Stavy vyšších rádov

Rád 0: emisná tabuľka e určuje $\Pr(S_i|A_i)$

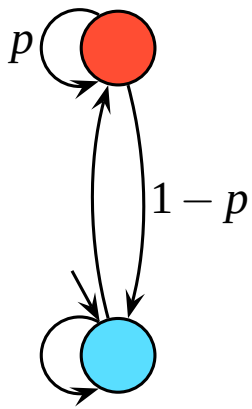
Rád 1: e určuje $\Pr(S_i|A_i, S_{i-1})$

A_i	S_{i-1}	a	c	g	t
	a	0.24	0.23	0.34	0.19
	c	0.30	0.31	0.13	0.26
	g	0.27	0.28	0.28	0.17
	t	0.13	0.28	0.38	0.21
	a	0.30	0.18	0.27	0.25
	c	0.32	0.28	0.06	0.35
	g	0.27	0.22	0.27	0.24
	t	0.20	0.21	0.26	0.33

...

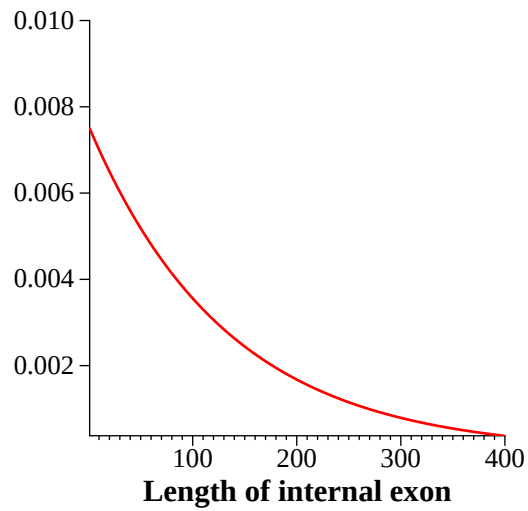
Na charakterizovanie exónov, intrónov atď používame rád 4-5.

Modeling length distributions

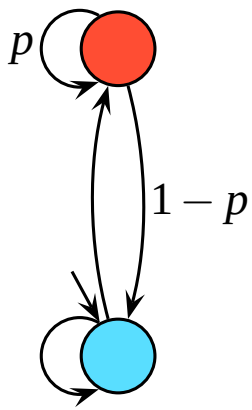


What is the length distribution of red segments generated by the model?

$$\Pr(\text{red segment of length } \ell) = p^{\ell-1}(1-p)$$

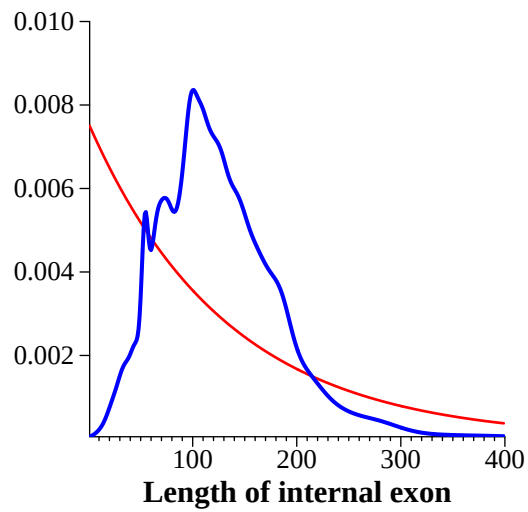


Modeling length distributions



What is the length distribution of red segments?

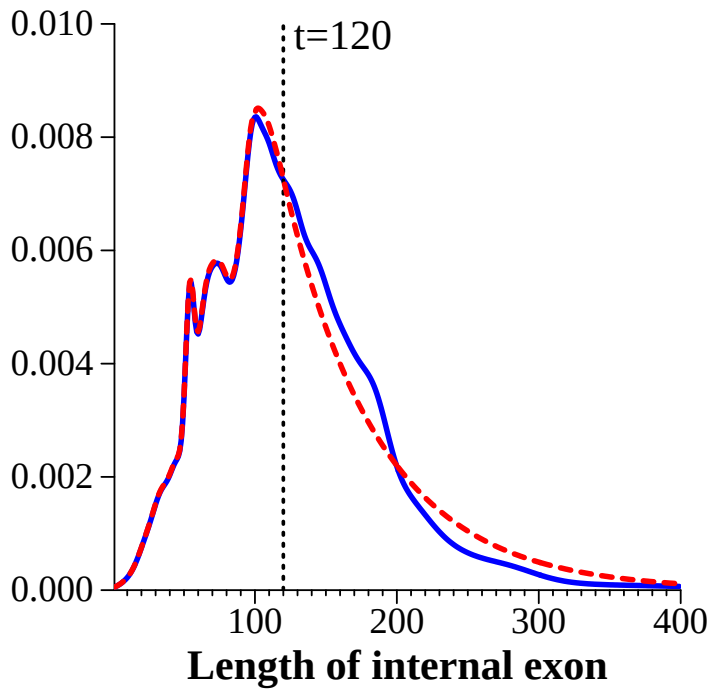
$$\Pr(\text{red segment of length } \ell) = p^{\ell-1}(1-p)$$



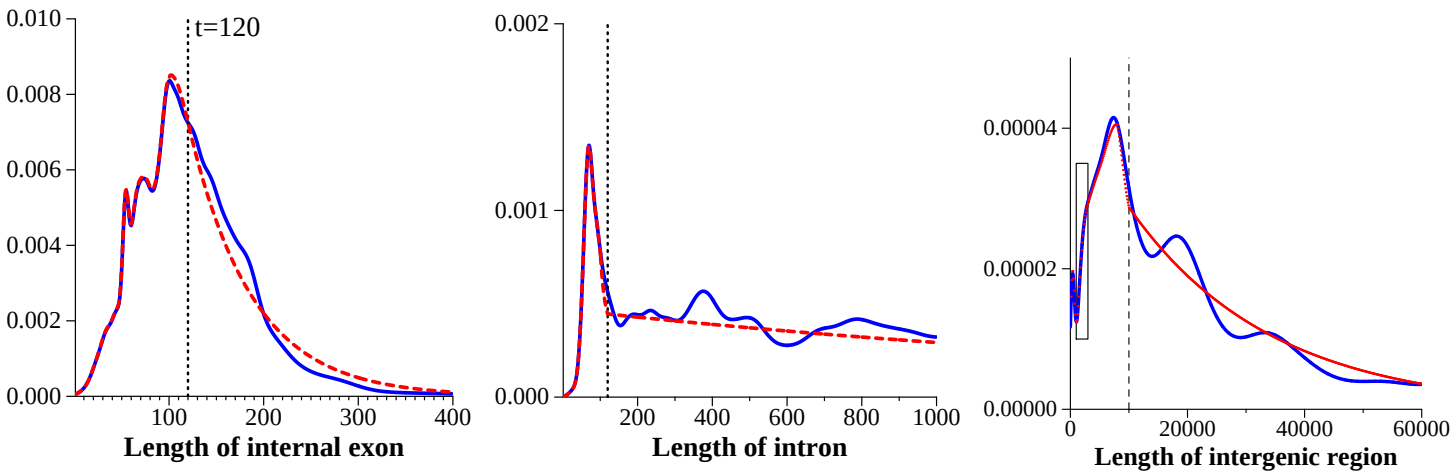
- Geometric distributions: bad model of real world; $O(n)$ time [Viterbi 1967]
- Arbitrary distributions: faithful model; $O(n^2)$ time [Rabiner 1989]
- Will show: geometric tails: better model; $O(nt)$ time.

Geometric tail distributions

- head (lengths $< t$): specify explicitly
- tail (lengths $\geq t$): geometrically decaying



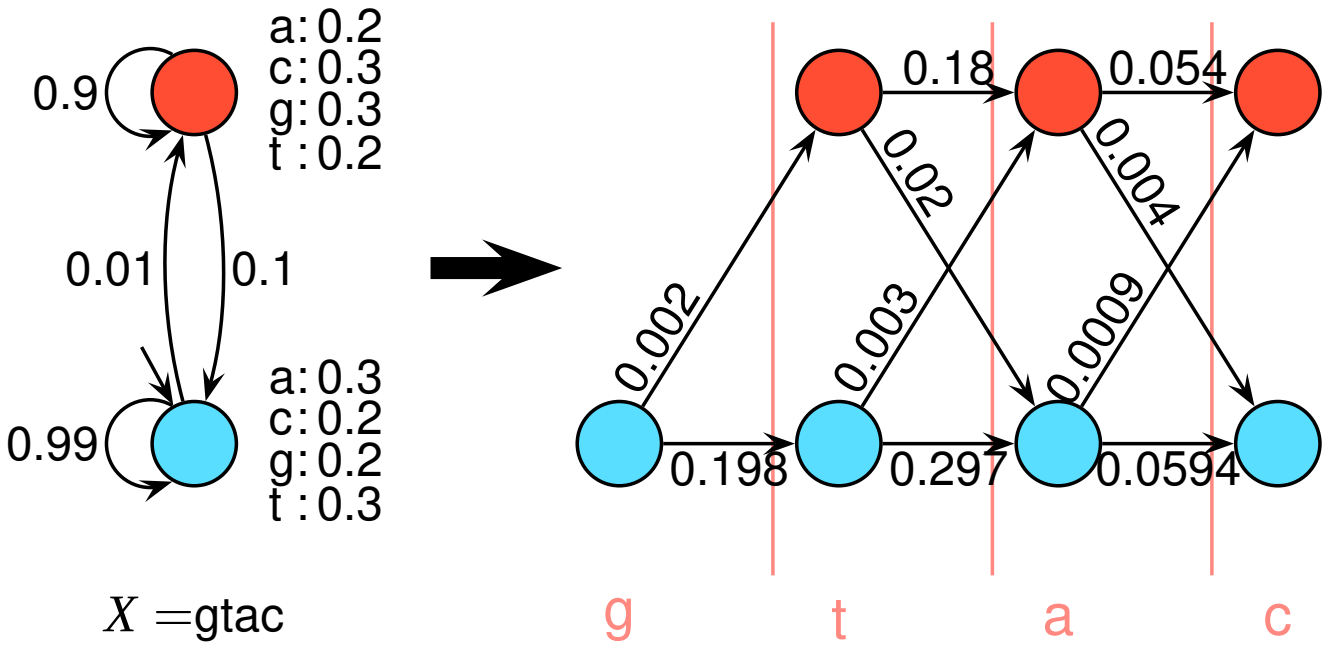
Geometric tail is a good approximation



- $O(nt)$ works for exons and introns
- Intergenic regions ($t \approx 10000$):
  - Use less accurate approximation
  - Better running time: $O(n\sqrt{t})$

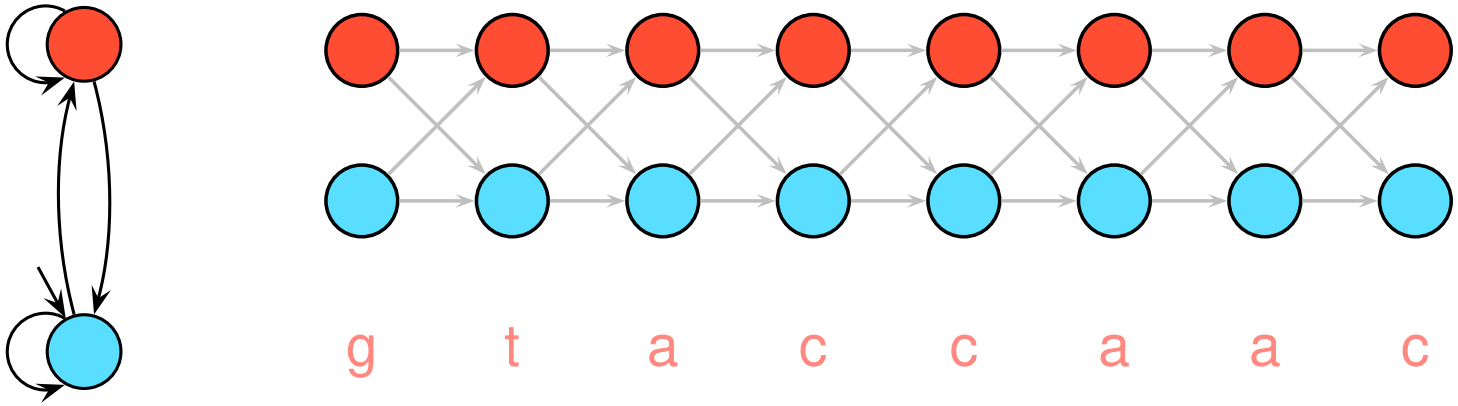
Viterbi algorithm: the most probable state path [Viterbi 1967]

(but geometric length distributions only)

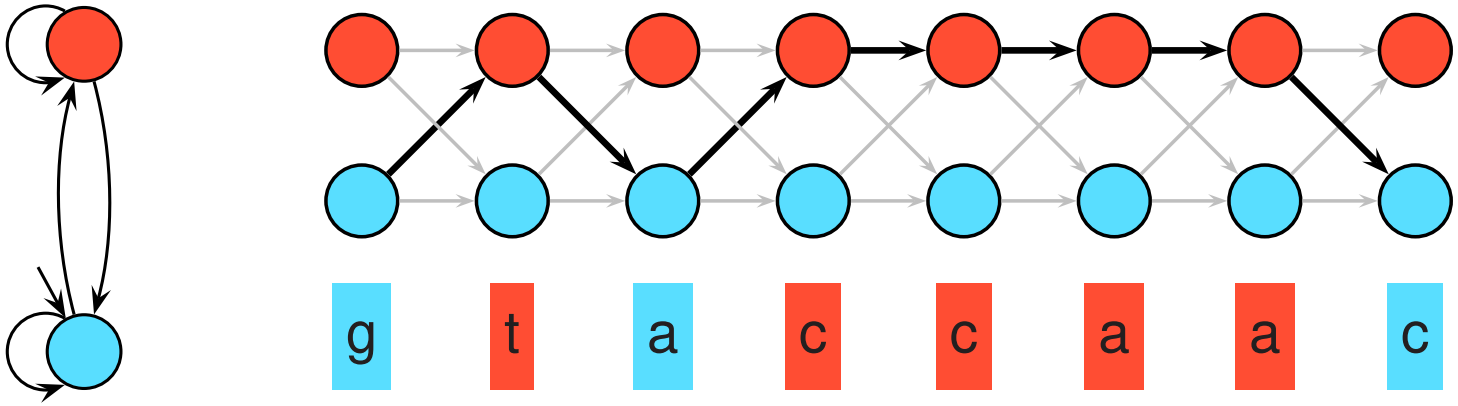


- Take $-\log$ of weights and compute shortest path in DAG
- Running time: $O(n)$

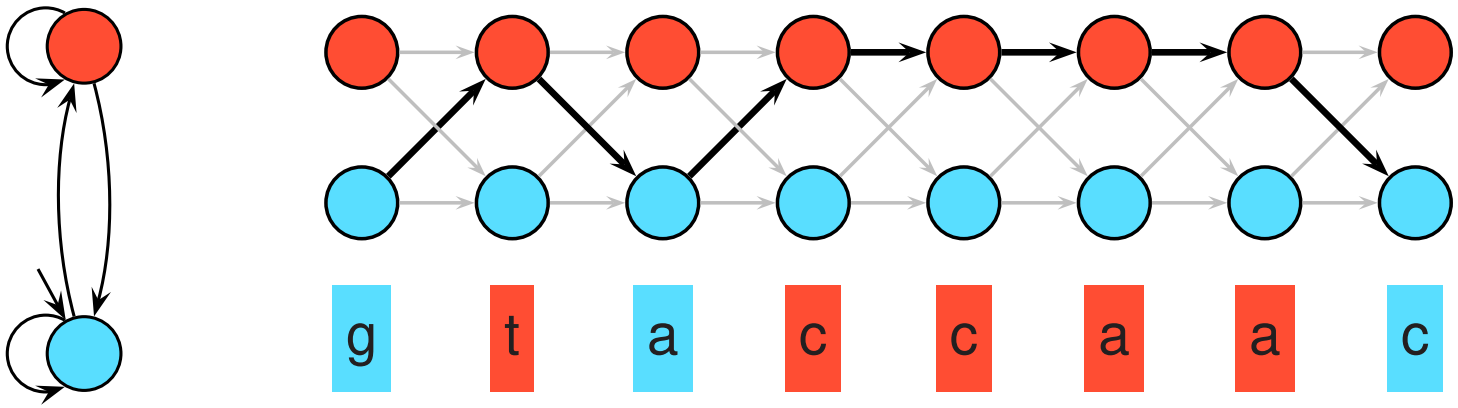
Viterbi algorithm – $O(n)$ time



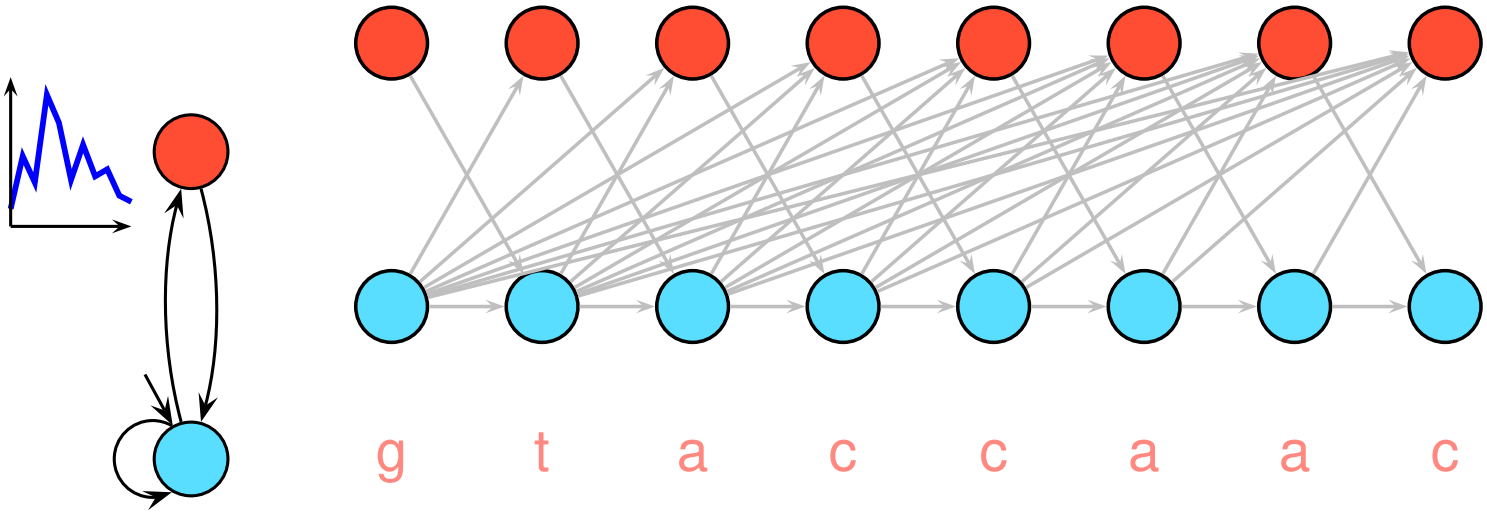
Viterbi algorithm – $O(n)$ time



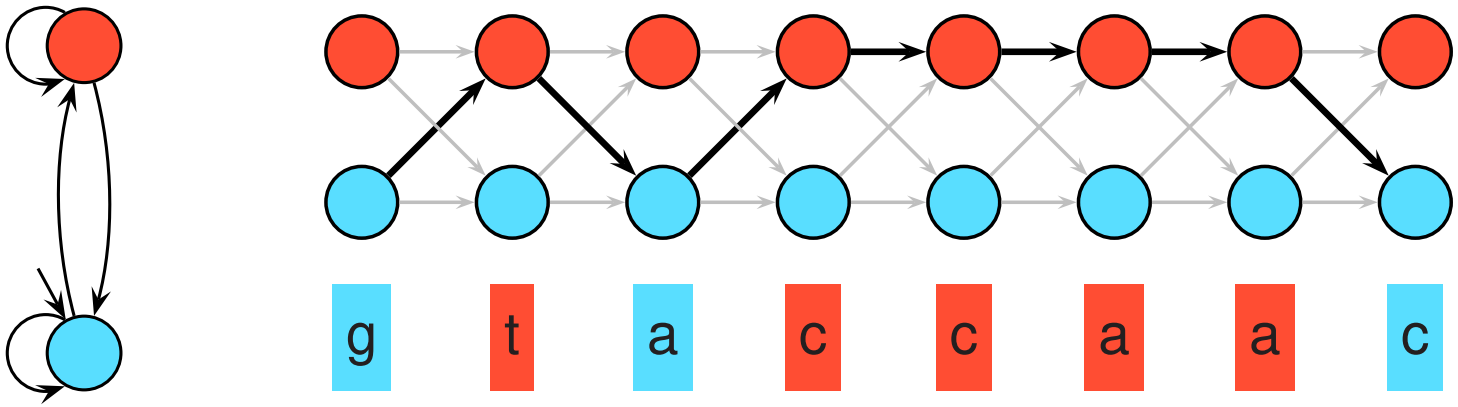
Viterbi algorithm – $O(n)$ time



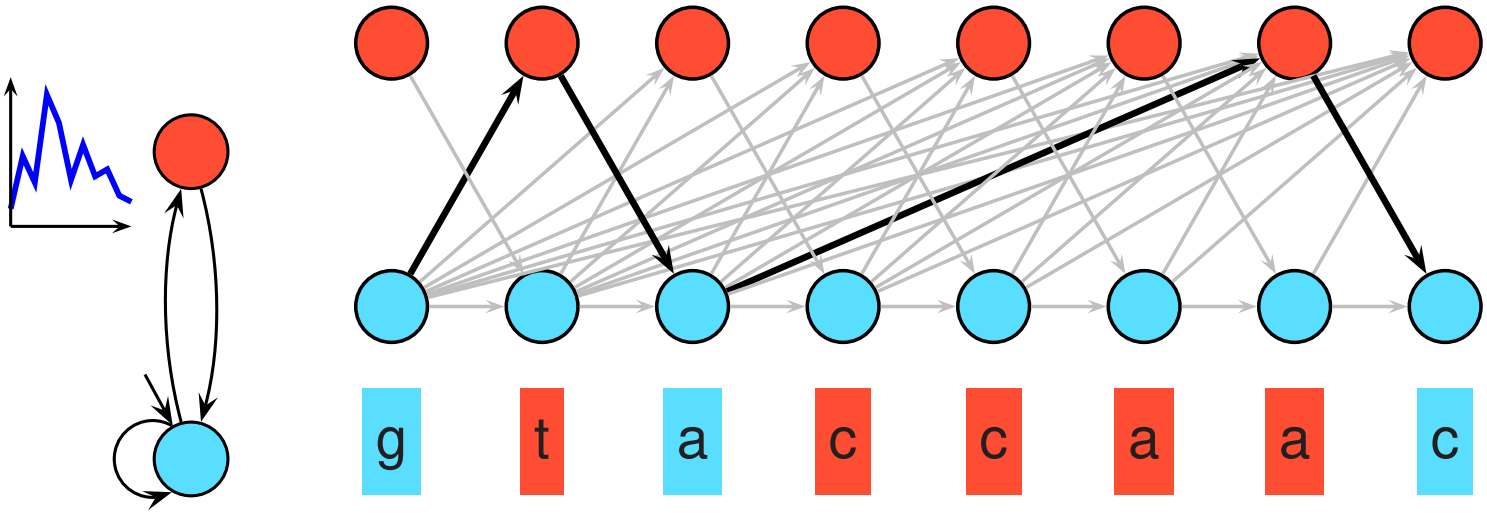
Generalized Viterbi algorithm [Rabiner, 1989]



Viterbi algorithm – $O(n)$ time

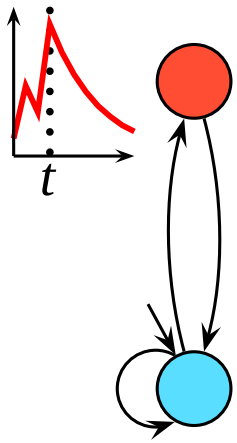


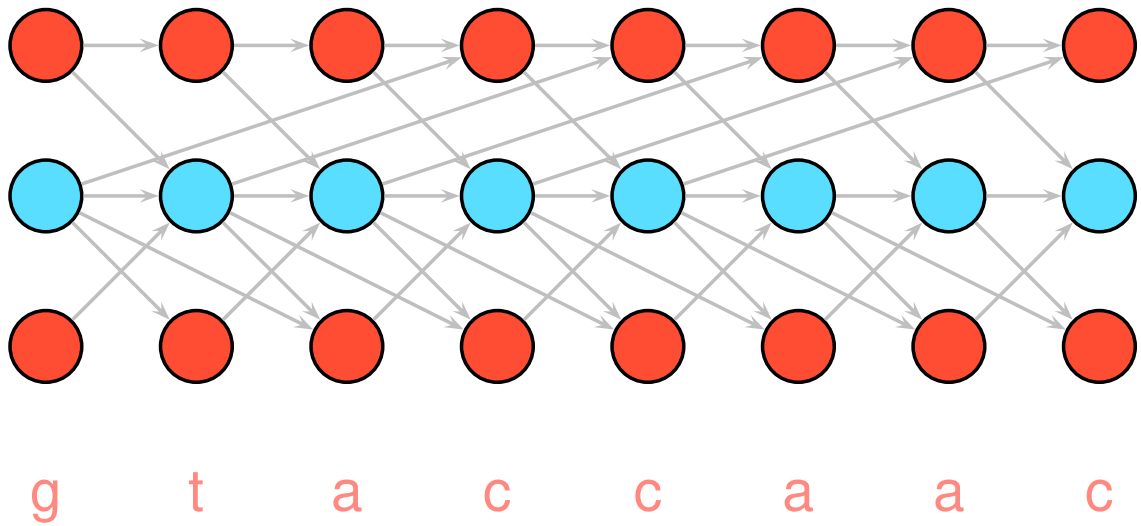
Generalized Viterbi algorithm – $O(n^2)$ time



Combining two algorithms

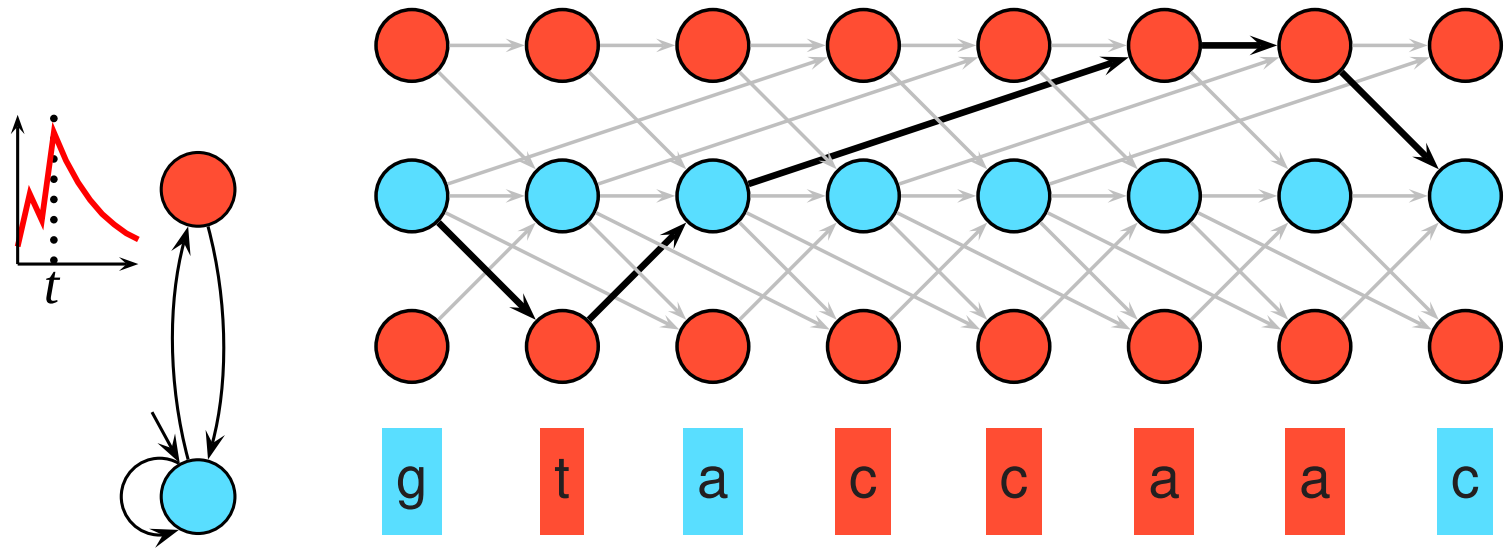
Assumption: Length distribution – geometric tail starting at $t = 3$





Combining two algorithms

Assumption: Length distribution – geometric tail starting at $t = 3$



Running time: $O(nt)$

Modeling length distributions – summary

- Change from $O(n)$ to $O(n^2)$ – any length distribution you want
- Instead: trade-off between model faithfulness and running time
  - Approximate by geometric tail: $O(nt)$ time
  - If t is too large: $O(n\sqrt{t})$ time

Signals in gene finding

- Conserved sequences of fixed length that appear at boundaries of exons and other important places.
- Our interest: replace section of HMM by more realistic generative model giving
  - High probability to actual signals
  - Low probability to decoys (sites which are not signals)

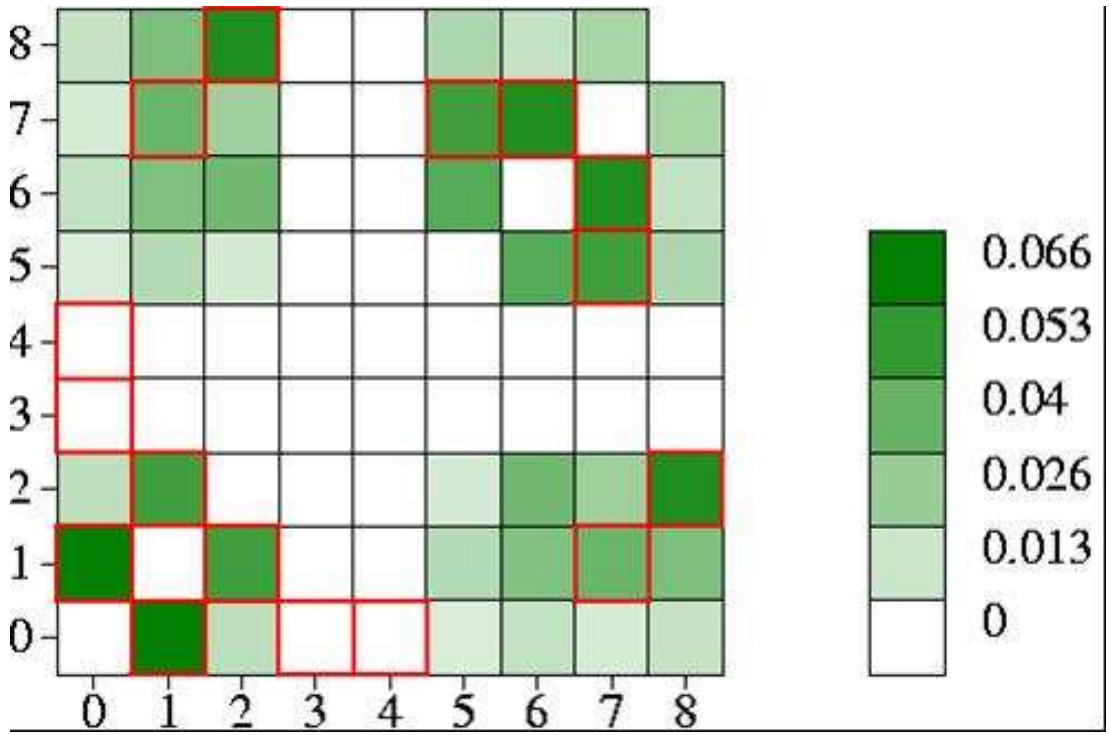
Example: Position Weight Matrix (PWM)

	0	1	2	3	4	5	6	7	8
A	.38	.62	.12	0	0	.71	.73	.11	.21
C	.31	.10	.04	0	0	.02	.06	.06	.10
G	.18	.12	.77	1	0	.24	.08	.75	.14
T	.13	.16	.07	0	1	.03	.13	.08	.55

Main challenge: dependencies within signal

How much more information,

if we consider pairs instead of individual positions?



(darker is better)

Signals as DAGs

- vertices = signal positions
- edges = "dependencies" between positions

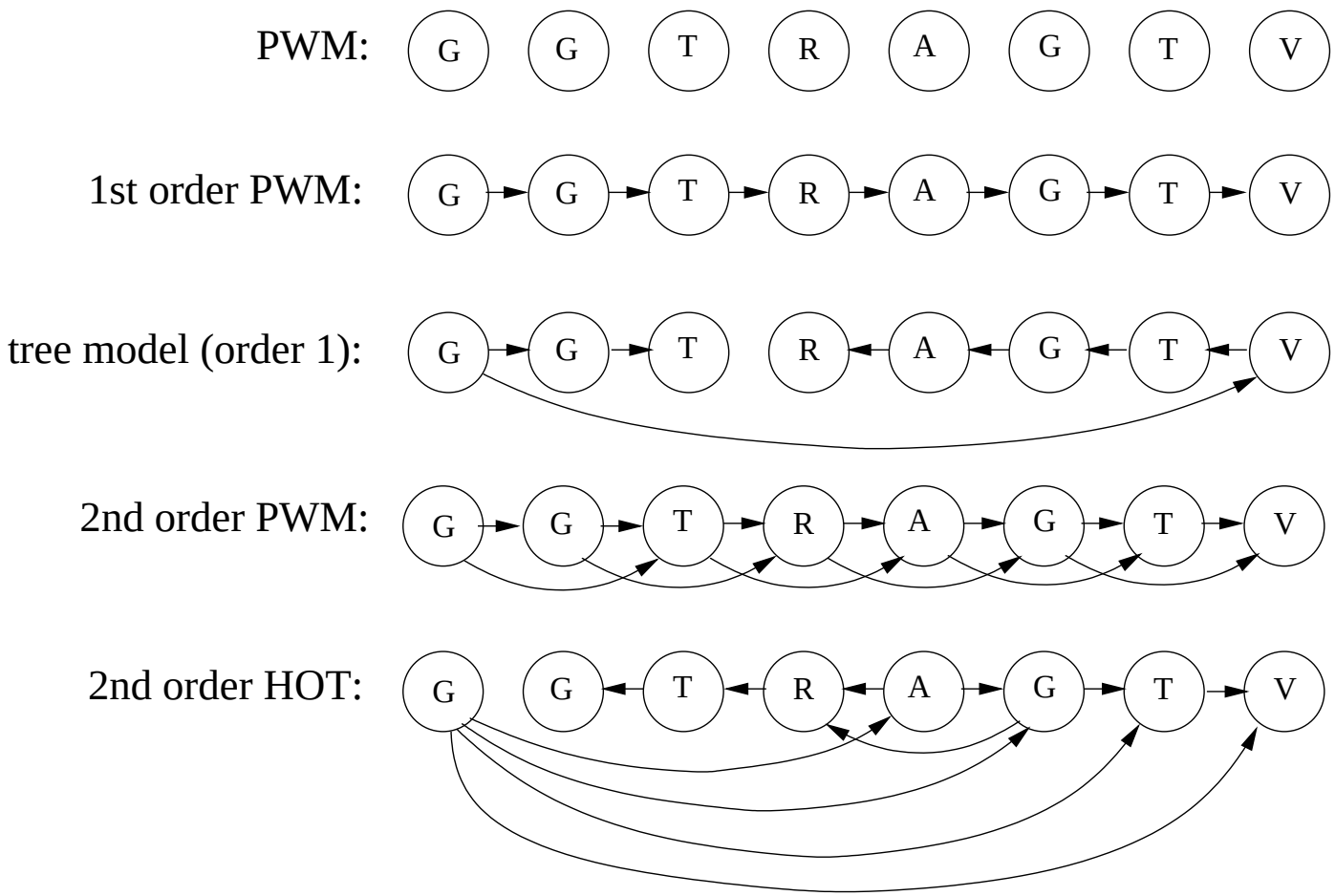
To generate signal by model M

- Generate characters at signal positions in topological order
- Model specifies for each position i:

$$\Pr[S_i = x_i \mid S_{j_1} = x_{j_1}, \ldots, S_{j_k} = x_{j_k}],$$

where $j_1, \ldots, j_k$ are predecessors of i in the DAG

Examples of generative models for donor signal



Estimating model parameters (training)

Maximum-likelihood approach: Find model that maximizes joint probability of generating all signals in the training set.

1. determine best topology
2. compute probability tables (count frequencies – easy)

Note:

- The amount of data needed to train model given by a graph
  – grows exponentially with maximum in-degree
  – does not depend on number of vertices or topology
- Limit in-degree to avoid overfitting

Training HOT models

Task: Given a training set $S_1, \ldots, S_\ell$, find model topology with maximum in-degree k that maximizes likelihood of $S_1, \ldots, S_\ell$.

Optimization problem:

1. Create a hypergraph $\mathcal{H}$:
   – vertices = signal sites
   – hyperedge (T, h) for each h, T, s.t. $0 \leq |T| \leq k$
2. Compute cost of hyperedge (T, h) as $w_{T,h} = H(T \cup \{h\}) - H(T)$, where $H(X)$ is entropy over signal positions X
3. Find minimum directed spanning hypertree $\mathcal{M}$
   - For $k = 1$: Chow-Liu trees [Chow, Liu 1968]
   - For $k \geq 2$: NP-hard
4. Underlying graph of $\mathcal{M}$ is the optimal topology of HOT-k model

Training HOT models by integer programming

$b_{i,j}$ – ordering of sites in generative process

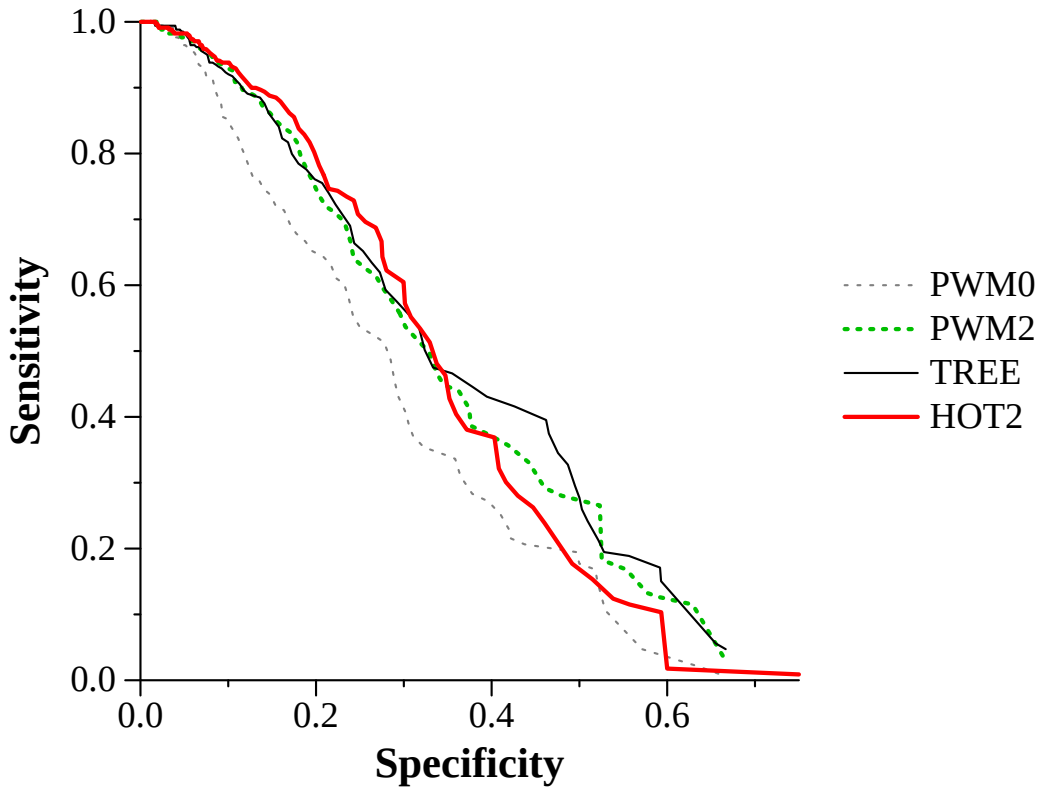
$a_{T,h}$ – was hyperedge (T, h) chosen?

$$\min \sum_{E=(T,h)} w_{T,h} a_{T,h}, \qquad \text{subject to:}$$

$$\begin{aligned}
b_{i,j} + b_{j,i} &= 1, \text{ for all pairs } i \text{ and } j, \\
b_{i,j} + b_{j,k} + b_{k,i} &\leq 2, \text{ for all triplets } i, j \text{ and } k, \\
a_{T,h} &\leq b_{x,h}, \text{ for all hyper edges } E = (T, h) \text{ and nodes } x \text{ in} \\
\sum_{E:E=(T,h)} a_{T,h} &= 1, \text{ for all nodes } h, \\
a_{T,h} &\in \{0, 1\}, \text{ for all hyperedges } E = (T, h), \\
b_{i,j} &\in \{0, 1\}, \text{ for pairs of nodes } i \text{ and } j.
\end{aligned}$$

Using signal models for discrimination

- Choose a threshold score for "true" predicted donor site
- Changing the threshold balances sensitivity vs. specificity

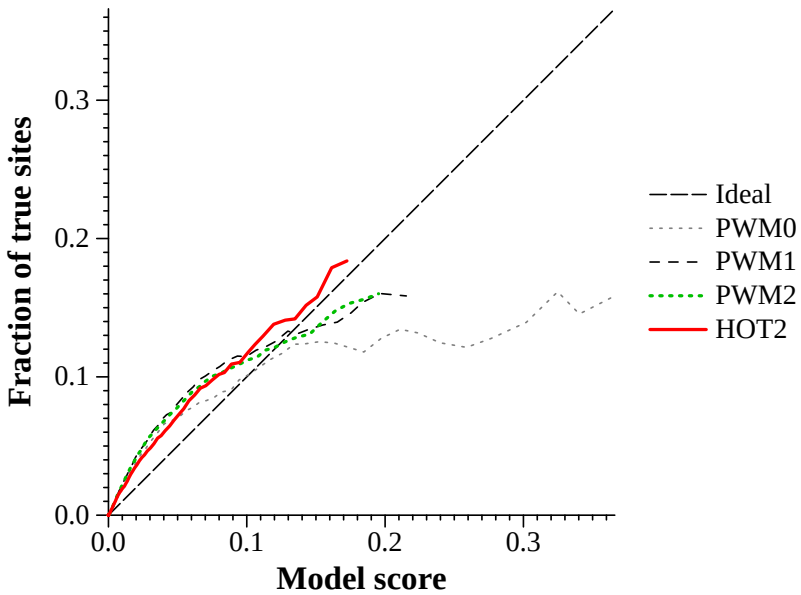


Reliability of the score

- If used in HMMs, the models are NOT used for discrimination
- Rather the scores are used in HMM inference

$\Rightarrow$ need to use different measure to evaluate signal models

- Score: given sequence S, estimate probability that S is a signal
- Can we rely on the value of the score?



Example:

- $Q_{0.066}$ = set of positions with score ≈ 0.066 in HOT2 (258 samples)
- 20 true donors in $Q_{0.066}$
- This is 7.8% $(20/258)$

Model	Correlation
PWM0	0.827
PWM1	0.890
PWM2	0.911
HOT2	0.955

Signals in gene finding – summary

- Main problem: how to capture dependencies between non-adjacent signal positions
- Traditional tradeoff: how many dependencies we can capture without running into overfitting
  (limited in-degree of vertex in the model)
- Many models can be represented as hypertrees (or Bayesian networks with fixed in-degree)
- Training HOT models is hard in general; however integer programming does reasonable job
- Both discrimination power and reliability of score are important measure of model performance

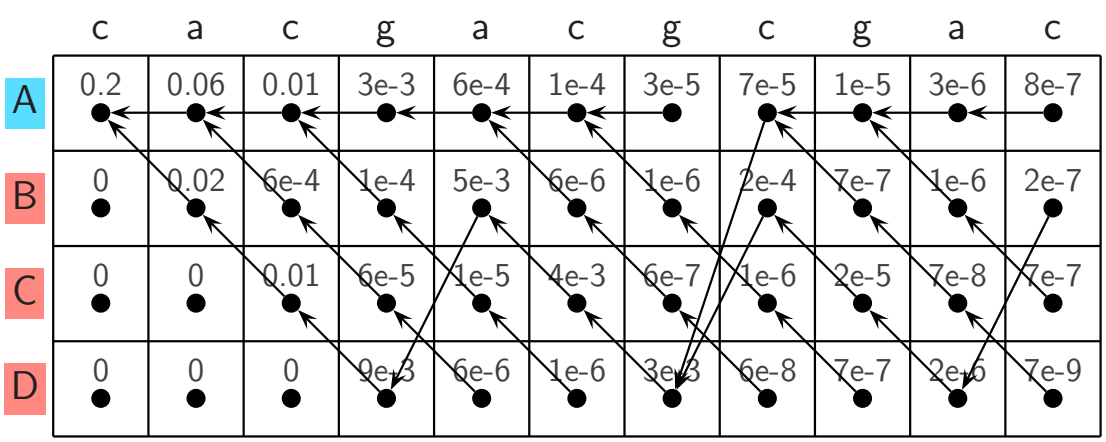
Viterbi algorithm

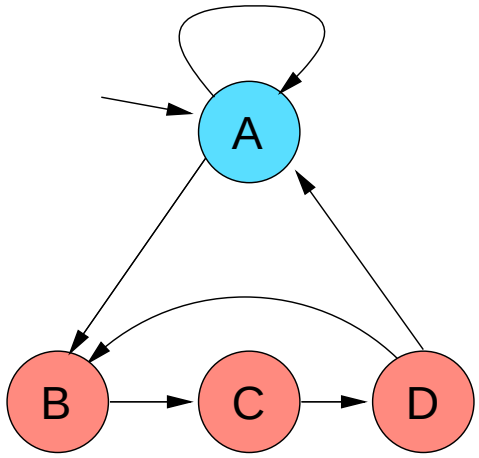
Dynamic programming:

- $P[i,j]$ — probability of generating $x_1, \ldots, x_i$ and ending in state j
- $P[i,j] = \max_k P[i-1,k] \cdot t(k,j) \cdot e(x_i,j)$

  $t(k,j)$: probability of transition $k \to j$

  $e(x_i,j)$: probability of emission of x_i in state j
- **For each $P[i,j]$ need to remember best previous state k (back pointers)**





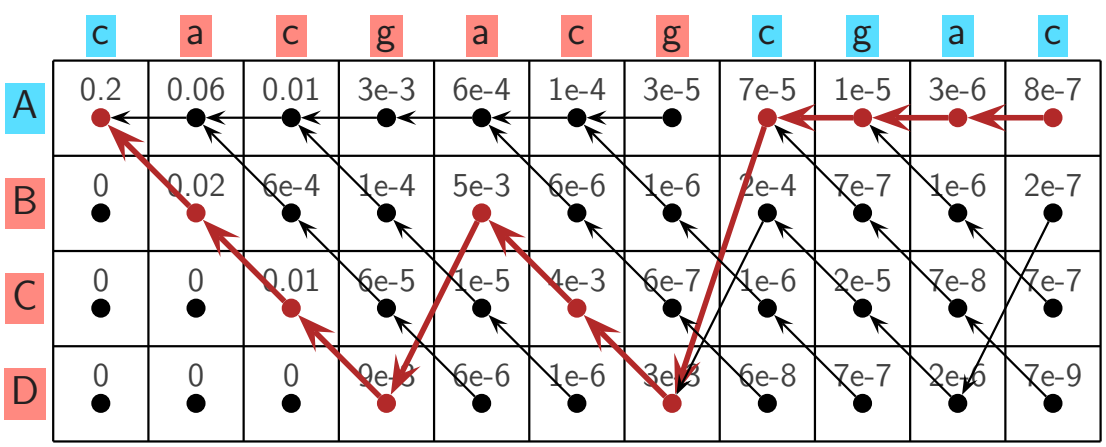
Viterbi algorithm

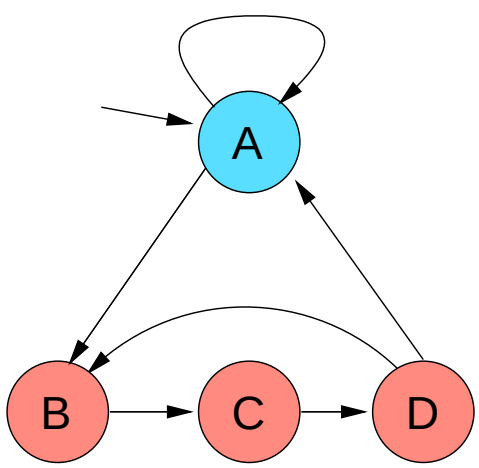
Dynamic programming:

- $P[i,j]$ — probability of generating $x_1, \ldots, x_i$ and ending in state j
- $P[i,j] = \max_k P[i-1,k] \cdot t(k,j) \cdot e(x_i,j)$

  $t(k,j)$: probability of transition $k \to j$

  $e(x_i,j)$: probability of emission of x_i in state j
- **For each $P[i,j]$ need to remember best previous state k (back pointers)**





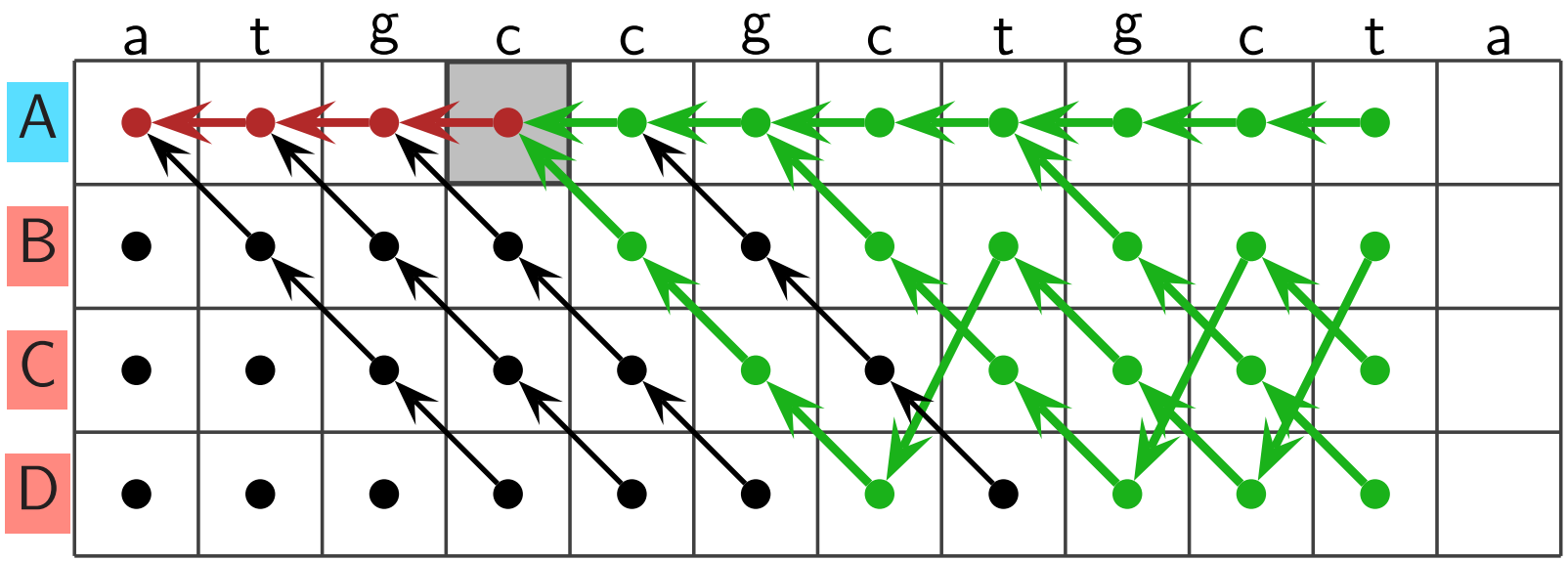
Space requirements of the Viterbi algorithm

- Need to remember back pointers for the whole sequence.
- **Example:** gene finding on 250 MB sequence
  with 100 state HMM $\Rightarrow$ **25 GB of internal memory**

Some solutions:

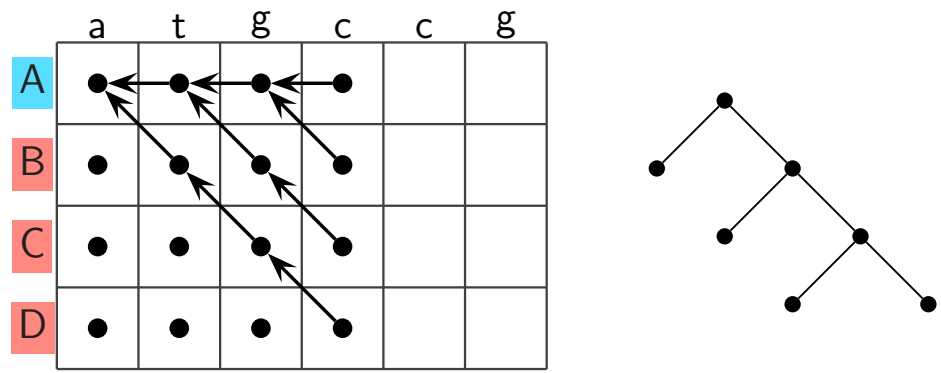
- **Split the sequence up into smaller chunks**
  - How to resolve "mismatches" on boundaries?
  - Cannot always give the optimal solution
- **Check pointing** [Grice et al. 1997]
  - $O(\sqrt[L]{n}m)$, factor L slow down
  - PLUS: Need to store the complete sequence all the time
- **This paper: on-line algorithm**,
  needs variable-size buffer, small most of the time

On-line Viterbi algorithm

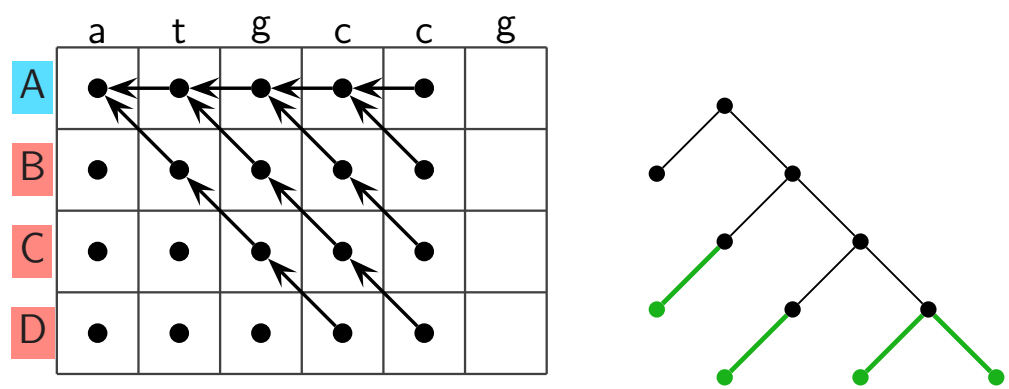


- Detect **coalescence points** efficiently
- Output the path left of the coalescence point
- Remove all the data left of the coalescence point

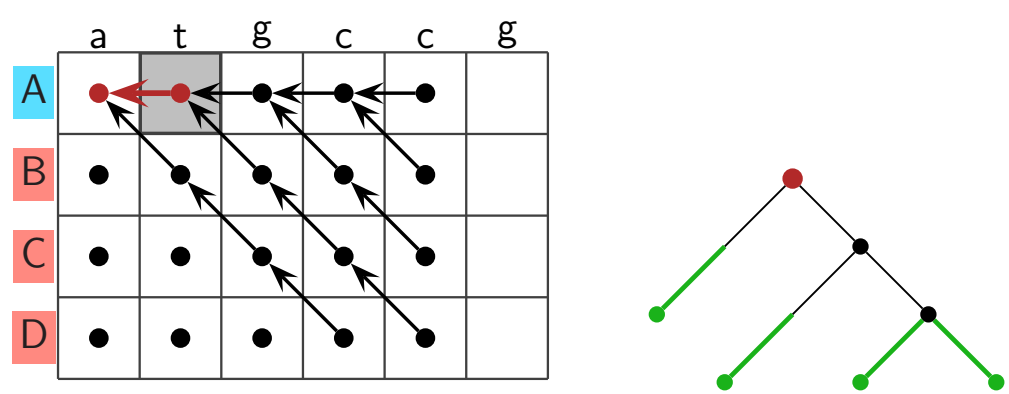
Efficient detection of coalescance points:
Maintain **compressed** backpointer tree



In each step, add **newly created** back pointers. . .



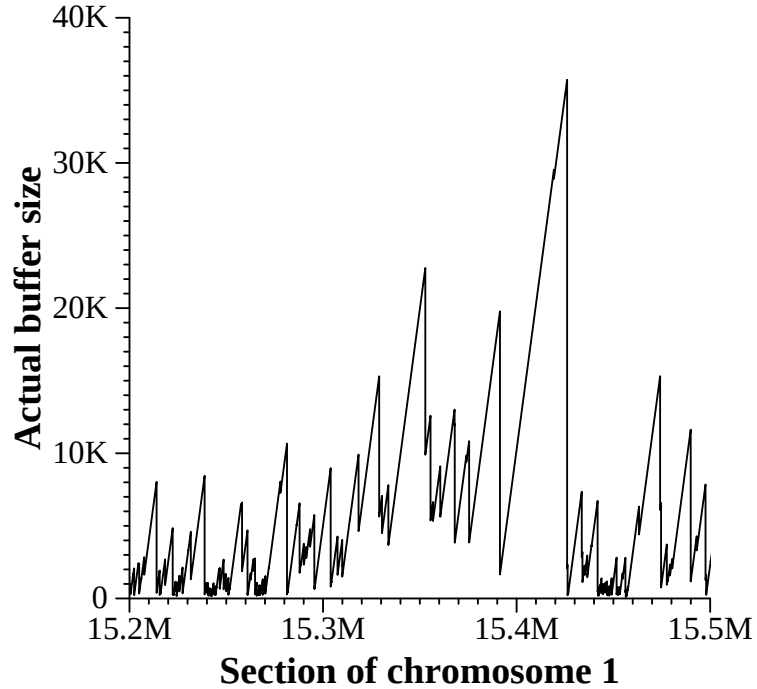
. . . remove unused branches, compress non-branching vertices.
Overhead: $O(m)$ space, $O(m)$ time in each step, $\approx 5\%$ slowdown



How much memory do we need?

Gene finding experiment:

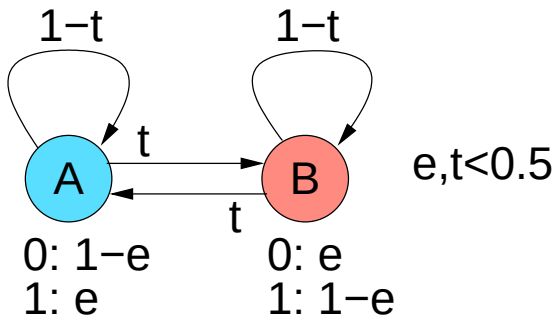
- 256 state HMM
- 20 MB human sequences
- Average buffer size: ≈ 11 kB
- Maximum buffer size: ≈ 222 kB
- Average **maximum** buffer size: ≈ 100 kB



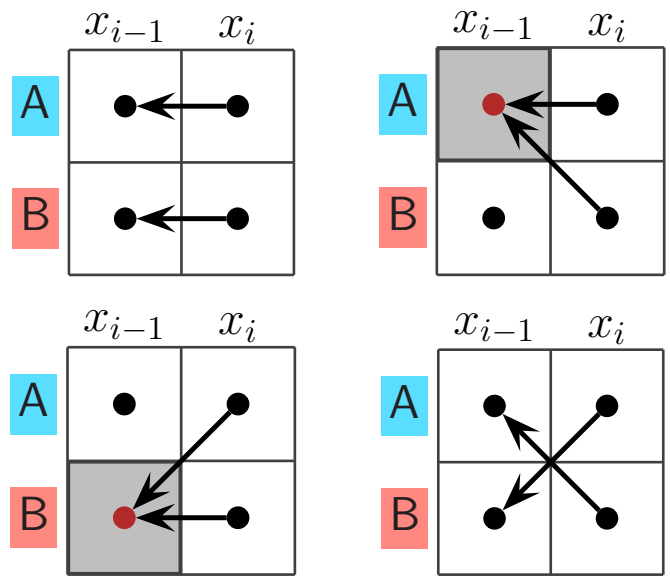
Estimating expected maximum buffer size:

- **Talk:** 2-state symmetric HMM, i.i.d. sequence, $O(\log n)$
- **Paper:** 2-state general HMM, HMM generated sequence

2-state symmetric HMM:



Possible backpointer configurations:



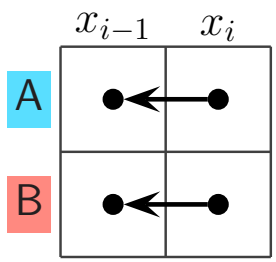
Which configuration?

depends on **ratio of** $P[i-1, A]$ **and** $P[i-1, B]$

Configurations of back-pointers

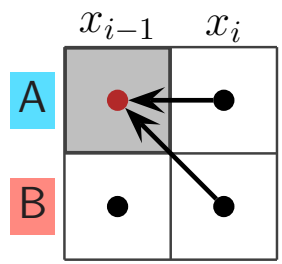
$$P_{i-1} = \frac{\log P[i-1,A] - \log P[i-1,B]}{\log(1-e) - \log e} \qquad L = \left\lceil \frac{\log(1-t) - \log t}{\log(1-e) - \log e} \right\rceil$$

$\boxed{-L < P_{i-1} < L}$



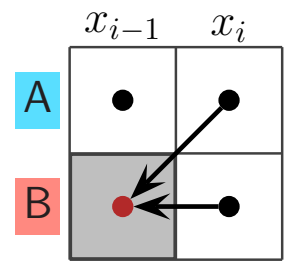
$P_i := P_{i-1} \pm 1,$
$+1$ if $x_i = 0,$
-1 if $x_i = 1$

$\boxed{P_{i-1} \geq L}$



coalescence!
$P_i := L \pm 1$
depending on x_i

$\boxed{P_{i-1} \leq -L}$



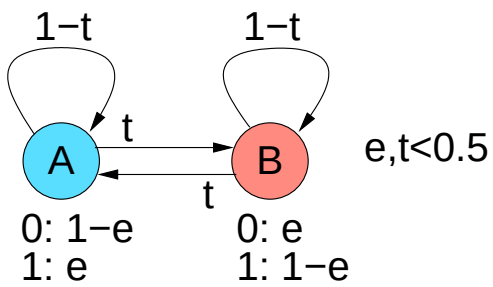
coalescence!
$P_i := -L \pm 1$
depending on x_i

Consider uniform i.i.d. generated random sequence $x_1, \ldots, x_n$:

$\Rightarrow$ variable P_i is a **random walk** on interval $(-L, L)$

$\Rightarrow$ **run:** time between two coalescence points

How long are runs?



- **Expected length:** $\left\lceil 2\frac{\log(1-t)-\log t}{\log(1-e)-\log e} \right\rceil - 1$
- **Run length distribution:**

  R_ℓ: occurence of run of length $2\ell+1$ or $2\ell+2$

  $\boxed{b \cdot \alpha^{2\ell} \leq \Pr(R_\ell) \leq c \cdot \alpha^{2\ell}}$, for some $b, c > 0, \alpha < 1$

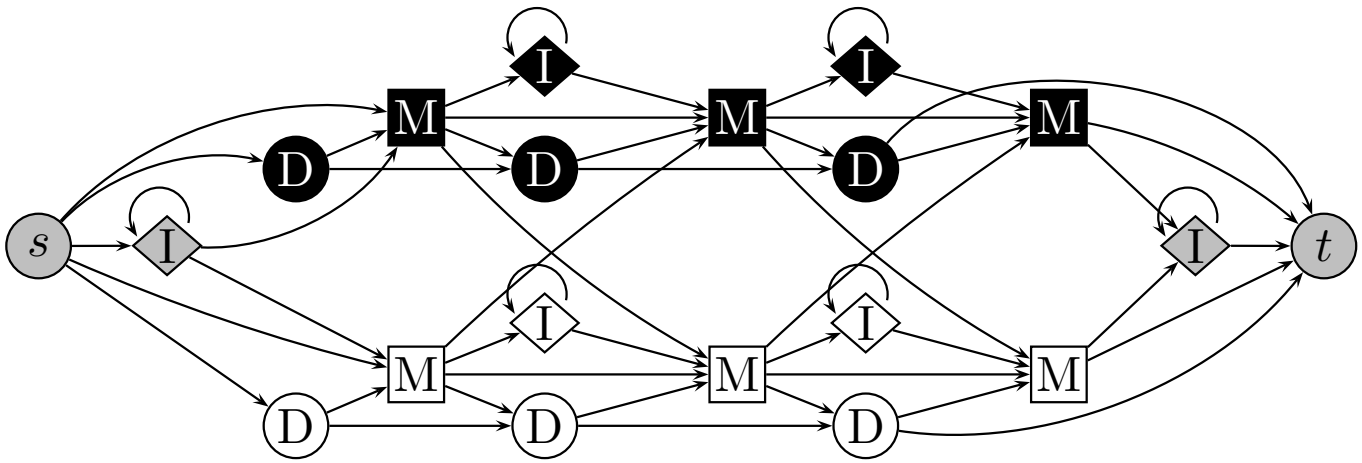
  geometrically decaying function

  (from random walk theory [Feller 1968])

Expected maximum buffer size

- Lengths of runs sum up to sequence length n
- Runs geometrically decaying and independent
- Expected buffer size $=$ length of the longest run
- **Extreme value theory for coin head runs**
  [Guibas, Odlyzko 1980; Gordon, Schilling, Waterman 1986]
- Modified for geometrically decaying functions
- **Result:** $\Theta(\log n)$

Annotation issues in jumping HMMs



State path: alignment of sequence to subtype profiles

Annotation: segments of inputs emitted by subtype profiles

Problems with most probable annotation:

— probably hard to decode

— many annotations with slightly shifted boundaries

Change the objective function for decoding

Gain function [Hamada et al. 2009]

$G(A, A')$ measures accuracy of A wrt. correct annotation A'

Examples:

Identity: score 1 iff A completely correct, 0 otherwise

Pointwise: score +1 for every correct label in A

Boundary: score +1 for every correct boundary, $-\gamma$ for incorrect boundary

	Identity	Pointwise	Boundary
$A =$  $A' =$ 	1	5	4
$A =$  $A' =$ 	0	4	$3 - \gamma$

Optimizing expected gain

Goal: find annotation A that maximizes

$$E_{A'|X}[G(A, A')] = \sum_{A'} G(A, A')P(A'|X)$$

Identity gain function: Viterbi algorithm

Pointwise gain function: Posterior decoding (forward-backward)

Boundary gain function: [Gross et al. 2007]

The choice of gain function is application-dependent