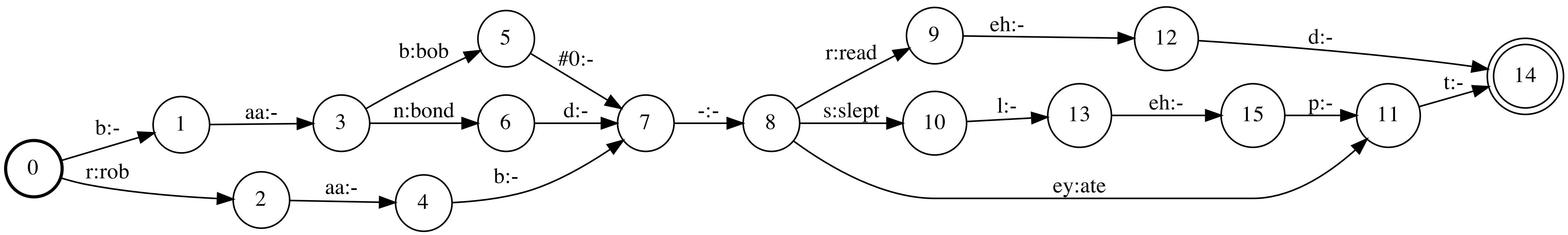
Automatic Speech Recognition (CS753)

Lecture 19: Search, Decoding and Lattices

Instructor: Preethi Jyothi
Mar 27, 2017

Recap: Static and Dynamic Networks

- Static network: Build compact decoding graph using WFST optimisation techniques.



- Dynamic networks:
  - Dynamically build the graph with active states on the fly
  - On-the-fly composition with the language model acceptor G

Static Network Decoding

- Expand the whole network prior to decoding.
- The individual transducers H, C, L and G are combined using composition to build a static decoding graph.
- The graph is further optimised by weighted determinization and minimisation.
  - $D = \pi_\varepsilon(\min(\det(\tilde{H} \circ \det(\tilde{C} \circ \det(\tilde{L} \circ G)))))$
  - The final optimised network is typically 3-5 times larger than the language model G
- Becomes impractical for very large vocabularies

Searching the graph

- Two main decoding algorithms adopted in ASR systems:

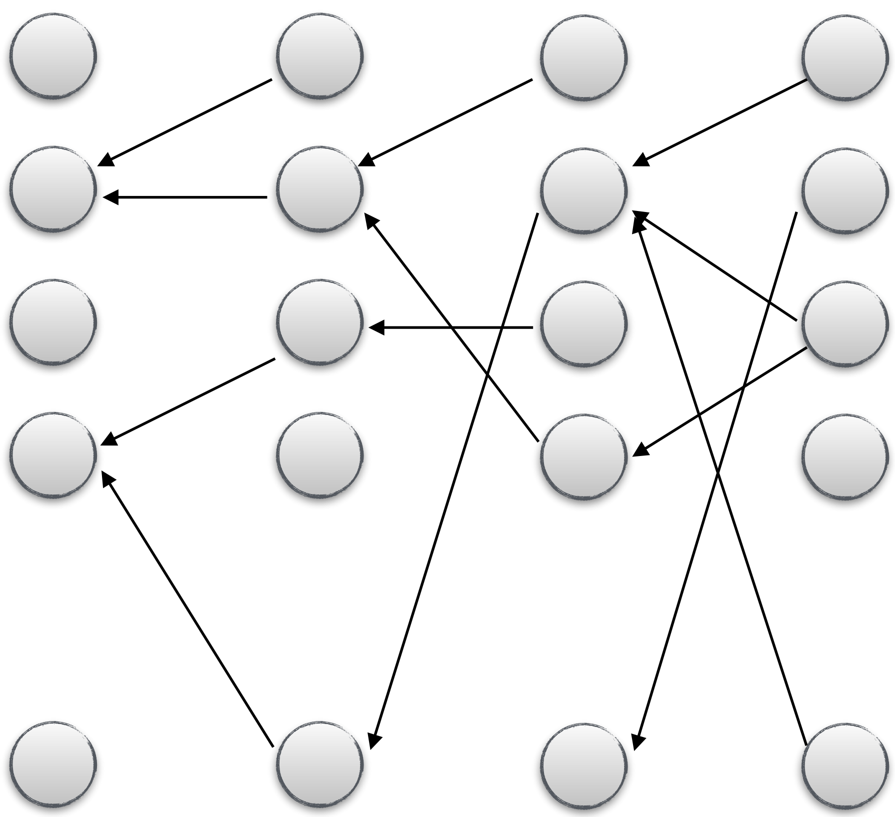
    1. Viterbi beam search decoder

    2. A* stack decoder

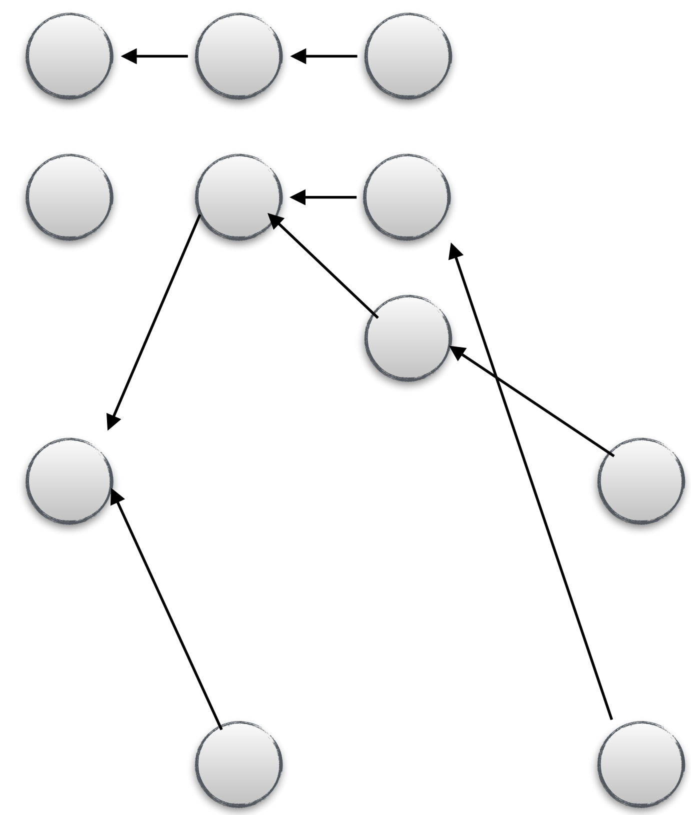
Viterbi beam search decoder

- Time-synchronous search algorithm:
  - For time t, each state is updated by the best score from all states in time t-1
- Beam search prunes unpromising states at every time step.
- At each time-step t, only retain those nodes in the time-state trellis that are within a fixed threshold δ (beam width) of the score of the best hypothesis.

Trellis with full Viterbi & beam search



No beam search



With beam search

Beam search algorithm

Initialization: c*urrent states* := *initial state*

while (*current states* do not contain the *goal state*) do:

> *successor states* := NEXT(*current states*)
> where NEXT is next state function
>
> score the *successor states*
>
> set *current states* to a pruned set of *successor states* using beam width δ
> only retain those *successor states* that are within δ times the best path weight

A* stack decoder

- So far, we considered a time-synchronous search algorithm that moves through the observation sequence step-by-step

- A* stack decoding is a time-asynchronous algorithm that proceeds by extending one or more hypotheses word by word (i.e. no constraint on hypotheses ending at the same time)

- Running hypotheses are handled using a *stack* which is a priority queue sorted on scores. Two problems to be addressed:

    1. Which hypotheses should be extended? (Use A*)

    2. How to choose the next word used in the extensions? (fast-match)

Recall A* algorithm

- To find the best path from a node to a goal node within a weighted graph,
  - A* maintains a tree of paths until one of them terminates in a goal node
  - A* expands a path that minimises `f(n) = g(n) + h(n)` where `n` is the final node on the path, `g(n)` is the cost from the start node to `n` and `h(n)` is a heuristic determining the cost from `n` to the goal node
  - `h(n)` must be *admissible* i.e. it shouldn't overestimate the true cost to the nearest goal node

A* stack decoder

- So far, we considered a time-synchronous search algorithm that moves through the observation sequence step-by-step

- A* stack decoding is a time-asynchronous algorithm that proceeds by extending one or more hypotheses word by word (i.e. no constraint on hypotheses ending at the same time)

- Running hypotheses are handled using a *stack* which is a priority queue sorted on scores. Two problems to be addressed:

    1. Which hypotheses should be extended? (Use A*)

    2. How to choose the next word used in the extensions? (fast-match)

Which hypotheses should be extended?

- A* maintains a priority queue of partial paths and chooses the one with the highest score to be extended

- Score should be related to probability: For a word sequence W given an acoustic sequence O, score $\propto \Pr(O|W)\Pr(W)$

- But not exactly this score because this will be biased towards shorter paths

- A* evaluation function based on f(p) = g(p) + h(p) for a partial path p where
  g(p) = score from the beginning of the utterance to the end of p
  h(p) = estimate of best scoring extension from p to end of the utterance

- An example of h(p): Compute some average probability *prob* per frame (over a training corpus). Then h(p) = *prob* × (T-t) where t is the end time of the hypothesis and T is the length of the utterance

A* stack decoder

- So far, we considered a time-synchronous search algorithm that moves through the observation sequence step-by-step

- A* stack decoding is a time-asynchronous algorithm that proceeds by extending one or more hypotheses word by word (i.e. no constraint on hypotheses ending at the same time)

- Running hypotheses are handled using a *stack* which is a priority queue sorted on scores. Two problems to be addressed:

  1. Which hypotheses should be extended? (Use A*)

  2. How to choose the next word used in the extensions? (fast-match)

Fast-match

- Fast-match: Algorithm to quickly find words in the lexicon that are a good match to a portion of the acoustic input

- Acoustics are split into a front part, A, (accounted by the word string so far, W) and the remaining part A'. Fast-match is to find a small subset of words that best match the beginning of A'.

    - Many techniques exist: 1) Rapidly find Pr(A'|w) for all w in the vocabulary and choose words that exceed a threshold
    2) Vocabulary is pre-clustered into subsets of acoustically similar words. Each cluster is associated with a centroid. Match A' against the centroids and choose subsets having centroids whose match exceeds a threshold

[B et al.]: Bahl et al., Fast match for continuous speech recognition using allophonic models, 1992

A* stack decoder

function STACK-DECODING() **returns** *min-distance*

Initialize the priority queue with a null sentence.
Pop the best (highest score) sentence s off the queue.
If (s is marked end-of-sentence (EOS)) output s and terminate.
Get list of candidate next words by doing fast matches.
For each candidate next word w:
    Create a new candidate sentence $s+w$.
    Use forward algorithm to compute acoustic likelihood L of $s+w$
    Compute language model probability P of extended sentence $s+w$
    Compute "score" for $s+w$ (a function of L, P, and ???)
    if (end-of-sentence) set EOS flag for $s+w$.
    Insert $s+w$ into the queue together with its score and EOS flag

Image from [JM]: Jurafsky & Martin, SLP 2nd edition, Chapter 10

Example (1)

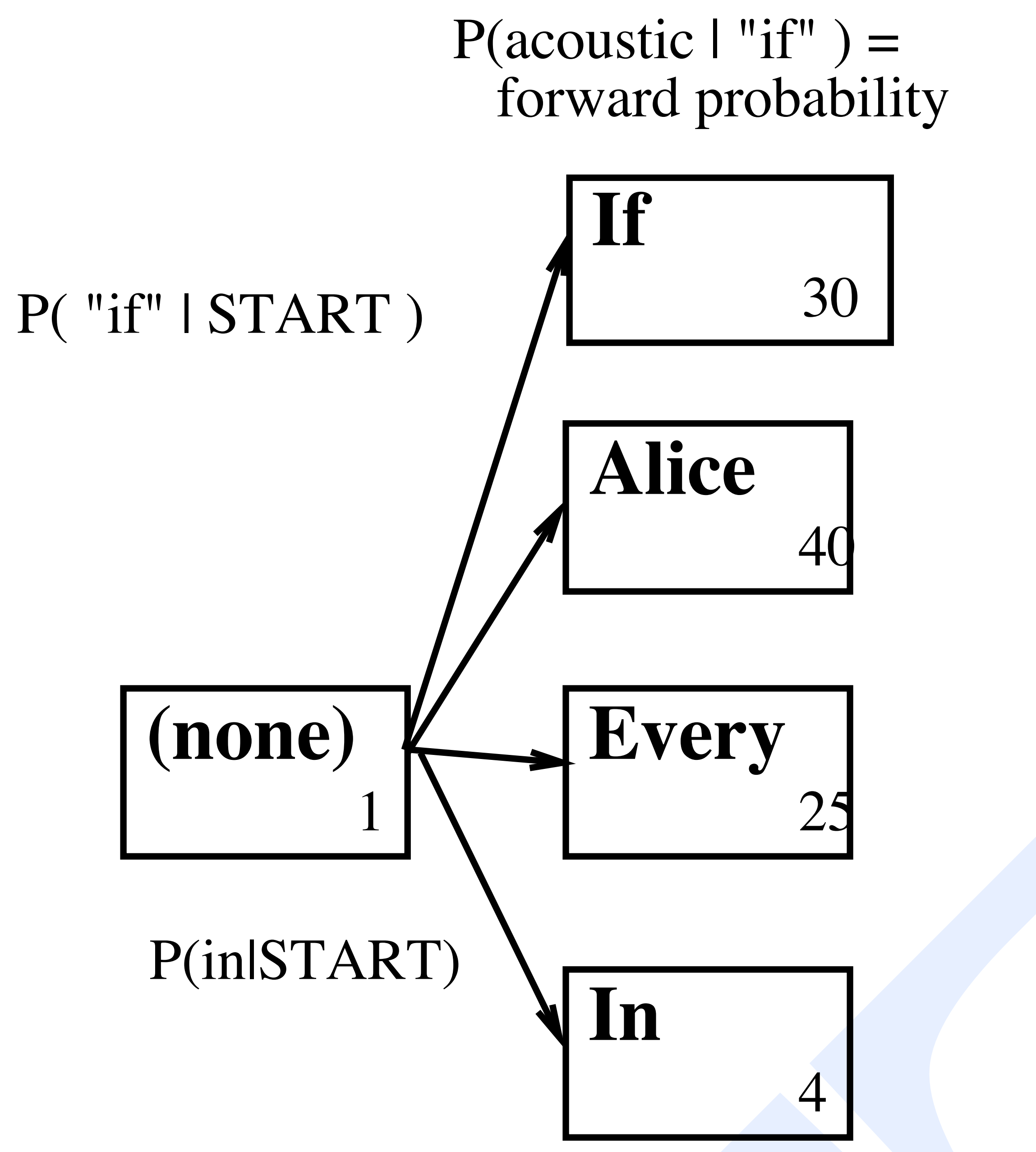


Image from [JM]: Jurafsky & Martin, SLP 2nd edition, Chapter 10

Example (2)

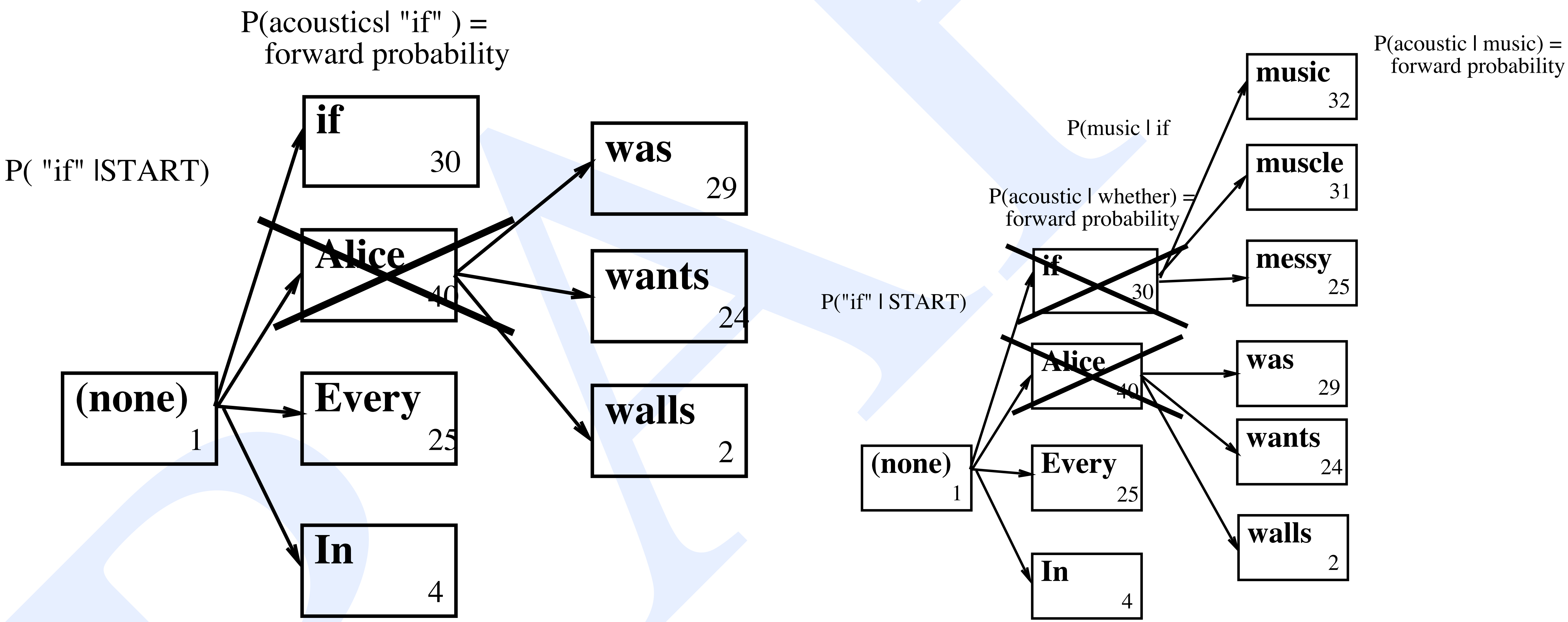


Image from [JM]: Jurafsky & Martin, SLP 2nd edition, Chapter 10

A* vs Beam search

- Nowadays Viterbi beam search is the more popular paradigm for ASR tasks
- A* is used to search through lattices
- How are lattices generated?

Lattice Generation

- Say we want to decode an utterance, U, of T frames.
- Construct a sausage acceptor for this utterance, X, with T+1 states and arcs for each context-dependent HMM state at each time-step
- Search the following composed machine for the best word sequence corresponding to U:

$$D = X \circ HCLG$$

Lattice Generation

- For all practical applications, we have to use beam pruning over D such that only a subset of states/arcs in D are visited. Call this resulting pruned machine, B.

- Word lattice, say L, is a further pruned version of B defined by a lattice beam, β. L satisfies the following requirements:

    - L should have a path for every word sequence within β of the best-scoring path in B

    - All scores and alignments in L correspond to actual paths through B

    - L does not contain duplicate paths with the same word sequence

Word Confusion Networks

Word confusion networks are normalised word lattices that provide alignments for a fraction of word sequences in the word lattice

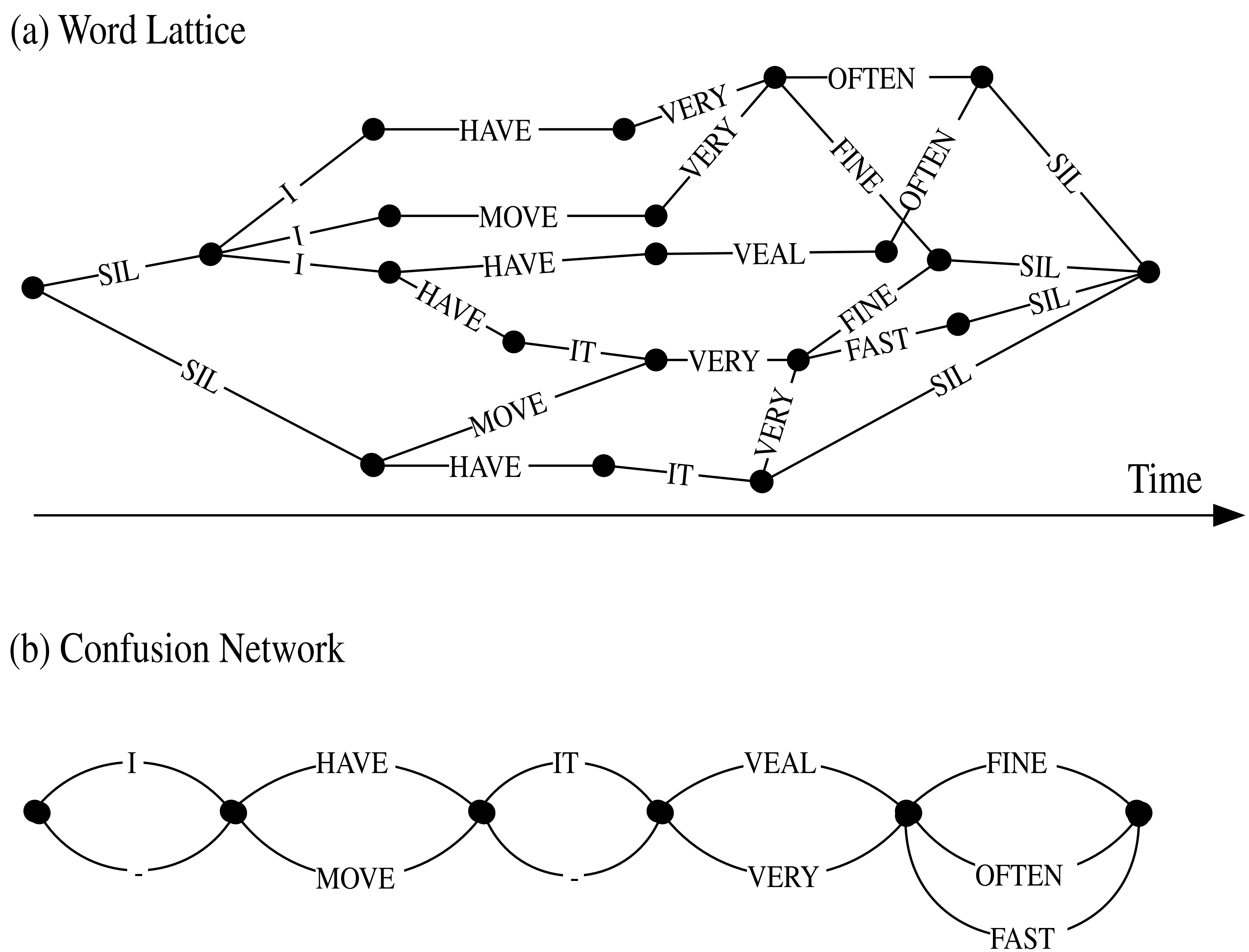


Image from [GY08]: Gales & Young, Application of HMMs in speech recognition, NOW book, 2008

Constructing word confusion network

- Links of a confusion network are grouped into confusion sets and every path contains exactly one link from each set

- This clustering is done in two stages:

    1. Links that correspond to the same word and overlap in time are combined

    2. Links corresponding to different words are clustered into confusion sets. Clustering algorithm is based on phonetic similarity, time overlap and word posteriors. More details in [LBS00]

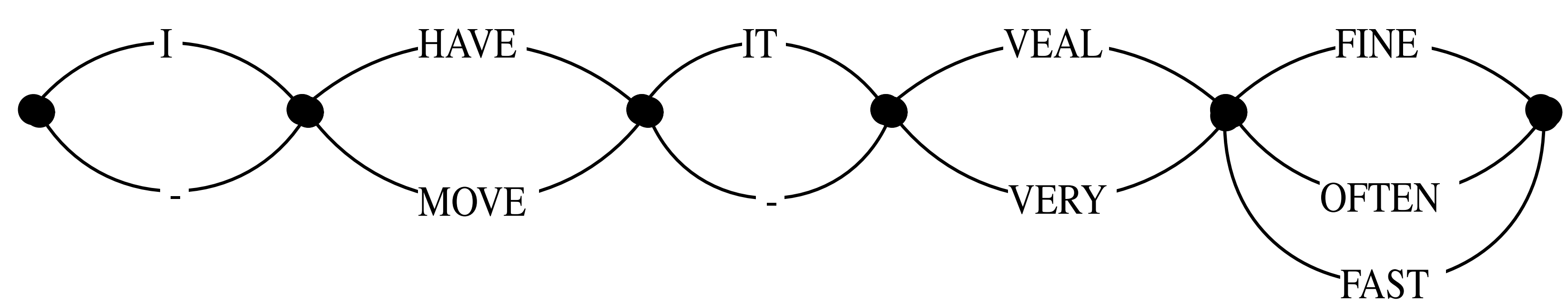


Image from [LBS00]: L. Mangu et al., “Finding consensus in speech recognition”, Computer Speech & Lang, 2000