

Topic:

Search - Part I + II

Readings:

- (1) X.Huang/Acero/Hon: Chapter 12,13
- (2) Sakoe: Two-level DP-matching (in Waibel/Lee, pp 180ff)
- (3) Ney: One-stage DP (in Waibel/Lee, pp 188ff)

Major Parts of these slides were provided by Monika Woszczyna

- Part 1 What is search about, Motivation
DTW, Viterbi, Continuous SR: Two-level DTW, One-stage DP
Search Strategies, Stack Decoder
- Part 2 Optimization strategies, Tree search, Pruning
Search with LM / Grammars
N-best hypotheses,
Speeding up Search

Search (Part 1)

- The Search in Automatic Speech Recognition
- DTW review \Rightarrow pattern based recognition, Optimizations
- Viterbi review \Rightarrow model based recognition, Optimizations
- Continuous speech recognition
 - Reasons against predicting word boundaries
 - Two level DP
 - One stage DP, Search strategies, stack decoder
- Optimization: How to waste not too much Computation Time
 - Tree-Search, Pruning, Pruning with Beamsearch
- Search with LM / Grammar
- Multi-Pass Searches, Problems and Examples
- Producing more than one Hypothesis, Problems
- Speeding up the Search
- Search with Context-Dependent Models

What Is This All About?

Fundamental Equation of Speech Recognition:

Observe a sequence of feature vectors X , Find the most likely word sequence W

While W is represented as example pattern or as a sequence of states in an HMM

$$\arg \max_W P(W | X) = \arg \max_W \frac{P(W) p(X | W)}{\cancel{P(X)}}$$

$p(X|W)$: Acoustic Model

$P(W)$: Language Model

argmax_W : How to efficiently try all possible W 's? = Search/Decoding

Search in Automatic Speech Recognition

- The entire set of possible HMM state sequences / sequences of pattern is called the **search space**
- Typical search spaces have
 - 1,000 time frames (10sec speech) and 500,000 HMM states
 - With an average of 25 words per sentence (e.g. WSJ) and a vocabulary of 64,000 words, more possible word sequences than the universe has atoms!
- ↑ It is not feasible to compute the most likely sequence of words by evaluating the likelihoods of all possible sequences
- We need an *intelligent* algorithm that scans the search space and finds the best (or at least a very good) hypothesis
- This problem is referred to **search** or **decoding**

Search in Automatic Speech Recognition

Isolated Word Recognition:

- For every applicable vocabulary word:
 - Compute it's score (DTW)
 - Compute it's likelihood (forward algorithm)
- Report the word with the best score/likelihood

Problems, Optimization, Drawbacks

Continuous Speech Recognition:

- Not only find probability of words ...
... BUT ...
- ... also find the word sequence
(i.e. a sequence through the huge state graph)

Problems, Strategies, Optimization, Drawbacks

DTW review

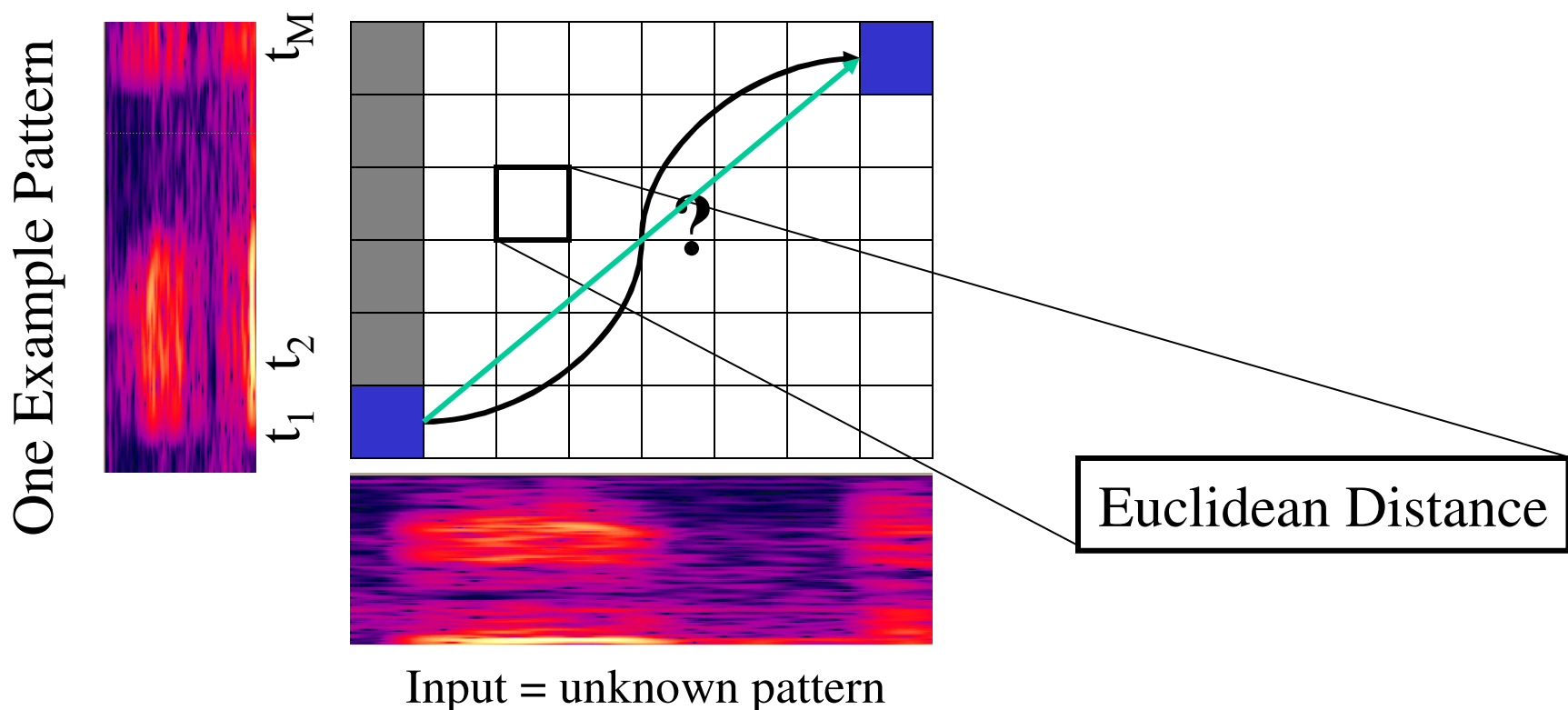
Goal: identify example pattern that is most similar to unknown input

⇒ compare patterns of different length

Note: all patterns are preprocessed ⇒ 100-120 vectors / second of speech

DTW: Find alignment between unknown input and the example pattern that minimizes the overall distance

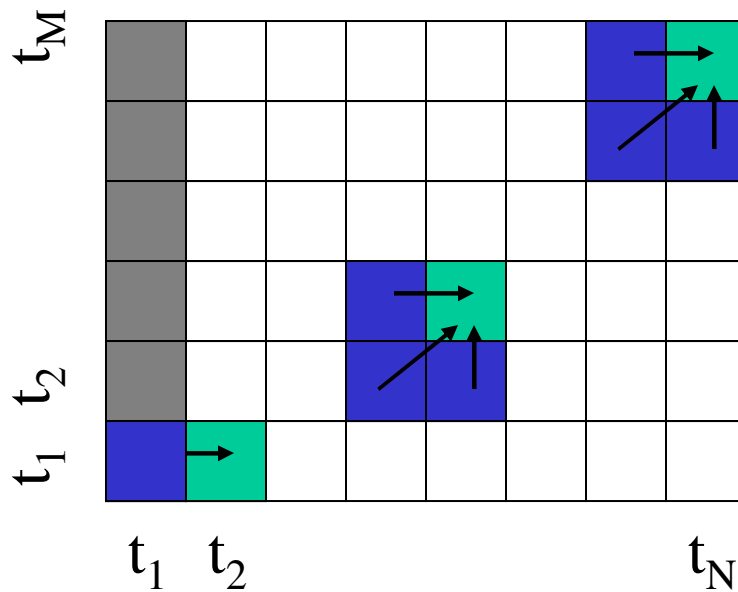
Compute total vector distance, but which frame-pairs ?



DTW review

Example Pattern 1

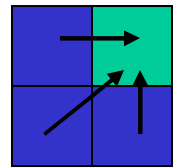
Apply transition function



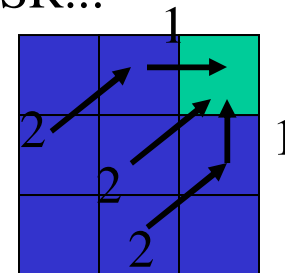
Unknown pattern

Editing Distance,
for spell-checkers..

$$C(x,y) = \min (\begin{aligned} &C(x-1,y) + d_{\text{ins}}(x,y), \\ &C(x-1,y-1) + d_{\text{sub}}(x,y), \\ &C(x,y-1) + d_{\text{del}}(x,y) \end{aligned})$$



Weighted Distance
for SR..:



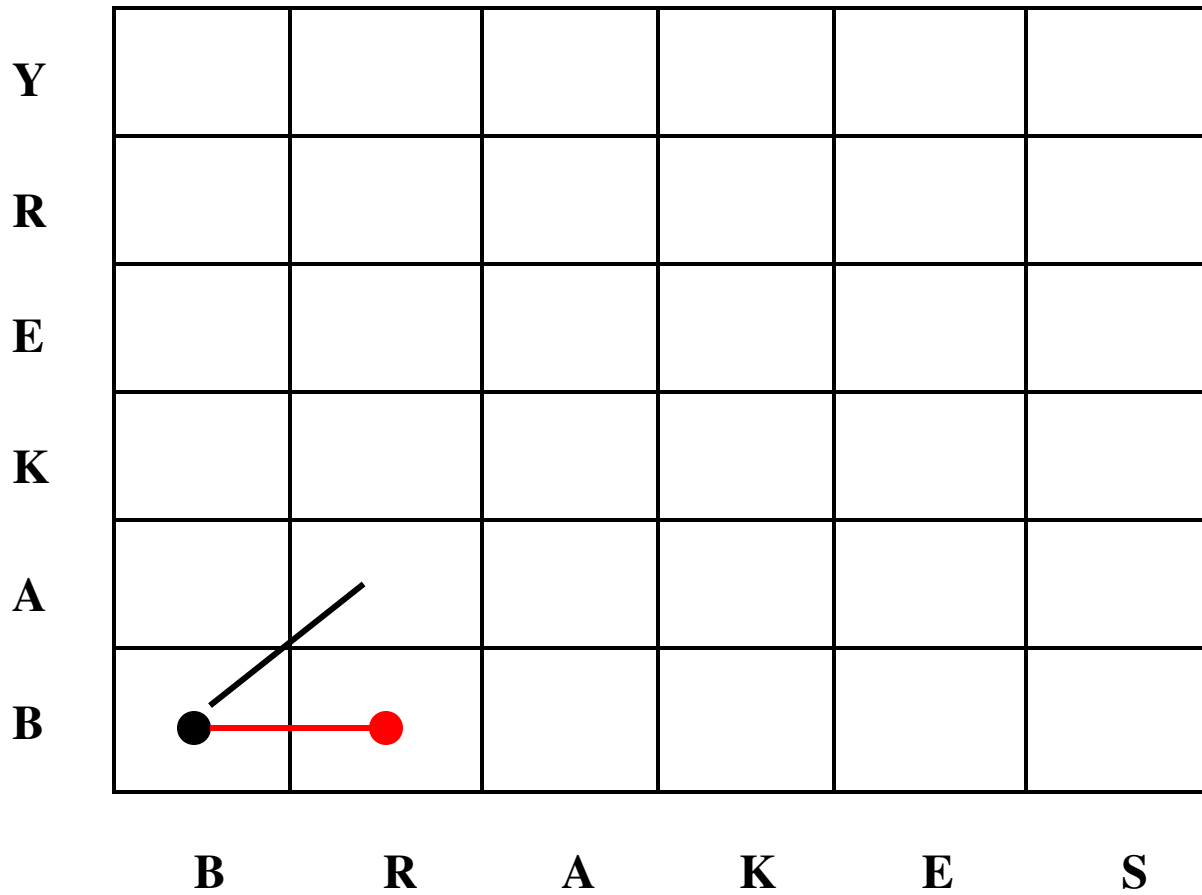
Levinshtein

Editing Distance between 'brakes' and 'bakery'

Y						
R						
E						
K						
A						
B	●					
	B	R	A	K	E	S

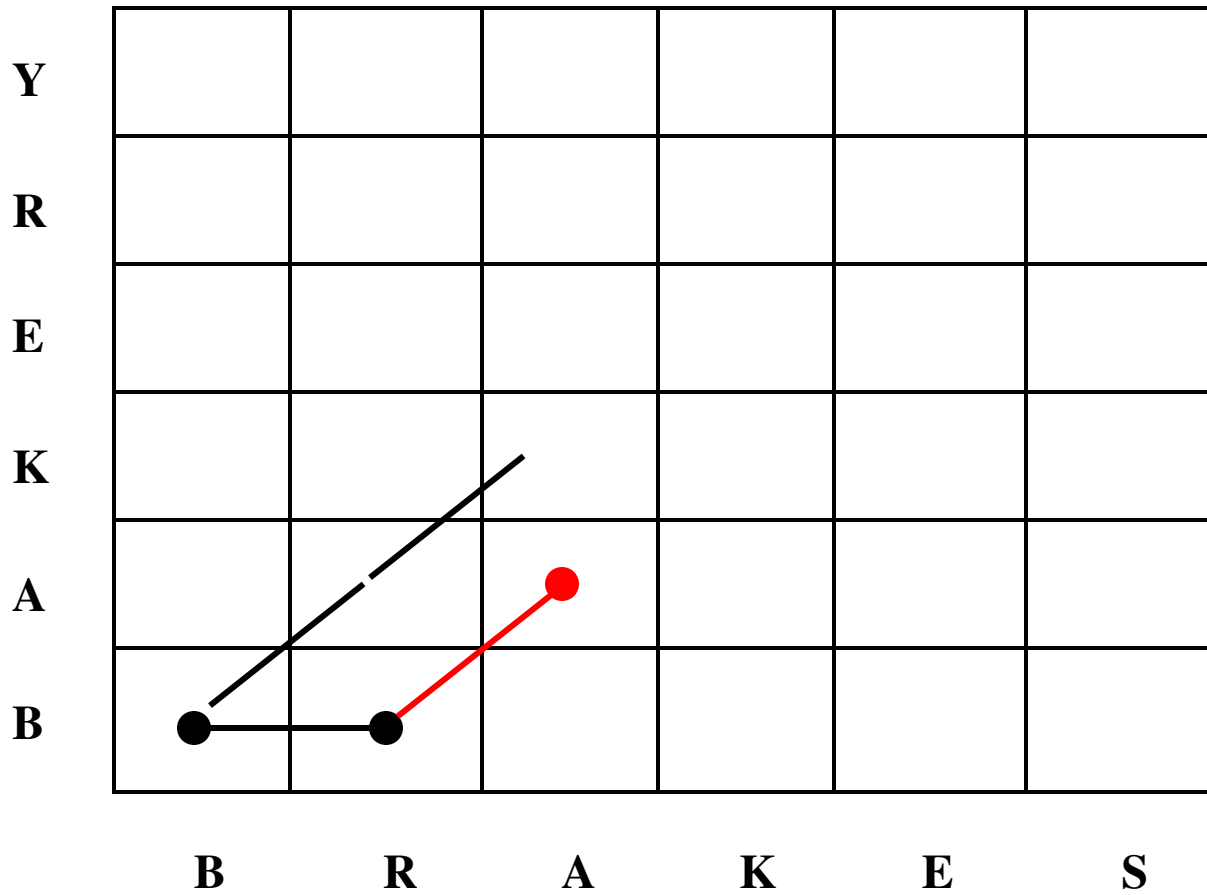
B=B, move to next

Levinshtein



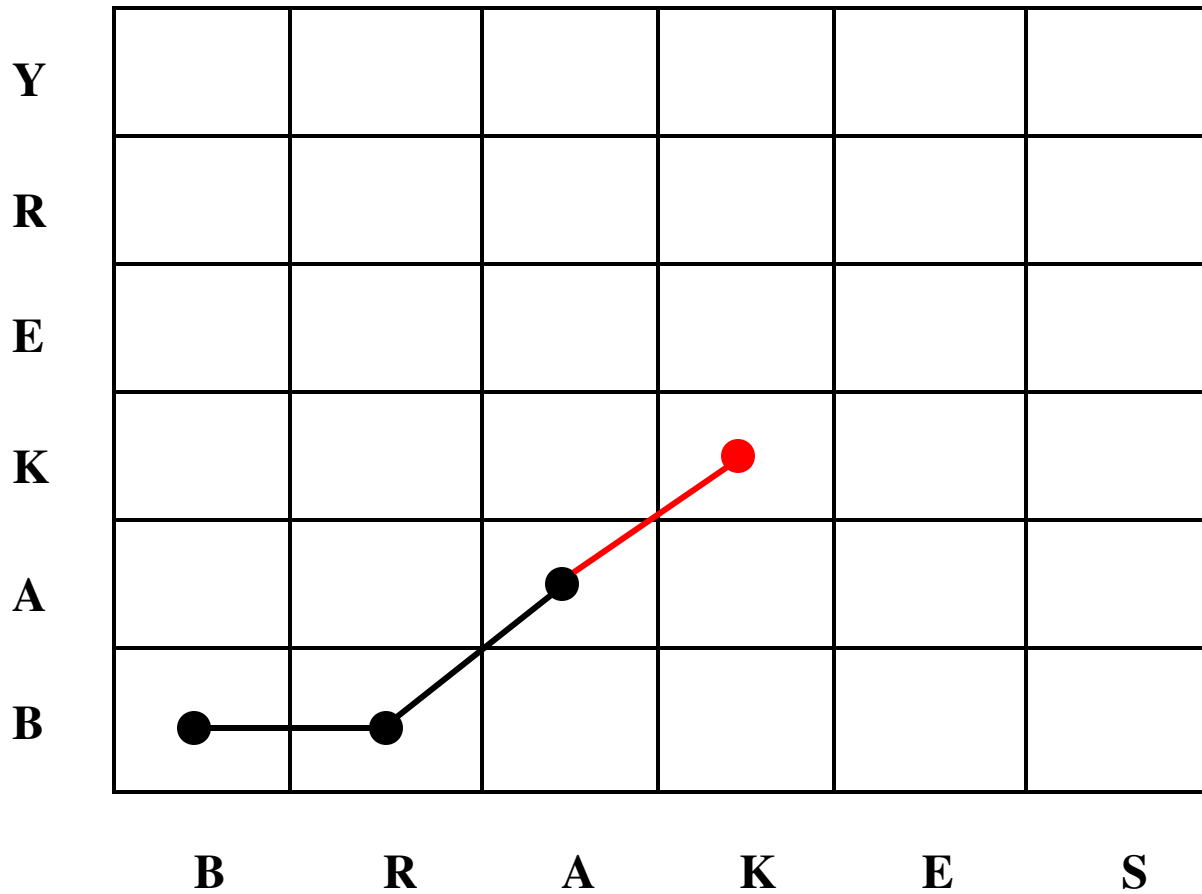
Insert character $x_2=R$

Levinshtein



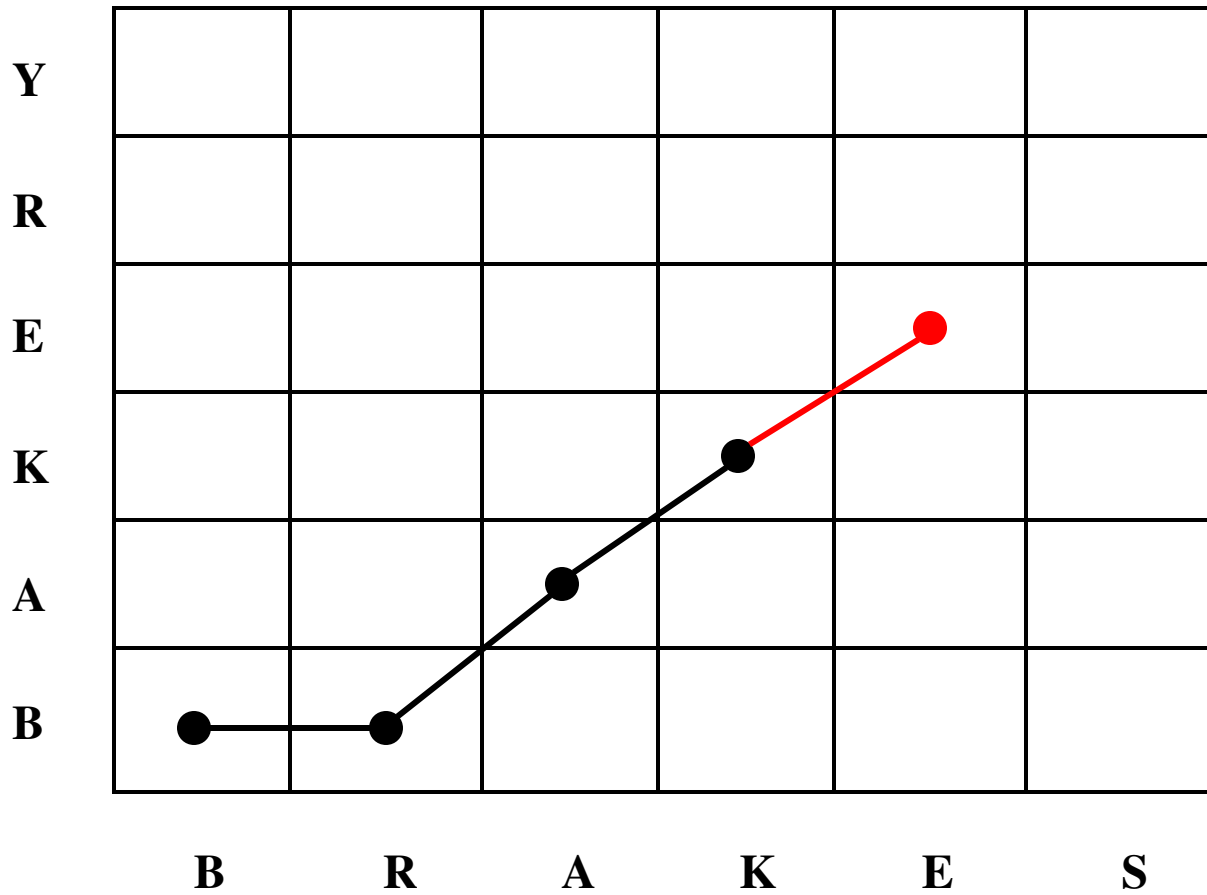
A=A, move to next

Levinshtein



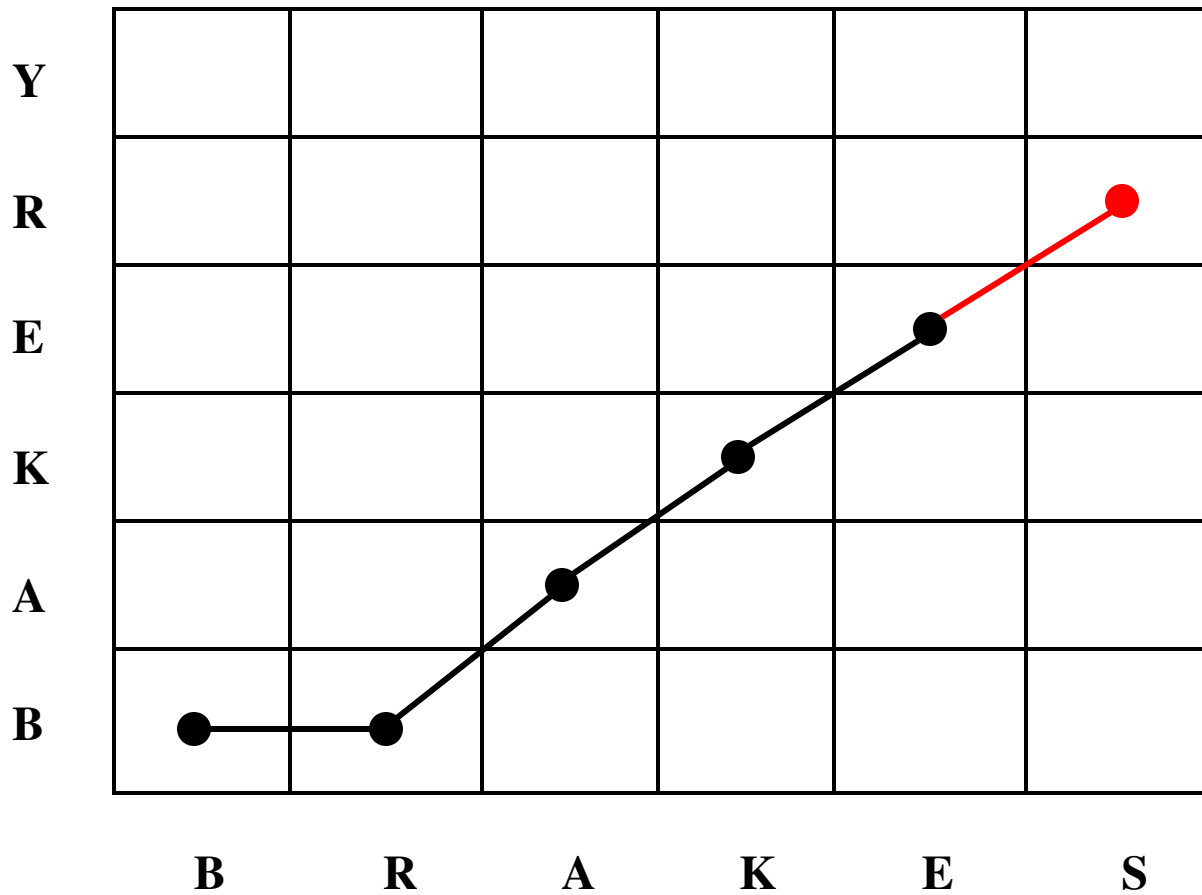
K=K, move to next

Levinshtein



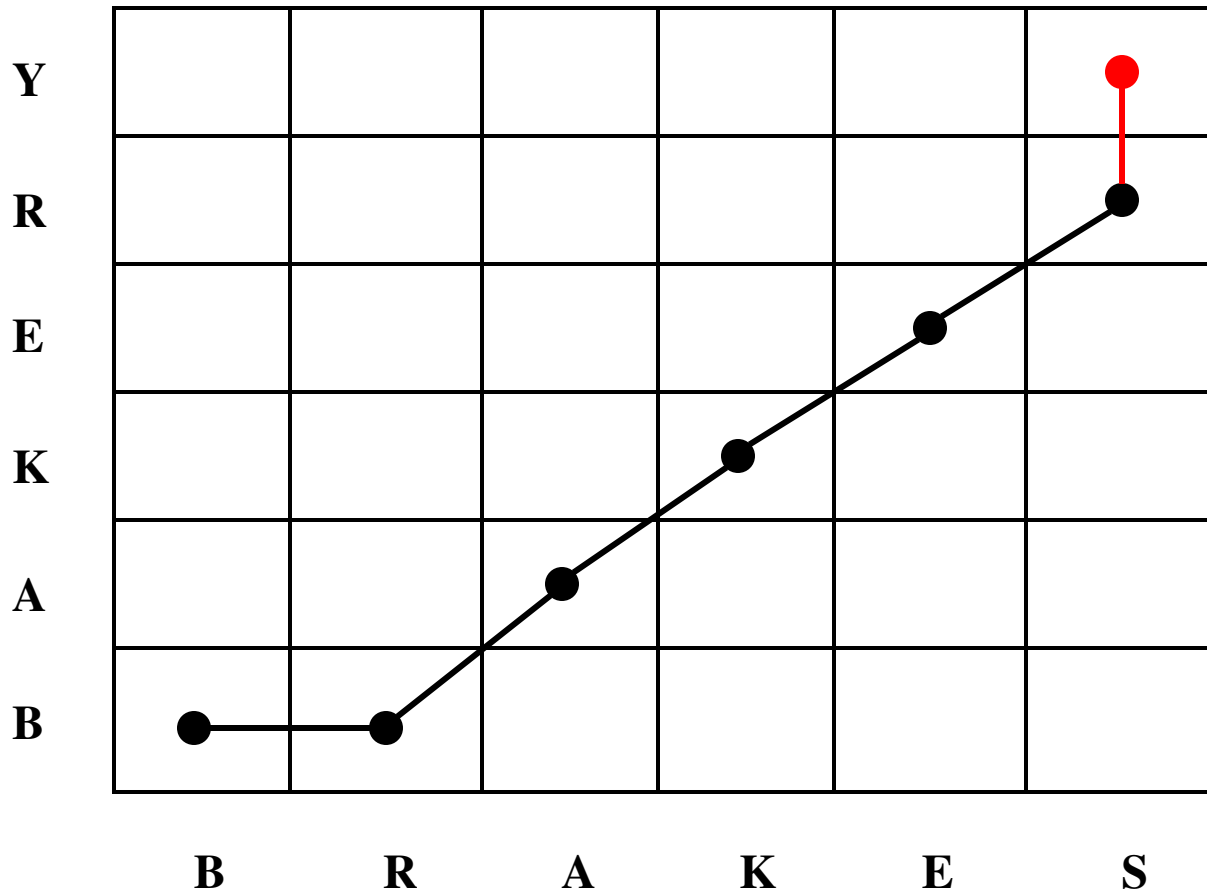
E=E, move to next

Levinshtein



substitute character $x_6 = S$
with character $y_5 = R$

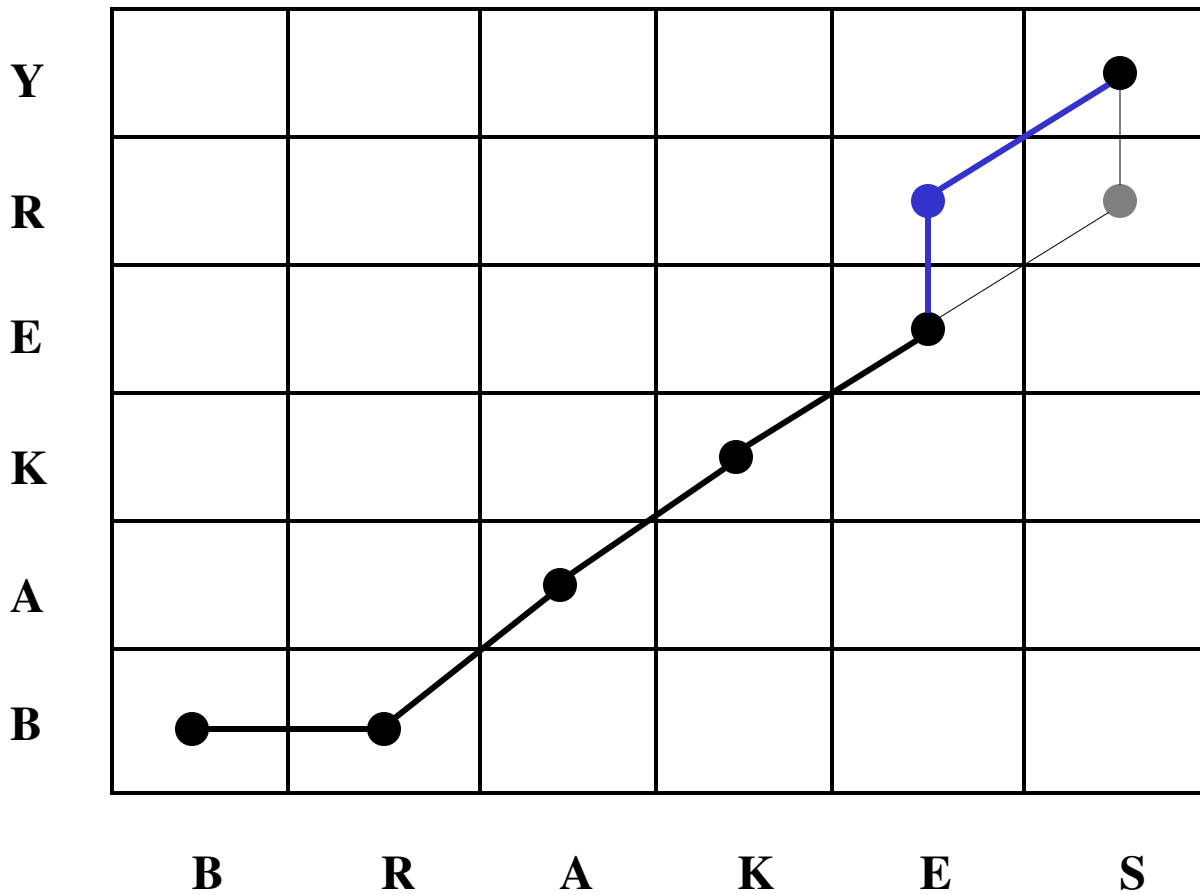
Levinshtein



delete character $y_6 = Y$

Levinshtein

Sequence is not necessarily unique!



delete character $y_5 = R$

substitute character $x_6 = S$ with $y_6 = Y$

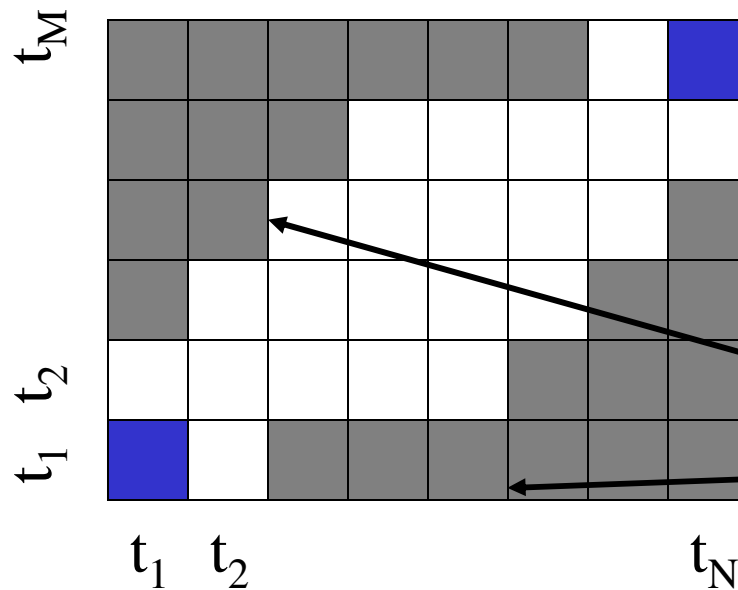
DTW optimization

How to save computation time?

Solution: Add constraint

- path should be close to diagonal,
- define start and end point
- no two successive horizontal or vertical steps

Example Pattern 1

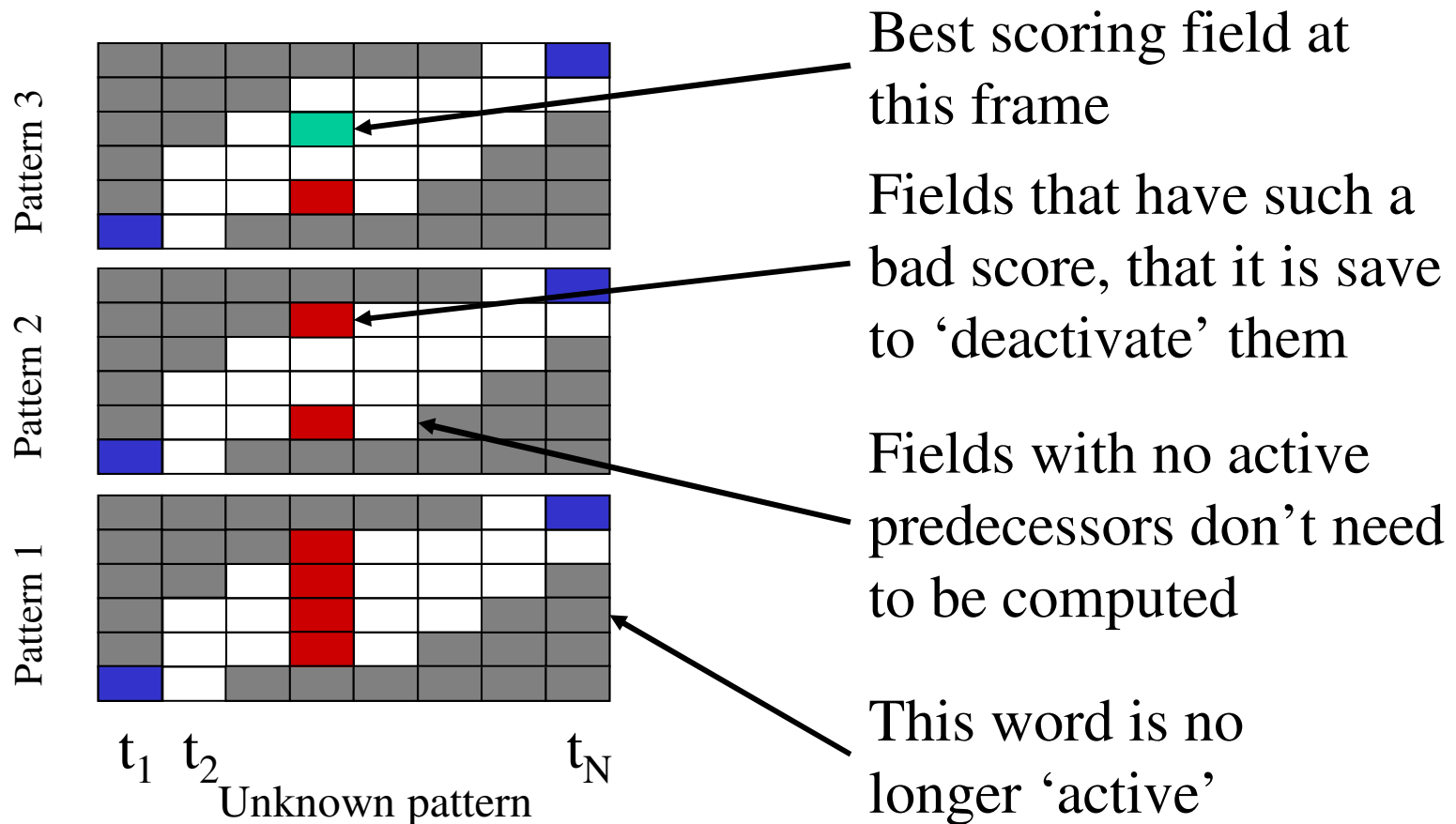


Shaded fields cannot be reached and don't need to be computed (inactive)

Unknown pattern

DTW optimization (2)

Pruning beam: Eliminate losers (score > best score + Δ)



DTW Summary

Optimization of DTW:

- Usually, only interested in final score
- Algorithm requires only values in current and previous frame
- Don't allocate new storage but overwrite stuff that is no longer needed
- For most transition patterns, one frame is enough

Drawbacks of DTW:

- Does not generalize over multiple training patterns
- Speaker dependent
- Need example(s) for each word from each speaker
- Gets computationally expensive for large vocabularies

Viterbi Review

Hidden Markov Models ?!

Observation Probabilities ?!

Differences between Forward/Backward, Viterbi and DTW

Duration modeling

Viterbi Optimization

Viterbi review

Viterbi: find probability along most likely state sequence

The following **view** may differ a bit from previous explanations.

Multiplications are slow and can lead to floating point overflow \Rightarrow log:

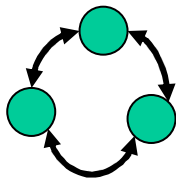
$$\log (\max (P(a) * P(b)))$$

$$\max (\log (P(a) * P(b)))$$

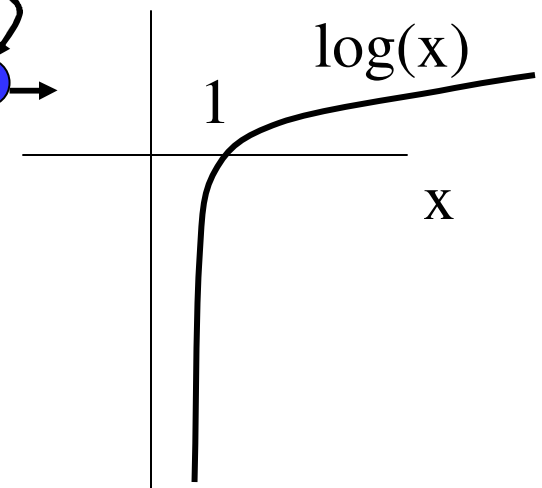
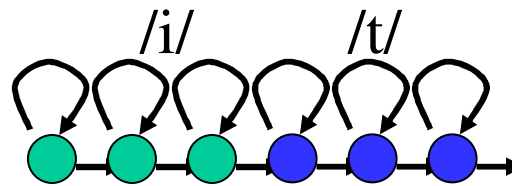
$$\max (\log(P(a)) + \log(P(b)))$$

$$\min (-\log(P(a)) + -\log(P(b)))$$

Ergodic (urn examples)

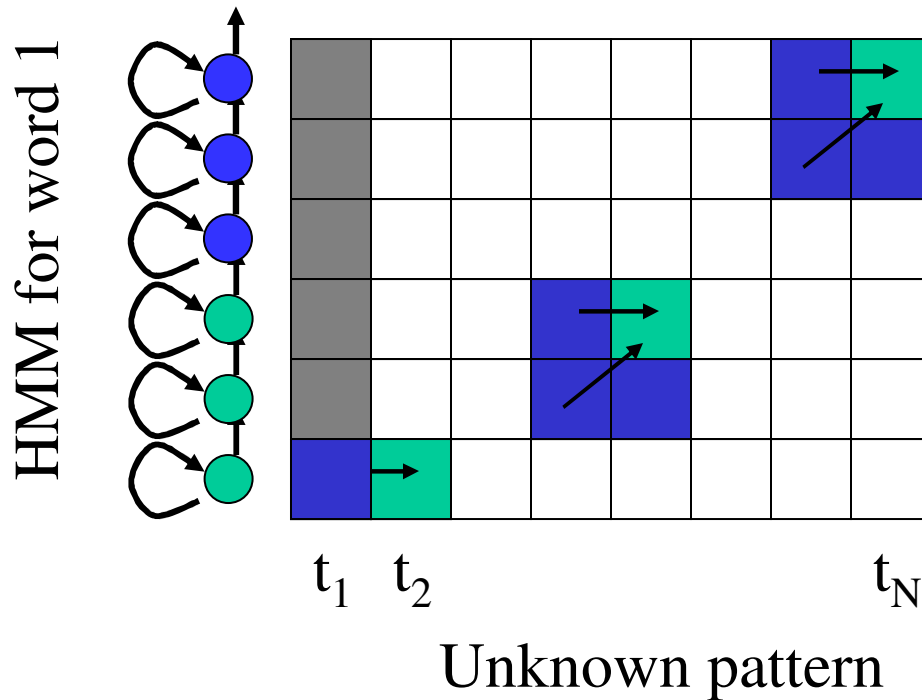


Topology for speech:



Viterbi review

Compute state transitions



Without transition probabilities:

$$-\log (P(x,y)) = \min ($$
$$-\log (P(x-1,y)) + d(x,y),$$
$$-\log (P(x-1,y-1)) + d(x,y)$$
$$)$$
$$d(x,y) = \log(p(\text{frame}_x | s_y))$$

\Rightarrow a lot like DTW

Viterbi alignment & DTW

Decisions on best predecessor state are **final**

- independent of later scores

Local scores and transition penalties only

- independent of all previous decisions

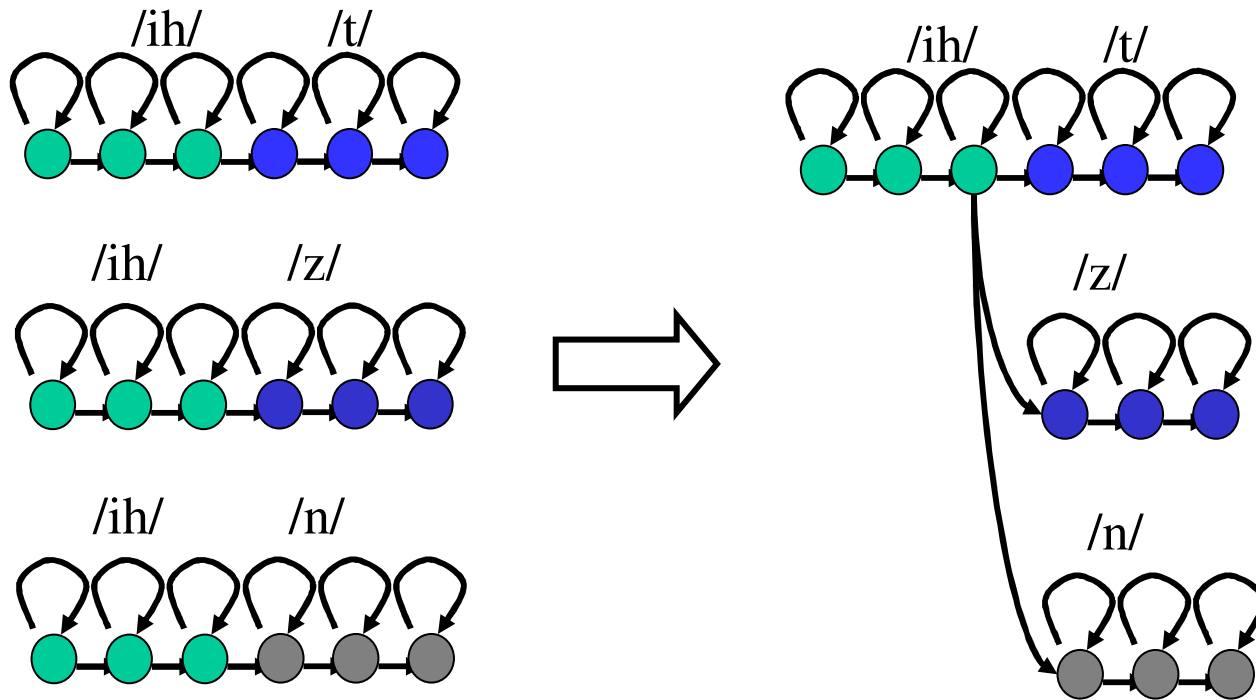
To use information that violates these assumptions:

- either decisions have to be postponed
(usually by making state copies)
- or global optimum no longer found
- Example:
 - explicit duration models
 - language models

Viterbi optimization

Tree search:

- All words that start with the same model start with exactly the same computations

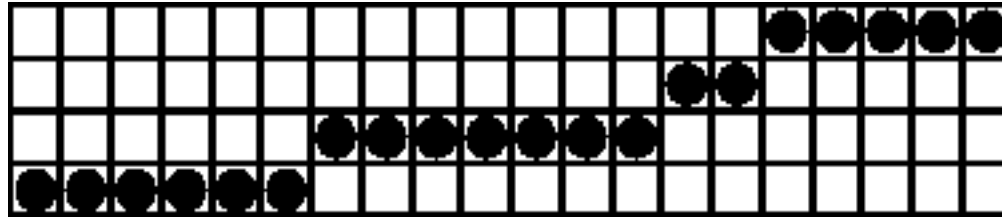


Isolated Word Recognition

For isolated word recognition:

- Compute $P(w|X)$ for all w ; report $\operatorname{argmax}_w p(X|w) \cdot P(w)$
- Search algorithms:

•
Viterbi:



Distance = \sum local distances

• Forward:

					$\alpha_T(n)$
					...
					$\alpha_T(2)$
					$\alpha_T(1)$

Distance = $-\log \sum_j \alpha_j(T)$

From Isolated to Continuous Speech

Sloppier

Higher speaking rate

Combinatorial explosion of things that can be said

Spontaneous effects: restarts, fragments, noise

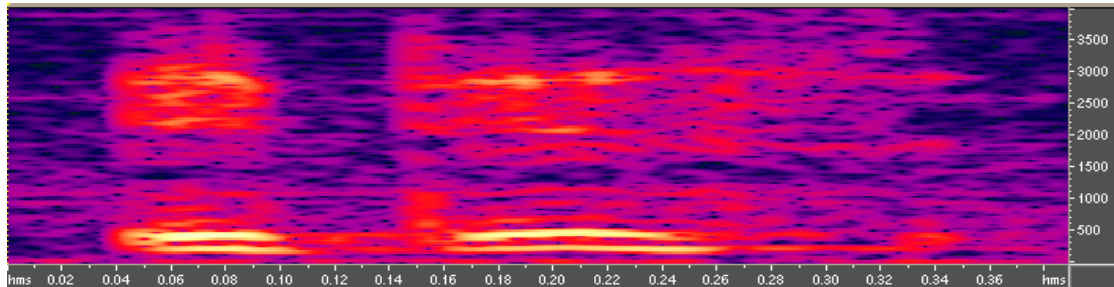
Co-articulation: Did you \Rightarrow dija ..

Segmentation: how to find word boundaries

Solution: reduce to known problems

Plan 1: Cut Continuous Speech Into Single Words

- Write magic algorithm that cuts preprocessed speech into 1-word chunks
- Run DTW/Viterbi on each chunk
- BUT: Where **are** the boundaries????

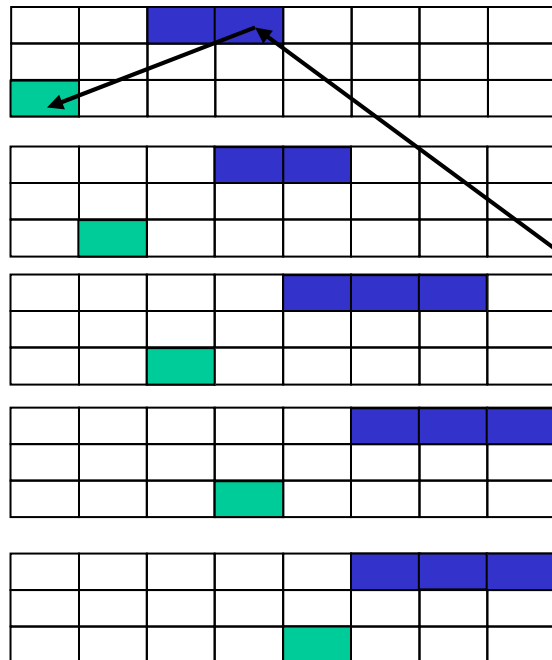


- No reliable segmentation algorithm for detecting word boundaries other than doing recognition itself, since:
 - Co-articulation between words
 - Hesitations within words
 - Hard decisions lead to accumulating errors
- ⇒ integrated approach works better

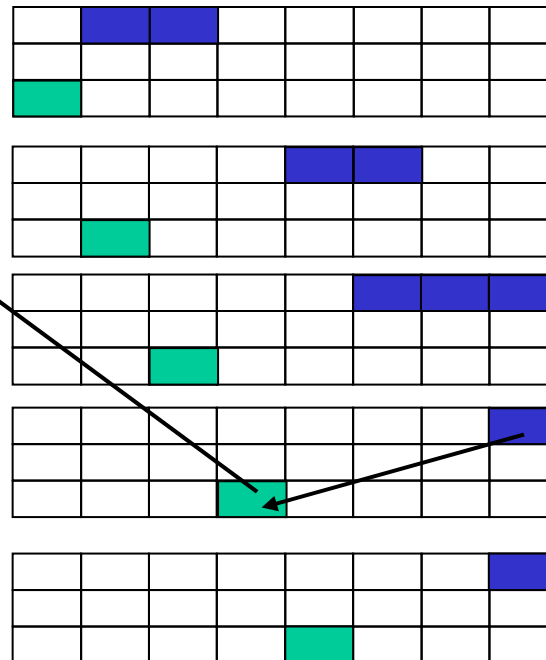
Plan 2: Two level DTW



- Level 1 DTW: For each point in time (or each remotely likely word begin time)
Compute DTW with open end within all words
- Level 2 DTW: For each word end and start point from first DTW
Combine words in second DTW based on calculated with-in scores
- Problem: Extremely costly for larger vocabularies and longer words

Pattern 1



Pattern 2



-  Assumed word begin frame
-  Active corresponding word end frames (with scores)

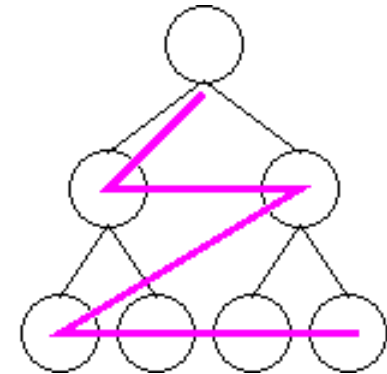
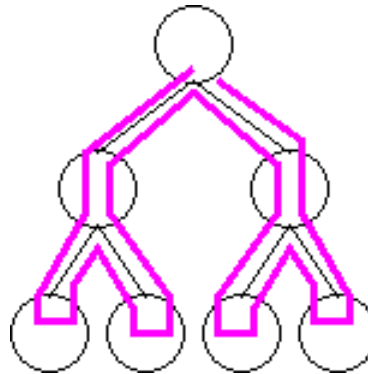
Plan 3: Depth First Search

Well known search strategies:

depth first search

vs.

breadth first search:

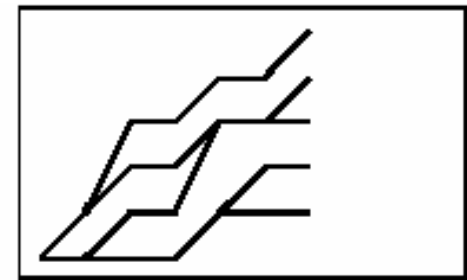
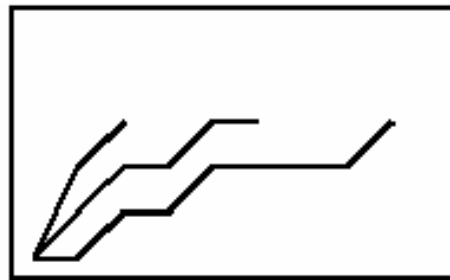


In speech recognition, this corresponds roughly to:

time-asynchronous

vs.

time-synchronous



In time-synchronous search, we expand every hypothesis simultaneously frame by frame.
In time-asynchronous search, we expand partial hypotheses that have different durations.

A* Search

A* search = example of best-first search

= even more specifically: admissible best-first search

- Expand the node first which gives best hope of leading to the best path to the goal G
- Decision about the best node is taken according to an evaluation function $h(N)$ estimating the remaining distance from the node N to the goal G
- If this heuristic function $h(N)$ is an **underestimate** of the true distance from N to G , the best-first strategy is guaranteed to find an optimal solution, if one exists (it is admissible).
- In this case the best-first algorithm is called **A* search**
- When the heuristic function is close to the true remaining distance, the search will find the optimal solution without too much effort (informed search)

A* with stack decoder

Time-asynchronous search:

Typical Implementation: **stack decoder**

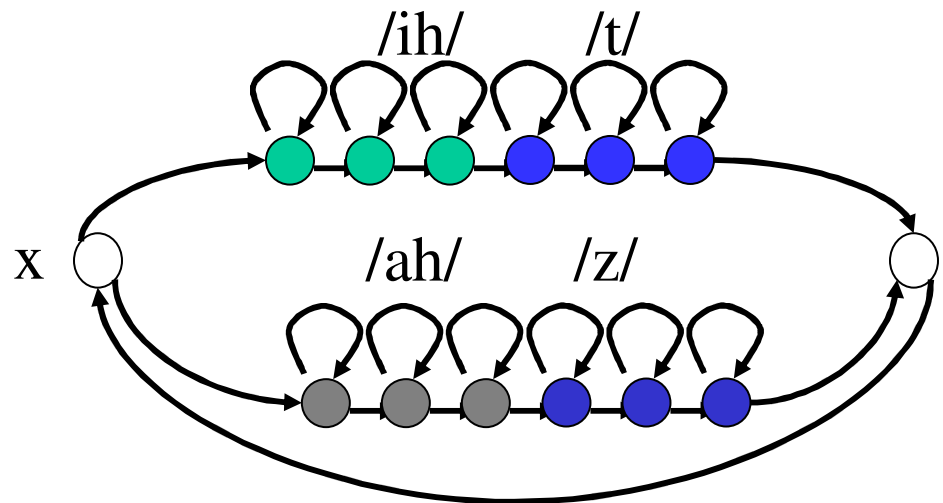
= **A***, estimation function *h* is based on forward probabilities

- Sort all partial hypotheses according to their accumulated score
- Expand the hypothesis with the highest success-expectation
- Success expectation = accumulated score + expectation of remainder score (A*)
- Expansion of a hypothesis increases its score (decreases likelihood)

=> some other hypothesis might become the best

Stack (after 5 frames):

- x ih(1) ih(2) ih(2) ih(3) t(1)
- x ih(1) ih(2) ih(2) ih(3) ih(3)
- x ih(1) ih(2) ih(2) ih(2)
- x ih(1) ih(2) ih(3) ih(3)
- x ih(1) ih(2) ih(3) t(1) t(1)
- x ih(1) ih(1)
- x ah(1)



Pros&Cons for Stack-decoder

Advantage of stack-decoder:

- If we are lucky, only very few states will be expanded
- A stack element can "easily" store the full history of a hypothesis
- finds optimal solution if one exists (admissible)

Disadvantage of stack-decoder:

- Compare scores of hypotheses over a different number of input vectors
- Use function to estimate score accumulated at end of utterance
- To estimate this score accurately is almost the same problem as recognition
- Stack decoder often used as second pass, using first pass with other algorithm for this estimation (fast-match, detailed-match)

Viterbi Beam Search vs A* Stack Decoder

Advantage of beam search:

- No heuristic required to find the next best node
- Attractive when including different knowledge sources in a time synchronous fashion
- Easy to compare scores since the paths have same length
- Appropriate for parallel implementation

Disadvantage of beam search:

- Not admissible

Viterbi vs A*

- 80's quite controversial discussion whether Viterbi or A* better
- Nowadays Viterbi beam search (with the help of efficient pruning techniques) is the preferred method for all SR tasks, A* is taken for search through n-best and lattice structures

Plan 4: One stage Dynamic Programming

Within words:

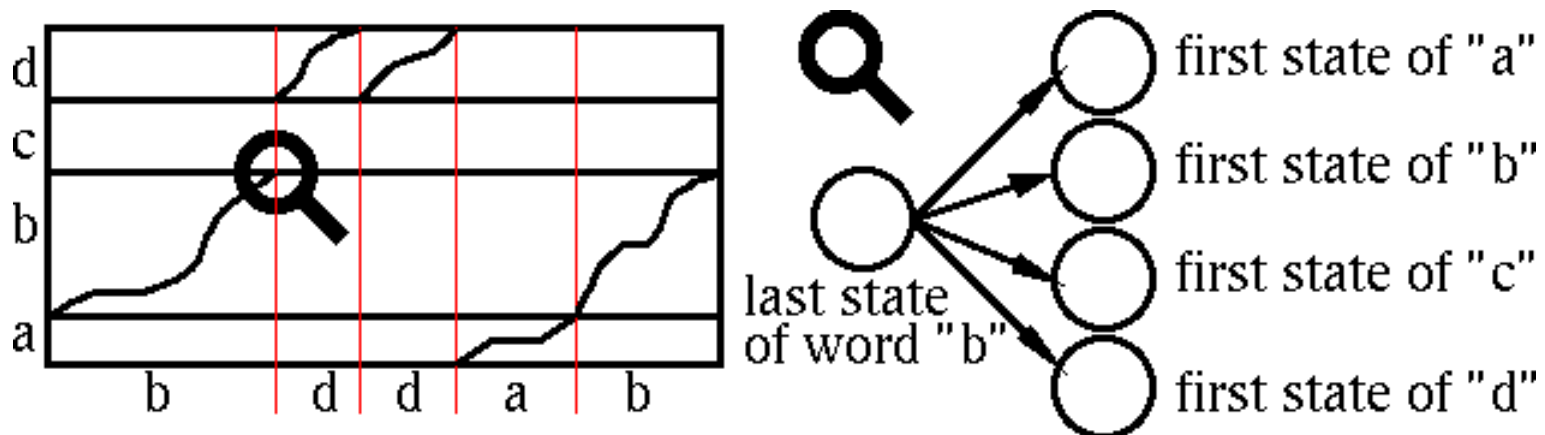
- Viterbi

Between words:

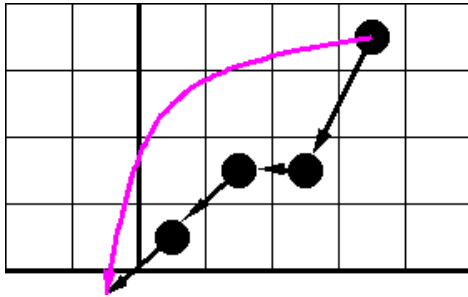
- For the first state of each words, consider the best last state of all words as predecessor.
- This last state or the first state in the current word is predecessor.

first state means any state that can be the first of a word

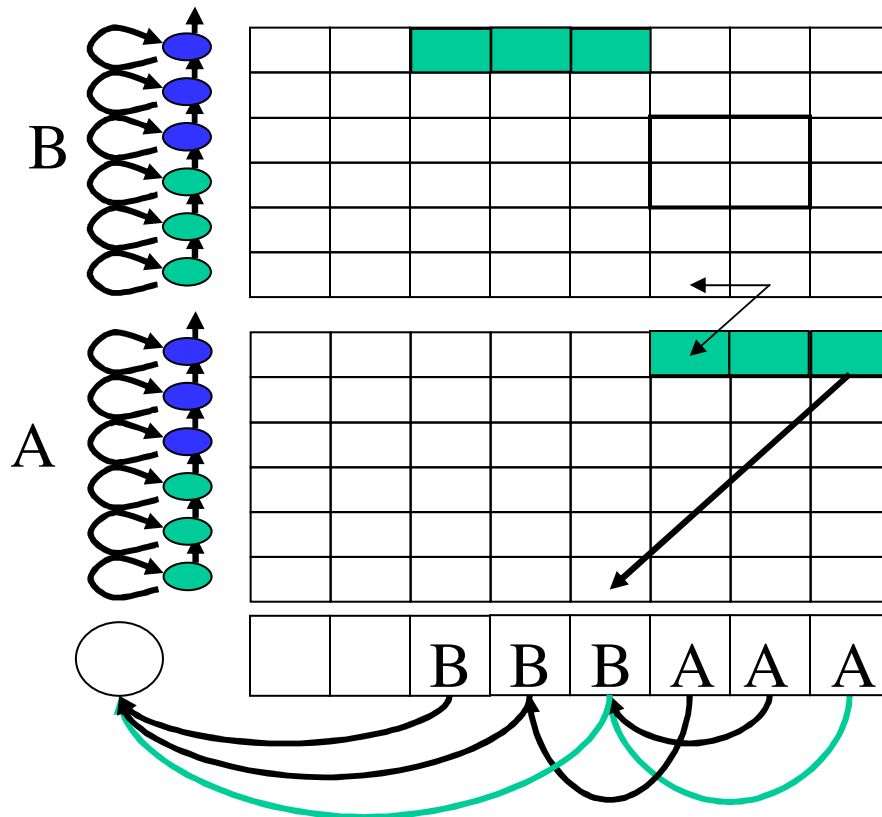
last state means any state that has a transition out of the word



One stage: Find the Backtrace



- In order to reconstruct the best word sequence: find the best predecessor word to the current word end, I.e **Store backpointers for each word end**
- Which was the best predecessor word
- In which frame was the transition from this predecessor made into the current word



- In every matrix cell, remember "best" previous word
- If in-word path, it is not needed, forget in-word backpointers
- Find hypothesis by following the word-to-word backpointers (backtrace)

■ Best word-end

Backtrace:

word sequence = B A