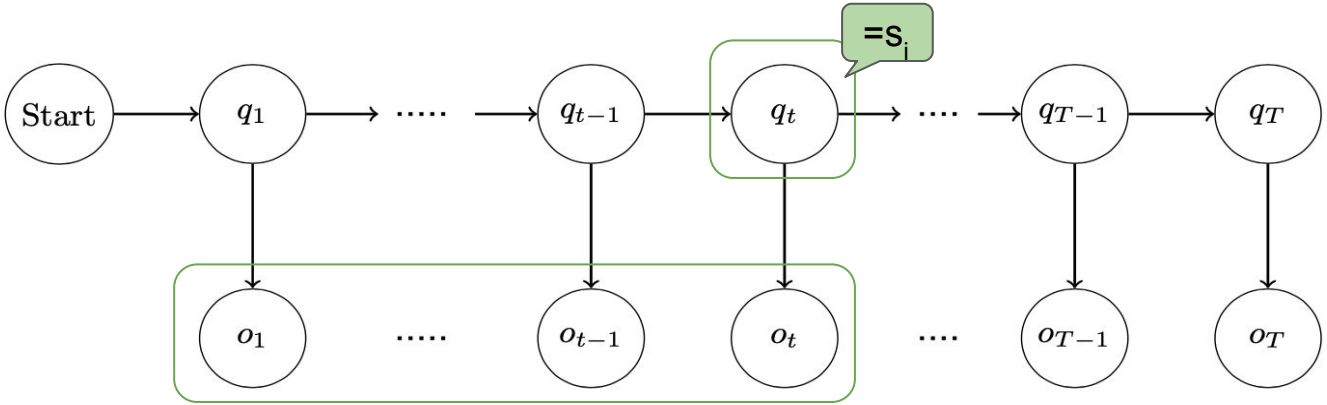


Dynamic programming to find the optimal Q which maximizes Pr(O,Q|λ)

Subproblem: the probability of the **best** path for $o_1 \ldots o_t$ that ends at state i.

$$w_t(i) = \max_{q_1,\ldots,q_{t-1}} Pr(o_1, o_2, \ldots, o_t, q_t = s_i)$$

The Viterbi Algorithm to Find Best Path



Dynamic programming to find the optimal Q which maximizes Pr(O,Q|λ)

Subproblem: the probability of the **best** path for $o_1 \ldots o_t$ that ends at state i.

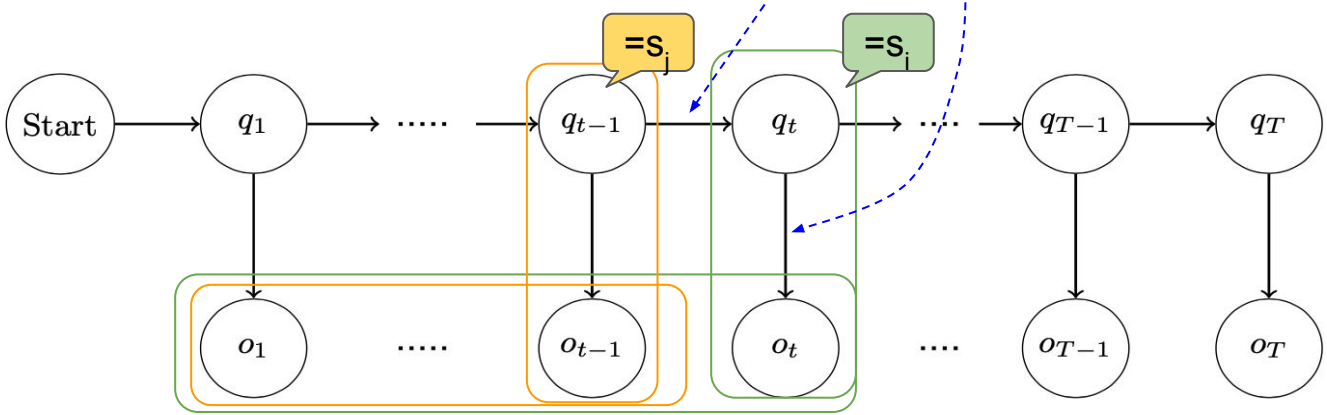
$$w_t(i) = \max_{q_1,\ldots,q_{t-1}} Pr(o_1, o_2, \ldots, o_t, q_t = s_i)$$

The Viterbi Algorithm to Find Best Path

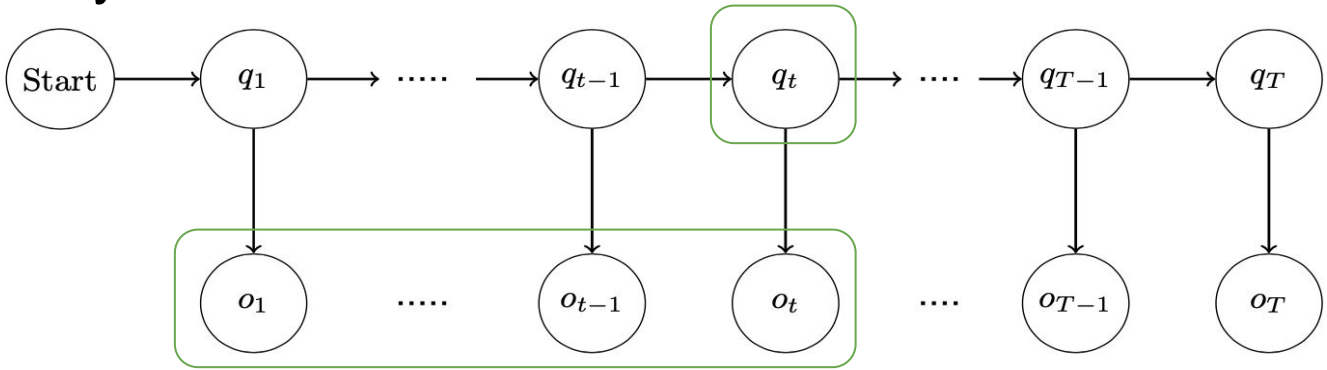
$w_t(i)$: = the probability of the **best** path for $o_1 \ldots o_t$ that ends at state i.

Smaller subproblem: calculate $w_{t-1}(j)$

$$w_t(i) = \max_{1 \le j \le N} w_{t-1}(j) a_{ji} b_i(o_t)$$



Why the DP works?



$$w_t(i) = \max_{q_1, q_2, \ldots, q_{t-1}} Pr(o_1, o_2, \ldots, o_{t-1}, o_t, q_1, q_2, \ldots, q_{t-1}, q_t=s_i)$$

→ Chain rule: P(A,B)=P(A|B)P(B)

$$= \max_{q_1, q_2, \ldots, q_{t-1}} \{ Pr(o_t, q_t=s_i | o_1, o_2, \ldots, o_{t-1}, q_1, q_2, \ldots, q_{t-1}) \, Pr(o_1, o_2, \ldots, o_{t-1}, q_1, q_2, \ldots, q_{t-1}) \}$$

→ Markov property

$$= \max_{q_1, q_2, \ldots, q_{t-1}} \{ Pr(o_t, q_t=s_i | q_{t-1}) \, Pr(o_1, o_2, \ldots, o_{t-1}, q_1, q_2, \ldots, q_{t-1}) \}$$

$$= \max_j \{ Pr(o_t, q_t=s_i | q_{t-1}=s_j) \max_{q_1, q_2, \ldots, q_{t-2}} Pr(o_1, o_2, \ldots, o_{t-1}, q_1, q_2, \ldots, q_{t-2}, q_{t-1}= s_j) \}$$

$$= \max_j \{ a_{ji} \, b_i(o_t) \, w_{t-1}(j) \}$$

The Viterbi Algorithm to Find Best Path

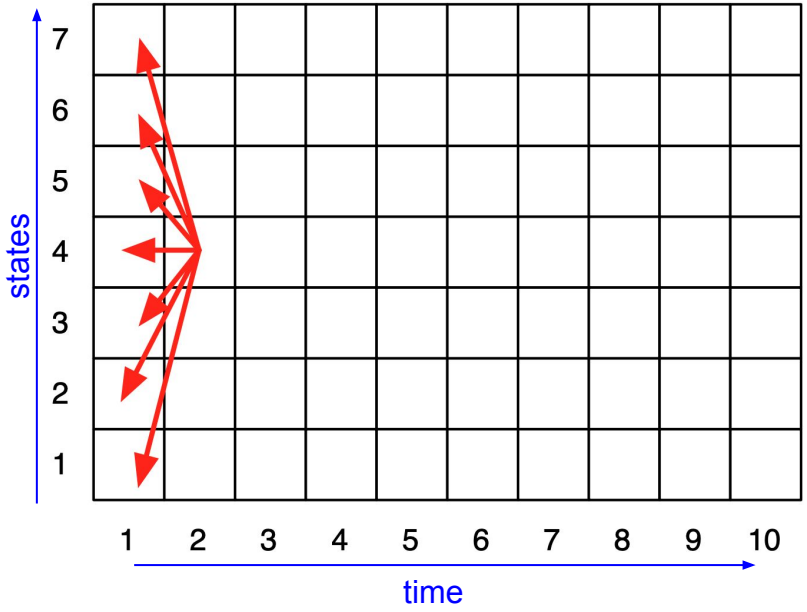
$w_t(i)$: = the probability of the **best** path for $o_1 \ldots o_t$ that ends at state i.

Recurrence:

$$w_t(i) = \max_{1 \le j \le N} w_{t-1}(j) a_{ji} b_i(o_t)$$

Base case:

$$w_1(i) = \pi_i b_i(o_1)$$



The Viterbi Algorithm to Find Best Path

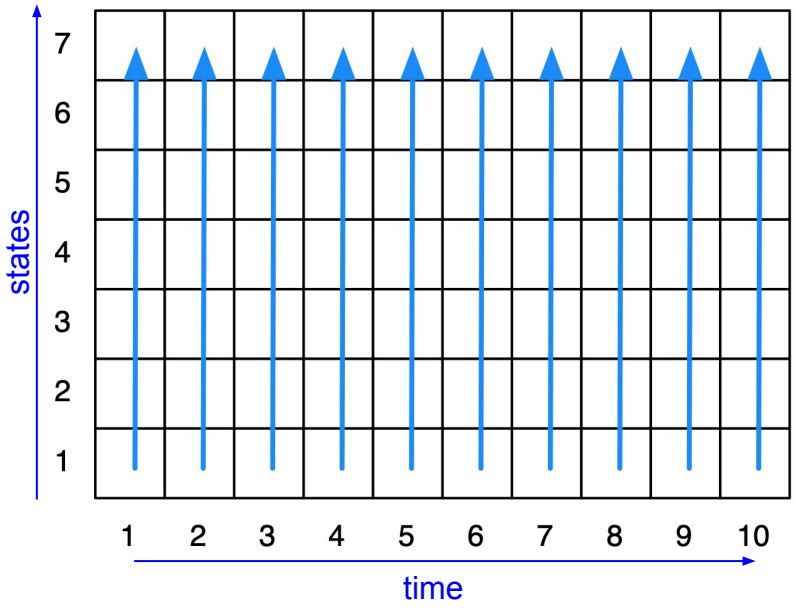
$w_t(i)$: = the probability of the **best** path for $o_1 \ldots o_t$ that ends at state i.

Recurrence:

$$w_t(i) = \max_{1 \le j \le N} w_{t-1}(j) a_{ji} b_i(o_t)$$

Base case:

$$w_1(i) = \pi_i b_i(o_1)$$



The Viterbi Algorithm to Find Best Path

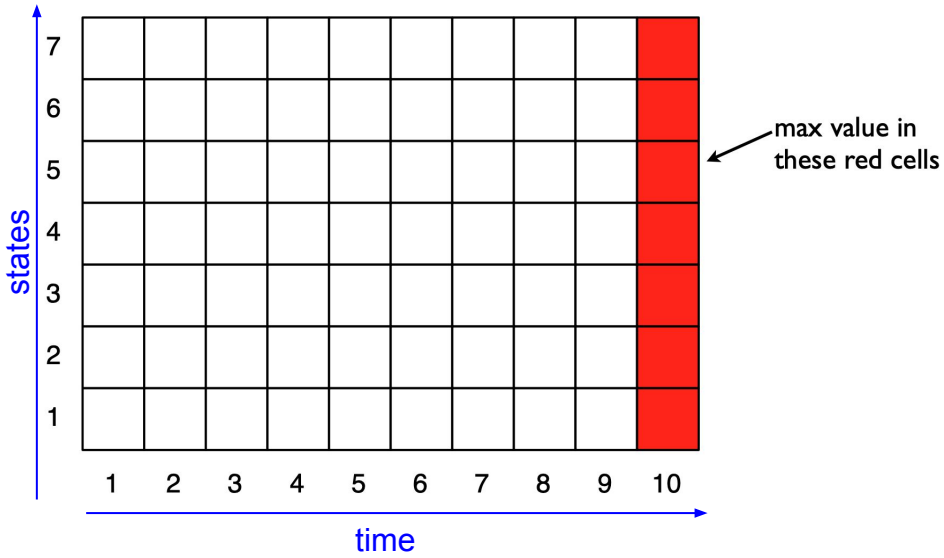
$w_t(i)$: = the probability of the **best** path for $o_1 \ldots o_t$ that ends at state i.

Recurrence:

$$w_t(i) = \max_{1 \le j \le N} w_{t-1}(j) a_{ji} b_i(o_t)$$

Base case:

$$w_1(i) = \pi_i b_i(o_1)$$



Running Time

- # of subproblems = $O(T|S|)$, where T is the length of the sequence.
- Time to solve a subproblem = $O(|S|)$
- Total running time: $O(T|S|^2)$

Using Logs

Typically, we take the log of the probabilities to avoid multiplying a lot of (small) terms:

$$\boxed{\log(ab) = \log(a) + \log(b)}$$

$$\begin{aligned}
\log(w_t(i)) &= \max_{1 \le j \le N} \{\log(w_{t-1}(j) \cdot a_{ji} \cdot b_i(o_t))\} \\
&= \max_{1 \le j \le N} \{\log(w_{t-1}(j)) + \log(a_{ji}) + \log(b_i(o_t))\}
\end{aligned}$$

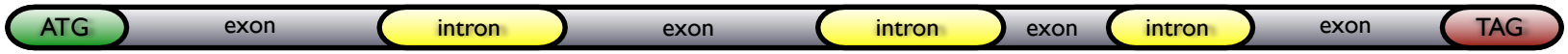
Why do we want to avoid multiplying lots of terms?

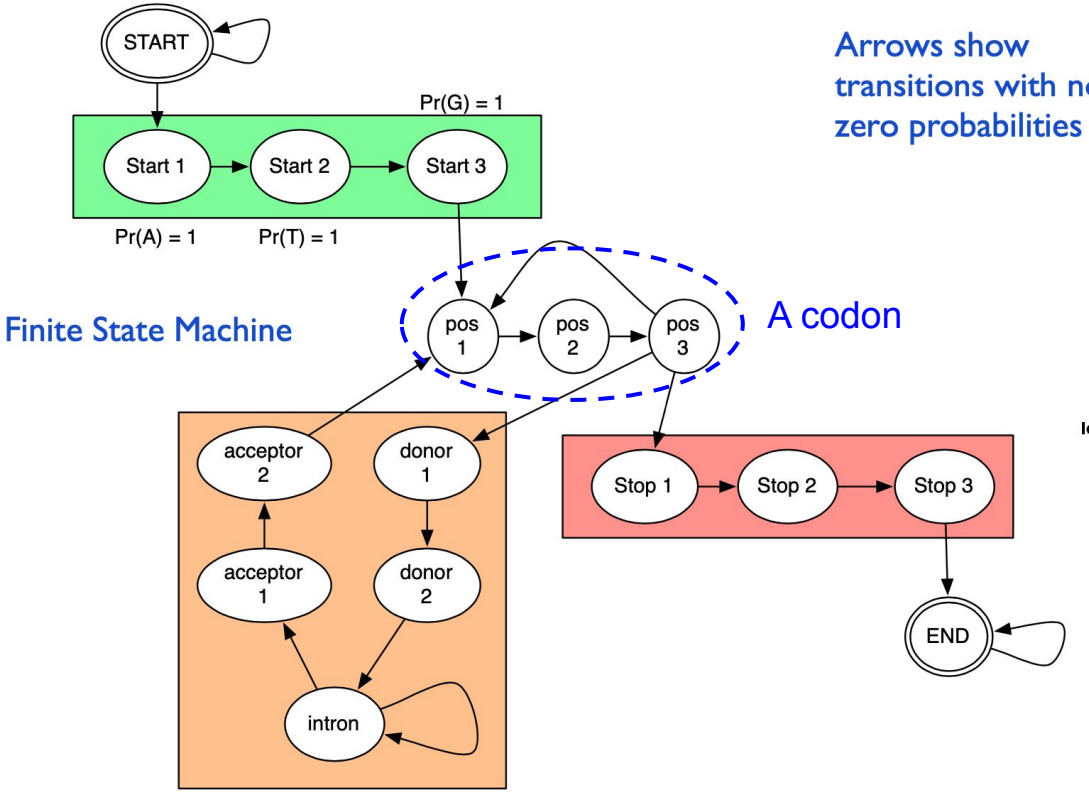
Multiplying leads to very small numbers: 0.1 x 0.1 x 0.1 x 0.1 x 0.1 = 0.00001

This can lead to underflow.

Taking logs and adding keeps numbers bigger.

An example of application in gene finding





Arrows show transitions with non-zero probabilities

Second Letter

1st letter	U		C		A		G		3rd letter
U	UUU	Phe	UCU	Ser	UAU	Tyr	UGU	Cys	U
	UUC		UCC		UAC		UGC		C
	UUA	Leu	UCA		UAA	Stop	UGA	Stop	A
	UUG		UCG		UAG	Stop	UGG	Trp	G
C	CUU	Leu	CCU	Pro	CAU	His	CGU	Arg	U
	CUC		CCC		CAC		CGC		C
	CUA		CCA		CAA	Gln	CGA		A
	CUG		CCG		CAG		CGG		G
A	AUU	Ile	ACU	Thr	AAU	Asn	AGU	Ser	U
	AUC		ACC		AAC		AGC		C
	AUA		ACA		AAA	Lys	AGA	Arg	A
	AUG	Met	ACG		AAG		AGG		G
G	GUU	Val	GCU	Ala	GAU	Asp	GGU	Gly	U
	GUC		GCC		GAC		GGC		C
	GUA		GCA		GAA	Glu	GGA		A
	GUG		GCG		GAG		GGG		G

The scoring problem with an HMM

Given an HMM (with known parameters $\boldsymbol{\lambda}$), what's the probability of an observed sequence O being generated from this HMM?

$$Pr(O|\lambda) = \sum_{Q} Pr(O, Q|\lambda)$$

The scoring problem with an HMM

Given an HMM (with known parameters **λ**), what's the probability of an observed sequence O being generated from this HMM?

$$Pr(O|\lambda) = \sum_{Q} Pr(O, Q|\lambda)$$

Recall that in the decoding problem, we want to find Q*:

$$Q* = \text{argmax}_{Q} Pr(O, Q|\lambda)$$

The Forward algorithm

Recall

$$w_t(i) = \max_{q_1,\ldots,q_{t-1}} Pr(o_1, o_2, ..., o_t, q_t = s_i)$$

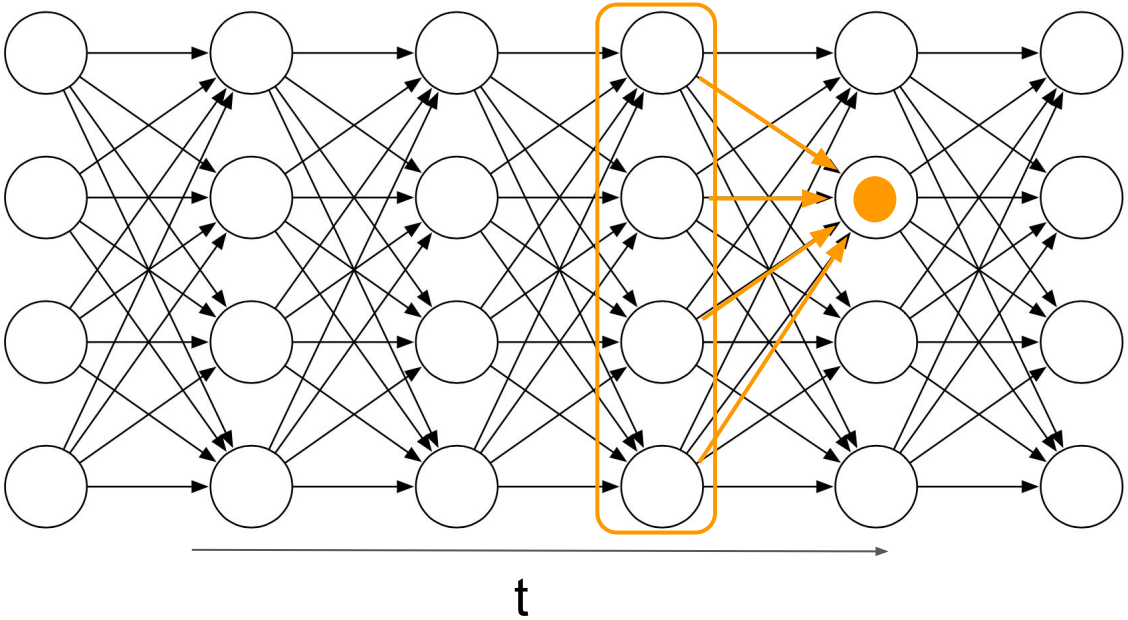
Now

$$\begin{aligned} \alpha_t(i) &= Pr(o_1, ..., o_t, q_t = s_i|\lambda) \\ &= \sum_{q_1,\ldots,q_{t-1}} Pr(o_1, ..., o_t, q_1, ..., q_{t-1}, q_t = s_i|\lambda) \end{aligned}$$

The Forward algorithm

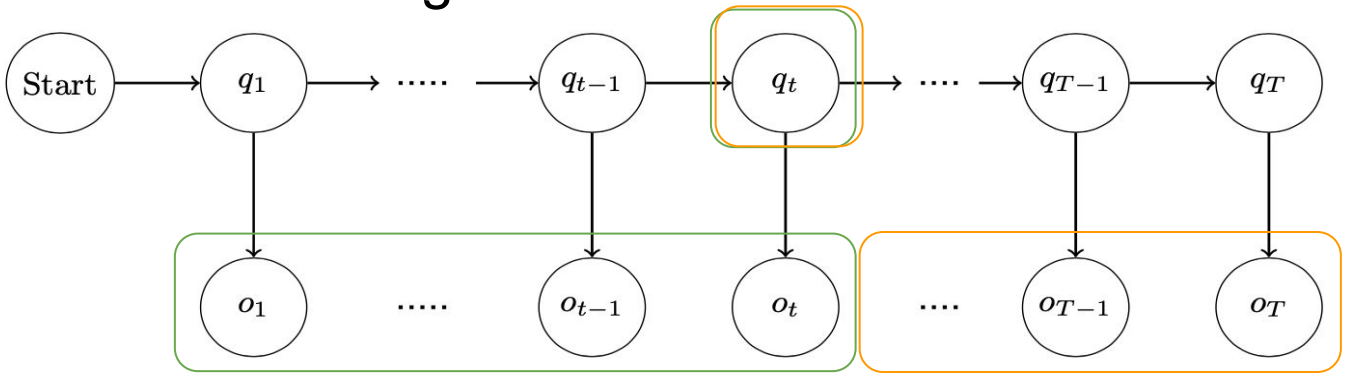
	Base case	Recurrence
Viterbi	$w_1(i) = \pi_i b_i(o_1)$	$w_t(i) = \max_{1 \le j \le N} w_{t-1}(j) a_{ji} b_i(o_t)$
Forward	$\alpha_1(i) = \pi_i b_i(o_1)$	$\alpha_t(i) = \sum_{1 \le j \le N} \alpha_{t-1}(j) a_{ji} b_i(o_t)$

The Forward algorithm



$$\alpha_t(i) = \sum_{1 \leq j \leq N} \alpha_{t-1}(j) a_{ji} b_i(o_t)$$

The Backward algorithm



$$\alpha_t(i) = p(o_1 \cdots o_t, q_t = s_i | \lambda)$$

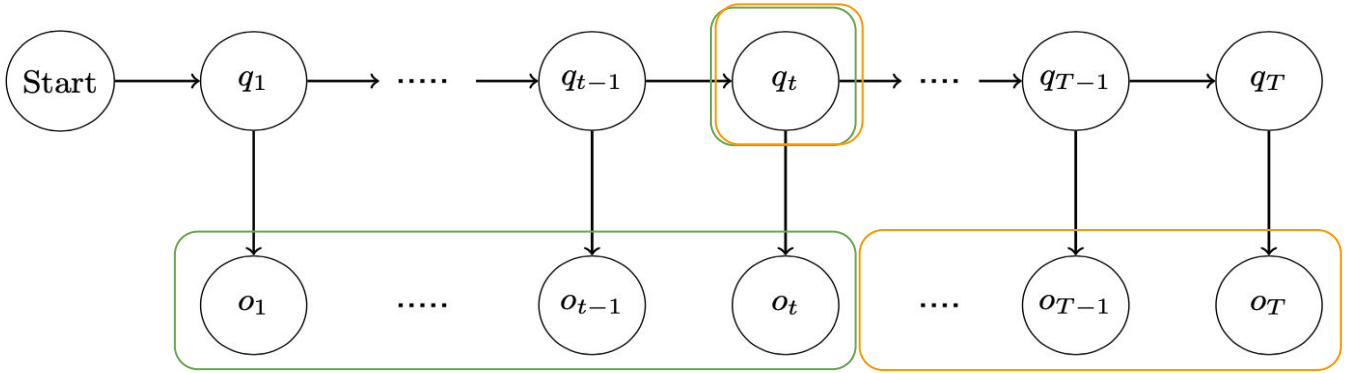
$$\alpha_t(i) = \sum_{1 \le j \le N} \alpha_{t-1}(j) a_{ji} b_i(o_t)$$

$$\beta_t(i) = p(o_{t+1} \cdots o_T, q_t = s_i | \lambda)$$

$$\beta_{t-1}(i) = \sum_{j=1}^{N} \beta_t(j) a_{ij} b_j(o_t)$$

$$\beta_T(i) = 1, 1 \le i \le N$$

Forward-Backward algorithm



$$\alpha_t(i) = p(o_1 \cdots o_t, q_t = s_i|\lambda)$$

$$\beta_t(i) = p(o_{t+1} \cdots o_T, q_t = s_i|\lambda)$$

$$\gamma_t(i) = Pr(q_t = s_i|O) = \frac{Pr(q_t = s_i, O)}{Pr(O)} = \frac{\alpha_t(i) \cdot \beta_t(i)}{\sum_{j=1}^{N} Pr(q_t = s_j, O)} = \frac{\alpha_t(i) \cdot \beta_t(i)}{\sum_{j=1}^{N} \alpha_t(j) \cdot \beta_t(j)}$$

Bayes' Theorem

Hidden Markov Models

Main algorithms with an HMM

- Viterbi algorithm → decoding/detection/matching problem
- Forward algorithm → scoring problem
- Backward algorithm → scoring problem
- Forward-backward / Baum-Welch algorithm → training problem

The training problem

When $\boldsymbol{\lambda}$ is unknown, we need to learn $\boldsymbol{\lambda}$ from data.

- When both observation sequence O and state sequence Q are known
  Maximum likelihood estimation

- When O is known but Q is unknown
  Expectation-Maximization (EM) algorithm

The training problem

When both O and Q are known:

$$a_{ij} = \frac{f_{ij}}{\sum_j f_{ij}} \qquad b_i(v_k) = \frac{h_{ik}}{\sum_k h_{ik}}$$

When O is known and Q is unknown:

The Baum-Welch algorithm (EM)

Baum-Welch algorithm

Objective:

$$\lambda^* = \arg\max_{\lambda} p(O|\lambda) = \arg\max_{\lambda} \sum_{Q} p(O, Q|\lambda)$$

Define two counters:

$$\boxed{\gamma_t(i) = p(q_t = s_i|O, \lambda)}$$
$$= \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)}$$

$$\boxed{\xi_t(i,j) = p(q_t = s_i, q_{t+1} = s_j|O, \lambda)}$$
$$= \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)}$$
$$= \gamma_t(i)\frac{a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\beta_t(i)}$$

Baum-Welch algorithm

Objective:

$$\lambda^* = \arg\max_{\lambda} p(O|\lambda) = \arg\max_{\lambda} \sum_{Q} p(O, Q|\lambda)$$

Expectation Maximization

- E-step:
  - Expected number of transitions from state s_i: $\sum_{t=1}^{T} \gamma_i(t)$
  - Expected number of transitions from state s_i to s_j : $\sum_{t=1}^{T} \xi_t(i,j)$
- M-step:
  - Update model parameters

Baum-Welch algorithm

M-step:

$$\pi_i = \text{ expected number of times in state } s_i \text{ at time } t = 1$$
$$= \gamma_1(i)$$

$$b_i(k) = \frac{\text{expected number of times in state } s_i \text{ observing } v_k}{\text{expected number of times in state } s_i}$$
$$= \frac{\sum_{t=1}^{T} I_{o_t = v_k} \gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)}$$

$$\begin{cases} 1, \text{ if } o_t = v_k \\ 0, \text{ else} \end{cases}$$

$$a_{ij} = \frac{\text{expected number of transitions from state } s_i \text{ to state } s_j}{\text{expected number of transitions from state } s_i}$$
$$= \frac{\sum_{t=1}^{T} \xi_t(i,j)}{\sum_{t=1}^{T} \gamma_t(i)}$$

Profile HMM for multiple sequence alignment

Profiles

- To summarize an MSA:

```
S1    ACG-TT-GA
S2    ATC-GTCGA
S3    ACGCGA-CC
S4    ACGCGT-TA
```

Column in the alignment

Character

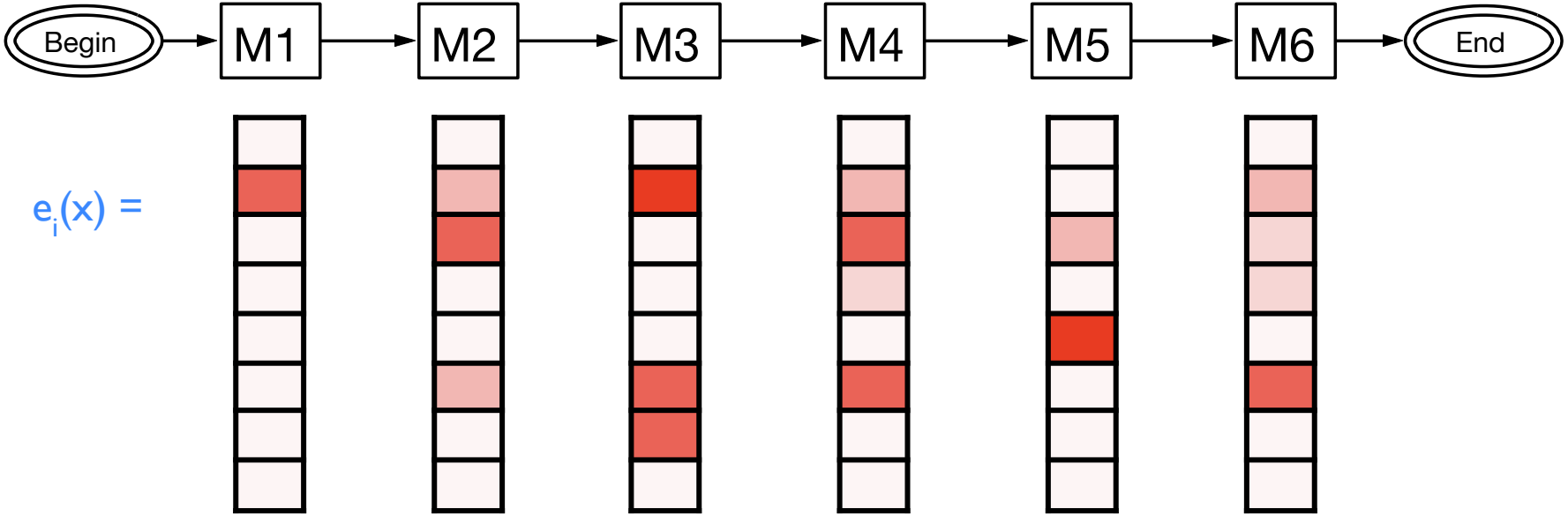
	1	2	3	4	5	6	7	8	9
A	1	0	0	0	0	0.25	0	0	0.75
C	0	0.75	0.25	0.5	0	0	0.25	0.25	0.25
G	0	0	0.75	0	0.75	0	0	0.5	0
T	0	0.25	0	0	0.25	0.75	0	0.25	0
-	0	0	0	0.5	0	0	0.75	0	0

Fraction of time given column had the given character

A Simple HMM

- A profile is equivalent to a simple HMM:

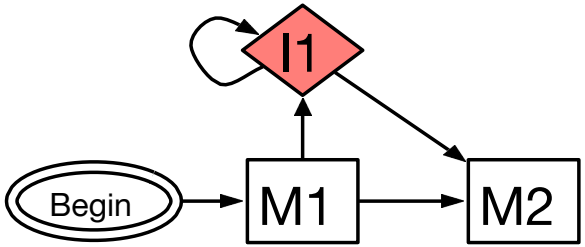
No choice about which state to visit.



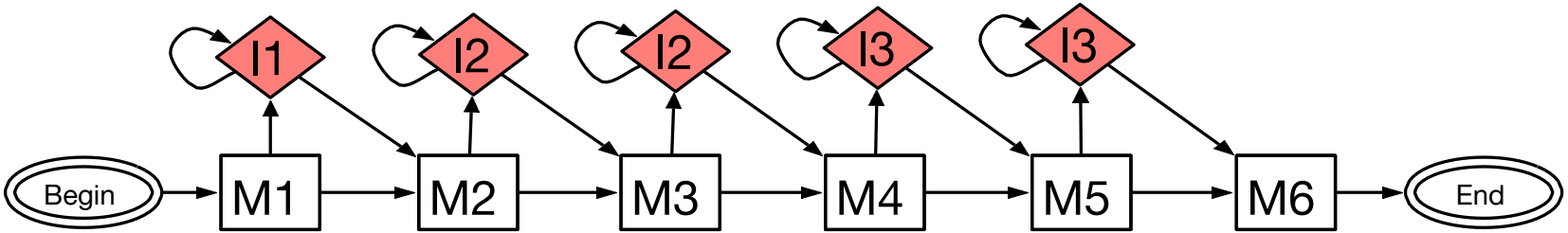
Emission probabilities given by Sequence Profile

Handling Insertions

characters in the sequence that are not in the profile



The "I" state allows any number of non-profile characters to be output.

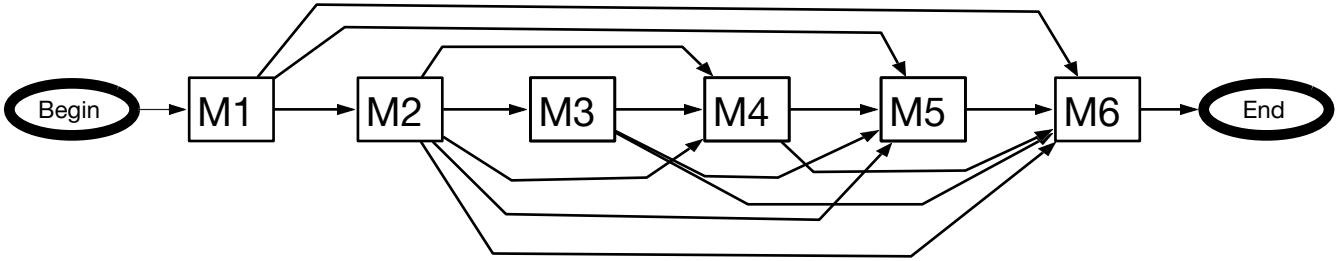


Handling Deletions

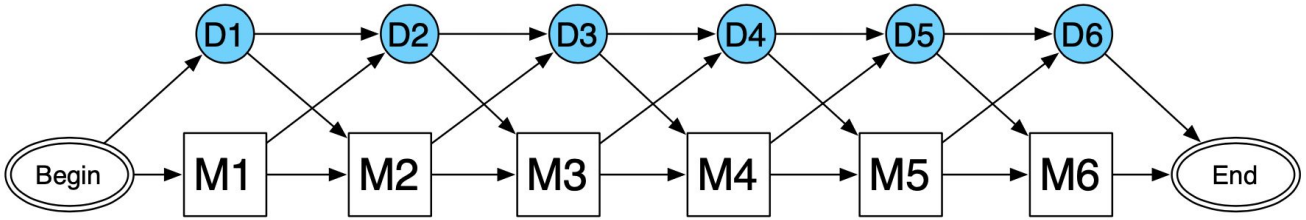
positions in the profile that are not matched in the string

We *could* add $O(n^2)$ edges that allow us to skip any number of match states.

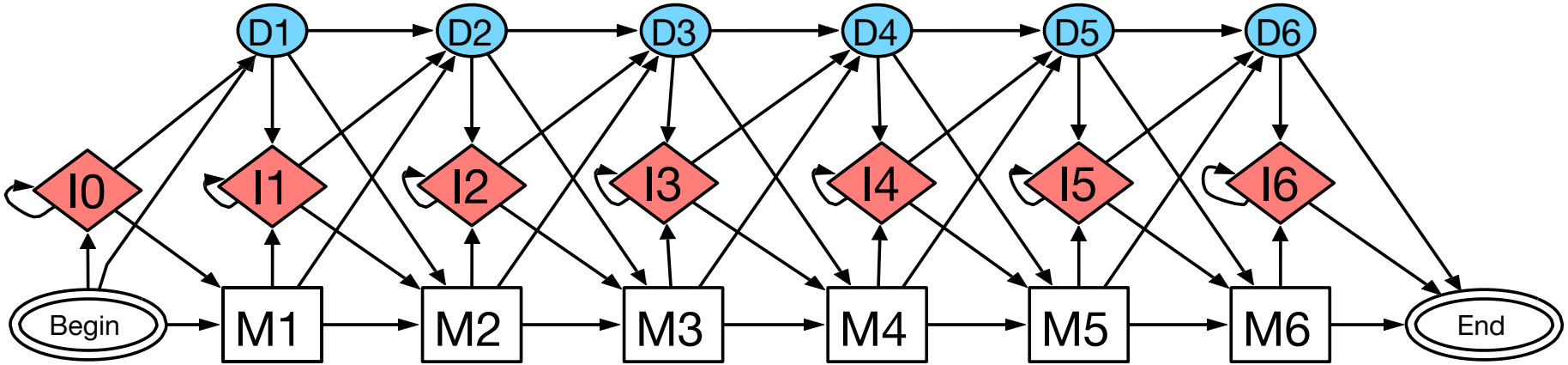
But this is too many edges.



Instead we add some delete states that don't emit any characters:

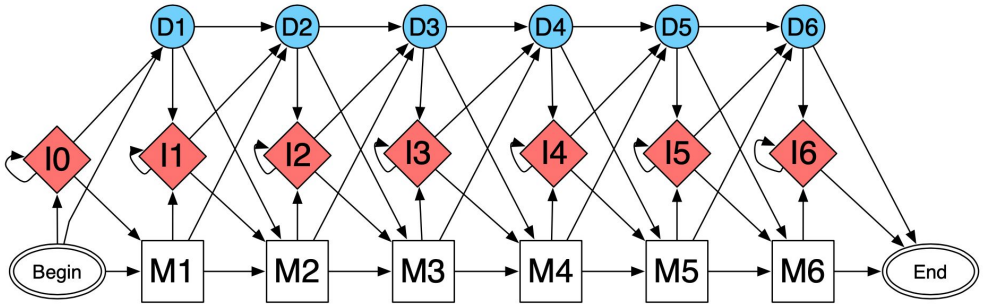


Combining Insertions & Deletions



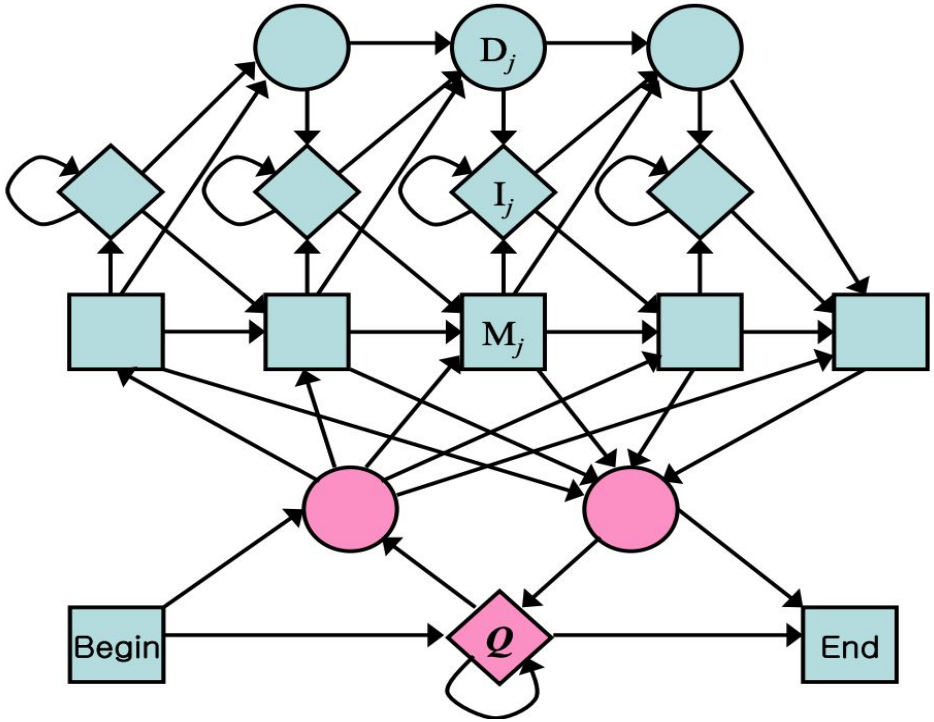
Example: MSA

```
A  T  G  C  T  G  -
M1 M2 I2 M3 M4 M5 D6

A  -  C  T  C  T  G  A  T
M1 D2 I2 I2 M3 M4 M5 I5 M6

C  T  C  A  -  T
M1 M2 M3 M4 D5 M6
```



	M1		M2		M3		M4		M5		M6	
	A		T	G–	C		T		G	–	–	
	A		–	TC	C		T		G	A	T	
	C		T	––	C		A		–	–	T	

Variant for non-global alignments



Profile HMM

- In the example above we assume that we already have a profile HMM with all parameters.
- In practice we need to first learn the parameters as well as decide the best number of matching states (that is, the length of the profile HMM) of the profile HMM.
- To infer the state sequences, there is an extended version of the Viterbi algorithm which we can use.

Summary

Main algorithms with an HMM

- Viterbi algorithm → decoding/detection/matching problem
- Forward algorithm → scoring problem
- Backward algorithm → scoring problem
- Forward-backward / Baum-Welch algorithm → training problem

Profile HMM

- An HMM model designed to perform MSA

Further readings

- Ghahramani, Z. Probabilistic machine learning and artificial intelligence. *Nature* **521**, 452–459 (2015)

- R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*,
  pp. 46–66 for algorithms on a standard HMM model including parameter estimation;
  pp. 102–113 for constructing a profile HMM and Forward and Viterbi algorithms on a profile HMM;
  pp. 149–154 for applying profile HMM to MSA, and the training (parameter estimation) of a profile HMM.
  pp. 323–325 for the EM (expectation maximization) algorithm.