Recap: Construct Static Network

• 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. \((D\) can get very large for LVCSR tasks)
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 $\delta$ (beam width) of the score of the best hypothesis.
Beam search algorithm

Initialization: current 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 $\delta$
only retain those successor states that are within $\delta$ times the best path weight
Trellis with full Viterbi & beam search

No beam search

With beam search
Beam search over the decoding graph

Say $\delta = 2$

Score of arc:

- $-\log P(O_1|x_1)$
+ graph cost

$x_1$: the
$x_{200}$: the
$x_2$: a

$O_1$, $O_2$, $O_3$, $O_T$
Searching the graph

- Two main decoding algorithms adopted in ASR systems:
  1. Viterbi beam search decoder
  2. A* stack decoder
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 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

Nice animations: http://www.redblobgames.com/pathfinding/a-star/introduction.html
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 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 $g(p) = \text{score from the beginning of the utterance to the end of } p$ $h(p) = \text{estimate of best scoring extension from } p \text{ to end of the utterance}$

- An example of $h(p)$: Compute some average probability $\text{prob}$ per frame (over a training corpus). Then $h(p) = \text{prob} \times (T-t)$ 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.
Example (1)

P(acoustic | "if") = forward probability

\[
P("if" | START) = 30
\]

\[
P("if" | START) = 40
\]

\[
P("if" | START) = 25
\]

\[
P("if" | START) = 4
\]
Example (2)

Figure 10.8 The beginning of the search for the sentence "If music be the food of love". At this early stage, Alice is the most likely hypothesis. (It has a higher score than the other hypotheses.)

Figure 10.9 The next steps of the search for the sentence "If music be the food of love". In (a) we've now expanded the "if" node and added three extensions which have a relatively high score; the highest-scoring node is "START if", which is not on the START Alice path at all. In (b) we've expanded the "if" node. The hypothesis "START if music" then has the highest score.

Instead, we use the $A^*$ evaluation function (Nilsson, 1980; Pearl, 1984)

$$f^*(p) = g(p) + h^*(p)$$

where $f^*(p)$ is the estimated score of the best complete path (complete sentence) which starts with the partial path $p$. In other words, it is an estimate of how well this path would do if we let it continue through the sentence. The $A^*$ algorithm builds this.
Moving on to multi-pass decoding

- We learned about two algorithms (beam search & A*) via which one can search through the decoding graph in a first-pass decoding pass

- However, some models are too expensive to implement in first-pass decoding (e.g. RNN-based LMs)

- Multi-pass decoding:
  - First, use simpler model (e.g. Ngram LMs) to find most probable word sequences
  - Rescore first-pass hypotheses using complex model to find the best word sequence
Multi-pass decoding with N-best lists

- Simple algorithm: Modify the Viterbi algorithm to return the N-best word sequences for a given speech input

---

Image from [JM]: Jurafsky & Martin, SLP 2nd edition, Chapter 10
Multi-pass decoding with N-best lists

• Simple algorithm: Modify the Viterbi algorithm to return the N-best word sequences for a given speech input

<table>
<thead>
<tr>
<th>Rank</th>
<th>Path</th>
<th>AM logprob</th>
<th>LM logprob</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>it’s an area that’s naturally sort of mysterious</td>
<td>-7193.53</td>
<td>-20.25</td>
</tr>
<tr>
<td>2.</td>
<td>that’s an area that’s naturally sort of mysterious</td>
<td>-7192.28</td>
<td>-21.11</td>
</tr>
<tr>
<td>3.</td>
<td>it’s an area that’s not really sort of mysterious</td>
<td>-7221.68</td>
<td>-18.91</td>
</tr>
<tr>
<td>4.</td>
<td>that scenario that’s naturally sort of mysterious</td>
<td>-7189.19</td>
<td>-22.08</td>
</tr>
<tr>
<td>5.</td>
<td>there’s an area that’s naturally sort of mysterious</td>
<td>-7198.35</td>
<td>-21.34</td>
</tr>
<tr>
<td>6.</td>
<td>that’s an area that’s not really sort of mysterious</td>
<td>-7220.44</td>
<td>-19.77</td>
</tr>
<tr>
<td>7.</td>
<td>the scenario that’s naturally sort of mysterious</td>
<td>-7205.42</td>
<td>-21.50</td>
</tr>
<tr>
<td>8.</td>
<td>so it’s an area that’s naturally sort of mysterious</td>
<td>-7195.92</td>
<td>-21.71</td>
</tr>
<tr>
<td>9.</td>
<td>that scenario that’s not really sort of mysterious</td>
<td>-7217.34</td>
<td>-20.70</td>
</tr>
<tr>
<td>10.</td>
<td>there’s an area that’s not really sort of mysterious</td>
<td>-7226.51</td>
<td>-20.01</td>
</tr>
</tbody>
</table>

• Problem: N-best lists aren’t as diverse as we’d like. And, not enough information in N-best lists to effectively use other knowledge sources
Multi-pass decoding with lattices

- ASR lattice: Weighted automata/directed graph representing alternate word hypotheses from an ASR system

- Lattice is a (heavily) pruned reduction of the decoding graph
Multi-pass decoding with confusion networks

- **Confusion networks/sausages**: Lattices that show competing/confusible words and can be used to compute posterior probabilities at the word level.
Word Confusion Networks

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

(a) Word Lattice

(b) Confusion Network

Image from [GY08]: Gales & Young, Application of HMMs in speech recognition, NOW book, 2008
Word posterior probabilities in the word confusion network

- Each arc in the confusion network is marked with the posterior probability of the corresponding word $w$
- First, find the link probability of $w$ from the word lattice:
  - Joint probability of a path $a$ (corr. to word sequence $w$) and acoustic observations $O$: $\Pr(a, O) = \Pr_{AM}(O|a)\Pr_{LM}(w)$
  - For each link $l$, the joint probabilities of all paths through $l$ are summed to find the link probability:
    $$\Pr(l|O) = \frac{\sum_{a \in A} \Pr(a, O)}{\Pr(O)}$$
Constructing word confusion network

- Second step in estimating word posteriors is the clustering of links that correspond to the same word/confusion 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].

![Diagram of word confusion network](image_url)
Another use for confusion networks: 

*System Combination*
System Combination

- Combining recognition outputs from multiple systems to produce a hypothesis that is more accurate than any of the original systems

- Most widely used technique: ROVER [ROVER].

- 1-best word sequences from each system are aligned using a greedy dynamic programming algorithm

- Voting-based decision made for words aligned together

- Can we do better than just looking at 1-best sequences?

Image from [ROVER]: Fiscus, Post-processing method to yield reduced word error rates, 1997
System Combination

- Combining recognition outputs from multiple systems to produce a hypothesis that is more accurate than any of the original systems

- Most widely used technique: ROVER [ROVER].
  
  - 1-best word sequences from each system are aligned using a greedy dynamic programming algorithm
  
  - Voting-based decision made for words aligned together

- Could align confusion networks instead of 1-best sequences

Table: Examples of output from various systems

<table>
<thead>
<tr>
<th>System</th>
<th>Example Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>bbn1.ctm</td>
<td>there's a lot of like societies is for engineers and lakes think and like</td>
</tr>
<tr>
<td>cmu-isl1.ctm</td>
<td>there's the labs @ like societies @ for women of engineers</td>
</tr>
<tr>
<td>cu-htk2.ctm</td>
<td>there's the last @ like societies @ for true of women through engineers</td>
</tr>
<tr>
<td>dragon1.ctm</td>
<td>was a lot of @ like society's @ is @ for women through engineers</td>
</tr>
<tr>
<td>sril.ctm</td>
<td>there's a lot of @ like societies @ for women through engineers</td>
</tr>
</tbody>
</table>