

# Automatic Speech Recognition

JHU Summer School on Human Language Technology

Izhak Shafran  
Barton 309, ECE, JHU  
`zakshafran@jhu.edu`

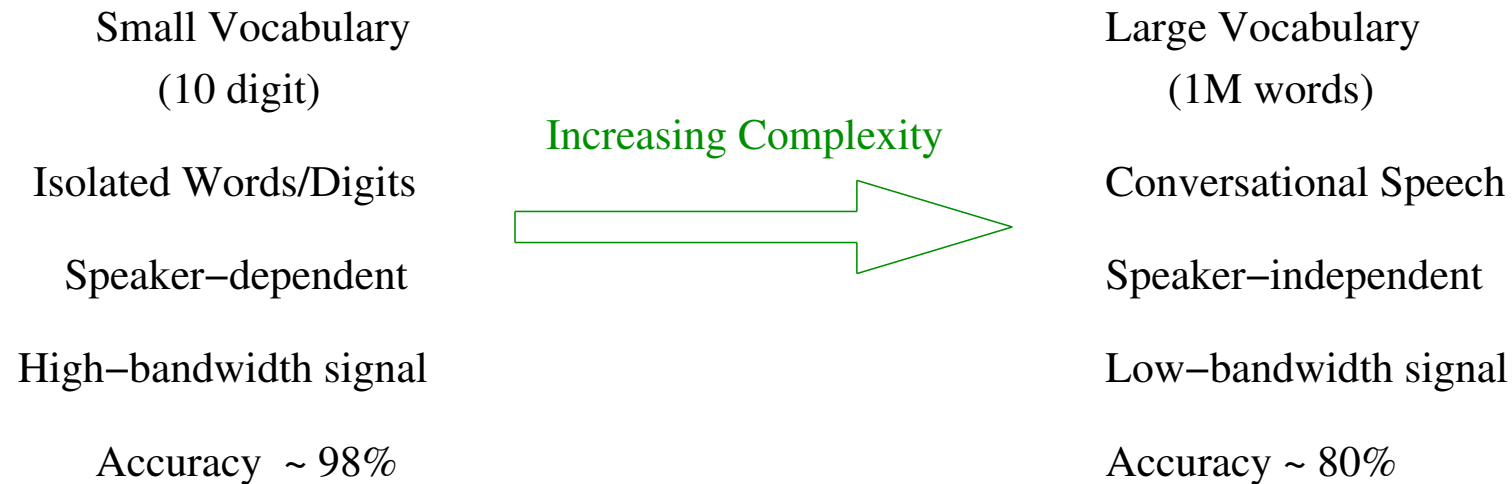
27 June, 2005

## Overview

- Introduction.
- Quick Tutorial on HMMs.
- Building a Simple Speech Recognition System.

## ASR Tasks

- Performance & complexity of an ASR system depends on the task.



In almost all cases, the underlying model used is a Hidden Markov Model.

**Focus of this lab:** Speaker-independent continuous digit recognition.

## A Quick Tutorial on HMMs

Any pattern recognition system needs to tackle 2 questions:

- How to measure the distance between two sequences?
- How to pick the best match from all the candidates?

Three popular techniques for sequence recognition are:

- Edit Distance + Dynamic Programming  
e.g. spell check, score error rate, gene matching.
- Vector Distance + Dynamic Time Warping  
e.g. simple isolated digit recognition.
- Hidden Markov Models + Viterbi Algorithm  
e.g. speech and handwriting recognition.

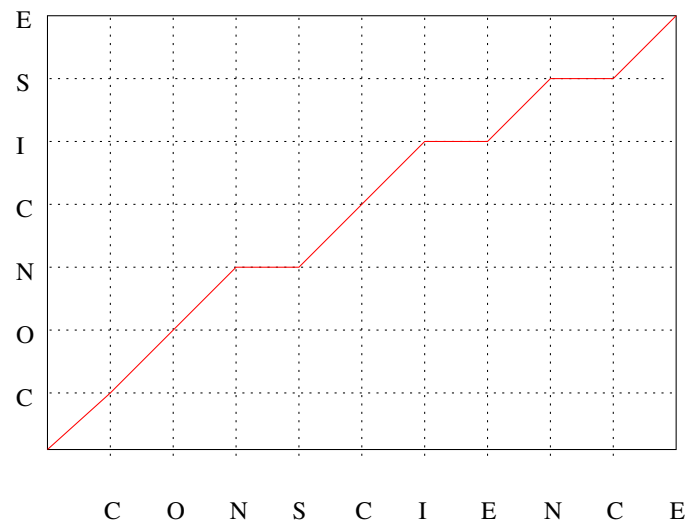
## Edit Distance: For Sequences of Symbols

Consider two sequences  $x = \textit{conscience}$  and  $y = \textit{concise}$  of lengths  $m = 10$  and  $n = 7$  respectively.

There are several possible alignments - e.g.

$(1, 1), (2, 2), (3, 3), (4, -), (5, 4), (6, 5), (7, -), (8, 6), (9, -), (10, 7)$

An alignment can be represented as a two dimensional matrix.



## Computing the Cost

Now, to score an alignment, we need to define elementary costs – cost of transitions and cost of being in aligned states.

- Cost of transition,  $(i_{k-1}, j_{k-1}) \rightarrow (i_k, j_k) := d_T(i_k, j_k | i_{k-1}, j_{k-1})$
- Cost of staying in an aligned state,  $(i_k, j_k) := d_S(i_k, j_k)$
- Combined cost at time step  $k$ ,

$$d(i_k, j_k | i_{k-1}, j_{k-1}) = d_T(i_k, j_k | i_{k-1}, j_{k-1}) + d_S(i_k, j_k)$$

- Cost of a complete alignment,

$$D := \sum_{k=1}^K d(i_k, j_k | i_{k-1}, j_{k-1})$$

## Computational Cost

Consider,  $|x| = m$  and  $|y| = n$ , let the start and end states be matched and  $m > n$ .

Then, the total number of alignments is  $\binom{m+n}{m} = \frac{(m+n)!}{m!n!}$ .

To get a sense of what this means, let's get an approximation.

Using Sterling's formula,  $\log(n!) \approx n \log(n) - n$   
total number of alignments  $> 2^{2n}!!$

Each alignment requires  $N$  additions, so  
total computational cost  $> N2^{2n}!!$

For  $n = 50$ , this is  $\approx 5 \times 10^{31}$ .

## Bellman's Optimality Principle (1957)

- Let  $(s, t) \xrightarrow{*} (u, v)$  denote the best path that begins and ends at arbitrary nodes  $(s, t)$  and  $(u, v)$ .
- Let  $(s, t) \xrightarrow{*}^{(w, x)} (u, v)$  denote the best path segment from  $(s, t)$  to  $(u, v)$  that also passes through  $(w, x)$ .
- Then, for any  $0 \leq s, w, u \leq m$  and  $0 \leq t, x, v \leq n$ ,

$$(s, t) \xrightarrow{*}^{(w, x)} (u, v) = (s, t) \xrightarrow{*} (w, x) \oplus (w, x) \xrightarrow{*} (u, v)$$

where  $\oplus$  denotes concatenation of the path segments.

## Dynamic Programming

Consequences of Bellman's optimality principle for matching patterns with Markovian dependence.

$$(0, 0) \xrightarrow[*]{(i_{k-1}, j_{k-1})} (i_k, j_k) = (0, 0) \xrightarrow[*]{(i_{k-1}, j_{k-1})} (i_{k-1}, j_{k-1}) \oplus (i_{k-1}, j_{k-1}) \xrightarrow[*]{(i_k, j_k)} (i_k, j_k)$$

That is,

$$D_{\min}(i_k, j_k) = \min_{(i_{k-1}, j_{k-1})} \{D_{\min}(i_{k-1}, j_{k-1}) + d((i_k, j_k) | (i_{k-1}, j_{k-1}))\}$$

where  $D_{\min}(0, 0) := 0$ .

Note, the application usually defines the constraints of possible elementary step  $(i_{k-1}, j_{k-1}) \longrightarrow (i_k, j_k)$ .

## Dynamic Programming

Apart from obtaining a score, often, we would also like to extract the best alignment. How can we do that?

Define  $\Psi(i_k, j_k)$  as the index of the predecessor node to  $(i_k, j_k)$  on  $(0, 0) \xrightarrow{*} (i_k, j_k)$ .

Clearly, if we know the predecessor node to any node on  $(0, 0) \xrightarrow{*} (i_k, j_k)$ , then the entire path can be obtained by *backtracking* from the terminal node,  $(i_k, j_k) = (M, N)$ .

## Dynamic Programming

Example: *Edit distance* or *Levenshtein distance* (1965).

- Cost of being in an aligned state,  $d_S(a, b) = 1 - \delta(a, b)$ .
- Transitions allowed are insertions, deletions and substitutions. And, they have equal cost.

Thus,

$$D_{\min}(i_k, j_k) = \min\left\{ \begin{array}{l} D_{\min}(i_{k-1}, j_{k-1}) + d_S(i_k, j_k), \\ D_{\min}(i_k, j_{k-1}) + d_S(-, j_k), \\ D_{\min}(i_{k-1}, j_k) + d_S(i_k, -) \end{array} \right\}$$

where  $D_{\min}(0, 0) := 0$ .

Computational cost:  $O(NM)$ .

## Dynamic Time Warping: For Sequences of Vectors

Consider two vector sequences,  $h(t)$  and  $r(t)$  of lengths  $m$  and  $n$  respectively.

Cost of alignment:

$$D := \sum_{k=1}^K d(i_k, j_k | i_{k-1}, j_{k-1})$$

Two costs for each time step: transition + being in current state

$$d(i_k, j_k | i_{k-1}, j_{k-1}) = d_T(i_k, j_k | i_{k-1}, j_{k-1}) * d_V(h(i_k), r(j_k))$$

Cost of staying in an aligned state:

$$d_S(i_k, j_k) = d_V(h(i_k), r(j_k))$$

where  $d_V$  could be Euclidean distance, Itakura distance, ....

## Distances

**Itakura Distance:** Let  $x$  and  $y$  be the LPC coefficients for the “observed” and the “reference” sequence. Let  $R$  be the autocorrelation matrix of the input signal. Then, the Itakura distance is defined as the log of the LPC residual errors.

$$D(x, y) = \log \frac{E_x}{E_y} = \frac{x^T R x}{y^T R y}$$

**Mahalanobis Distance:** Let  $x$  and  $y$  be the input feature vector, again for the “observed” and the “reference”.

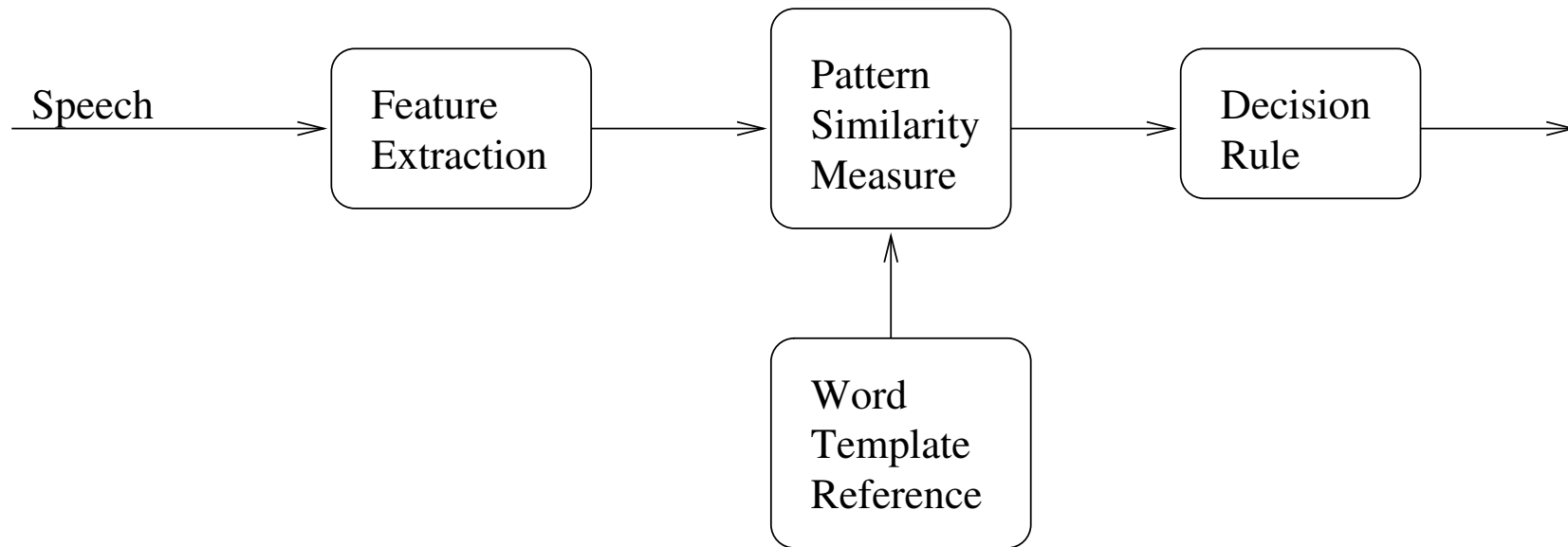
$$D(x, y) = (x - y)^T \Sigma^{-1} (x - y)$$

## Dynamic Time Warping

### Search constraints

- End points are constrained to match:  $(h(1), r(1))$  and  $(h(M), r(N))$ . Alternatively, the constraints may be a bit more relaxed as in *Bridle algorithm*.
- Monotonicity:  $i_{k-1} \leq i_k$  and  $j_{k-1} \leq j_k$ . In other words, no going back in time  $t$ .
- Local neighborhood constraints: types of transitions allowed  $(i_{k-1}, j_{k-1}) \longrightarrow (i_k, j_k)$ .
- Alternatively, the costs may be normalized. However, certain normalizations are not compatible with DP.

## Isolated Word Recognition



- Whole word templates.
- But, difficult to train and capture variations within a template.

Check out the cool demo!

## Matching Multiple Patterns

- Search space can be represented as a directed graph.
- Or, as a prefix tree.
- Patterns can be searched in time-synchronous manner. That is, pursue all paths in parallel.
- Or, apply depth-first search.
- Or, by best-first search, a greedy search using a queue to store partial paths (not constrained by DP assumptions).

## Pruning and Beam Search

Both greedy and dynamic programming can take advantage of optimal substructure. How?

Let  $\Phi(i, j)$  be the best path from node  $i$  to node  $j$ , ending at a particular time,  $t$ .

If  $k$  is a node in  $\Phi(i, j)$ , then  $\Phi(i, j) = \min_k \Phi(i, k) + \Phi(k, j)$ .

Solution to subpaths need to be computed only once, and suboptimal partial paths can be discarded.

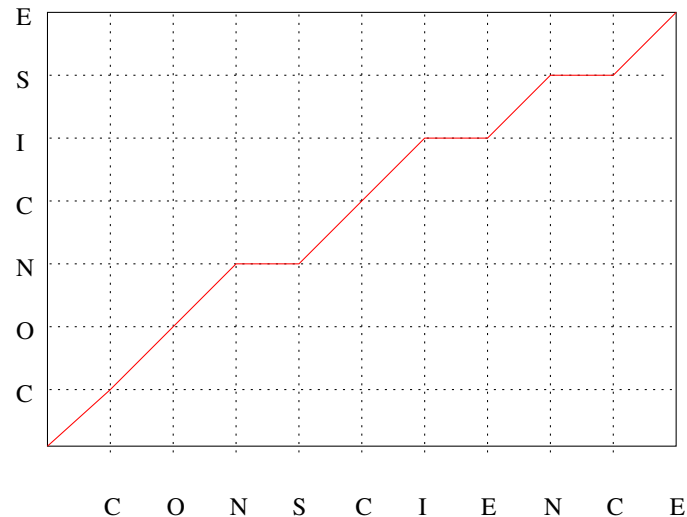
Types of pruning: beam width can be static or dynamic based on relative scores.

In greedy search, partial path score  $\Phi(1, k)$  can be augmented with an underestimate of future cost  $\hat{\Phi}(k)$ , giving A\* search.

Search can also produce N-best hypothesis.

# HMM: Mapping Vector Sequences to Symbols

Recall, dynamic time warping algorithm.



$$d(i_k, j_k | i_{k-1}, j_{k-1}) = d_T(i_k, j_k | i_{k-1}, j_{k-1}) * d_V(h(i_k), r(j_k))$$

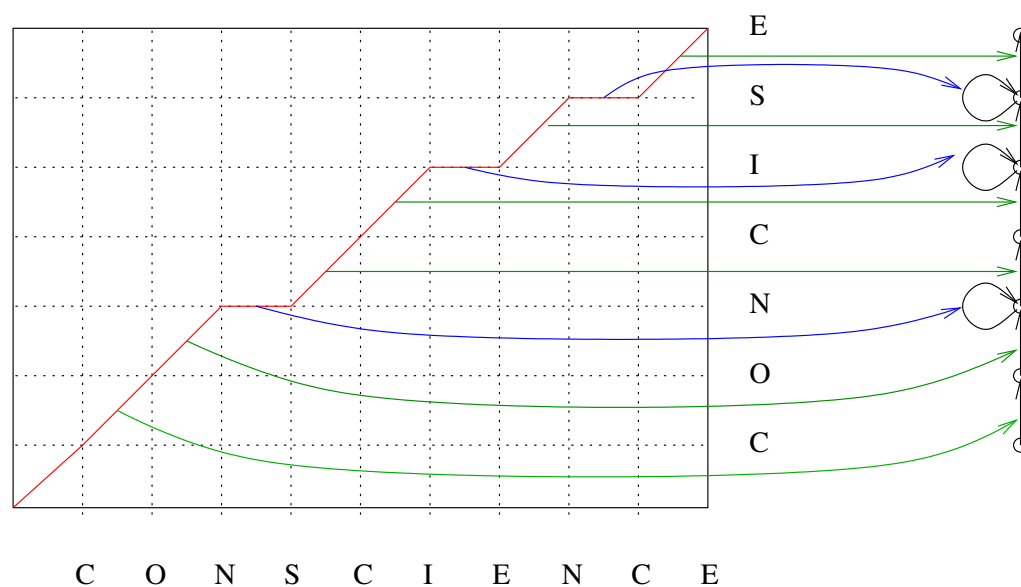
$$D := \sum_{k=1}^K d(i_k, j_k | i_{k-1}, j_{k-1})$$

where  $d_V(h(i_k), r(j_k))$  is a standard distance metric such as Euclidean, Mahalanobis, Itakura ....

With a few steps the **DTW can be morphed into an HMM.**

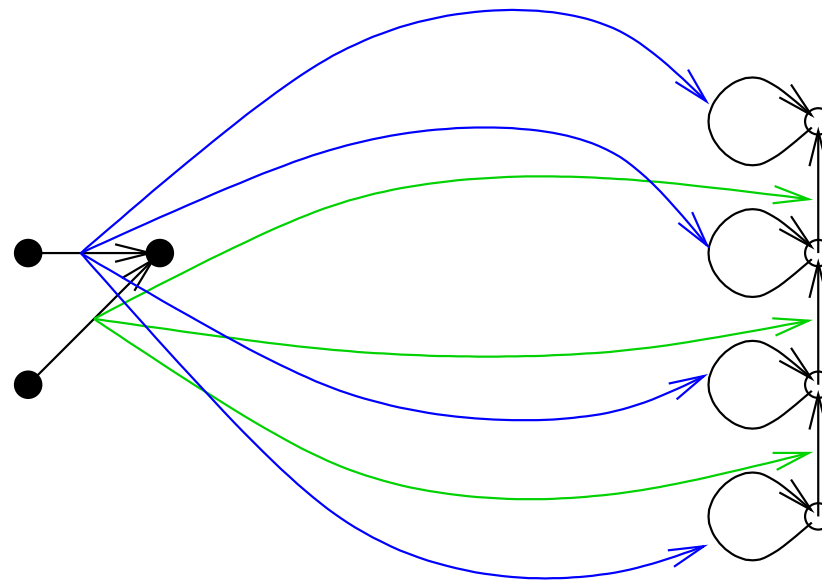
## DTW Transitions

Now, replacing the template elements with states, every horizontal path element with a self loop and other transitions as shown, we have.



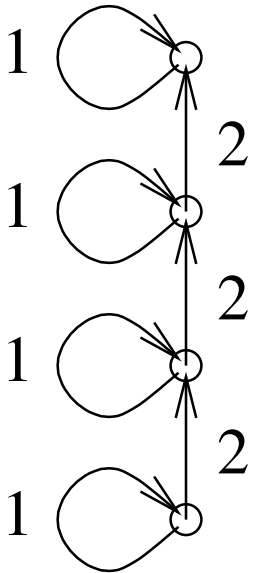
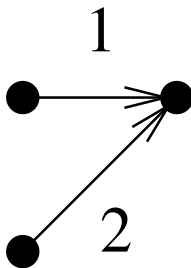
## DTW Transitions

In general, path constraints can be mapped to transitions of a finite state machine (FSM).



# DTW Transitions

Apart from encoding the path constraints, the transitions weights can also be included in the FSM.



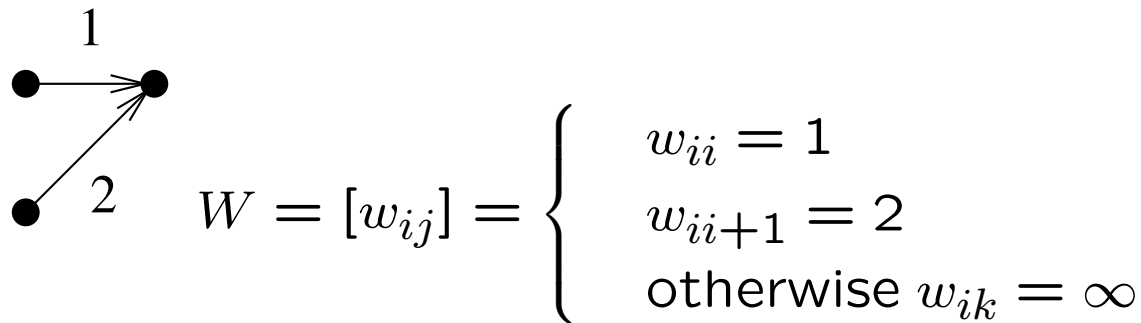
## DTW Transition Matrix

Alternatively, the path constraints can be represented as a transition matrix  $W$  of the FSM.

$$W = [w_{ij}]$$

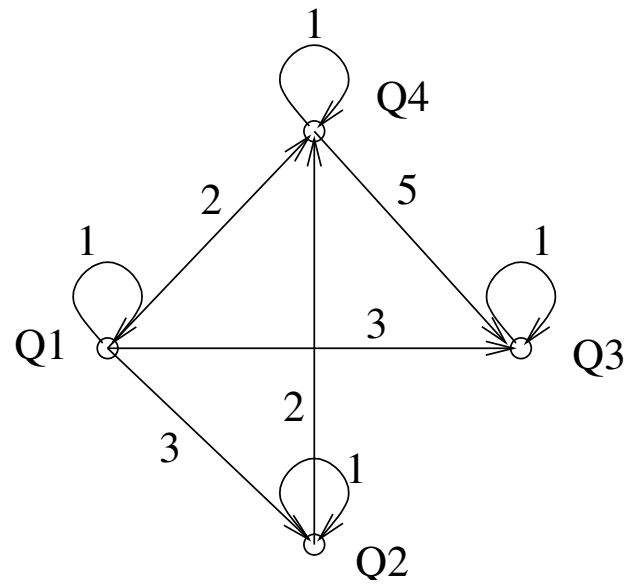
where  $w_{ij}$  is the weight or cost of going from state  $i$  to  $j$ .

Example:



## Transition Matrix

Another example:



$$W = \begin{pmatrix} 1 & 3 & 3 & 2 \\ \infty & 1 & \infty & 2 \\ \infty & \infty & 1 & \infty \\ \infty & \infty & 5 & 1 \end{pmatrix}$$

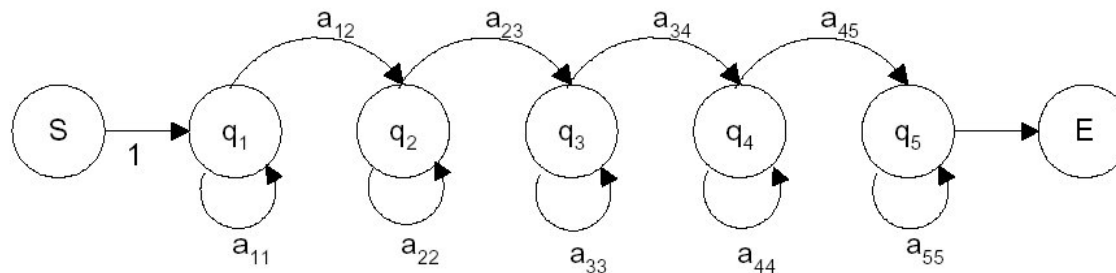
## Probabilistic Transitions

Let  $w_{ij} = -\log c_{ij}$ . Consider the following transformation of the weights.

$$A = [a_{ij}]$$

$$a_{ij} = \frac{c_{ij}}{\sum_j c_{ij}} = \frac{\exp(-w_{ij})}{\sum_j \exp(-w_{ij})}$$

Note  $w_{ij} = \infty$  implies  $a_{ij} = 0$ . Assuming  $w_{ij} > 0$ , we have  $0 \leq a_{ij} \leq 1$  and  $\sum_j a_{ij} = 1$ . Thus,  $a_{ij}$ s are transition probabilities.



Now, we have a *stochastic finite state automata*, a finite state machine with probabilistic transitions.

## DTW to HMM

- Replace transition costs with transition probabilities.
- Replace cost of being in a state (e.g. Mahalanobis distance) with its probabilistic counterpart (e.g. Gaussian).

The cost in DTW is:

$$D := \sum_{k=1}^K d_T(i_k, j_k | i_{k-1}, j_{k-1}) * d_V(h(i_k), r(j_k))$$

Its probabilistic counterpart is:

$$D := \prod_{k=1}^K P(h(i_k) | q(j_k)) a_{j_{k-1}j_k}$$

$$\log D = \sum_{k=1}^K [\log P(h(i_k) | q(j_k)) + \log a_{j_{k-1}j_k}]$$

## Bayes Decision Rule: DTW vs HMMs

In DTW based recognition, the decision rule is:

$$W = \min_W D(X, W)$$

On the other hand, the Bayes decision rule is:

$$W = \max_W P(X, W)$$

Both measures are strikingly similar. But ...

- DTW allows 2-path transitions which HMMs don't.
- HMM allows cyclic paths while DTW doesn't.
- HMMs can support complex distance metric to evaluate the probability (score) of being in a state (e.g. GMM).
- Parameters of an HMM can be estimated using simple and efficient algorithms.