Reminder: class web links and infrastructure

- Check in with our web page (http://j.ee.washington.edu/~bilmes/classes/ee596a_winter_2013/) for up to date announcements, homeworks, etc.
Reminder: class web links and infrastructure

- Check in with our web page (http://j.ee.washington.edu/~bilmes/classes/ee596a_winter_2013/) for up to date announcements, homeworks, etc.

- All homeworks will be due via our dropbox (https://catalyst.uw.edu/collectit/dropbox/bilmes/25379)
Reminder: class web links and infrastructure

- Check in with our web page (http://j.ee.washington.edu/~bilmes/classes/ee596a_winter_2013/) for up to date announcements, homeworks, etc.
- All homeworks will be due via our dropbox (https://catalyst.uw.edu/collectit/dropbox/bilmes/25379)
- All questions should be posted to our discussion board (https://catalyst.uw.edu/gopost/board/bilmes/31332)
Reminder: class web links and infrastructure

- Check in with our web page (http://j.ee.washington.edu/~bilmes/classes/ee596a_winter_2013/) for up to date announcements, homeworks, etc.
- All homeworks will be due via our dropbox (https://catalyst.uw.edu/collectit/dropbox/bilmes/25379)
- All questions should be posted to our discussion board (https://catalyst.uw.edu/gopost/board/bilmes/31332)
- You can contact me anonymously if you wish via anonymous email (https://catalyst.uw.edu/umail/form/bilmes/4144)
Readings are in a sub-directory “reading_drafts” directly below our web page (http://j.ee.washington.edu/~bilmes/classes/ee596a_winter_2013/).
Readings are in a sub-directory “reading_drafts” directly below our web page (http://j.ee.washington.edu/~bilmes/classes/ee596a_winter_2013/).

uid is this class name (lower case) and pwd are the quarter/year of the class.
Readings are in a sub-directory “reading_drafts” directly below our web page (http://j.ee.washington.edu/~bilmes/classes/ee596a_winter_2013/).

uid is this class name (lower case) and pwd are the quarter/year of the class.

Note, the PDF file is password protected. Send me email if you have trouble (adobe reader should have no problems reading it).
Cumulative Outstanding Reading

- Read $k$-best section in 'doc.pdf'
- Read new $k$-best file that is being emailed out.
Why $k$-best

Why might we want to compute the $k$-best?

- **Confidence:** we would like to make decisions based on the HMM and know how confident the HMM is. Multiple ways to compute confidence.
Why \( k \)-best

Why might we want to compute the \( k \)-best?

- Confidence: we would like to make decisions based on the HMM and know how confident the HMM is. Multiple ways to compute confidence.

- **Score concentration.**

\[
P(\mathbf{q}_{1:t} | \mathbf{x}_{1:t})
\]
Why \( k \)-best

Why might we want to compute the \( k \)-best?

- **Confidence:** we would like to make decisions based on the HMM and know how confident the HMM is. Multiple ways to compute confidence.

- **Score concentration.**

- **Diversity of paths:** path composition based confidence.
Why \( k \)-best

Why might we want to compute the \( k \)-best?

- Confidence: we would like to make decisions based on the HMM and know how confident the HMM is. Multiple ways to compute confidence.
- Score concentration.
- Diversity of paths: path composition based confidence.
- Typical set theory.
Why \( k \)-best

Why might we want to compute the \( k \)-best?

- Confidence: we would like to make decisions based on the HMM and know how confident the HMM is. Multiple ways to compute confidence.
- Score concentration.
- Diversity of paths: path composition based confidence.
- Typical set theory.
- Localized versions of the above.

\[ \text{Diversity}(t) = \# \text{ of dist. states in the set } \bigcup_{h} X^*_{t}(h) \]

\[ h = 1, \ldots, H \]
Operations on Sorted $k$-tuples

- define the process of merging, sorting, and then truncating the ordered tuple to retain the largest $k$ elements, as the operation

$$\oplus : \mathbb{R}_+^k \times \mathbb{R}_+^k \rightarrow \mathbb{R}_+^k.$$  

- That is, the above sequence of operations defines the operation

$$s = (s^1, s^2, \ldots, s^k) = (s^1_1, s^2_1, \ldots, s^k_1) \oplus (s^1_2, s^2_2, \ldots, s^k_2) \quad (8.20)$$

$$= s_1 \oplus s_2 = k\text{-truncate}(\text{sort}(\text{merge}(s_1, s_2))) \quad (8.21)$$

- Hence, if $k = 1$, then $s_1 \oplus s_2 = s$ with $s^1 = \max(s^1_1, s^1_2)$. So this is just the max operator.
Scalar conversion

- operation of converting a non-negative scalar to a sorted $k$-tuple
- For $r \in R_+$, which is the set of non-negative reals, we have

$$\overrightarrow{\mathbb{R}}_+^k(r) = (r, 0, 0, \ldots, 0)$$  \hspace{1cm} (8.31)

where there are $k - 1$ zeros following the $r$.

- Thus, any scalar value is considered a $k$-tuple with potentially only one non-zero entry.

- E.g., the value 3 is represented as $(3, 0, 0)$.

- Thus, we can repeatedly “insert” elements $r_1, r_2, \ldots, r_\ell \in \mathbb{R}_+$ (not necessarily sorted) into a $k$-tuple by forming:

$$\overrightarrow{\mathbb{R}}_+^k(r_1) \oplus \overrightarrow{\mathbb{R}}_+^k(r_2) \oplus \cdots \oplus \overrightarrow{\mathbb{R}}_+^k(r_\ell) = (r_{\sigma_1}, r_{\sigma_2}, \ldots, r_{\sigma_k})$$  \hspace{1cm} (8.32)

where $r_{\sigma_1} \geq r_{\sigma_2} \geq \cdots \geq r_{\sigma_k}$ and $r_{\sigma_k}$ is not strictly less than any of the elements that may have been truncated from $r_1, r_2, \ldots, r_\ell$. 
Algebraic Properties

- Note that this operation \( \oplus \) has a number of properties.
- For any \( s_1, s_2, s_3 \in \mathbb{R}_+^k \), we have that
  
  \[
  s_1 \oplus s_2 \in \mathbb{R}_+^k \\
  s_1 \oplus s_2 = s_2 \oplus s_1 \\
  (s_1 \oplus s_2) \oplus s_3 = s_1 \oplus (s_2 \oplus s_3)
  \]
  (closure) \hspace{1cm} (commutativity) \hspace{1cm} (associativity) \hspace{1cm} (8.31) \hspace{1cm} (8.32) \hspace{1cm} (8.33)

- Also, there exists an additive identity, call it \( \emptyset \), such that
  
  \[
  s_1 \oplus \emptyset = s_1
  \]
  (8.34)

  with \( \emptyset = (0, 0, \ldots, 0) \).

- Therefore, \((\mathbb{R}_+^k, \oplus)\) itself forms a *commutative semi-group*. It is a semi-group rather than a group since we are not requiring an “additive” inverse.
(Left) Scalar multiplication

Given $r \in \mathbb{R}_+$, and $s \in \mathbb{R}^k_+$, define the scalar left-multiplication as an operation $\mathbb{R}_+ \times \mathbb{R}^k_+ \rightarrow \mathbb{R}^k_+$ called as follows:

$$r \cdot s = (rs^1, rs^2, \ldots, rs^k)$$  \hspace{1cm} (8.32)

I.e., we scalar multiply each element in $s$ by $r$, and since $r \geq 0$, the order does not change.

When clear from the context, we drop the $\cdot$ notation as is done in standard scalar-vector multiplication, so that $r \cdot s = rs$. 
Properties of scalar mult

- Note that for all $r_1, r_2 \in \mathbb{R}_+$, and $s_1, s_2 \in \mathbb{R}_+^k$, we have that

\[
  r_1(s_1 \oplus s_2) = r_1s_1 \oplus r_1s_2 \quad \text{distributive property} \\
  (r_1 + r_2)s_1 = (r_1s_1) + (r_2s_1) \quad \text{distributive property} \\
  (r_1r_2)s_1 = r_1(r_2s_1) \quad \text{associativity}
\]
Vector (tuple) addition

- For $s_1, s_2 \in \mathbb{R}_+^k$, we also define standard vector addition as $s = (s^1, s^2, \ldots, s^k) = s_1 + s_2$ where $s^i = s^i_1 + s^i_2$.

- Since input operands are both sorted, so is result. I.e., since $s^1_1 \geq s^2_1$ and $s^1_2 \geq s^2_2$ we have $s^1 = s^1_1 + s^1_2 \geq s^2_1 + s^2_2 = s^2$.

- Note that $s_1 + s_2$ using the “+” operator is standard vector addition, while $s_1 \oplus s_2$ is the merge, sort, and truncate operation we mentioned above.
Because of the distributed property above, we have that

\begin{align*}
\bigoplus_{q_1:T} p(x_{1:T} | q_{1:T}) &= \bigoplus_{q_1:T} p(x_{1:T} | q_{1:T}) \\
&= \bigoplus_{q_1:T} p(x_{1:T} | q_{1:T}) \\
&= \bigoplus_{q_1:T} p(x_{1:T} | q_{1:T}) \\
\end{align*}
A $k$-best forward equations recursion

\[ \alpha_1^{(k)}(q) = \overrightarrow{R}_+^k \left( p(\bar{x}_1|Q_t = q) \right) \]

and for $t > 1$:

\[ \alpha_t^{(k)}(q) = p(\bar{x}_t|Q_t = q) \bigoplus_r p(Q_t = q|Q_{t-1} = r) \alpha_{t-1}^{(k)}(r) \]  \hspace{1cm} (8.37)

Note:

- $\alpha_t^{(k)}(q) \in \overrightarrow{R}_+^k$ is a $k$-tuple for all $q, t$.
- Hence, for each $(t, q)$, need a tuple of size $k$ to store the $k$ values, unlike standard ($k = 1$) Viterbi (a 3D tensor).
If, after the final time $T$, we then compute

$$\bigoplus_{q_T} \alpha_T^{(k)}(q_T) = (p(\bar{x}_{1:T}, q_{1:T}^*(1)), p(\bar{x}_{1:T}, q_{1:T}^*(2)), \ldots, p(\bar{x}_{1:T}, q_{1:T}^*(k)))$$

(8.39)

we’ll get the desired values, i.e., the scores of the $k$-best paths through the HMM.

But how do we get the $k$-best paths?
arg $k$ max

- max returns the max and argmax returns the argument that achieves the max.
arg \( k \) max

- max returns the max and argmax returns the argument that achieves the max.
- \( \oplus \) returns the max \( k \) elements.
arg \( k \) max

- max returns the max and argmax returns the argument that achieves the max.
- \( \oplus \) returns the max \( k \) elements.
- we can generalize \( \oplus \) as well in the same way. For
\textbf{arg} \( k \) \textbf{max}

- \texttt{max} returns the \texttt{max} and \texttt{argmax} returns the argument that achieves the \texttt{max}.
- \( \oplus \) returns the \texttt{max} \( k \) elements.
- we can generalize \( \oplus \) as well in the same way. For
- \textit{We call this} \texttt{arg}\( \oplus \), pronounced “arg-k-max”.

\textbullet{} Note that \texttt{arg}\( \oplus \) has an implicit (but not expressed) dependence on \( k \).

\textbullet{} If we have \( s_1, s_2 \in \mathbb{R}^k_+ \), and \( k = 1 \), then

\begin{equation}
\text{arg}\oplus (s_1, s_2) \tag{8.1}
\end{equation}

\textbullet{} should produce the index, 1 or 2, depending on which of \( s_1 \) and \( s_2 \) contain the \texttt{max}.
**arg** \( k \) **max**

- \( \text{max} \) returns the max and \( \text{argmax} \) returns the argument that achieves the max.
- \( \oplus \) returns the max \( k \) elements.
- We can generalize \( \oplus \) as well in the same way. For \( \text{arg} \oplus \), pronounced “arg-k-max”.
- Note that \( \text{arg} \oplus \) has an implicit (but not expressed) dependence on \( k \).
arg $k$ max

- max returns the max and argmax returns the argument that achieves the max.
- $\oplus$ returns the max $k$ elements.
- we can generalize $\oplus$ as well in the same way. For
- We call this arg$\oplus$, pronounced “arg-$k$-max”.
- Note that arg$\oplus$ has an implicit (but not expressed) dependence on $k$.
- If we have $s_1, s_2 \in \mathbb{R}^k_+$, and $k = 1$, then

$$\arg\oplus(s_1, s_2)$$ (8.1)

should produce the index, 1 or 2, depending on which of $s_1$ and $s_2$ contain the max.
With $k > 1$, $\text{arg}^{\oplus}$ should produce a set of $k$ indices indicating where the $k$-largest values are.
\[ \text{arg} \ k \ \text{max} \]

- With \( k > 1 \), \( \text{arg} \oplus \) should produce a set of \( k \) indices indicating where the \( k \)-largest values are.
- With \( k > 1 \) the values could come from either \( s_1 \) or \( s_2 \), so the indices themselves have to be a 2-tuple, stating
\( \text{arg } k \max \)

- With \( k > 1 \), \( \text{arg}\oplus \) should produce a set of \( k \) indices indicating where the \( k \)-largest values are.
- With \( k > 1 \) the values could come from either \( s_1 \) or \( s_2 \), so the indices themselves have to be a 2-tuple, stating
  - which \( k \)-tuple (either \( s_1 \) or \( s_2 \)) the entry lies in and
arg $k$ max

- With $k > 1$, arg$\oplus$ should produce a set of $k$ indices indicating where the $k$-largest values are.
- With $k > 1$ the values could come from either $s_1$ or $s_2$, so the indices themselves have to be a 2-tuple, stating
  1. which $k$-tuple (either $s_1$ or $s_2$) the entry lies in and
  2. where in that selected $k$-tuple the entry is.
arg $k$ max

- With $k > 1$, arg$\oplus$ should produce a set of $k$ indices indicating where the $k$-largest values are.
- With $k > 1$ the values could come from either $s_1$ or $s_2$, so the indices themselves have to be a 2-tuple, stating
  1. which $k$-tuple (either $s_1$ or $s_2$) the entry lies in and
  2. where in that selected $k$-tuple the entry is.
- In other words, arg$\oplus$ returns a list of pairs of integers, where each pair is (a tuple-identifier $i$) and a position $1 \leq j \leq k$ within that tuple.
arg $k$ max

- With $k > 1$, $\text{arg} \oplus$ should produce a set of $k$ indices indicating where the $k$-largest values are.
- With $k > 1$ the values could come from either $s_1$ or $s_2$, so the indices themselves have to be a 2-tuple, stating
  1. which $k$-tuple (either $s_1$ or $s_2$) the entry lies in and
  2. where in that selected $k$-tuple the entry is.
- In other words, $\text{arg} \oplus$ returns a list of pairs of integers, where each pair is (a tuple-identifier $i$) and a position $1 \leq j \leq k$ within that tuple.
- List of pairs ordered: the first pair indexes the tuple and tuple entry of the largest element in the ordered lists of numbers; the second pair indexes the tuple and tuple entry of the second largest element in the ordered lists of numbers; and so on.
For example, suppose $k = 3$, then

$$\arg \max_k \{((3.0, 2.0, 1.0), (6.0, 5.0, 2.0)) = \left\{ (2, 1), (2, 2), (1, 1) \right\}$$ (8.2)
For example, suppose $k = 3$, then

$$\text{arg} \oplus ((3.0, 2.0, 1.0), (6.0, 5.0, 2.0)) = \{((2, 1), (2, 2), (1, 1))\} \quad (8.2)$$
For example, suppose $k = 3$, then

$$\text{arg}^\oplus((3.0, 2.0, 1.0), (6.0, 5.0, 2.0)) = \{(2, 1), (2, 2), (1, 1)\}$$ (8.2)

Another example with $k = 4$ would be:

$$\text{arg}^\oplus((4.0, 3.0, 2.1, 0.5), (7.0, 6.0, 2.0, 1.0), (5.0, 2.0, 1.0, 1.0))$$

$$= \left\{ (2, 1), (2, 2), (3, 1), (1, 1) \right\}$$ (8.3)

(8.4)
For example, suppose $k = 3$, then

$$\text{arg} \oplus ((3.0, 2.0, 1.0), (6.0, 5.0, 2.0)) = \{((2, 1), (2, 2), (1, 1))\} \quad (8.2)$$

Another example with $k = 4$ would be:

$$\text{arg} \oplus ((4.0, 3.0, 2.1, 0.5), (7.0, 6.0, 2.0, 1.0), (5.0, 2.0, 1.0, 1.0))$$

$$= \{((2, 1), (2, 2), (3, 1), (1, 1))\} \quad (8.3)$$
Another example, suppose $k = 3$, then

$$\arg\max_k \{ (3.0, 2.0, 1.0), (6.0, 5.0, 3.0) \}$$

$$\Rightarrow \{ ((2,1), (2,2), (2,3)), ((2,1), (2,2), (1,1)) \}$$

Hence, we can have more than list of pairs that achieves the same max $k$ values. Thus, $\arg\max_k$ should be interpreted as a set of lists, each list in the set is a list of pairs.
Another example, suppose $k = 3$, then

$$\text{arg}\,\, \max_{\mathbf{w}}$$

$$= \{((2, 1), (2, 2), (1, 1)) \cup (2, 1), (2, 2), (2, 3))\}$$
Another example, suppose $k = 3$, then

$$\text{arg}^{\oplus}((3.0, 2.0, 1.0), (6.0, 5.0, 3.0)) = \{(2, 1), (2, 2), (1, 1)\}, \{(2, 1), (2, 2), (2, 3)\}$$

Hence, we can have more than list of pairs that achieves the same max $k$ values.
Another example, suppose $k = 3$, then

$$\text{arg} \oplus ((3.0, 2.0, 1.0), (6.0, 5.0, 3.0))$$

$$= \{ ((2, 1), (2, 2), (1, 1)) , ((2, 1), (2, 2), (2, 3)) \}$$

Hence, we can have more than list of pairs that achieves the same max $k$ values.

thus, $\text{arg} \oplus$ should be interpreted as a set of lists, each list in the set is a list of pairs.
Backtracking recursion

- Consider now the following recursion, where at each step we are computing a set of $k$ pairs of integers.

$$
\tilde{\alpha}_t^{(k)}(q) \in \arg\max_r p(Q_t = q \mid Q_{t-1} = r) \alpha_{t-1}^{(k)}(r)
$$

(8.7)
Backtracking recursion

- Consider now the following recursion, where at each step we are computing a set of $k$ pairs of integers.

$$\tilde{\alpha}_t^{(k)}(q) \in \arg\max_r p(Q_t = q | Q_{t-1} = r) \alpha_{t-1}^{(k)}(r)$$  \hspace{1cm} (8.7)

- Where is the use of the observation scores $p(\bar{x}_t | q_t)$? Discuss.
Consider now the following recursion, where at each step we are computing a set of $k$ pairs of integers.

$$\hat{\alpha}_t^{(k)}(q) \in \arg\max_r p(Q_t = q | Q_{t-1} = r) \alpha_{t-1}^{(k)}(r)$$  \hspace{1cm} (8.7)

Where is the use of the observation scores $p(x_t | q_t)$? Discuss. Also, note the use of “$\in$”.
Consider now the following recursion, where at each step we are computing a set of $k$ pairs of integers.

$$
\tilde{\alpha}_t^{(k)}(q) \in \arg\max_r p(Q_t = q | Q_{t-1} = r) \alpha_{t-1}^{(k)}(r)
$$

(8.7)

Where is the use of the observation scores $p(x_t | q_t)$? Discuss.

Also, note the use of "$\in$".

Collectively, \[ \{ \tilde{\alpha}_t^{(k)}(q) \} \] is a $|D_Q| \times T$ matrix of $k$-tuples of pairs (so $2kTN$ entries in total).
Backtracking recursion

Consider now the following recursion, where at each step we are computing a set of $k$ pairs of integers.

$$
\tilde{\alpha}_t^{(k)}(q) \in \arg\max_r p(Q_t = q \mid Q_{t-1} = r)\alpha_{t-1}^{(k)}(r)
$$

(8.7)

Where is the use of the observation scores $p(x_t \mid q_t)$? Discuss.

Also, note the use of “$\in$”.

Collectively, \( \{ \tilde{\alpha}_t^{(k)}(q) \} \) is a \(|D_Q| \times T\) matrix of $k$-tuples of pairs (so $2kTN$ entries in total).

The set of pairs at $\tilde{\alpha}_t^{(k)}(q)$ identify the location of the $k$ largest items in the collective set of $k$-tuples in the previous time step that: 1) lead to entry $(q, t)$ in the matrix, and 2) have accounted for the transition matrix $p(q \mid r)$ for $r$ ranging over the first element in every pair $\tilde{\alpha}_t^{(k)}(q)$. 
Backtracking recursion

Example of two successive time steps:

Where are the back pointers?
Backtracking recursion

For discussion.

<table>
<thead>
<tr>
<th>State</th>
<th>Time t-1</th>
<th>Best Rank</th>
<th>State</th>
<th>Time t</th>
<th>Best Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td></td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(1,1) (1,2) (1,3) (2,2)
Backtracking recursion

- We can denote this list of pairs via:

\[ \tilde{\alpha}_t^{(k)}(q) = \left( (\tilde{q}^1, \tilde{k}^1), (\tilde{q}^2, \tilde{k}^2), \ldots, (\tilde{q}^k, \tilde{k}^k) \right) \]  

(8.8)

where \( 1 \leq \tilde{q}^i \leq |D_Q|, 1 \leq \tilde{k}^i \leq k \)
Backtracking recursion

- We can denote this list of pairs via:

\[ \tilde{\alpha}_t^{(k)}(q) = \left( (\tilde{q}^1, \tilde{k}^1), (\tilde{q}^2, \tilde{k}^2), \ldots, (\tilde{q}^k, \tilde{k}^k) \right) \]  \hspace{1cm} (8.8)

where \(1 \leq \tilde{q}^i \leq |D_Q|, \ 1 \leq \tilde{k}^i \leq k\)

- denote quantities like the \(\ell^{th}\) entry in this tuple of pairs using double arguments \("(q)(\ell)"\) as in:

\[ (\tilde{q}, \tilde{k}) = \tilde{\alpha}_t^{(k)}(q)(\ell) \]  \hspace{1cm} (8.9)

where \(1 \leq \ell \leq k, \ 1 \leq \tilde{q} \leq |D_Q|, \text{ and } 1 \leq \tilde{k} \leq k\).
Backtracking recursion

- At the very last time step $T$, we can then compute

$$
\tilde{\alpha}^{(k)}_T \in \arg\max_r \alpha^{(k)}_T (r)
$$

which then gives a length-$k$ list of pairs of states and position in the $k$-tuple at that state.

\[
\begin{align*}
\left\{ \alpha^{(k)}_{t}(i) \right\}_{t, i}
\end{align*}
\]
Backtracking recursion

- At the very last time step \( T \), we can then compute

\[
\tilde{\alpha}_T^{(k)} \in \arg\oplus r \alpha_T^{(k)}(r)
\]  

which then gives a length-\( k \) list of pairs of states and position in the \( k \)-tuple at that state.

- The states in this list correspond to the states of the \( k \) maximum paths through the HMM, and the index into the \( k \)-tuple gives the position within the \( k \)-tuple from which we can obtain the value.
### Backtracking recursion

- At the very last time step $T$, we can then compute

$$
\tilde{\alpha}^{(k)}_T \in \arg\oplus_{r} \alpha^{(k)}_T (r)
$$

(8.10)

which then gives a length-$k$ list of pairs of states and position in the $k$-tuple at that state.

- The states in this list correspond to the states of the $k$ maximum paths through the HMM, and the index into the $k$-tuple gives the position within the $k$-tuple from which we can obtain the value.

- Note also that the observation scores ($p(\bar{x}_T|q)$ over $q$) at time $T$ are finally (implicitly) used here.
Recall again standard Viterbi backtracking from lecture 5, on next slide:
MPE/Viterbi path - summary

Forward Equations

\[
\begin{align*}
\alpha_q^m(1) &= p(\bar{x}_1|Q_1 = q) \\ 
\alpha_q^m(t) &= p(\bar{x}_t|Q_t = q) \max_r p(Q_t = q|Q_{t-1} = r)\alpha_r^m(t-1)
\end{align*}
\] (8.21)

(8.22)

And the forward equation for storing the back indices:

\[
\tilde{\alpha}_q^m(t) \in \arg\max_r p(Q_t = q|Q_{t-1} = r)\alpha_r^m(t-1)
\] (8.23)

Backward algorithm, to compute the Viterbi path

1. Compute \(q^*_T \in \arg\max_q \alpha_q^m(T)\)
2. for \(t = T \ldots 2\) do
3. Set \(q^*_{t-1} \leftarrow \tilde{\alpha}_{q^*_t}^m(t)\)
At the final $T$ step, we need to find the $k$ top scoring positions, like above.

\[
\left( (q_T^1, k_T^1), (q_T^2, k_T^2), \ldots, (q_T^k, k_T^k) \right) = \tilde{\alpha}_T^{(k)} \in \arg\oplus r \alpha_T^{(k)}(r) 
\]  

(8.11)
$k$-best Backtracking

- At the final $T$ step, we need to find the $k$ top scoring positions, like above.

$$
\left( (q_T^1, k_T^1), (q_T^2, k_T^2), \ldots, (q_T^k, k_T^k) \right) = \tilde{\alpha}_T^{(k)} \in \arg\max_r \alpha_T^{(k)}(r)
$$

(8.11)

- Once we have these $k$ state-element pairs, we look up the index at each of those $k$ state-element pairs, to get a new set of $k$ state-element pairs. That is, we construct a new $k$-tuple of pairs as follows:

$$
\left( \tilde{\alpha}_T^{(k)}(q_T^1)(k_T^1), \tilde{\alpha}_T^{(k)}(q_T^2)(k_T^2), \ldots, \tilde{\alpha}_T^{(k)}(q_T^k)(k_T^k) \right)
$$

(8.12)
$k$-best Backtracking

- At the final $T$ step, we need to find the $k$ top scoring positions, like above.

$$
\left( (q_{T^1}^*, k_{T^1}^*), (q_{T^2}^*, k_{T^2}^*), \ldots, (q_{T^k}^*, k_{T^k}^*) \right) = \tilde{\alpha}_{T}^{(k)} \in \operatorname{arg} \oplus \alpha_T^{(k)}(r)
$$

(8.11)

- Once we have these $k$ state-element pairs, we look up the index at each of those $k$ state-element pairs, to get a new set of $k$ state-element pairs. That is, we construct a new $k$-tuple of pairs as follows:

$$
\left( \tilde{\alpha}_{T}^{(k)}(q_{T^1}^*)(k_{T^1}^*), \tilde{\alpha}_{T}^{(k)}(q_{T^2}^*)(k_{T^2}^*), \ldots, \tilde{\alpha}_{T}^{(k)}(q_{T^k}^*)(k_{T^k}^*) \right)
$$

(8.12)

Note that each entry in this $k$-tuple is still a pair, but it a pair that points to an entry in the collection of $k$-tuples at time $T - 1$. 
$k$-best Backtracking

- Each entry in a $k$-tuple at time $T - 1$ also contains a pair, so the above $k$-tuple of pairs can be converted into a new tuple of pairs.

\[
\left( (q_{T-1}^1, k_{T-1}^1), (q_{T-1}^2, k_{T-1}^2), \ldots, (q_{T-1}^k, k_{T-1}^k) \right) = \left( \hat{\alpha}_T^{(k)}(q_T^1)(k_T^1), \hat{\alpha}_T^{(k)}(q_T^2)(k_T^2), \ldots, \hat{\alpha}_T^{(k)}(q_T^k)(k_T^k) \right)
\] (8.13)
\(k\)-best Backtracking

- Each entry in a \(k\)-tuple at time \(T - 1\) also contains a pair, so the above \(k\)-tuple of pairs can be converted into a new tuple of pairs.

\[
\left( (q^*_1 T_{-1}, k^*_1 T_{-1}), (q^*_2 T_{-1}, k^*_2 T_{-1}), \ldots, (q^*_k T_{-1}, k^*_k T_{-1}) \right) \\
= \left( \tilde{\alpha}^{(k)}_T (q^*_1 T), \tilde{\alpha}^{(k)}_T (q^*_2 T), \ldots, \tilde{\alpha}^{(k)}_T (q^*_k T) \right) \tag{8.13}
\]

- This can then be used to construct the \(k\)-tuple of pairs at time step \(T - 2\) as follows:

\[
\left( (q^*_1 T_{-2}, k^*_1 T_{-2}), (q^*_2 T_{-2}, k^*_2 T_{-2}), \ldots, (q^*_k T_{-2}, k^*_k T_{-2}) \right) \\
= \left( \tilde{\alpha}^{(k)}_{T-1} (q^*_1 T_{-1}), \tilde{\alpha}^{(k)}_{T-1} (q^*_2 T_{-1}), \ldots, \tilde{\alpha}^{(k)}_{T-1} (q^*_k T_{-1}) \right) \tag{8.14}
\]
$k$-best Backtracking algorithm

The above backtracking algorithm can be written as follows:

```
1 Compute \( ((q^{*1}_T, k^{*1}_T), (q^{*2}_T, k^{*2}_T), \ldots, (q^{*k}_T, k^{*k}_T)) = \check{\alpha}_T^{(k)} \in \arg\oplus_r \alpha_T^{(k)}(r) \)
2 for \( t = T \ldots 2 \) do
3 \[ \text{Set} \quad ((q^{*1}_{t-1}, k^{*1}_{t-1}), (q^{*2}_{t-1}, k^{*2}_{t-1}), \ldots, (q^{*k}_{t-1}, k^{*k}_{t-1})) \leftarrow \\
\quad \left( \check{\alpha}_t^{(k)}(q^*_t)(k^*_t), \check{\alpha}_t^{(k)}(q^{*2}_t)(k^{*2}_t), \ldots \check{\alpha}_t^{(k)}(q^{*k}_t)(k^{*k}_t) \right) \]
```
Complete $k$-best Forward/Backward algorithm

1  **forwards pass:**
2  **for** $q = 1 \ldots |D_Q|$  **do**
3  \[ \alpha_1^{(k)}(q) \leftarrow \mathbb{R}_+^k \left( p(\bar{x}_1|Q_t = q) \right) ; \]
4  **for** $t = 2 \ldots T$  **do**
5  **for** $q = 1 \ldots |D_Q|$  **do**
6  \[ \alpha_t^{(k)}(q) \leftarrow p(\bar{x}_t|Q_t = q) \bigoplus_r p(Q_t = q|Q_{t-1} = r) \alpha_{t-1}^{(k)}(r) ; \]
7  \[ \tilde{\alpha}_t^{(k)}(q) \in \arg\bigoplus_r p(Q_t = q|Q_{t-1} = r) \alpha_{t-1}^{(k)}(r) ; \]
8  \[ \tilde{\alpha}_T^{(k)} \in \arg\bigoplus_r \alpha_T^{(k)}(r) ; \]
9  **backwards pass:**
10  **Identify** \( ((q_T^1, k_T^1), (q_T^2, k_T^2), \ldots, (q_T^k, k_T^k)) = \tilde{\alpha}_T^{(k)} ; \)
11  **for** $t = T \ldots 2$  **do**
12  **Set** \( ((q_{t-1}^1, k_{t-1}^1), (q_{t-1}^2, k_{t-1}^2), \ldots, (q_{t-1}^k, k_{t-1}^k)) \leftarrow \)
13  \[ \left( \tilde{\alpha}_t^{(k)}(q_t^1)(k_t^1), \tilde{\alpha}_t^{(k)}(q_t^2)(k_t^2), \ldots \tilde{\alpha}_t^{(k)}(q_t^k)(k_t^k) \right) ; \]
\[ p_c(\mathcal{L}_{i:1:T}^{\#}(1)) = p_c(\mathcal{L}_{i:1:T}^{\#}(n)) \]

\[ \text{dist}(\mathcal{L}_{i:1:T}^{\#}(y^+1), \mathcal{L}_{i:1:T}^{\#}(y)) = \frac{1}{\alpha} \]

\[ \vdots \]
Analysis

- one forward pass of $T$ steps and one backwards pass of $T$ steps, so the compute is still linear in $T$. 
Analysis

- one forward pass of $T$ steps and one backwards pass of $T$ steps, so the compute is still linear in $T$.
- Also, at each time step $t$ we need storage for $k|D_Q|$ cells, so the memory is also still linear in $T$.
Analysis

- one forward pass of $T$ steps and one backwards pass of $T$ steps, so the compute is still linear in $T$.
- Also, at each time step $t$ we need storage for $k|D_Q|$ cells, so the memory is also still linear in $T$.
- Finding the single max entry at each step for a given state costs $O(|D_Q|)$ (leading to $O(|D_Q|^2)$ per time step).

Finding the $k$ max entries out of $N$ can be done in $O(N)$ using a variant of the quick-sort algorithm — does so in arbitrary order (i.e., it doesn't find the $k$ top in sorted order).

Hence, finding the $k$ max entries for a state can still be done in $O(k|D_Q|)$ time. Overall, $O(k|D_Q|^2)$ per time step.

To sort the top $k$ items, we pay an additional $O(k \log k)$ (but typically $k \ll |D_Q|$). Additional $|D_Q|k \log k$ time.

Therefore, the overall time-complexity is now $O\left(T(|D_Q|^2) + |D_Q|^{2k} + |D_Q|^2 k \log k\right)$. 
Analysis

- one forward pass of $T$ steps and one backwards pass of $T$ steps, so the compute is still linear in $T$.
- Also, at each time step $t$ we need storage for $k|D_Q|$ cells, so the memory is also still linear in $T$.
- Finding the single max entry at each step for a given state costs $O(|D_Q|)$ (leading to $O(|D_Q|^2)$ per time step).
- Finding the $k$ max entries out of $N$ can be done in $O(N)$ using a variant of the quick-sort algorithm — does so in arbitrary order (i.e., it doesn’t find the $k$ top in sorted order).
Analysis

- one forward pass of $T$ steps and one backwards pass of $T$ steps, so the compute is still linear in $T$.
- Also, at each time step $t$ we need storage for $k|D_Q|$ cells, so the memory is also still linear in $T$.
- Finding the single max entry at each step for a given state costs $O(|D_Q|)$ (leading to $O(|D_Q|^2)$ per time step).
- Finding the $k$ max entries out of $N$ can be done in $O(N)$ using a variant of the quick-sort algorithm — does so in arbitrary order (i.e., it doesn’t find the $k$ top in sorted order).
- Hence, finding the $k$ max entries for a state can still be done in $O(k|D_Q|)$ time. Overall, $O(k|D_Q|^2)$ per time step.
Analysis

- one forward pass of $T$ steps and one backwards pass of $T$ steps, so the compute is still linear in $T$.
- Also, at each time step $t$ we need storage for $k|D_Q|$ cells, so the memory is also still linear in $T$.
- Finding the single max entry at each step for a given state costs $O(|D_Q|)$ (leading to $O(|D_Q|^2)$ per time step).
- Finding the $k$ max entries out of $N$ can be done in $O(N)$ using a variant of the quick-sort algorithm — does so in arbitrary order (i.e., it doesn’t find the $k$ top in sorted order).
- Hence, finding the $k$ max entries for a state can still be done in $O(k|D_Q|)$ time. Overall, $O(k|D_Q|^2)$ per time step.
- To sort the top $k$ items, we pay an additional $O(k \log k)$ (but typically $k \ll |D_Q|$). Additional $|D_Q|k \log k$ time.
Analysis

- one forward pass of $T$ steps and one backwards pass of $T$ steps, so the compute is still linear in $T$.
- Also, at each time step $t$ we need storage for $k|D_Q|$ cells, so the memory is also still linear in $T$.
- Finding the single max entry at each step for a given state costs $O(|D_Q|)$ (leading to $O(|D_Q|^2)$ per time step).
- Finding the $k$ max entries out of $N$ can be done in $O(N)$ using a variant of the quick-sort algorithm — does so in arbitrary order (i.e., it doesn’t find the $k$ top in sorted order).
- Hence, finding the $k$ max entries for a state can still be done in $O(k|D_Q|)$ time. Overall, $O(k|D_Q|^2)$ per time step.
- To sort the top $k$ items, we pay an additional $O(k \log k)$ (but typically $k \ll |D_Q|$). Additional $|D_Q|k \log k$ time.
- Therefore, the overall time-complexity is now $O(T(|D_Q|^2k + |D_Q|k \log k))$. 
at each \((t, q)\) position, we need now to store a \(k\)-tuple and in each \(k\)-tuple entry we need three values (the value and the two integers).
at each $(t, q)$ position, we need now to store a $k$-tuple and in each $k$-tuple entry we need three values (the value and the two integers).

Therefore, the memory has gone from $O(T|D_Q|)$ to $O(kT|D_Q|)$, which is $k$-times worse.
Analysis - memory

- at each \((t, q)\) position, we need now to store a \(k\)-tuple and in each \(k\)-tuple entry we need three values (the value and the two integers).
- Therefore, the memory has gone from \(O(T|D_Q|)\) to \(O(kT|D_Q|)\), which is \(k\)-times worse.
- There are other interesting time-space tradeoffs as well with dynamic models, such as the Island algorithm as we will soon see (Wednesday’s lecture).
at each \((t, q)\) position, we need now to store a \(k\)-tuple and in each \(k\)-tuple entry we need three values (the value and the two integers).

Therefore, the memory has gone from \(O(T|D_Q|)\) to \(O(kt|D_Q|)\), which is \(k\)-times worse.

There are other interesting time-space tradeoffs as well with dynamic models, such as the Island algorithm as we will soon see (Wednesday’s lecture).

Other algorithms for computing \(k\)-best that do not require \(k\) times the memory, but they do require \(k\) times the time/memory – we cover this next.
Another $k$-best algorithm

- This algorithm, memory stays $O(TN^2)$ (no extra factor of $k$).
Another $k$-best algorithm

- This algorithm, memory stays $O(TN^2)$ (no extra factor of $k$).
- Additional time cost is not too bad (as we will see).
Another $k$-best algorithm

- This algorithm, memory stays $O(TN^2)$ (no extra factor of $k$).
- Additional time cost is not too bad (as we will see).
- Algorithm is based entirely on max-marginals.
The max marginals over the hidden nodes in an HMM are defined as follows.

\[ m_t(q) = \max_{q_1:t-1, q_{t+1:T}} p(q_1:t-1, Q_t = q, q_{t+1:T}, \bar{x}_{1:T}) \]  \hspace{1cm} (8.15)

\[ = \max_{q_1:T | q_t = q} p(q_1:T, \bar{x}_{1:T}) \]  \hspace{1cm} (8.16)
Max marginals

- The max marginals over the hidden nodes in an HMM are defined as follows.

\[
m_t(q) = \max_{q_{1:t-1},q_{t+1:T}} p(q_{1:t-1}, q_t = q, q_{t+1:T}, \bar{x}_{1:T})
\]  

\[
= \max_{q_{1:T} | q_t = q} p(q_{1:T}, \bar{x}_{1:T})
\]  

(8.15) (8.16)

- The max marginals over the node pairs, or graphical model edges, in an HMM are defined as follows.

\[
m_{t,t+1}(q, q') = \max_{q_{1:t-1}, q_{t+2:T}} p(q_{1:t-1}, q_t = q, q_{t+1} = q', q_{t+2:T}, \bar{x}_{1:T})
\]  

\[
= \max_{q_{1:T} | (q_t, q_{t+1}) = (q, q')} p(q_{1:T}, \bar{x}_{1:T})
\]  

(8.17) (8.18)
Max marginals

- The max marginals over the hidden nodes in an HMM are defined as follows.

\[
m_t(q) = \max_{q_1:t-1, q_{t+1:T}} p(q_{1:t-1}, Q_t = q, q_{t+1:T}, \bar{x}_{1:T})
\]

\[
= \max_{q_1:T \mid q_t = q} p(q_1:T, \bar{x}_{1:T})
\]

(8.15) (8.16)

- The max marginals over the node pairs, or graphical model edges, in an HMM are defined as follows.

\[
m_{t,t+1}(q, q') = \max_{q_1:t-1, q_{t+2:T}} p(q_{1:t-1}, Q_t = q, Q_{t+1} = q', q_{t+2:T}, \bar{x}_{1:T})
\]

\[
= \max_{q_1:T \mid (q_t, q_{t+1}) = (q, q')} p(q_1:T, \bar{x}_{1:T})
\]

(8.17) (8.18)

- Can we compute them efficiently?
Recall the forward max equations from before.

\[
\alpha^m_q(1) = p(\bar{x}_1|Q_1 = q) \quad (8.19)
\]

\[
\alpha^m_q(t) = p(\bar{x}_t|Q_t = q) \max_r p(Q_t = q|Q_{t-1} = r) \alpha^m_r(t - 1) \quad (8.20)
\]

and

\[
\tilde{\alpha}^m_t(q) \in \text{argmax}_r p(Q_t = q|Q_{t-1} = r) \alpha^m_{t-1}(r) \quad (8.21)
\]
Recall the forward max equations from before.

\[
\alpha_q^m(1) = p(\bar{x}_1 | Q_1 = q) \tag{8.19}
\]

\[
\alpha_q^m(t) = p(\bar{x}_t | Q_t = q) \max_r p(Q_t = q | Q_{t-1} = r) \alpha_r^m(t-1) \tag{8.20}
\]

and

\[
\tilde{\alpha}_t^m(q) \in \arg\max_r p(Q_t = q | Q_{t-1} = r) \alpha_{t-1}^m(r) \tag{8.21}
\]

Recall also how we can view the \(\alpha_q^m(t)\) as the max marginals up to time \(t\), as in the following

\[
\alpha_t^m(q) = p^m(x_{1:t}, Q_t = q) = \max_{q_{1:t-1}} p(\bar{x}_{1:t}, q_{1:t-1}, Q_t = q) \tag{8.22}
\]
Backwards max-marginal

- Can also define a backwards recursion similar to the $\beta$ calculation, replace the sum with the max

\[
\beta_T^m(q) = 1
\]

\[
\beta_t^m(q) = \max_r p(\bar{x}_{t+1}|Q_{t+1} = r)p(Q_{t+1} = r|Q_t = q)\beta_{t+1}^m(r)
\]  

(8.23)  

(8.24)
Can also define a backwards recursion similar to the $\beta$ calculation, replace the sum with the max

$$
\beta_T(q) = 1
$$
$$
\beta_t^m(q) = \max_r p(\bar{x}_{t+1}|Q_{t+1} = r)p(Q_{t+1} = r|Q_t = q)\beta_{t+1}^m(r)
$$

In this case, we have the following interpretation of $\beta_t^m(q)$

$$
\beta_t^m(q) = \max_r p(\bar{x}_{t+1}|Q_{t+1} = r)p(Q_{t+1} = r|Q_t = q)\beta_{t+1}^m(r)
$$

$$
= \max_r p(\bar{x}_{t+1}|Q_{t+1} = r)p(Q_{t+1} = r|Q_t = q) \max_{\bar{x}_{t+1:T}, q_{t+1:T} | Q_t = q} p(\bar{x}_{t+1:T}, q_{t+1:T} | Q_t = q)
$$

$$
= \max_{\bar{x}_{t+1:T}, q_{t+1:T} | Q_t = q} p(\bar{x}_{t+1:T}, q_{t+1:T} | Q_t = q)
$$
How to compute singleton max marginal using forward/backward

- Now consider the max marginals we defined above

\[ m_t(q) = \max_{q_{1:T} \mid q_t = q} p(q_{1:T}, \bar{x}_{1:T}) \tag{8.30} \]

\[ = \max_{q_{1:T} \mid q_t = q} p(q_{1:t}, \bar{x}_{1:t}, q_{t+1:T}, \bar{x}_{t+1:T}) \tag{8.31} \]

\[ = \max_{q_{1:T} \mid q_t = q} p(\bar{x}_{t+1:T}, q_{t+1:T} \mid q_{1:t}, \bar{x}_{1:t}) p(q_{1:t}, \bar{x}_{1:t}) \tag{8.32} \]

\[ = \max_{q_{1:T} \mid q_t = q} p(\bar{x}_{t+1:T}, q_{t+1:T} \mid q_t) p(q_{1:t}, \bar{x}_{1:t}) \tag{8.33} \]

\[ = \max_{q_{t+1:T} \mid q_t = q} p(\bar{x}_{t+1:T}, q_{t+1:T} \mid q_t) \max_{q_{1:t} \mid q_t = q} p(q_{1:t}, \bar{x}_{1:t}) \tag{8.34} \]

\[ = \beta_t^m(q) \alpha_t^m(q) \tag{8.35} \]
How to compute edge max marginal using forward/backward

The pairwise (edge) max marginal is

\[ m_{t,t+1}(q, q') \]

\[ = \max_{q_1:T \mid (q_t, q_{t+1}) = (q, q')} p(q_{1:T}, \bar{x}_{1:T}) \]

\[ = \max_{q_1:T \mid (q_t, q_{t+1}) = (q, q')} p(q_{1:t-1}, q_t, q_{t+1}, q_{t+2:T}, \bar{x}_{1:t}, \bar{x}_{t+1} \bar{x}_{t+2:T}) \]

\[ = p(x_{t+1} | q') \max_{q_{t+2:T} \mid q_{t+1} = q'} p(\bar{x}_{t+2:T}, q_{t+2:T} \mid q_{t+1}) p(q' \mid q) \max_{q_1:t \mid q_t = q} p(q_1:t, \bar{x}_{1:t}) \]

\[ = p(x_{t+1} | q') \beta_{t+1}^m (q') p(q' \mid q) \alpha_t^m (q) \]

Therefore, either max marginal is easily obtained with the max versions of the forward and backward recursions.
Computing Viterbi from max-margin - unique case

Proposition 8.4.1

If the Viterbi path is unique, meaning $p(\bar{x}_{1:T}, q^*_{1:T}(1)) > p(\bar{x}_{1:T}, q^*_{1:T}(k))$ for any $k > 1$, then we have the relationship:

$$\forall t, \{q^*_t(1)\} = \text{argmax}_q m_t(q). \quad (8.41)$$

Proof.

Let $p^1 = \max_{q_{1:T}} p(q_{1:T}, \bar{x}_{1:T})$ be the score value of the Viterbi path. Then we clearly have that, for any $t$,

$$p^1 = \max_q m_t(q) = m_t(q_t(1)) \quad (8.42)$$

where $q_t(1) = \text{argmax}_q m_t(q)$. Since Viterbi path is unique, we have:

$$m_t(q_t(1)) > m_t(q) \quad \forall q \neq q_t(1) \quad (8.43)$$

which means that no value other than $q_t(1)$ can be part of a Viterbi path. Hence, $q_t(1) = q^*_t(1)$ is the Viterbi path value at time $t$. $\square$
Proposition 8.4.2

Let $q_t^*(1)$ be the state value of any Viterbi path of an HMM at time $t$. Then we can find a state value $q_{t+1}^*(1)$ at $t + 1$ such that $(q_t^*(1), q_{t+1}^*(1))$ is a pair of state values of a Viterbi path using the following procedure:

$$q_{t+1}^*(1) \in \arg\max_q m_{t,t+1}(q_t^*(1), q).$$

(8.44)

Proof.

The quantity $\arg\max_{r,q} m_{t,t+1}(r, q) = \{(r_i^*, q_i^*)\}_i$ is a set of pairs, each of which is compatible with some Viterbi path. Moreover, any Viterbi path must, at times $t$ and $t + 1$, have value corresponding to one of the pairs. Therefore, there is some $i$ such that $q_t^*(1) = r_i^*$, and the argmax in Equation (8.44) chooses the corresponding $q_i^*$ which hence is compatible with some Viterbi path.
Computing Viterbi from max-margin - any case

- We start this process with the singleton max-marginal on the left

\[ q_t^*(1) \in \arg \max_q m_{t-1,t}(q_{t-1}(1), q), \]  

(8.46)

and then repeat the following recursion, for \( t = 2 \ldots T \), as follows

\[ q_1^*(1) \in \arg \max_q m_1(q), \]  

(8.45)

which, thanks to Proposition 8.4.2, is guaranteed to be a Viterbi path.
Computing Viterbi from max-margin - any case

- We start this process with the singleton max-marginal on the left

\[ q_1^*(1) \in \arg\max_q m_1(q), \]  

(8.45)

and then repeat the following recursion, for \( t = 2 \ldots T \), as follows

\[ q_t^*(1) \in \arg\max_q m_{t-1,t}(q_{t-1}(1), q), \]  

(8.46)

which, thanks to Proposition 8.4.2, is guaranteed to be a Viterbi path.

- Max marginals seem to be powerful. Can we compute \( k \)-best using just them?
Computing 2nd best from max-margin

- Suppose we have identified the Viterbi (1st best) path, i.e., $q_{1:T}^*(1) \in D_{Q_{1:T}}$ such that

$$p(\bar{x}_{1:T}, q_{1:T}^*(1)) \geq p(\bar{x}_{1:T}, q_{1:T})$$

(8.47)

for all $q_{1:T} \in D_{Q_{1:T}} \setminus \{q_{1:T}^*(1)\}$. 
Computing 2nd best from max-margin

Suppose we have identified the Viterbi (1st best) path, i.e., $q^*_1:T(1) \in D_{Q1:T}$ such that

$$p(\bar{x}_{1:T}, q^*_1:T(1)) \geq p(\bar{x}_{1:T}, q_1:T)$$ \hspace{1cm} (8.47)

for all $q_1:T \in D_{Q1:T} \setminus \{q^*_1:T(1)\}$.

2nd best path must exist within the set of sequences

$$D_{Q1:T}(1) \triangleq D_{Q1:T} \setminus \{q^*_1:T(1)\}$$ \hspace{1cm} (8.48)
Computing 2nd best from max-margin

- Suppose we have identified the Viterbi (1st best) path, i.e., $q_{1:T}^*(1) \in D_{Q_1:T}$ such that

\[ p(\bar{x}_{1:T}, q_{1:T}^*(1)) \geq p(\bar{x}_{1:T}, q_{1:T}) \]  

(8.47)

for all $q_{1:T} \in D_{Q_1:T} \setminus \{q_{1:T}^*(1)\}$.

- 2nd best path must exist within the set of sequences

\[ D_{Q_1:T}(1) \triangleq D_{Q_1:T} \setminus \{q_{1:T}^*(1)\} \]  

(8.48)

- and, 2nd best path $q_{1:T}^*(2)$ must have some difference with the 1st best path $q_{1:T}^*(1)$. 
Computing 2nd best from max-margin

- partition $D_{Q_1:T}(1)$ into separate sets of paths based on where difference between 1st and 2nd best path occurs.
Computing 2nd best from max-margin

- partition $D_{Q_1:T}(1)$ into separate sets of paths based on where difference between 1st and 2nd best path occurs.
- The first time where difference exists between the best and second best path may be either:
Computing 2nd best from max-margin

- partition $D_{Q_1:T}(1)$ into separate sets of paths based on where difference between 1st and 2nd best path occurs.
- The first time where difference exists between the best and second best path may be either:
  - at time 1,
Computing 2nd best from max-margin

- partition \( D_{Q_1:T}(1) \) into separate sets of paths based on where difference between 1st and 2nd best path occurs.
- The first time where difference exists between the best and second best path may be either:
  - at time 1,
  - or at time 2 with there being no difference at time 1,
Computing 2nd best from max-margin

- partition $D_{Q_1:T}(1)$ into separate sets of paths based on where difference between 1st and 2nd best path occurs.
- The first time where difference exists between the best and second best path may be either:
  - at time 1,
  - or at time 2 with there being no difference at time 1,
  - or at time 3 with there being no difference either at time 1 or at time 2,
partition $D_{Q_1:T}(1)$ into separate sets of paths based on where difference between 1st and 2nd best path occurs.

The first time where difference exists between the best and second best path may be either:
- at time 1,
- or at time 2 with there being no difference at time 1,
- or at time 3 with there being no difference either at time 1 or at time 2,
- ...
Computing 2nd best from max-margin

- partition $D_{Q_1:T}(1)$ into separate sets of paths based on where difference between 1st and 2nd best path occurs.
- The first time where difference exists between the best and second best path may be either:
  - at time 1,
  - or at time 2 with there being no difference at time 1,
  - or at time 3 with there being no difference either at time 1 or at time 2,
  - ...
  - or at time $t$ with there being no difference from time 1 through time $t - 1$
Computing 2nd best from max-margin

- partition \(D_{Q_1:T}(1)\) into separate sets of paths based on where difference between 1st and 2nd best path occurs.

- The first time where difference exists between the best and second best path may be either:
  - at time 1,
  - or at time 2 with there being no difference at time 1,
  - or at time 3 with there being no difference either at time 1 or at time 2,
  - \(\ldots\)
  - or at time \(t\) with there being no difference from time 1 through time \(t - 1\)
  - etc. up to time \(T\).
Partition into blocks

That is, we partition $D_{Q_1:T}(1)$ into $T$ blocks of sequences as follows:
That is, we partition $D_{Q_1:T}(1)$ into $T$ blocks of sequences as follows:

$$D_{Q_1:T}(1,1) = \{ q_{1:T} \in D_{Q_1:T} : q_1 \neq q^*(1) \}$$ (8.49)
Partition into blocks

That is, we partition \( D_{Q_1:T}(1) \) into \( T \) blocks of sequences as follows:

\[
D_{Q_1:T}(1, 1) = \{ q_{1:T} \in D_{Q_1:T} : q_1 \neq q_1^*(1) \} \quad (8.49)
\]

\[
D_{Q_1:T}(1, 2) = \{ q_{1:T} \in D_{Q_1:T} : q_1 = q_1^*(1), q_2 \neq q_2^*(1) \} \quad (8.50)
\]
Partition into blocks

That is, we partition $D_{Q_1:T}(1)$ into $T$ blocks of sequences as follows:

$$D_{Q_1:T}(1, 1) = \{ q_{1:T} \in D_{Q_1:T} : q_1 \neq q_1^*(1) \} \quad (8.49)$$
$$D_{Q_1:T}(1, 2) = \{ q_{1:T} \in D_{Q_1:T} : q_1 = q_1^*(1), q_2 \neq q_2^*(1) \} \quad (8.50)$$
$$D_{Q_1:T}(1, 3) = \{ q_{1:T} \in D_{Q_1:T} : q_1 = q_1^*(1), q_2 = q_2^*(1), q_3 \neq q_3^*(1) \} \quad (8.51)$$
Partition into blocks

That is, we partition $D_{Q_1:T}(1)$ into $T$ blocks of sequences as follows:

$$D_{Q_1:T}(1, 1) = \{ q_{1:T} \in D_{Q_1:T} : q_1 \neq q_1^*(1) \} \quad (8.49)$$
$$D_{Q_1:T}(1, 2) = \{ q_{1:T} \in D_{Q_1:T} : q_1 = q_1^*(1), q_2 \neq q_2^*(1) \} \quad (8.50)$$
$$D_{Q_1:T}(1, 3) = \{ q_{1:T} \in D_{Q_1:T} : q_1 = q_1^*(1), q_2 = q_2^*(1), q_3 \neq q_3^*(1) \} \quad (8.51)$$

...
That is, we partition $D_{Q_1:T}(1)$ into $T$ blocks of sequences as follows:

$$D_{Q_1:T}(1, 1) = \{ q_{1:T} \in D_{Q_1:T} : q_1 \neq q_1^*(1) \} \quad (8.49)$$

$$D_{Q_1:T}(1, 2) = \{ q_{1:T} \in D_{Q_1:T} : q_1 = q_1^*(1), q_2 \neq q_2^*(1) \} \quad (8.50)$$

$$D_{Q_1:T}(1, 3) = \{ q_{1:T} \in D_{Q_1:T} : q_1 = q_1^*(1), q_2 = q_2^*(1), q_3 \neq q_3^*(1) \} \quad (8.51)$$

...\n
$$D_{Q_1:T}(1, T) = \{ q_{1:T} \in D_{Q_1:T} : q_1 = q_1^*(1), \ldots, q_{T-1} = q_{T-1}^*(1), q_T \neq q_T^*(1) \} \quad (8.53)$$
Partition into blocks

- That is, we partition $D_{Q_1:T}(1)$ into $T$ blocks of sequences as follows:

  $$D_{Q_1:T}(1, 1) = \{q_{1:T} \in D_{Q_1:T} : q_1 \neq q_1^*(1)\}$$  \hspace{1cm} (8.49)

  $$D_{Q_1:T}(1, 2) = \{q_{1:T} \in D_{Q_1:T} : q_1 = q_1^*(1), q_2 \neq q_2^*(1)\}$$  \hspace{1cm} (8.50)

  $$D_{Q_1:T}(1, 3) = \{q_{1:T} \in D_{Q_1:T} : q_1 = q_1^*(1), q_2 = q_2^*(1), q_3 \neq q_3^*(1)\}$$  \hspace{1cm} (8.51)

  $$\vdots$$

  $$D_{Q_1:T}(1, T) = \{q_{1:T} \in D_{Q_1:T} : q_1 = q_1^*(1), \ldots,\,$$

  $$q_{T-1} = q_{T-1}^*(1), q_T \neq q_T^*(1)\}$$  \hspace{1cm} (8.52)

- This is a partition into $T$ blocks, meaning:

  $D_{Q_1:T}(1, t) \cap D_{Q_1:T}(1, t') = \emptyset$ for $t \neq t'$ and that

  $$D_{Q_1:T}(1) = \bigcup_{t=1}^{T} D_{Q_1:T}(1, t).$$
2nd best path must exist in one of the above blocks. We follow the following strategy:
**Strategy**

- 2nd best path must exist in one of the above blocks. We follow the following strategy:
  - Find the block with the maximum score
Strategy

- 2nd best path must exist in one of the above blocks. We follow the following strategy:
  - Find the block with the maximum score
  - Find the best path within that discovered block.
2nd best path must exist in one of the above blocks. We follow the following strategy:

- Find the block with the maximum score
- Find the best path within that discovered block.
- Do this efficiently.
Find the block with the maximum score

- Define block scores, for each \( t \in \{1, 2, \ldots, T\} \), the following:

\[
p^m(D_{Q_{1:T}}(1, t)) \triangleq \max_{q_{1:T} \in D_{Q_{1:T}}(1, t)} p(q_{1:T}, \bar{x}_{1:T})
\]  

(8.55)
Find the block with the maximum score

- Define block scores, for each \( t \in \{1, 2, \ldots, T\} \), the following:

\[
p^m(D_{Q_1:T}(1, t)) \triangleq \max_{q_{1:T} \in D_{Q_1:T}(1,t)} p(q_{1:T}, \bar{x}_{1:T}) \tag{8.55}
\]

- can compute them using only the max marginals.

\[
p^m(D_{Q_1:T}(1, 1)) = \max_{q \neq q_{1}^{*}(1)} m_1(q) \tag{8.56}
\]

and for \( t \in \{2, 3, \ldots, T\} \)

\[
p^m(D_{Q_1:T}(1, t)) = \max_{q \neq q_{t}^{*}(1)} m_{t-1,t}(q_{t-1}^{*}(1), q) \tag{8.57}
\]
Find the block with the maximum score

- Define block scores, for each $t \in \{1, 2, \ldots, T\}$, the following:

$$p^m(D_{Q1:T}(1, t)) \triangleq \max_{q_1:T \in D_{Q1:T}(1,t)} p(q_1:T, \bar{x}_{1:T})$$  \hspace{1cm} (8.55)

- can compute them using only the max marginals.

$$p^m(D_{Q1:T}(1, 1)) = \max_{q \neq q_1^*(1)} m_1(q)$$  \hspace{1cm} (8.56)

and for $t \in \{2, 3, \ldots, T\}$

$$p^m(D_{Q1:T}(1, t)) = \max_{q \neq q_t^*(1)} m_{t-1,t}(q_{t-1}(1), q)$$  \hspace{1cm} (8.57)

- Finding the maximum block then is easy:

$$p(q_1^*:T(2), \bar{x}_{1:T}) = \max_t p^m(D_{Q1:T}(1, t))$$  \hspace{1cm} (8.58)
The maximum score block

- Suppose $t^{(2)} \in \arg\max_t p^m(D_{Q1:T}(1, t))$ is time that achieved the maximum.
The maximum score block

- Suppose \( t^{(2)} \in \arg \max_t p^m(D_{Q:1:T}(1, t)) \) is time that achieved the maximum.
- Then

\[
p(q^{*}_{1:T}(2), \bar{x}_{1:T}) = p^m(D_{Q:1:T}(1, t^{(2)})).
\]  

(8.59)

and \( q^{*}_{1:T}(2) \in D_{Q:1:T}(1, t^{(2)}) \).
The maximum score block

- Suppose \( t^{(2)} \in \operatorname{argmax}_t p^m(D_{Q_1:T}(1, t)) \) is time that achieved the maximum.

- Then

\[
p(q_{1:T}^*(2), \bar{x}_{1:T}) = p^m(D_{Q_1:T}(1, t^{(2)})). \tag{8.59}
\]

and \( q_{1:T}^*(2) \in D_{Q_1:T}(1, t^{(2)}) \).

- We can identify \( q_{1:T}^*(2) \) using the following procedure, part of which is a copy, and part of which is dynamic programming:

\[
q_t^*(2) = q_t^*(1) \quad \text{for } t = 1, \ldots, t^{(2)} - 1 \tag{8.60}
\]

\[
q_{t(2)}^*(2) \in \operatorname{argmax}_{q \neq q_{t(2)}^*(1)} m_{t-1,t}(q_{t(2)}^* - 1(2), q) \tag{8.61}
\]

\[
q_t(2) \in \operatorname{argmax}_q m_{t-1,t}(q_{t-1}^*(2), q) \quad \text{for } t = t^{(2)} + 1, \ldots, T \tag{8.62}
\]
Suppose $t^{(2)} \in \arg\max_t p^m(D_{Q_1:T}(1, t))$ is time that achieved the maximum.

Then

$$p(q^*_1:T(2), \bar{x}_{1:T}) = p^m(D_{Q_1:T}(1, t^{(2)})).$$

(8.59)

and $q^*_1:T(2) \in D_{Q_1:T}(1, t^{(2)})$.

We can identify $q^*_1:T(2)$ using the following procedure, part of which is a copy, and part of which is dynamic programming:

$$q_t^*(2) = q_t^*(1) \quad \text{for } t = 1, \ldots, t^{(2)} - 1$$

(8.60)

$$q_t^{(2)}(2) \in \arg\max_{q \neq q_{t(2)}^*(1)} m_{t-1, t}(q_{t^{(2)}-1}(2), q)$$

(8.61)

$$q_t(2) \in \arg\max_q m_{t-1, t}(q_{t-1}(2), q) \quad \text{for } t = t^{(2)} + 1, \ldots, T$$

(8.62)

- Time cost: only an additional $O(TN)$. Additional memory cost: $O(T)$. 
the 2nd best is $q^*_1:T(2) \in D_{Q_1:T}(1, t^{(2)})$ where $t^{(2)}$ is the index where there lies a difference between the first and second best path.
Finding the 3rd best

- the 2nd best is $q^{*}_{1:T}(2) \in D_{Q_{1:T}(1, t^{(2)})}$ where $t^{(2)}$ is the index where there lies a difference between the first and second best path.

- Third best is either in one of original blocks $\{D_{Q_{1:T}(1, t)}\}_{t \neq t^{(2)}}$ that did not contain the second best . . .

In latter case, must exist difference from both 1st and 2nd best. To identify it, can partition $D_{Q_{1:T}(1, t^{(2)})}$ into separate sets of paths based on where this difference happens.
Finding the 3rd best

- the 2nd best is $q^*_1:T(2) \in D_{Q_1:T}(1, t^{(2)})$ where $t^{(2)}$ is the index where there lies a difference between the first and second best path.

- Third best is either in one of original blocks $\{D_{Q_1:T}(1, t)\}_{t \neq t^{(2)}}$ that did not contain the second best . . .

- . . . or otherwise is in the same original block that contained the second best $q^*_1:T(3) \in D_{Q_1:T}(1, t^{(2)}) \setminus \{q^*_1:T(2)\}$.
Finding the 3rd best

- the 2nd best is \( q_1^*:T(2) \in D_{Q_1:T}(1, t(2)) \) where \( t(2) \) is the index where there lies a difference between the first and second best path.

- Third best is either in one of original blocks \( \{D_{Q_1:T}(1, t)\}_{t \neq t(2)} \) that did not contain the second best . . .

- . . . or otherwise is in the same original block that contained the second best \( q_1^*:T(3) \in D_{Q_1:T}(1, t(2)) \setminus \{q_1^*:T(2)\} \).

- In latter case, must exist difference from both 1st and 2nd best.
Finding the 3rd best

- the 2nd best is $q_{1:T}^*(2) \in D_{Q_{1:T}}(1, t^{(2)})$ where $t^{(2)}$ is the index where there lies a difference between the first and second best path.

- Third best is either in one of original blocks $\{D_{Q_{1:T}}(1, t)\}_{t \neq t^{(2)}}$ that did not contain the second best . . .

- . . . or otherwise is in the same original block that contained the second best $q_{1:T}^*(3) \in D_{Q_{1:T}}(1, t^{(2)}) \setminus \{q_{1:T}^*(2)\}$.

- In latter case, must exist difference from both 1st and 2nd best.

- To identify it, can partition $D_{Q_{1:T}}(1, t^{(2)})$ into separate sets of paths based on where this difference happens.
The first time (meaning the frame closest to $t = t^{(2)}$) at which there is a difference between the second best path $q_{1:T}^*(2)$ and third best path $q_{1:T}^*(2)$ may be either:
Finding the 3rd best - partitioning $D_{Q_{1:T}}(1, t^{(2)})$

- The first time (meaning the frame closest to $t = t^{(2)}$) at which there is a difference between the second best path $q_{1:T}^*(2)$ and third best path $q_{1:T}^*(2)$ may be either:
  - at time $t^{(2)}$, 

Finding the 3rd best - partitioning $D_{Q_1:T}(1, t^{(2)})$

- The first time (meaning the frame closest to $t = t^{(2)}$) at which there is a difference between the second best path $q^*_1:T(2)$ and third best path $q^*_1:T(2)$ may be either:
  - at time $t^{(2)}$,
  - or at time $t^{(2)} + 1$ with there being no difference at time $t^{(2)}$, ...
Finding the 3rd best - partitioning $D_{Q_1:T}(1, t^{(2)})$

- The first time (meaning the frame closest to $t = t^{(2)}$) at which there is a difference between the second best path $q^*_1:T(2)$ and third best path $q^*_1:T(2)$ may be either:
  - at time $t^{(2)}$,
  - or at time $t^{(2)} + 1$ with there being no difference at time $t^{(2)}$,
  - or at time $t^{(2)} + 2$ with there being no difference either at time $t^{(2)}$,
  - or at time $t^{(2)} + 1$, ...
The first time (meaning the frame closest to $t = t^{(2)}$) at which there is a difference between the second best path $q^{*}_{1:T}(2)$ and third best path $q^{*}_{1:T}(2)$ may be either:

- at time $t^{(2)}$,
- or at time $t^{(2)} + 1$ with there being no difference at time $t^{(2)}$,
- or at time $t^{(2)} + 2$ with there being no difference either at time $t^{(2)}$,
- or at time $t^{(2)} + 1$,
- ...
Finding the 3rd best - partitioning $D_{Q_1:T}(1, t^{(2)})$

- The first time (meaning the frame closest to $t = t^{(2)}$) at which there is a difference between the second best path $q_{1:T}^*(2)$ and third best path $q_{1:T}^*(2)$ may be either:
  - at time $t^{(2)}$,
  - or at time $t^{(2)} + 1$ with there being no difference at time $t^{(2)}$,
  - or at time $t^{(2)} + 2$ with there being no difference either at time $t^{(2)}$,
  - or at time $t^{(2)} + 1$,
  - ... 
  - or at time $t > t^{(2)}$ with there being no difference from time $t^{(2)}$ through time $t - 1$
Finding the 3rd best - partitioning $D_{Q_1:T}(1, t^{(2)})$

- The first time (meaning the frame closest to $t = t^{(2)}$) at which there is a difference between the second best path $q^*_{1:T}(2)$ and third best path $q^*_{1:T}(2)$ may be either:
  - at time $t^{(2)}$,
  - or at time $t^{(2)} + 1$ with there being no difference at time $t^{(2)}$,
  - or at time $t^{(2)} + 2$ with there being no difference either at time $t^{(2)}$, or at time $t^{(2)} + 1$,
  - ... 
  - or at time $t > t^{(2)}$ with there being no difference from time $t^{(2)}$ through time $t-1$
  - etc. up to time $T$. 

Finding the 3rd best - partitioning $D_{Q_1:T}(1, t^{(2)})$

We partition $D_{Q_1:T}(1, t^{(2)})$ into $T - t' + 1$ subsets $s$

$$D_{Q_1:T}(1, t^{(2)}), D_{Q_1:T}(1, t^{(2)}, t^{(2)} + 1), \ldots, D_{Q_1:T}(1, t^{(2)}, T) \quad (8.63)$$

where
Finding the 3rd best - partitioning $D_{Q_1:T}(1, t^{(2)})$

We partition $D_{Q_1:T}(1, t^{(2)})$ into $T - t' + 1$ subsets $s$

$$D_{Q_1:T}(1, t^{(2)}, t^{(2)}), D_{Q_1:T}(1, t^{(2)}, t^{(2)} + 1), \ldots, D_{Q_1:T}(1, t^{(2)}, T) \quad (8.63)$$

where

$$D_{Q_1:T}(1, t^{(2)}, t^{(2)}) = \left\{ q_{1:T} \in D_{Q_1:T}(1, t^{(2)}) : q_{t^{(2)}} \neq q_{t^{(2)}(2)}^{*} \right\} \quad (8.64)$$
Finding the 3rd best - partitioning $D_{Q_1:T}(1, t^{(2)})$

We partition $D_{Q_1:T}(1, t^{(2)})$ into $T - t' + 1$ subsets $s$

$$D_{Q_1:T}(1, t^{(2)}, t^{(2)}), D_{Q_1:T}(1, t^{(2)}, t^{(2)} + 1), \ldots, D_{Q_1:T}(1, t^{(2)}, T)$$  \hspace{1cm} (8.63)

where

$$D_{Q_1:T}(1, t^{(2)}, t^{(2)}) = \left\{ q_{1:T} \in D_{Q_1:T}(1, t^{(2)}) : q_{t^{(2)}} \neq q_{t^{(2)}}^{*}(2) \right\}$$ \hspace{1cm} (8.64)

$$D_{Q_1:T}(1, t^{(2)}, t^{(2)} + 1) = \left\{ q_{1:T} \in D_{Q_1:T}^{1,t^{(2)}} : q_{t^{(2)}} = q_{t^{(2)}}^{*}(2), q_{t^{(2)}+1} \neq q_{t^{(2)}+1}^{*}(2) \right\}$$ \hspace{1cm} (8.65)
Finding the 3rd best - partitioning $D_{Q_1:T}(1, t^{(2)})$

We partition $D_{Q_1:T}(1, t^{(2)})$ into $T - t' + 1$ subsets $s$

$$D_{Q_1:T}(1, t^{(2)}, t^{(2)}), D_{Q_1:T}(1, t^{(2)}, t^{(2)} + 1), \ldots, D_{Q_1:T}(1, t^{(2)}, T)$$ (8.63)

where

$$D_{Q_1:T}(1, t^{(2)}, t^{(2)}) = \left\{ q_{1:T} \in D_{Q_1:T}(1, t^{(2)}) : q_{t^{(2)}} \neq q_{t^{(2)}}^{*} (2) \right\}$$ (8.64)

$$D_{Q_1:T}(1, t^{(2)}, t^{(2)} + 1) = \left\{ q_{1:T} \in D_{Q_1:T}^{1,t^{(2)}} : q_{t^{(2)}} = q_{t^{(2)}}^{*} (2), q_{t^{(2)} + 1} \neq q_{t^{(2)} + 1}^{*} (2) \right\}$$ (8.65)

...
Finding the 3rd best - partitioning $D_{Q_{1:T}}(1, t^{(2)})$

We partition $D_{Q_{1:T}}(1, t^{(2)})$ into $T - t' + 1$ subsets $s$

$$D_{Q_{1:T}}(1, t^{(2)}, t^{(2)})$, $D_{Q_{1:T}}(1, t^{(2)}, t^{(2)} + 1), \ldots, D_{Q_{1:T}}(1, t^{(2)}, T)$$ (8.63)

where

$$D_{Q_{1:T}}(1, t^{(2)}, t^{(2)}) = \left\{ q_{1:T} \in D_{Q_{1:T}}(1, t^{(2)}) : q_{t^{(2)}} \neq q_{t^{(2)}}^{*}(2) \right\}$$ (8.64)

$$D_{Q_{1:T}}(1, t^{(2)}, t^{(2)} + 1) = \left\{ q_{1:T} \in D_{Q_{1:T}}^{1,t^{(2)}} : q_{t^{(2)}} = q_{t^{(2)}}^{*}(2), q_{t^{(2)}+1} \neq q_{t^{(2)}+1}^{*}(2) \right\}$$ (8.65)

$$\ldots$$

$$D_{Q_{1:T}}(1, t^{(2)}, T) = \left\{ q_{1:T} \in D_{Q_{1:T}}(1, t^{(2)}) : q_{t^{(2)}} = q_{t^{(2)}}^{*}(2), \ldots, 
q_{T-1} = q_{T-1}^{*}(2), q_T \neq q_T^{*}(2) \right\}$$ (8.66)
Finding the 3rd best - partitioning $D_{Q_{1:T}}(1, t^{(2)})$

In other words, we have another characterization of $D_{Q_{1:T}}(1, t^{(2)}, t^{(2)})$ as:

$$D_{Q_{1:T}}(1, t^{(2)}, t^{(2)}) = \left\{ q_{1:T} : q_1 = q_1^*(1), q_2 = q_2^*(1), \ldots, \right\}$$

$$q_{t^{(2)}-1} = q_{t^{(2)}-1}^*(1), q_{t^{(2)}} \neq q_{t^{(2)}}^*(1), q_{t^{(2)}} \neq q_{t^{(2)}}^*(2) \right\}$$
Finding the 3rd best - partitioning $D_{Q_1:T}(1, t^{(2)})$

- In other words, we have another characterization of $D_{Q_1:T}(1, t^{(2)}, t^{(2)})$ as:

$$D_{Q_1:T}(1, t^{(2)}, t^{(2)}) = \left\{ q_{1:T} : q_1 = q_1^*(1), q_2 = q_2^*(1), \ldots, \right.$$  

$$q_{t^{(2)}-1} = q_{t^{(2)}-1}^*(1), q_{t^{(2)}} \neq q_{t^{(2)}}^*(1), q_{t^{(2)}} \neq q_{t^{(2)}}^*(2) \right\}$$

(8.67)

- and also that for $t > t^{(2)}$,

$$D_{Q_1:T}(1, t^{(2)}, t) = \left\{ q_{1:T} : q_1 = q_1^*(1), q_2 = q_2^*(1), q_{t^{(2)}-1} = q_{t^{(2)}-1}^*(1), \ldots, \right.$$  

$$q_{t^{(2)}} \neq q_{t^{(2)}}^*(1),$$  

$$q_{t^{(2)}} = q_{t^{(2)}}^*(2), q_{t^{(2)}+1} = q_{t^{(2)}+1}^*(2), \ldots, q_{t-1} = q_{t-1}^*(2), q_t \neq q_t^*(2) \right\}$$

(8.69)
Thus, the blocks \( \{D_{Q_1:T}(1, t)\}_{t \neq t^{(2)}} \) and \( \{D_{Q_1:T}(1, t^{(2)}, t)\}_{t \geq t^{(2)}} \) constitute the partitioning of \( D_{Q_1:T} \setminus \{q_1:T(1), q_1:T(2)\} \).
Thus, the blocks $\{D_{Q_1:T}(1, t)\}_{t \neq t^{(2)}}$ and $\{D_{Q_1:T}(1, t^{(2)}, t)\}_{t \geq t^{(2)}}$ constitute the partitioning of $D_{Q_1:T} \setminus \{q_{1:T}(1), q_{1:T}(2)\}$.

Use same strategy as before: score each block (based on max path within), find the (a) max, and then compute it within that one block.
Thus, the blocks \( \{D_{Q_1:T}(1, t)\}_{t \neq t^{(2)}} \) and \( \{D_{Q_1:T}(1, t^{(2)}, t)\}_{t \geq t^{(2)}} \) constitute the partitioning of \( D_{Q_1:T} \setminus \{q_{1:T}(1), q_{1:T}(2)\} \).

Use same strategy as before: score each block (based on max path within), find the (a) max, and then compute it within that one block.

We have max scores of previous blocks, we only need to find max scores of new blocks.
Scores of new blocks

- For a bit of notational simplicity, let's set \( t' = t^{(2)} \) in the following. Then for \( t = t' = t^{(2)} \) we have:

\[
p^m(D_{Q_1:T}(1, t', t')) = p^m(D_{Q_1:T}(1, t')) \max_{q \not\in \{q^*_t(1), q^*_t(2)\}} \frac{m_{t'-1,t'}(q^*_{t'-1}(2), q)}{m_{t'-1,t'}(q^*_{t'-1}(2), q^*_t(2))} \tag{8.70}
\]

and for \( t > t^{(2)} = t' \)

\[
p^m(D_{Q_1:T}(1, t', t)) = p^m(D_{Q_1:T}(1, t')) \max_{q \not= q^*_t(2)} \frac{m_{t-1,t}(q_{t-1}(2), q)}{m_{t-1,t}(q^*_{t-1}(2), q^*_t(2))} \tag{8.71}
\]
Scores of new blocks

For a bit of notational simplicity, let’s set $t' = t^{(2)}$ in the following. Then for $t = t' = t^{(2)}$ we have:

\[
p^m(D_{Q_1:T}(1, t', t')) = p^m(D_{Q_1:T}(1, t')) \frac{\max_{q \notin \{q^*_t(1), q^*_t(2)\}} m_{t' - 1, t'}(q^*_{t' - 1}(2), q)}{m_{t' - 1, t'}(q^*_{t' - 1}(2), q^*_t(2))}
\]

(8.70)

and for $t > t^{(2)} = t'$

\[
p^m(D_{Q_1:T}(1, t', t)) = p^m(D_{Q_1:T}(1, t')) \frac{\max_{q \neq q^*_t(2)} m_{t - 1, t}(q_{t - 1}(2), q)}{m_{t - 1, t}(q^*_{t - 1}(2), q^*_t(2))}
\]

(8.71)

The third most probable sequence has probability

\[
p(q^*_1(3), \bar{x}_{1:T}) = \max \left\{ \max_{t:t \neq t'} p^m(D_{Q_1:T}(1, t)), \max_{t:t \geq t'} p^m(D_{Q_1:T}(1, t', t)) \right\}
\]

(8.72)
Scores of new blocks

- For a bit of notational simplicity, let's set $t' = t^{(2)}$ in the following. Then for $t = t' = t^{(2)}$ we have:

$$p^m(D_{Q_1:T}(1, t', t')) = p^m(D_{Q_1:T}(1, t')) \max_{q \notin \{q^*_{t'}(1), q^*_{t'}(2)\}} \frac{m_{t'-1,t'}(q^*_{t'} - 1(2), q)}{m_{t'-1,t'}(q^*_{t'} - 1(2), q^*_{t'}(2))} \quad (8.70)$$

and for $t > t^{(2)} = t'$

$$p^m(D_{Q_1:T}(1, t', t)) = p^m(D_{Q_1:T}(1, t')) \max_{q \neq q^*_t(2)} \frac{m_{t-1,t}(q_{t-1}(2), q)}{m_{t-1,t}(q^*_{t-1}(2), q^*_{t}(2))} \quad (8.71)$$

- The third most probable sequence has probability

$$p(q^*_1:T(3), \bar{x}_{1:T}) = \max \left\{ \max_{t:t \neq t'} p^m(D_{Q_1:T}(1, t)), \max_{t:t \geq t'} p^m(D_{Q_1:T}(1, t', t)) \right\} \quad (8.72)$$

- Given new best block, use dynamic programming again to identify it.
General case

- the same pattern as above, iterative procedure to construct the next best.
General case

- the same pattern as above, iterative procedure to construct the next best.
- The basic pattern is:
General case

- the same pattern as above, iterative procedure to construct the next best.
- The basic pattern is:
- 1) look at the current partition of states sequences and find the block that contains the maximum;
General case

- the same pattern as above, iterative procedure to construct the next best.
- The basic pattern is:
  1) look at the current partition of states sequences and find the block that contains the maximum;
  2) partition that subset based on the set of possible differences;
General case

- the same pattern as above, iterative procedure to construct the next best.
- The basic pattern is:
  1) look at the current partition of states sequences and find the block that contains the maximum;
  2) partition that subset based on the set of possible differences;
  3) make sure that we have the computed max over all new blocks;
General case

- the same pattern as above, iterative procedure to construct the next best.
- The basic pattern is:
  1) look at the current partition of states sequences and find the block that contains the maximum;
  2) partition that subset based on the set of possible differences;
  3) make sure that we have the computed max over all new blocks;
  4) find the block with the maximum score that is maximum;
General case

- the same pattern as above, iterative procedure to construct the next best.
- The basic pattern is:
  1) look at the current partition of states sequences and find the block that contains the maximum;
  2) partition that subset based on the set of possible differences;
  3) make sure that we have the computed max over all new blocks;
  4) find the block with the maximum score that is maximum;
  5) use a dynamic programming recursion based on the max marginals to construct this new maximum assignment.
max marginal construction $O(TN^2)$
Compute Cost

- max marginal construction $O(TN^2)$
- block construction $O(kTN)$. 
Compute Cost

- max marginal construction $O(TN^2)$
- block construction $O(kTN)$.
- computing max scores: $O(kT \log(kT))$
Compute Cost

- max marginal construction $O(TN^2)$
- block construction $O(kTN)$.
- computing max scores: $O(kT \log(kT))$
- Dynamic programming: $O(kTN)$. 
Compute Cost

- max marginal construction $O(TN^2)$
- block construction $O(kTN)$.
- computing max scores: $O(kT \log(kT))$
- Dynamic programming: $O(kTN)$.
- Overall cost: overall computational costs of finding the $k$-best assignments is $O(TN^2 + kT \log(kT) + kTN)$. 
Memory Cost

- Storing max marginals: $O(TN^2)$
Memory Cost

- Storing max marginals: $O(TN^2)$
- Storing $k$-best paths: $O(Tk)$. 
Memory Cost

- Storing max marginals: $O(TN^2)$
- Storing $k$-best paths: $O(Tk)$.
- Storing the max scores: $O(kT)$
Memory Cost

- Storing max marginals: $O(TN^2)$
- Storing $k$-best paths: $O(Tk)$.
- Storing the max scores: $O(kT)$
- Overall: $O(TN^2 + kT)$. 
Comparison of previous and current algorithm

<table>
<thead>
<tr>
<th></th>
<th>compute</th>
<th>memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>original algorithm</td>
<td>$O(kTN^2)$</td>
<td>$O(kTN^2)$</td>
</tr>
<tr>
<td>new algorithm</td>
<td>$O(TN^2 + kT \log(kT) + kTN)$</td>
<td>$O(TN^2 + kT)$</td>
</tr>
</tbody>
</table>

**Table:** Comparison of computational and memory resources needed by the two algorithms.
Sources for Today’s Lecture

- “doc.pdf” sections 8.1 - 8.3