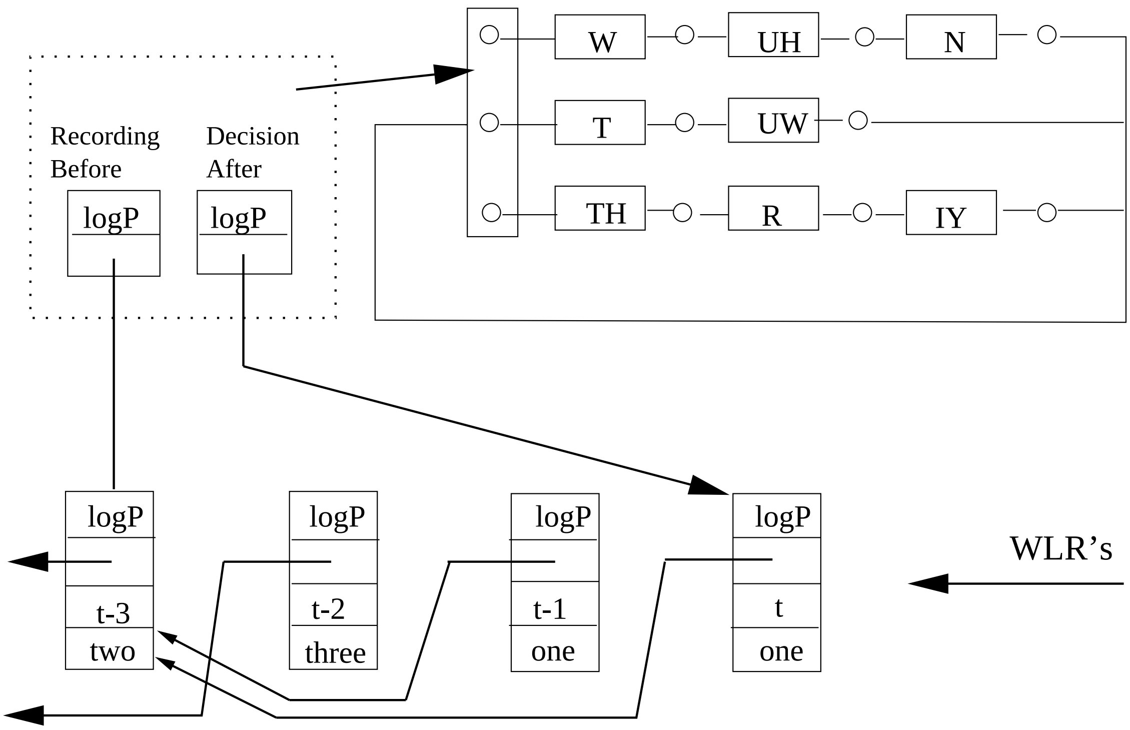
University of Cambridge

MPhil in Computer Speech Text & Internet Technology

Module: Speech Processing II

Lecture 8: Basic Search Revision



Lent 2003

Introduction

Recall that in speech recognition we are trying to find the word sequence, $\mathbf{W}$, that maximises $P(\mathbf{W}|\mathbf{O})$ where $\mathbf{O}$ is the observed speech.
From **Bayes' Theorem**:

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

The search (decoder) component needs to find this value. We have discussed and N-gram techniques earlier in this module. The basic property of these techniques is that *all* the knowledge (constraints) present can be represented in a network form.

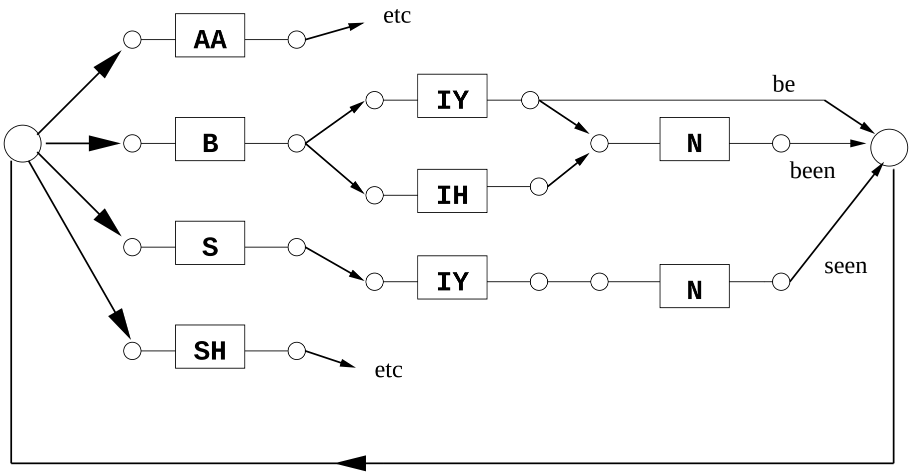
Hence we have

- Phone based HMMs (possibly context dependent)
- Pronunciation lexicon (dictionary)
- Language model

and in principle all the knowledge present can be compiled into a large HMM and the best (most likely) path found through the network.

In this lecture we will review basic Viterbi-style decoding for continuous speech recognition and discuss basic pruning techniques and continuous operation.

Simple No Grammar Network



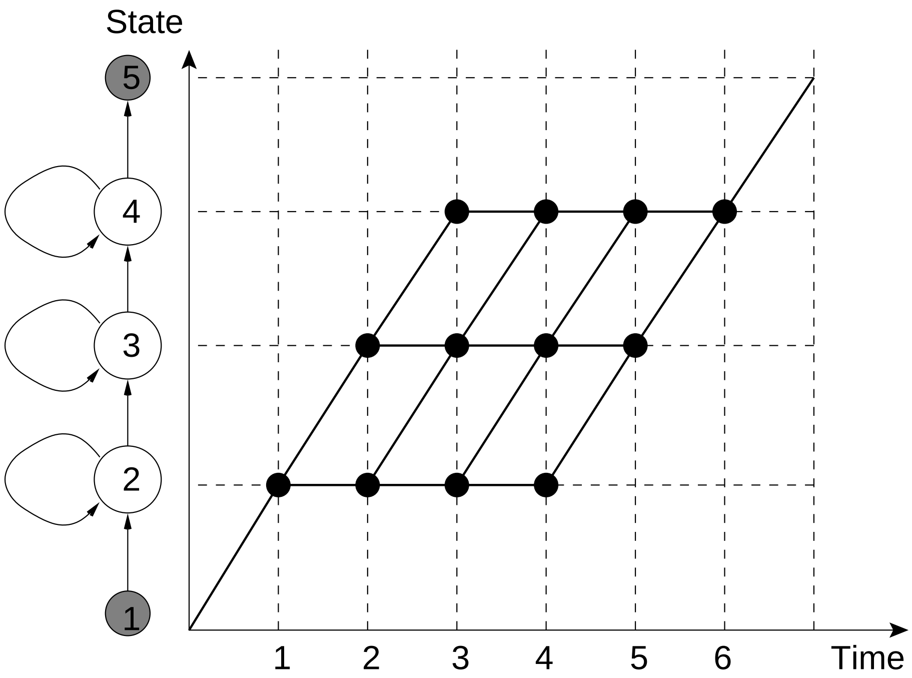
Notes

(a) The basic HMMs are duplicated when the full network is expanded: we talk of model *instances*. (Note able to *cache* log-likelihoods for duplicate copies).

(b) Example shows all words in a loop but can add extra finite language model structure.

(c) Network transitions may have probabilities attached to them.

Isolated Word Recognition



In theory we should compute the likelihood of each model generating the set of observations, $p(\mathbf{O}|\mathcal{M})$. For efficiency reasons we commonly approximate this by the *Viterbi* approximation using only the best state sequence. This likelihood will be denoted as $\hat{p}(\mathbf{O}|\mathcal{M})$.

We can write

$$\begin{aligned}\hat{p}(\mathbf{O}|\mathcal{M}) &= \max_{S}\{p(\mathbf{O}, S|\mathcal{M})\} \\ &= \max_{S}\{\prod_{t=1}^{6} a_{s(t-1)s(t)} b_{s(t)}(\mathbf{o}_t)\}\end{aligned}$$

where $s(1)\ldots s(6) = S$ and $s(0) = 1$. In addition we will be able to extract the best state sequence $\hat{S}$

$$\hat{S} = \arg\max_{S}\{p(\mathbf{O}, S|\mathcal{M})\}$$

Viterbi Algorithm

The best state sequence through trellis may be found by the *Viterbi Algorithm*. In a similar fashion to the *forward probability* we will introduce a new variable

$$\phi_j(t) = \max_S \left\{ p(\mathbf{o}_1, \ldots, \mathbf{o}_t, s(t) = j | \mathcal{M}) \right\}$$

We now need to define an efficient recursion. This may be achieved using

$$\phi_j(t) = \max_i \left\{ \phi_i(t-1) a_{ij} \right\} b_j(\mathbf{o}_t)$$

The algorithm is initialised with

$$\phi_j(0) = \begin{cases} 1, & j = 1 \\ 0, & j > 1 \end{cases}$$

In words: $\phi_j(t)$ represents probability of a partial path through the trellis and the *max* is over all paths ending in state j at time t.

In practice it is convenient to use logs to avoid underflow.

$$\psi_j(t) = \max_i \left\{ \psi_i(t-1) + \log(a_{ij}) \right\} + \log(b_j(\mathbf{o}_t))$$

This may be though of in terms of dynamic programming

$$\begin{array}{c}\mathrm{PartialPath} \\ \mathrm{to\ (j, t)}\end{array} = \begin{array}{c}\max \\ i\end{array} \left\{ \begin{array}{c}\mathrm{PartialPath} \\ \mathrm{to\ (i, t-1)}\end{array} + \begin{array}{c}\mathrm{Transition} \\ \mathrm{Cost\ ito\ j}\end{array} \right\} + \begin{array}{c}\mathrm{LocalSimilarity} \\ \mathrm{Measure}\end{array}$$

Paths

We would also like to be able to obtain the state sequence associated with $\hat{p}(\mathbf{O}|\mathcal{M})$. To do this we introduce the concept of a *path*. This is crucial to understanding continuous speech recognition.

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

- a score - usually a log likelihood
- a history - to record the preceding sequence of states

For isolated word recognition where we might be interested in finding the best state sequence the history should be at the state level. The extension to handle *continuous* speech recognition requires that the history be able to give the word (or phone) sequence.

The problem is to have a representation of the path that allows:

- a compact form of the history to be stored;
- at the end of the utterance the representation of the history can be mapped to the state/phone/word sequence.

To achieve this we will use **token passing** and **traceback**.

Token passing

The info describing the head of a (partial) path can be stored in a *token* containing

`LogP` log-likelihood of (partial) path
`Link` pointer to history information

All states of the expanded HMM network hold one token.

We can then define

- **start** token: `LogP`$= 0$, `Link=NULL (*)`
- **null** token: `LogP` $= -\infty$, `Link=NULL (*)`

At time t, the token in each model state represents a path through trellis covering input speech from time 1 to

(a) $t - \delta t$ for state 1

(b) t for states $2 \rightarrow N-1$

(c) $t + \delta t$ for state N

On completion, `LogP` of token in state N should be $\hat{p}(\mathbf{O}|\mathcal{M})$

Token Passing (cont)

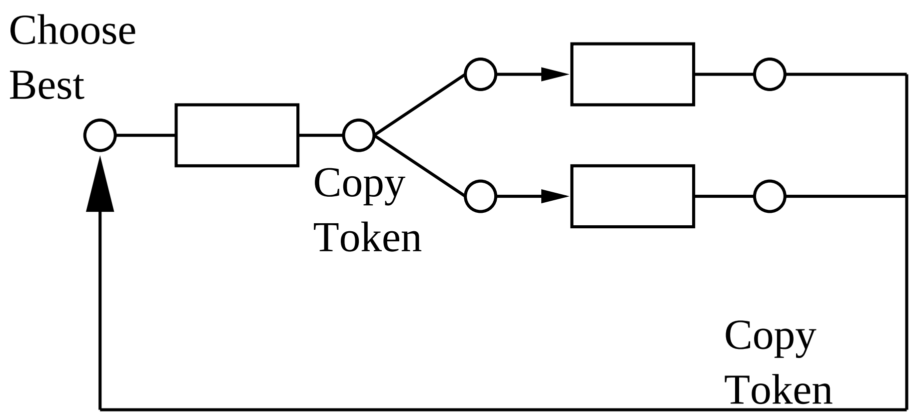
Initially consider *isolated word* recognition. We will only be interested in the word sequence (isolated word recognition, so we do not need to worry about the history).

Viterbi algorithm can be restated as:

```
Put a start token in entry node;
Put null tokens in all other nodes;
for each time t = 1 to T do
    /* Start of Step Model */
    for each state i < N do
        Pass a copy of the token Q in state i to
           all connecting states j;
        Q.LogP = Q.LogP + log(a_ij) + log(b_j(o_t))
    end;
    Discard all original tokens;
    for each state i < N do
        Find token in state i with max LogP
           and discard the rest
    end;
    for each state i connected to state N do
        Pass a copy of the token Q in state i to state N
        Q.LogP = Q.LogP+ log(a_iN)
    end;
    Find token in state N with max LogP
       and discard the rest;
    Put null token in entry state
    /* End of Step Model */
end;
```

Connected Unit Case

Extension of the token passing algorithm to deal with connected units is now simple. Algorithm becomes:




```
Put a start token in all network entry states;
Put null tokens in all other states;
for each time t = 1 to T do
    Step All Models;
    Propagate Exit Tokens to all connecting entry states;
    Record Decisions;
    Delete all but the best token in each entry state
end
```

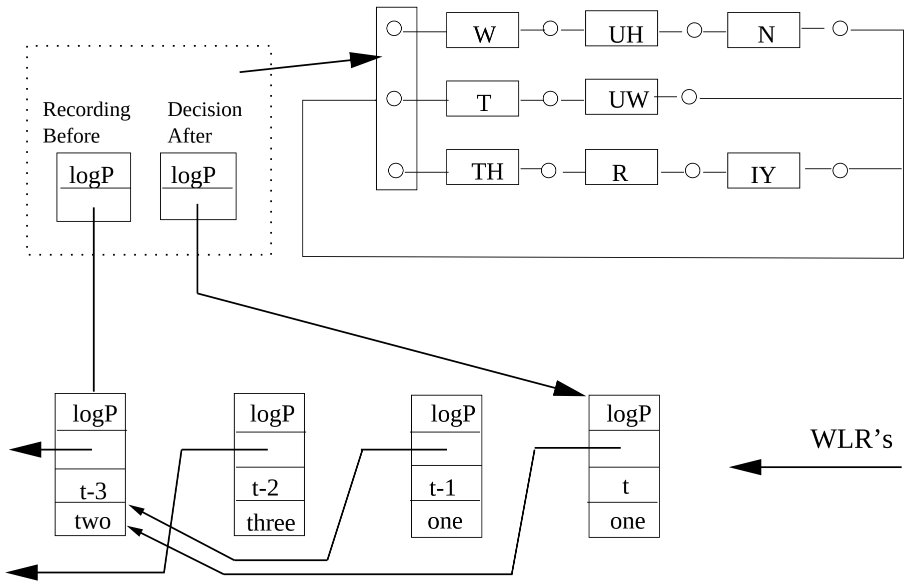
On completion, best token in exit states of all valid network final models represents most likely model sequence. Note that common entry states should be regarded as being a single entry state for this and subsequent algorithms.

We need to decide how `Record Decisions` should be implemented.

Record Decisions

This function is used to give a compact history representation (for this algorithm at the word level). The points at which tokens are propagated from the exit states of words to entry states of other words must be recorded. A *Word Link Record* (WLR) is used.

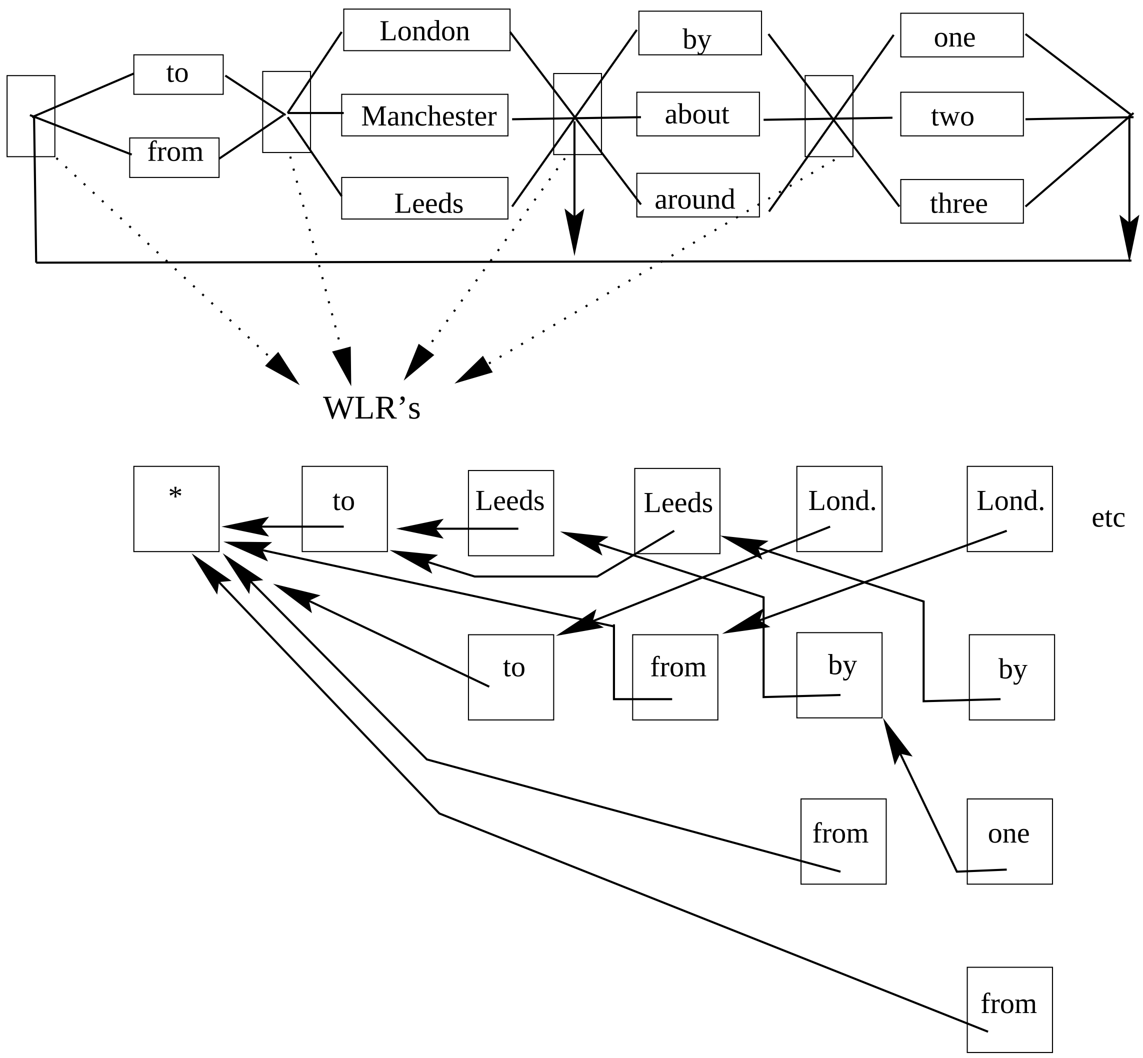
```
for each best token Q in each entry state at each t do
    Create a new WLR w containing:
             (1) a copy of Q
             (2) time t
             (3) identity of emitting word
    Q.link = w
end;
```



For simple loop network, 1 WLR is generated per speech frame.

Record Decisions (cont)

When syntax constraints are included, 1 WLR is generated for each *syntactically distinct node in the network.*



Traceback

Word link records have given a compact representation for the history. We now need to obtain the best word sequence.

After the final frame of speech has been processed

```
Examine WLRs generated at time T, find WLR with max LogP;
Print WLR.time, WLR.LogP, WLR.word;
while WLR.Link != NULL do
      WLR = WLR.Link;
      Print WLR.time, WLR.LogP, WLR.word;
end;
```

The traceback procedure yields:

1. best sequence (in reverse order)
2. word boundary locations
3. LogP for each individual word (can be calculated)

Performance Issues

For W words, N states/word (average), we need to perform

1. W x N internal token propagations
2. W external token propagations

for each time frame (typically 10ms)

Each internal propagation involves

(a) output prob calculation (may be shared)

(b) 2 or 3 add and compare ops for state instance

The internal propagation is (at least for medium vocabs) by state-output probability calculations. For N distinct state distributions, M Gaussians per state and vector size V, need to have $N \times M \times V \times 2$ multiply-adds per frame. If $N = 2000, M = 10, V = 39$ which is 1.5 million multiply-adds per 10ms frame. This is a already a lot. For a large vocab system (esp one with context-dependent models and a language model) the internal/external token propagation is at least as expensive (due to the very large number of state instances).

In practice, 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*.

Basic beam search algorithm

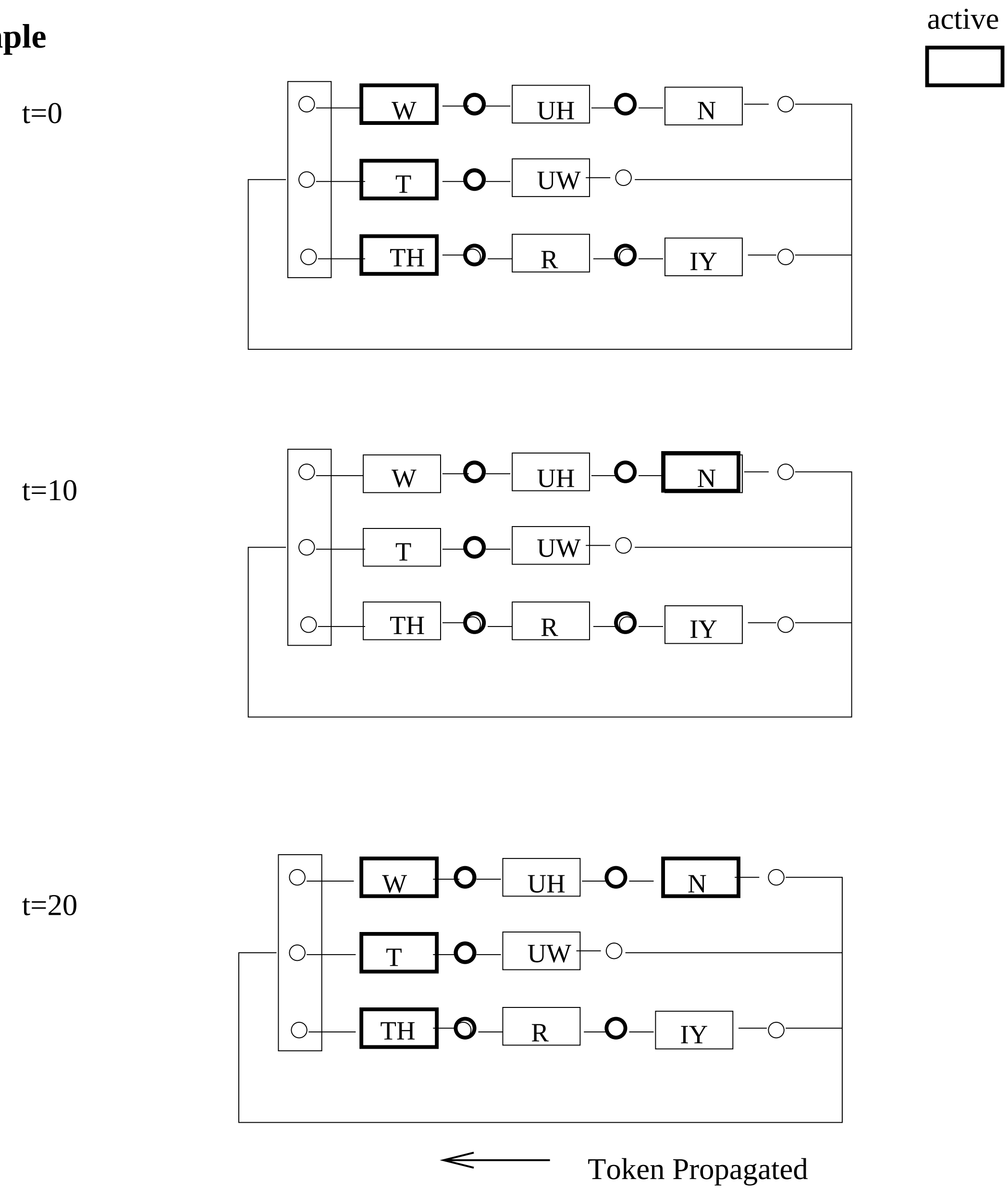
We would like to ensure that our search was *admissible* (i.e. the best path is *guaranteed* to be found). In practice this is too expensive, we will need to accept a certain level of *search errors*.

Each model instance is either *active* or *inactive*. The basic beam search algorithm is:

```
Set all entry models of all network initial words are active;
for each time t = 1 to T do
    for each active model w do
        Step Model;
        Find maximum LogP in w, lMax(w);
    end;
    Find global maximum LogP, gMax
    for each active model w do
        if lmax(w) < gMax - Thresh
           De-Activate w;
    end;
    PropagateExit Tokens to all connecting entry states
             if LogP > gMax - Thresh;
    Record Decisions;
    Delete all but the best token in each entry state;
    Re-Activate all entry models which have just
                received a new entry token;
end;
```

Beam Search Example

Example



Partial Traceback

During recognition, every active token represents a possible path. In the basic algorithm, we wait until end of speech and then *traceback* to find the best sequence. However, a large grammar could generate 100's of WLR's per frame. This results in a large memory management overhead and a waste of memory.

This is not necessary since:

1. not all WLR's lie on active paths;
2. when a word is recognised "well" - all active paths pass through that WLR (esp. when pruning is used)

The aim of *partial traceback* is to:

- remove all WLR's that do not lie on active paths
- if all active paths pass through the *same* node (model) in the network, output path upto that word.

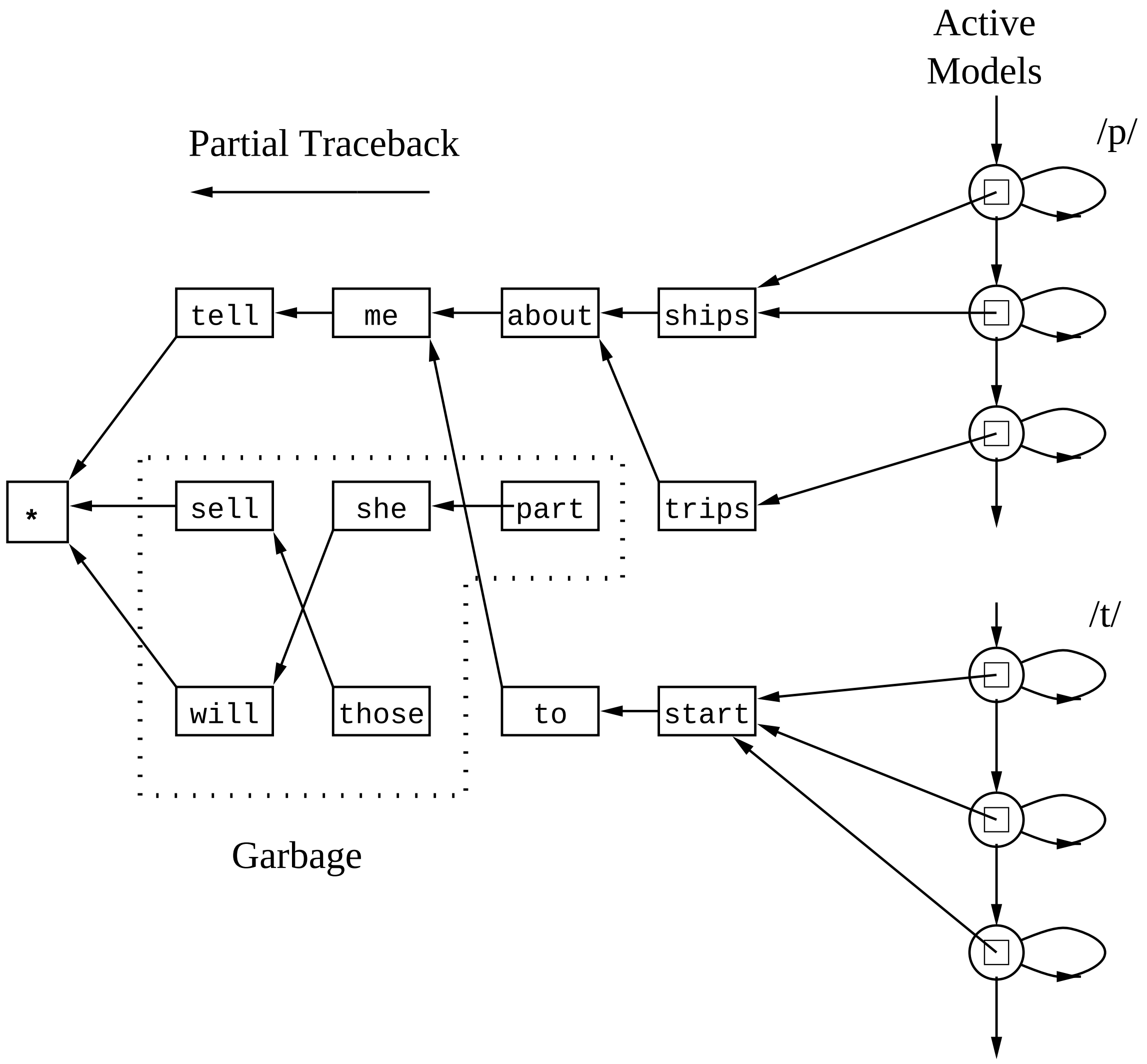
Partial Traceback Algorithm

Every WLR is given a usage counter ”nUse” to count the number of pointers to it.

```
for every WLR, wlr, do
    wlr.nUse = 0;
end;
for every active model w do
    for every token in w do
        for each WLR, wlr, on trace back path do
            wlr.nUse++
        end;
    end;
end;
for every WLR, wlr, do
    if wlr.nUse == 0 then
    delete wlr;
end;
for all active tokens, Q
    for each WLR, wlr, in traceback path from Q do
        if wlr.nUse == total active states then
           output remaining path;
           change current WLR to NULL (*)
        end;
    end;
end;
```

Note - the above algorithm allows a recogniser to run continuously withou having to wait until the speaker has finished before outputting something (look at how current speech products).

Partial Traceback Example



In the above example:

- 5 WLRs can be deleted as they are not on active paths;
- `tell me` can be printed.

Summary of Basic Viterbi Decoding

1. Time-synchronous beam search
2. All paths cover same region ($1 \rightarrow t$) of input speech hence they are simple to compare
3. If beam is wide enough, we can guarantee to find most likely word sequence with computation linear in the number of active words and length of input
4. Partial traceback and garbage collection make continuous operation possible
5. It is easy to implement

However, the basic algorithm is difficult to scale to large vocabularies and integrate in complex acoustic models and N-gram language models due to the very large fully-expanded state-space. We will examine some of these issues in future lectures.