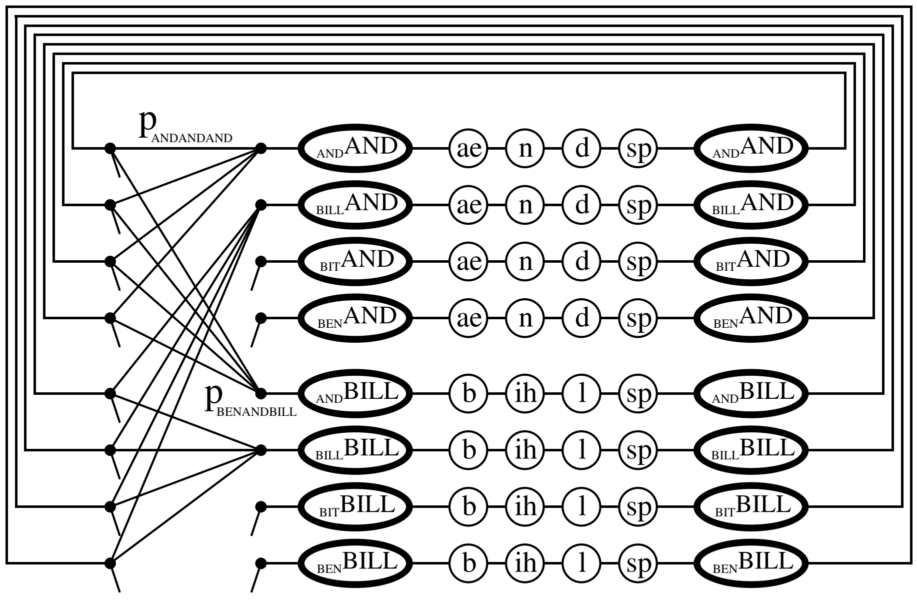
University of Cambridge

MPhil in Computer Speech Text & Internet Technology

Module: Speech Processing II

Lecture 9: Issues in LVCSR Search



Lent 2003

Introduction

This lecture discusses a number of issues that are important for large vocabulary recognition, particularly with N-gram language models.

Topics covered include:

- Requirements for an LVCSR decoder
- Language Model Application in Tree-Based Networks
- Word-End & Maximum Model Pruning
- Tree-Structured Lexicons
- Alternatives Strategies

The ideal decoder

An ideal decoder would be:

- **Efficient** - Computation resources must be reasonable. For dictation decoding must be better than real-time. Already seen *beam search* and *token passing*.
- **Accurate** - Decoder should always find the most likely state sequence- *admissible*. In order to keep the compute time down (e.g. using beam-search) almost always not possible. This results in *search errors*, which should be kept to a minimum.
- **Scalable** - Should be able to handle large vocabulary sizes.
- **Versatile** - Needs to be able to handle a variety of knowledge sources. For example N-gram language models and context dependent acoustic models.
- **Results** - Should be able to produce 1-best, N-best or lattices.

Time synchronous decoders using Viterbi recognition will be mainly discussed - in particular using the token passing paradigm.

Simple System

To illustrate some of the search issues 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:

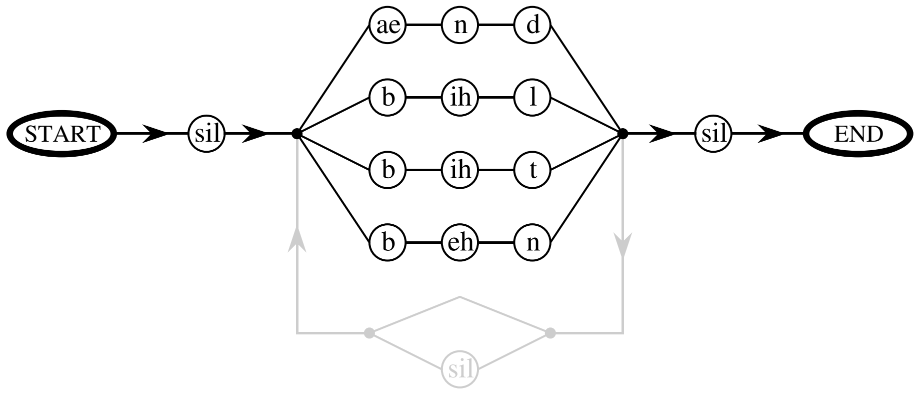
- Finite-state network (and unigram language model)
- Bigram language model
- Trigram language model
- Cross-word triphone context dependent models

Finite-State Network

Initially consider a system with:

- Monophone models
- Finite state grammar

This is shown below (link in grey shows the extension to continuous speech).



To extend this to use a unigram language model:

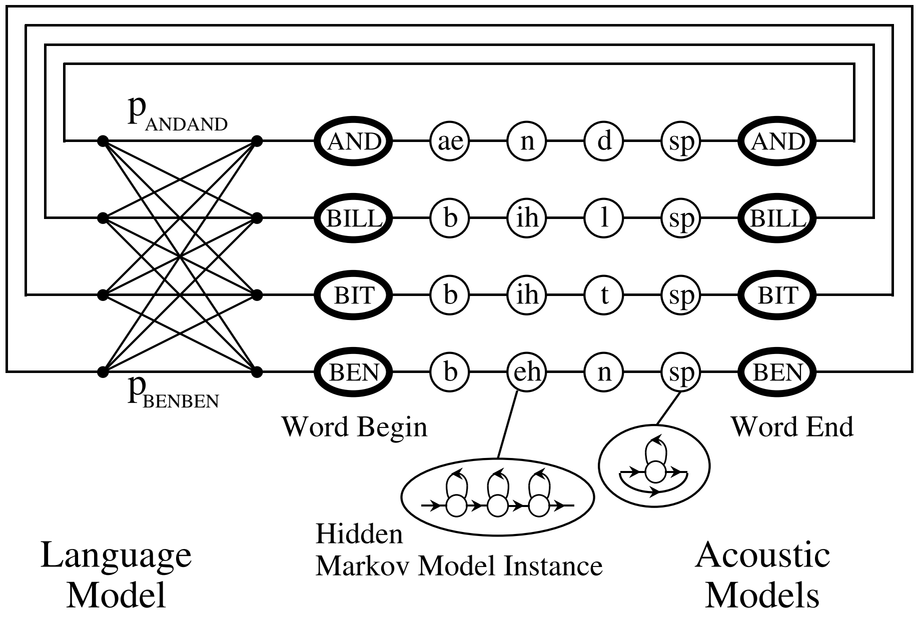
- Add the language model probability (e.g. $\log(P_{BEN})$) to the log-likelihood of the token at the start of each word.
- Language model information incorporated as soon as possible to aid pruning (beam search).

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.



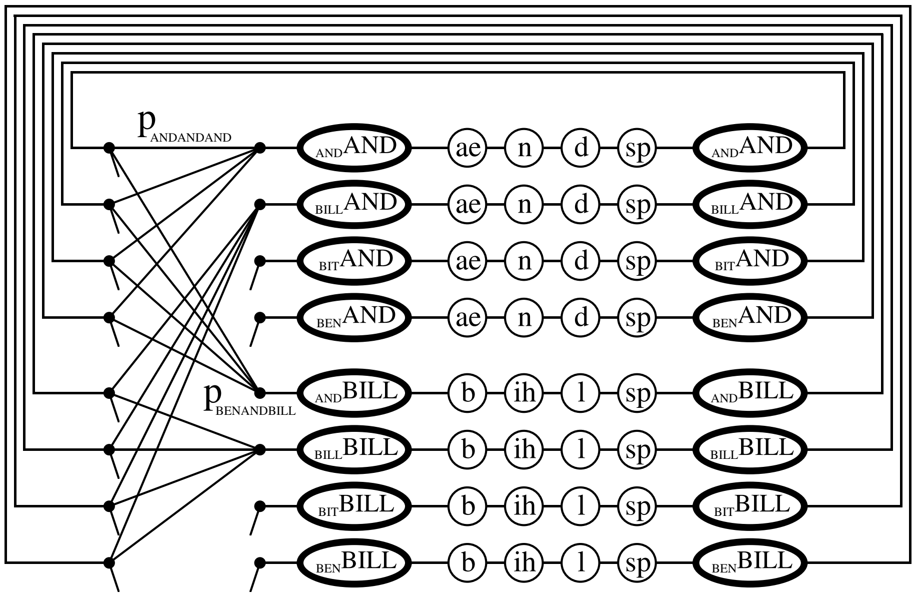
Modify network by:

- 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 word



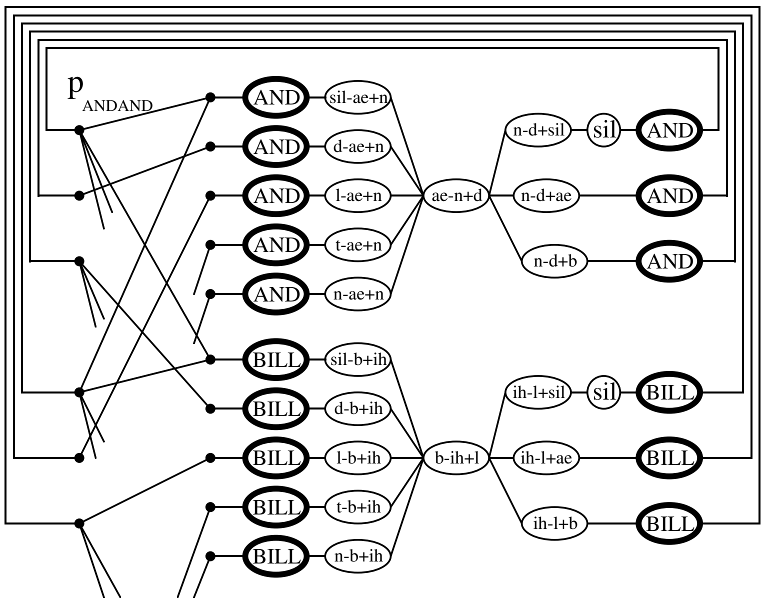
Modify network:

- 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 Models

Consider a system with:

- Cross word triphone 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.

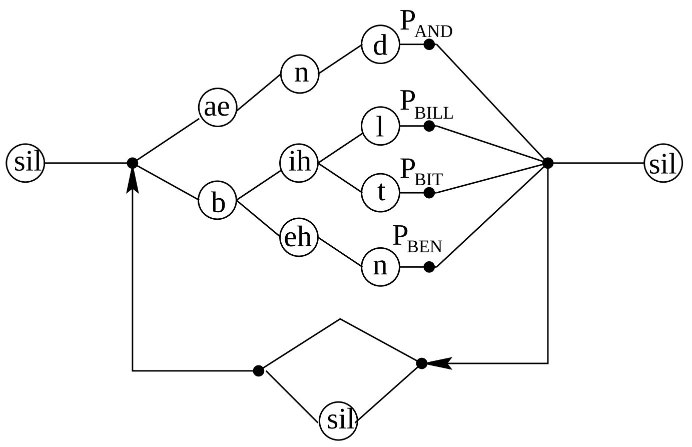
Model position in word	First	Second	Last but one	Last	Word End
Number active	3539	866	265	91	43
Proportion active	65.4%	16.0%	4.9%	1.7%	0.8%
Relative computation	76.0%	18.6%	5.7%	1.9%	

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

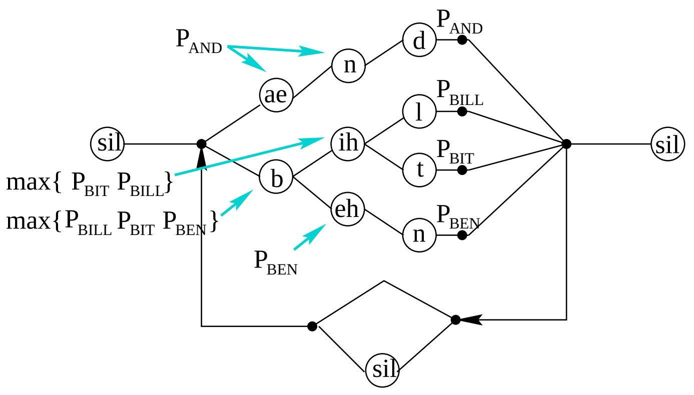
Problem with standard implementation

- At the end of each word there are typically many possible following words - most of these paths are rapidly pruned away.



Tree structuring the lexicon reduces cost, but

- the language model application is delayed until the end of the word - reduces the effectiveness of pruning.



May be overcome by early, approximate, application.

Dynamic Network Decoding

Examining the use of a trigram network and cross-word triphones the size of the network rapidly becomes impractical to store on current machines.

Rather than using a static network decoder, a *dynamic* network decoder may be used. The basic idea is to create new network nodes as required and rely on pruning of the search space to keep the number of active paths small. This allows more complex acoustic and language models and still integrates all knowledge in a single pass.

Basic description:

- tree structure network where possible;
- create new network nodes as needed;
- expand network to ensure enough context for language models and acoustic models;
- cross-word models triphone models only slightly more expensive than word internal models;
- relies of rapid pruning of search space;
- a general purpose dynamic network decoder can use arbitrary language model and acoustic model constraints (e.g. one in use at CUED uses pentaphone (quinphone) HMMs with a 4-gram LM).

For efficient pruning early application of the language model is required.

Alternatives to Viterbi

- **Fast-match procedures** - Look ahead a few frames and calculate an approximate match using for example monophone models. If this is unlikely, discard from the beam now.
- **Stack-decoders** - modify form of decoder to run in a *time asynchronous* fashion.
- **Multi-pass search** - run over data more than once. For example:
  Initially use word internal models/bigram. HMMs may be discrete or tied mixture. Use these simple models to generate multiple hypothesise (described in more detail in next lecture). Rescore hypothesise with more accurate cross-word models on an*N-best list* or *lattices* saved during the forward pass.

  Sometimes (e.g. BBN) run a very fast backwards pass to generate the final answers (and use a backwards dictionary, backwards data and a backwards LM).

Stack decoders

Continuous speech recognition in a stack decoder framework involves growing a tree of word hypotheses. In a simple implementation:

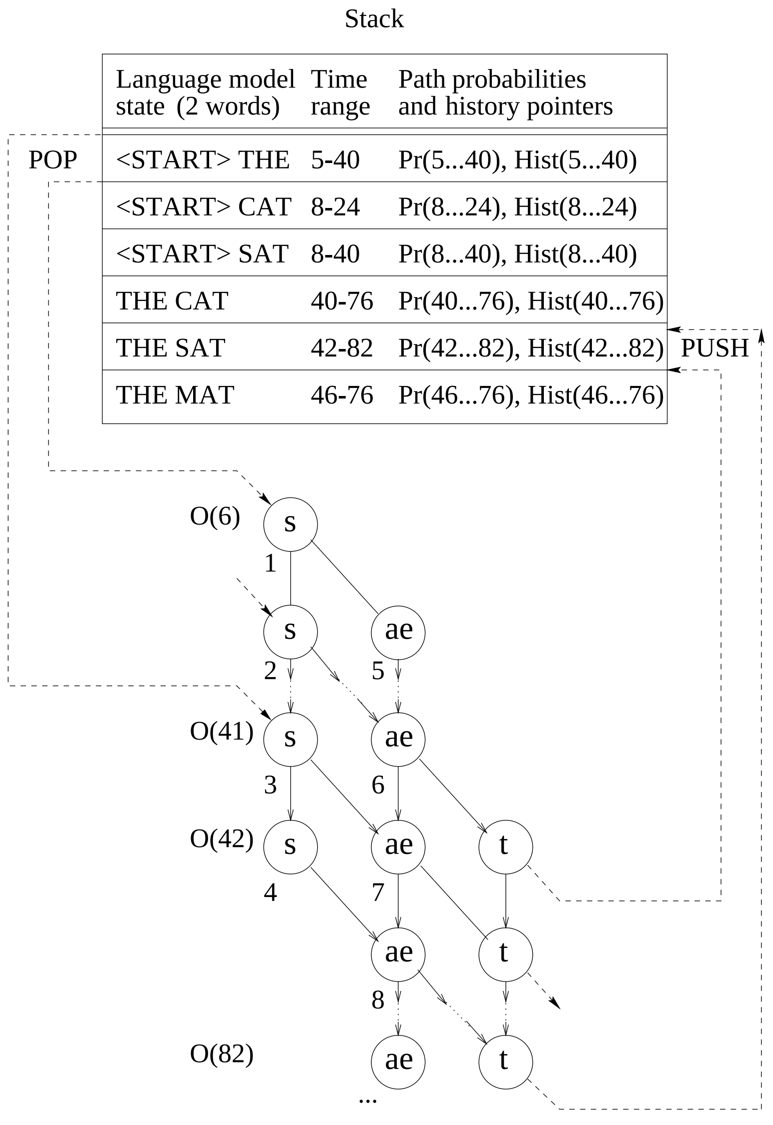
```
while not complete
  pop the top item from the stack
  expand the set of hypotheses
  for each extension
    if not seen before then
      add to stack
    else
      update appropriate item with these paths
```

Stack decoders allow the simpler incorporation of long-span language models since the whole word history is known for each stack item.

Stack ordering

- **Best first**: based on the log-likelihood of the hypothesised path, independent of length.
- **A* search**: as best first, but adds an estimate of the cost from current position to the end of the sentence (the least upper bound). Becomes very efficient as the LUB approximates the true path probability – but good estimates are hard to come by.

Example Stack



Stack

Language model state (2 words)	Time range	Path probabilities and history pointers
<START> THE	5-40	Pr(5...40), Hist(5...40)
<START> CAT	8-24	Pr(8...24), Hist(8...24)
<START> SAT	8-40	Pr(8...40), Hist(8...40)
THE CAT	40-76	Pr(40...76), Hist(40...76)
THE SAT	42-82	Pr(42...82), Hist(42...82)
THE MAT	46-76	Pr(46...76), Hist(46...76)

Time-first hypothesis extension

1. The partial hypothesis "<START> THE" is popped off the stack
2. Extend by each word (in this case "SAT")
3. Result is pushed back on the stack.