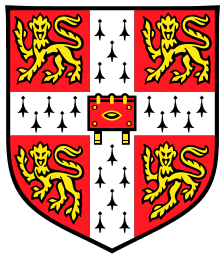
Engineering Part IIB: Module 4F11
Speech and Language Processing
Lecture 8 : Continuous Speech Recognition Search

Phil Woodland: pcw@eng.cam.ac.uk

Lent 2016



Cambridge University Engineering Department

Statistical Speech Recognition

The aim of speech recognition is to find the word string, **W**, such that it maximises $P(\mathbf{W}|\mathbf{O})$ where $\mathbf{O}$ is the observed acoustic data.

$$P(\mathbf{W}|\mathbf{O}) = \frac{p(\mathbf{O}|\mathbf{W})P(\mathbf{W})}{p(\mathbf{O})}$$

$p(\mathbf{O}\|\mathbf{W})$	obtained from the acoustic model
$P(\mathbf{W})$	obtained from the language model (LM)
$p(\mathbf{O})$	normalising factor

$p(\mathbf{O})$ is independent of the word sequence, so does not affect the choice of most probable word sequence.

The search (**decoder**) component needs to find the word sequence that maximises this value.

We will initially look at how to deal with continuous speech recognition with whole word models and then examine the issues in using (context-dependent) phone models and N-gram language models.

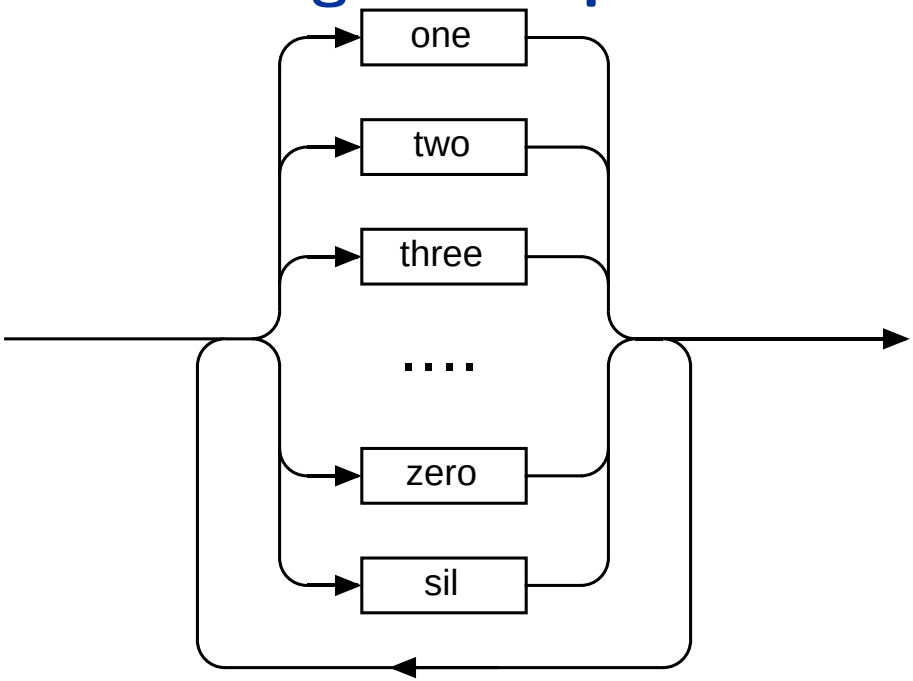
How Does CSR Work?

Continuous speech recognition (CSR) is difficult because the locations of the word boundaries are unknown: it is necessary to allow for the possibility of word boundaries at every frame in the utterance. Furthermore the spoken style of continuous speech increases the variability in the data.

- **Isolated** word recognition:
  - A *model of each word* was built.
  - The most likely word recognised.
- **Continuous** speech recognition:
  - Use a *sentence model* formed by joining HMMs together to form a network. The paths through the network define the set of legal word sequences. Each path is in effect a separate sentence model.
  - The most likely valid word sequence through the network is recognised

Note that if we have a network of a suitable structure we just need to use the **Viterbi** algorithm to find the best path/state sequence.

Digit Example

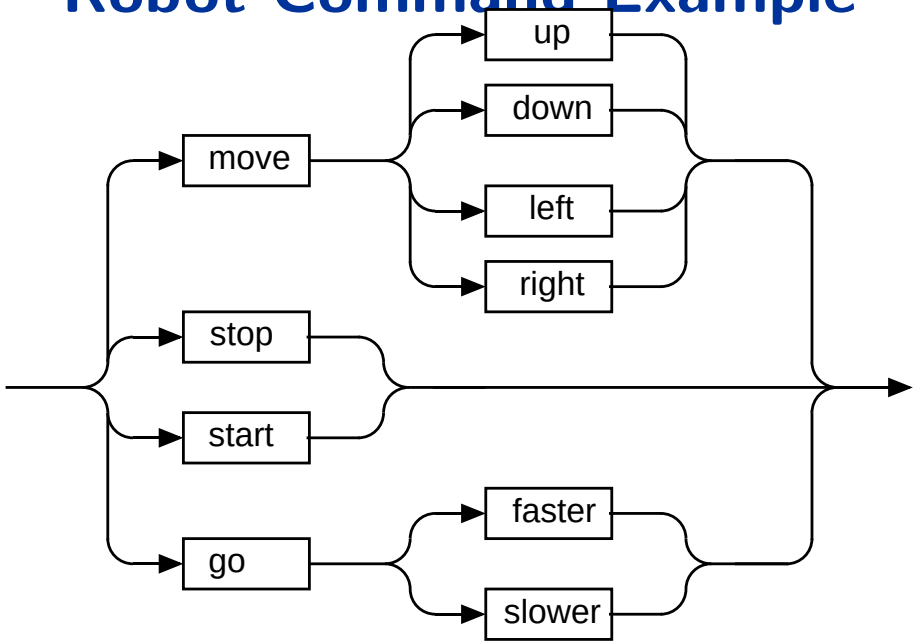


The diagram shows a network for connected digit recognition:

- Vocabulary of 10 digits (plus silence)
- Average branching factor of 10 (silence is a not counted)

Note that this allows an arbitrarily long digit sequence to be recognised.

Robot Command Example



The diagram shows a network defining some simple robot commands.

- Vocabulary of 10 (plus silence)
- Average branching factor of around 3

The robot command task is simpler in terms of average branching factor. This does not guarantee improved recognition performance!

Continuous Speech Recognition & Paths

In CSR the recognition network is treated as a single large HMM (a *composite model*) and the most likely state sequence is determined using the Viterbi algorithm. This state sequence then defines the recognised word sequence. For large vocabulary tasks, the *language model* is incorporated into the between-word transitions.

Recall the basic Viterbi equation:

$$\log \phi_j(t) = \max_{1 \leq k < N} \left[\log \left(\phi_k(t-1) \right) + \log \left(a_{kj} \right) \right] + \log \left(b_j(\mathbf{o}_t) \right)$$

A (partial) **path** represents an alignment of states with the frames of speech from the start of the utterance up to time t. Thus a path is represented by

- a **log likelihood** (here called `logP`)
- **a history** : to record the preceding sequence of states/words

Word-Level History

Note that for CSR we don't (normally) need to store the history at the state level. View the operation of the Viterbi algorithm in two steps at each time frame:

1. (word) HMM internal path extension
2. between-word (external) path extension (between word HMM exit and entry nodes, including LM probabilities)

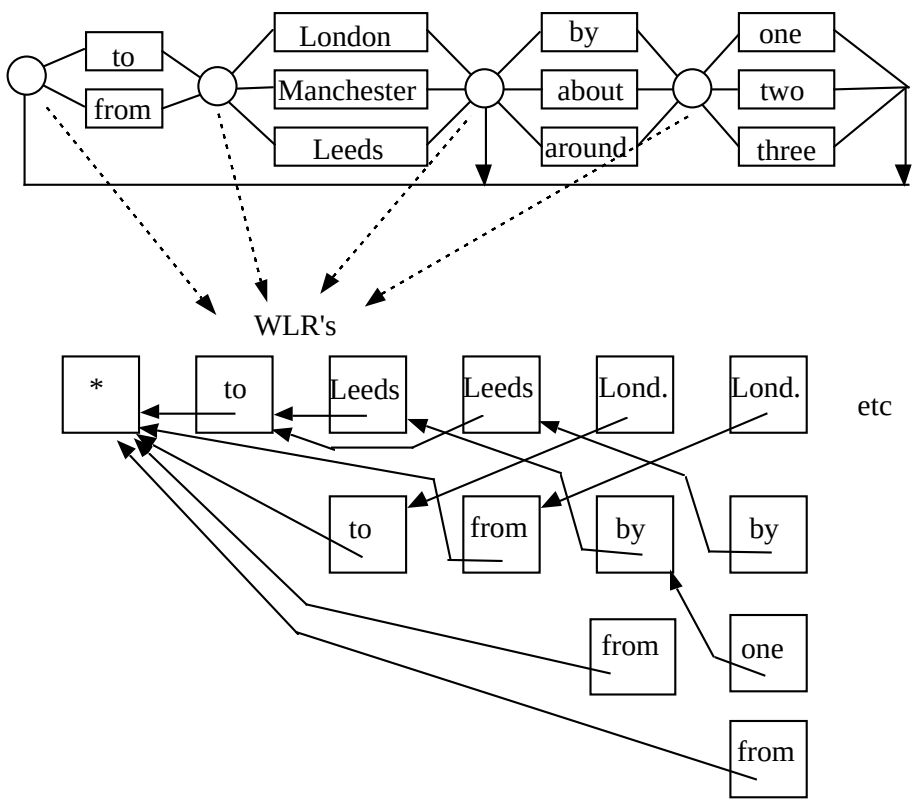
Only need to store information about the path extensions when the path is extended between words. History information is stored in **Word Link Records** (WLRs) for each potential word boundary.

Typically each WLR contains

- the `logP` score up to time generated;
- path id (pointer to a previous WLR);
- the identity of the word just left;
- the time generated (if the location of word boundaries is needed).

WLR Generation

- A new WLR is created for every syntactically distinct word boundary at every time frame (apart from the first few frames).
- Hence, for a simple word loop, just one WLR is recorded per frame.
- In the example shown four syntactically distinct boundaries lead to four WLRs being generated per frame.



Traceback

Once the end of the utterance is reached the path that has been extended to the network exit state (or the most likely path in an exit state) will contain the log likelihood (along the most likely state sequence) for the whole network and the WLRs generated along the way.

The final step is then to **trace back** through the WLRs to find the most likely sequence, starting from the WLR pointed to by the path extended to an exit state at time T. This yields

1. best word sequence (in reverse order)

2. word boundary locations (if time of WLR generation recorded)

3. logP for each individual word (from the difference in path scores between word ends).

Computational Cost

The Viterbi algorithm is guaranteed to find the best matching word sequence. In order to provide this guarantee, all paths must be explored no matter how unlikely they are and computationally this is very expensive. For example, suppose

- there are W words in a loop, each word has N emitting states
- each state has M mixture components (assume diagonal covariance matrices)
- the feature vector has V elements.

Then every time frame compute:

- $W \times N$ word internal path extensions - dominated by output probability computation - requires $M \times V \times 2$ multiplies per state;
- W word external path extensions.

Total cost word internal path extension is approx $W \times N \times M \times V \times 2$ multiplies. Even for a small 500 word system with $N = 10$, $M = 5$, $V = 39$, this is 4.7M multiplies per 10msec time frame.

Beam Search

A substantial reduction in computation can be achieved by only considering paths which are close in score to the best path. This is called **Beam Search**.

Every HMM state instance is either *active* or *inactive*. Initially, all network entry states are active and all other states are inactive. The beam search algorithm then works as follows

1. For each active state
   (a) Extend word internal paths as normal
   (b) Record max `logP` in any state, call this `gMax`

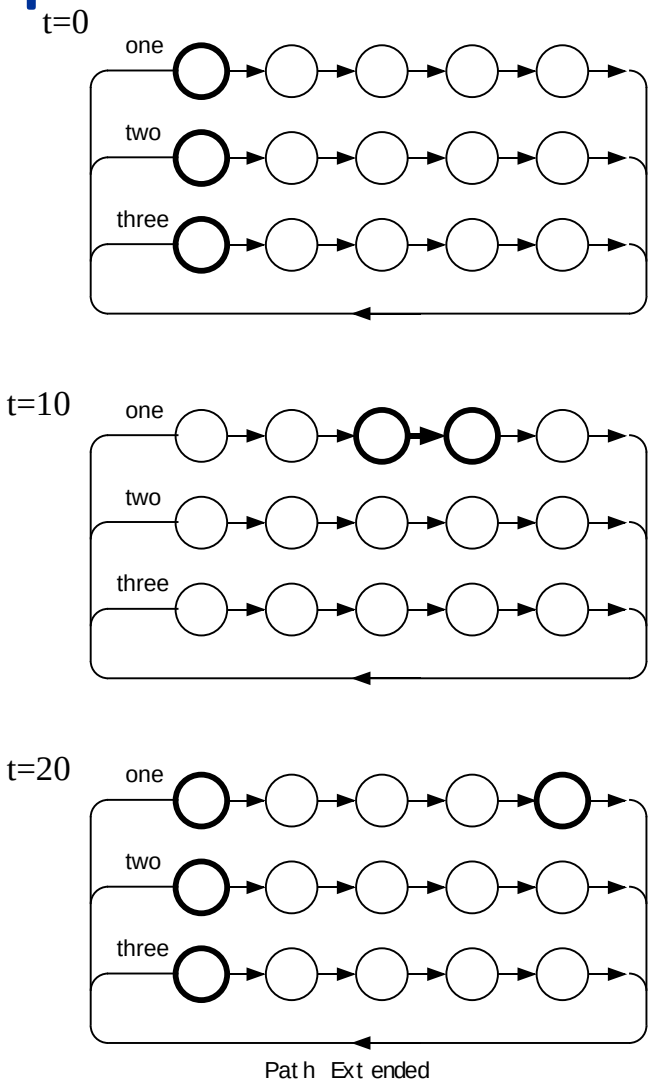
2. De-activate all states for which `logP < gMax - B` where `B` is the beam width

3. Extend paths via word external links only if they are within the beam i.e. `logP < gMax - B`

4. Re-activate all states which have a path extended to an HMM entry state.

Beam Search Example

- Diagram shows how pruning operates to reduce the number of active states and hence the total computation required.
- For large networks the use of pruning is vital to allow operation in reasonable time
- Note that similar ideas can also be applied in HMM training in forward pass (note network differences).



Simple Phone Based System

To illustrate some of the search issues of using N-gram language models with sub-word (phone) acoustic models, a simple 4 word vocabulary system will be used.

Lexicon (monophones):

AND	/ **ae n d** /
BILL	/ **b ih l** /
BIT	/ **b ih t** /
BEN	/ **b eh n** /

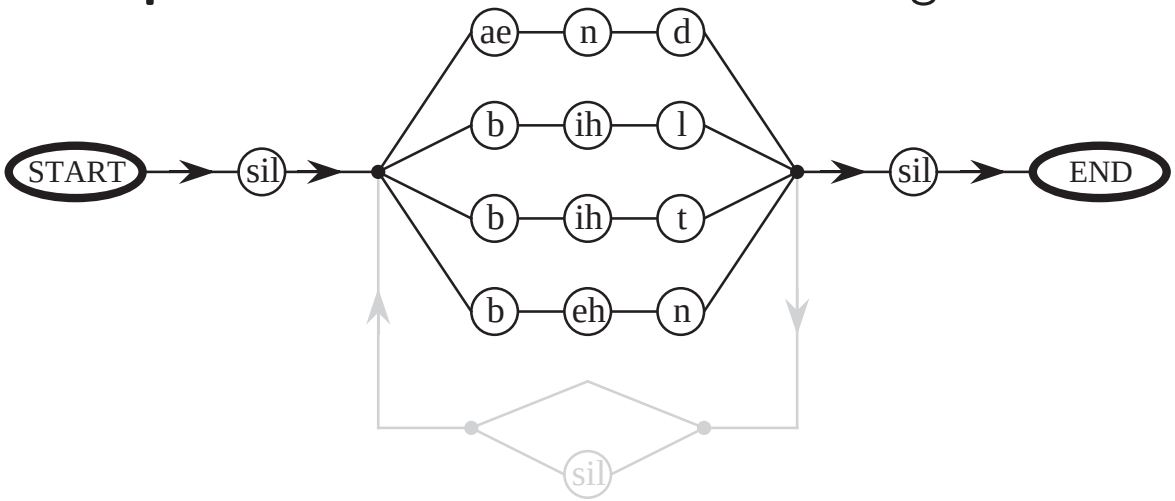
Using this example consider the problems involved in:

- Finite-state network (and unigram language model)
- Bigram language model
- Trigram language model
- Cross-word triphone context dependent models

Initial Network Structure

Initially consider a system with **Monophone** HMMs + Finite state grammar

This is shown in the figure (link in grey is extension to continuous speech)



To extend this to use a unigram language model:

- Add the language model probability (e.g. $\log(P_{BEN})$) to the log-likelihood of the path at the start of each word.
- Language model information incorporated as soon as possible to aid pruning (beam search).

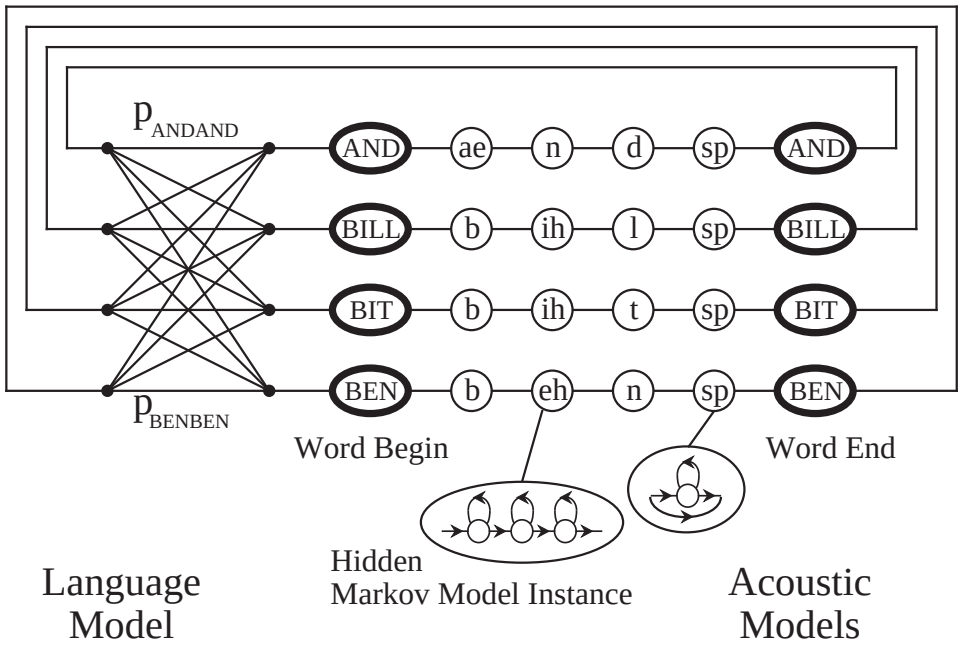
Bigram Language Model

Expanding to use bigrams is more complex:

- Language model probability is dependent on the previous word

Cannot have a single return silence loop.

- Apply bigram probability to start of each word
- Add separate loop back for each word
- Add optional silence model (*sp*) to end of each word.



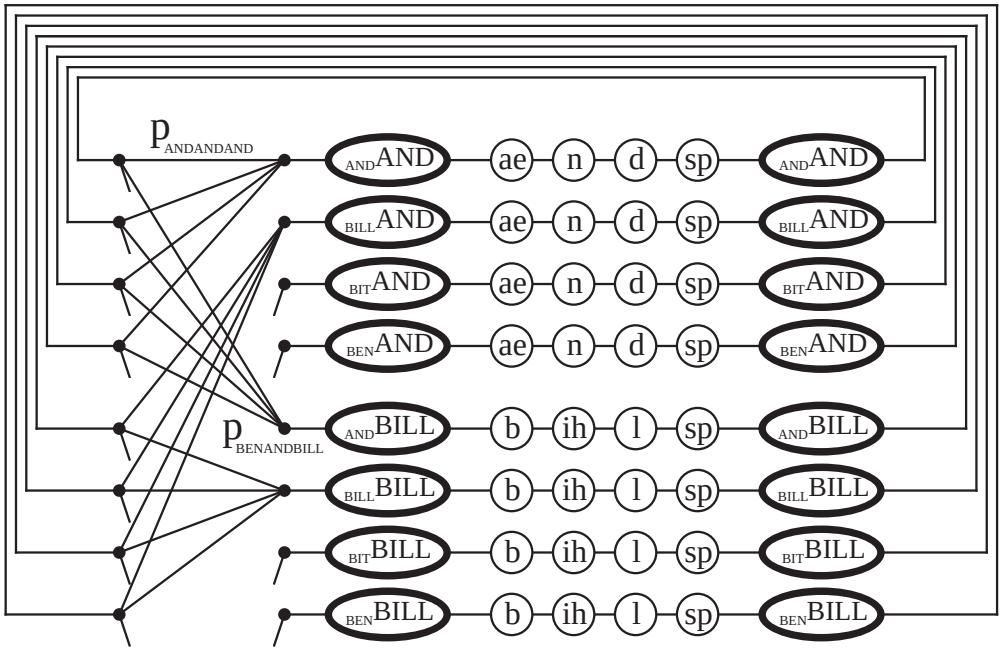
- Extension to word internal triphones is simple.

Trigram Language Model

Expanding to use trigrams is even more complex:

- Language model probability is dependent on the previous two words

- Duplicate each word according to previous word
- Apply trigram probability to start of each "duplicate" word
- Add separate loop back for each "duplicate" word



- Extension to word internal triphones is simple.

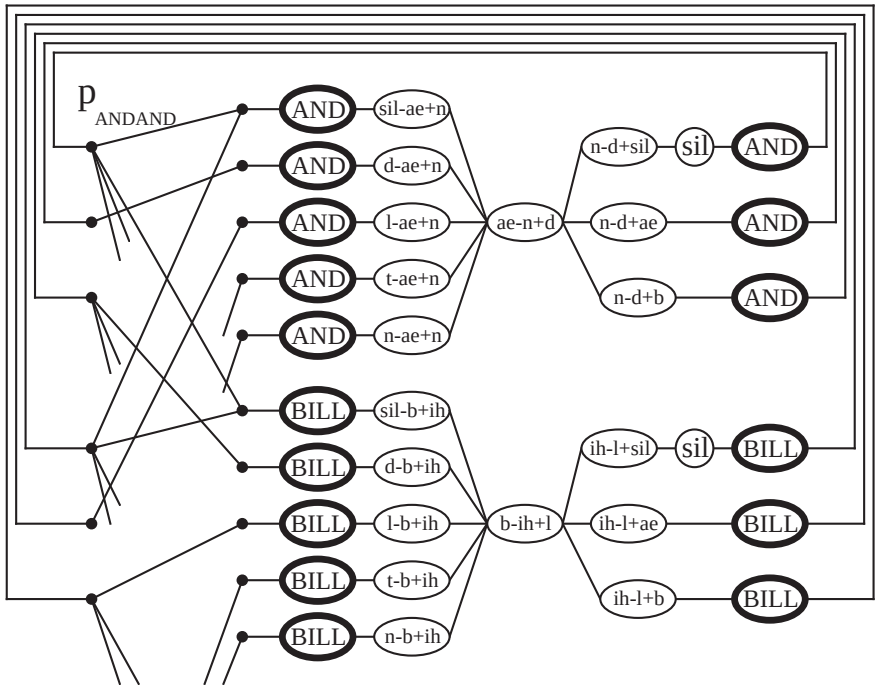
Cross Word Context Dependent Models

Consider a system with:

- Cross word triphone acoustic models + Bigram language model

Modify previous bigram network

- Make use of inter-word silence explicit in network
- Duplicate first phone of word according to the last phone of the previous word
- Duplicate last phone of word according to the first phone of the next word



Asymmetry in Pruned Search

When a decoder uses pruning with a fully-connected N-gram LM it is found that that most of the search effort is concentrated in the first phones of each word and relatively little at the word ends. For example the average number of active phones at different word positions for a 5000 word WSJ task (word-internal triphones with bigram) is shown below.

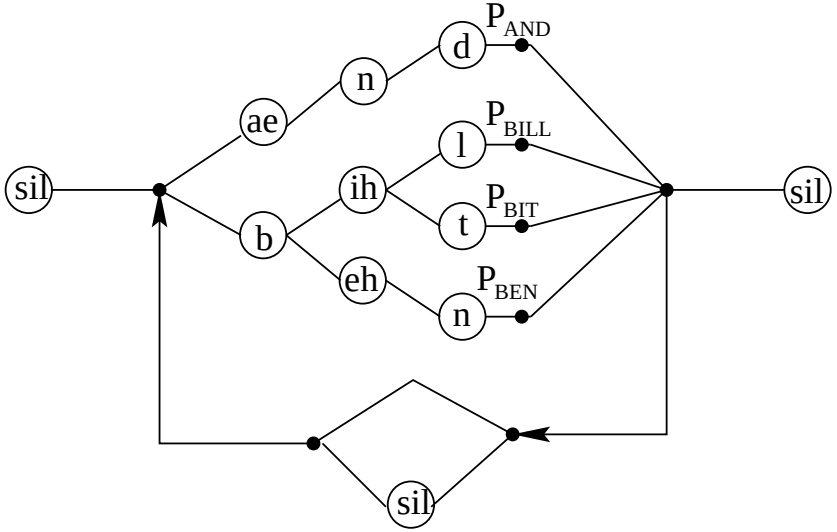
Model position in word	1st	2nd	Last but one	Last	Word End
Number active	3539	866	265	91	43
Proportion active	65.4%	16.0%	4.9%	1.7%	0.8%
Relative computation	76.0%	18.6%	5.7%	1.9%	

It can be seen that 95% of the computation is in the first two phones. Therefore to make an efficient decoder it is important to reduce the computation at the $start$ of words. This can effectively be done by tree-structuring the decoding network although this means that each word does not have a unique start in the network.

Tree Structured Lexicon

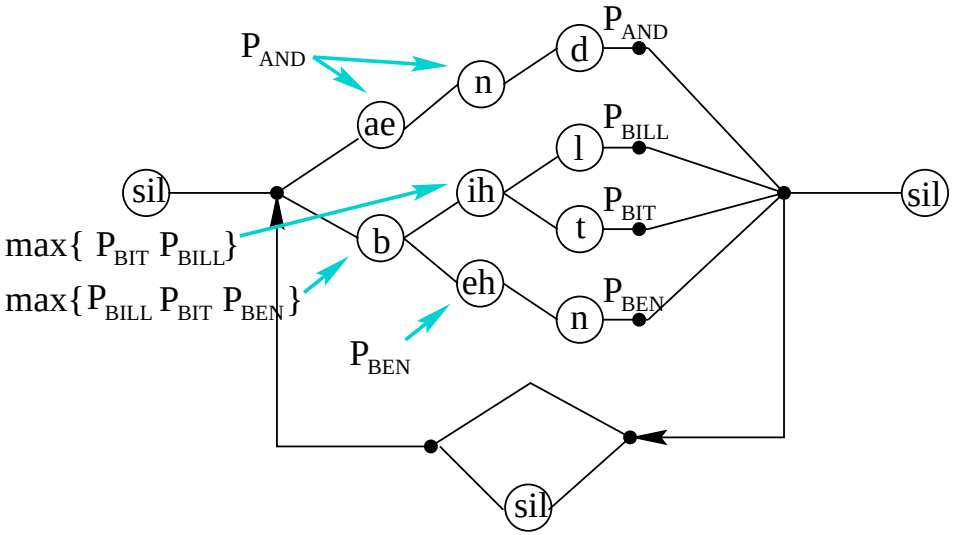
Problem with standard implementation

- At end of each word, typically many possible following words
- Most of these paths are rapidly pruned away.



Tree structuring the lexicon reduces search cost but

- the LM application is delayed until end of the word (reduces effects of pruning).
- may be overcome by early, approximate, LM application.



Summary of Viterbi CSR decoding

1. Time - synchronous beam search of composite HMM network

2. All paths cover same region $(0 \rightarrow t)$ of input speech hence they are simple to compare

3. If the beam is wide enough it can achieve minimal search errors

4. Incorporation of long-span language models and cross-word context models substantially increases complexity

5. Basic recognition network structures given above incl tree structuring: can be further optimised using general weighted-finite state techniques as described later in course.

An alternative to compiling all of the HMM/N-gram structure into a single decoding network is to use **Multi-pass search**. The data is processed initially with simple models to form a reduced search space (either **N-Best list** or **lattice**) via modified search and then full more complex models applied.

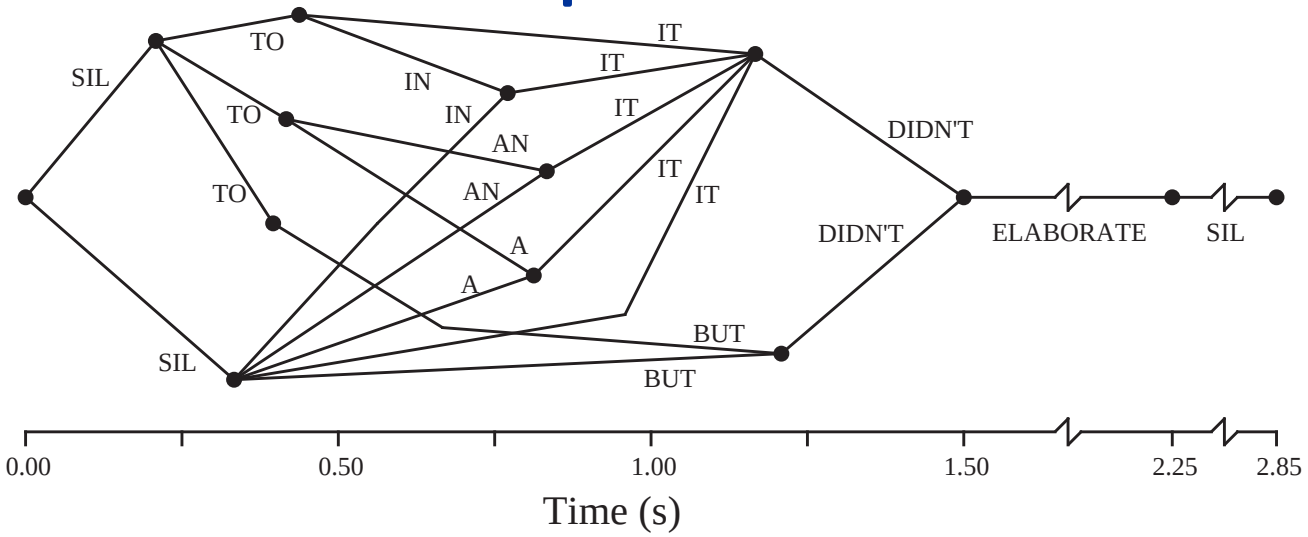
Multi-Pass Search Strategies

- Example of 1st/2nd pass:
  - recognise with word internal triphones and a bigram language model
  - rescore with cross-word context dependent phone models and a trigram LM
- **N-Best list**: produce list of possible sentences

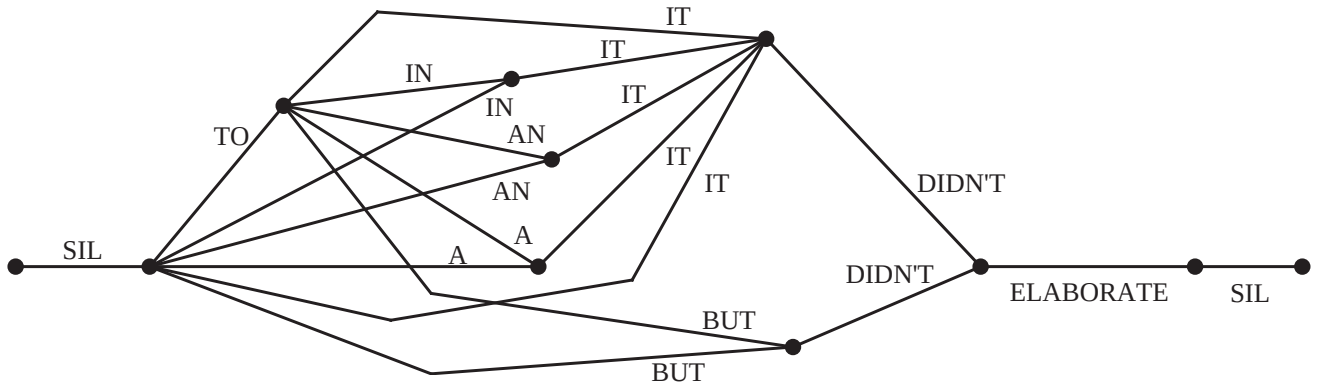
1	BUT DIDN'T ELABORATE
2	IT DIDN'T ELABORATE
⋮	⋮
100	TO IT DIDN'T ELABORATE

- **Lattice**: store
  - graph structure with with the associated words on nodes/arcs
  - language model probabilities (optional)
  - acoustic model probabilities (optional)
  - word boundary times (optional)

Example Lattices

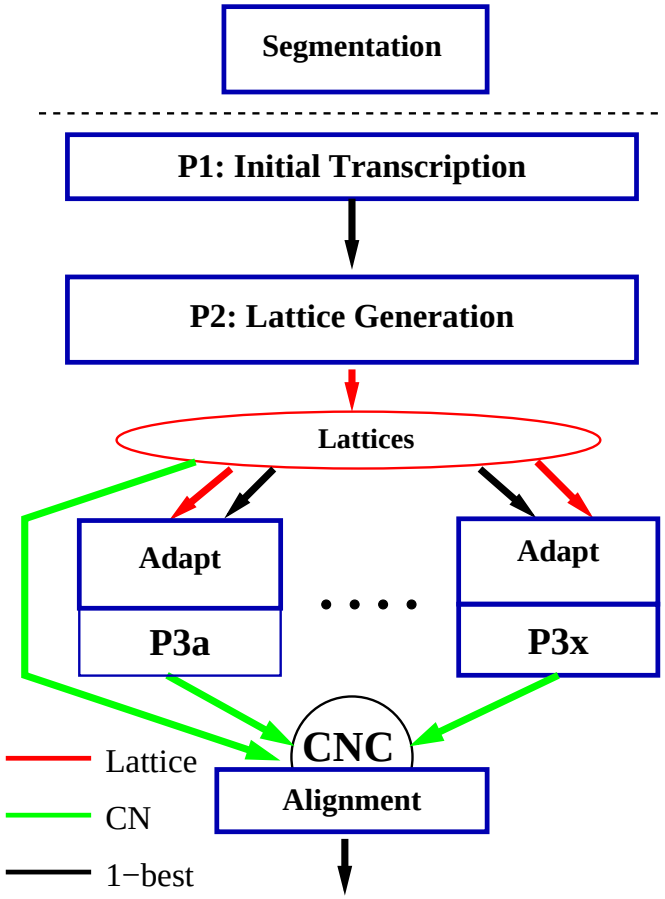


Cross-word context dependent HMMs + bigram language model



Record multiple alternate paths & merge to form word lattice

Typical state-of-the-art Transcription Architecture



- P1 used to generate initial hypothesis
- P1 hypothesis used for rapid adaptation
  - Initial adaptation
- P2: lattices generated for rescoring
  - apply complex LMs to trigram lattices
- P3 Unsupervised Adaptation
  - More sophisticated adaptation
- Use a reduced lattice form (confusion networks) to combine outputs

- Can run system just to P1, to P2 or complete system, with different pruning setting run-time and accuracy.

Summary & Outlook

- Viterbi search can be used for a variety of model types for continuous speech
- Can integrate all the models into a single search pass but
  - Network can get (much) more complex for different acoustic and language models
  - Network can be optimised using weighted finite state transducers (see subsequent lectures)
  - One alternative is to dynamically generate the network as needed (within pruning beam).
- Alternative in multi-pass search
  - Generate N-best or lattices with simpler acoustic/language models
  - Rescore lattices with more complex models
  - Can also include unsupervised adaptation step and is widely used in state-of-the-art systems