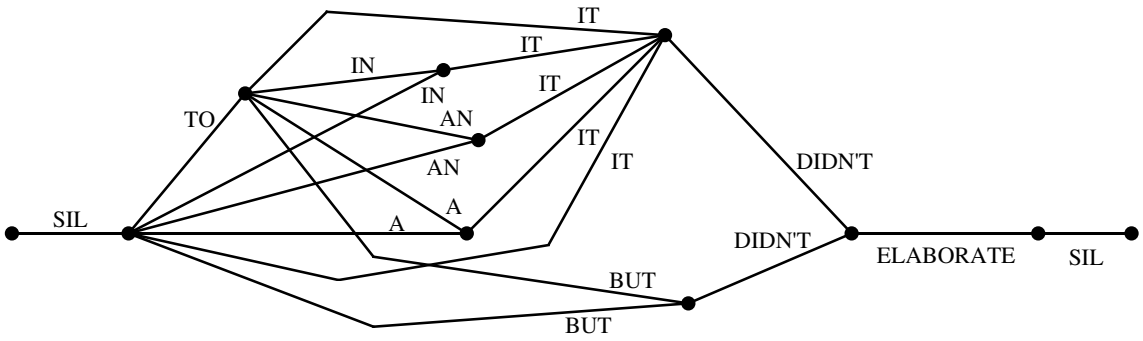
University of Cambridge

MPhil in Computer Speech Text & Internet Technology

Module: Speech Processing II

Lecture 10: Generation & Use of Multiple Solutions



Lent 2002

Introduction

It is often the case that a decoder will be required to produce multiple solutions rather than just the 1-Best output. This lecture discusses how this can be done and some ways in which such output can be used.

Topics covered include:

- Lattice & word-dependent N-best
- Multiple token search
- Lattice Quality
- Lattice Pruning
- Applying advanced LMs
- Acoustic Rescoring
- N-Best paradigm to integrate varied knowledge sources
- Confidence measures

Lattice/N-Best Generation

The standard Viterbi algorithm can be used to generate the log-likelihood and best path using a particular acoustic model set, language model and lexicon. In a multi-pass system we might want to use a more complex acoustic or language model in the decoding. In this case we need to be able to rescore multiple "good" paths, not just the best path (or what's the point!).

The multiple paths are either stored as

- **N-best list**
- **Word Lattice**

For the vast majority of LVCSR systems lattices are preferred, as they are a far more compact representation.

To generate multiple hypothesis the Viterbi algorithm need to be altered. In this lecture three schemes will be examined:

- **Lattice** N-best;
- **Sentence dependent** N-best;
- **Word dependent** N-best.

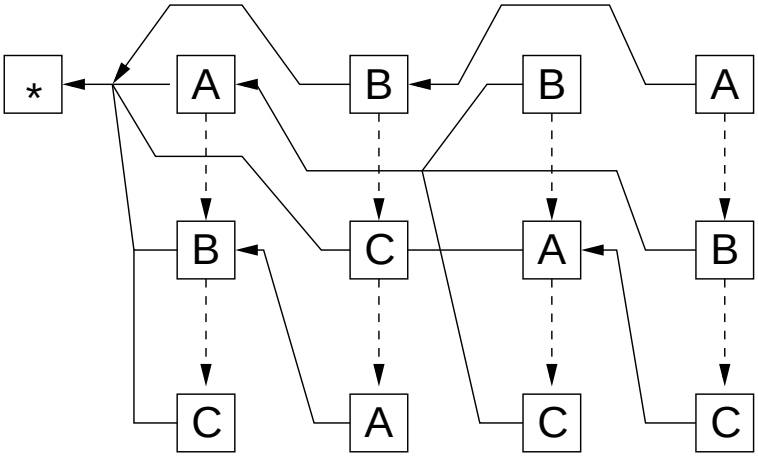
Lattice N-Best

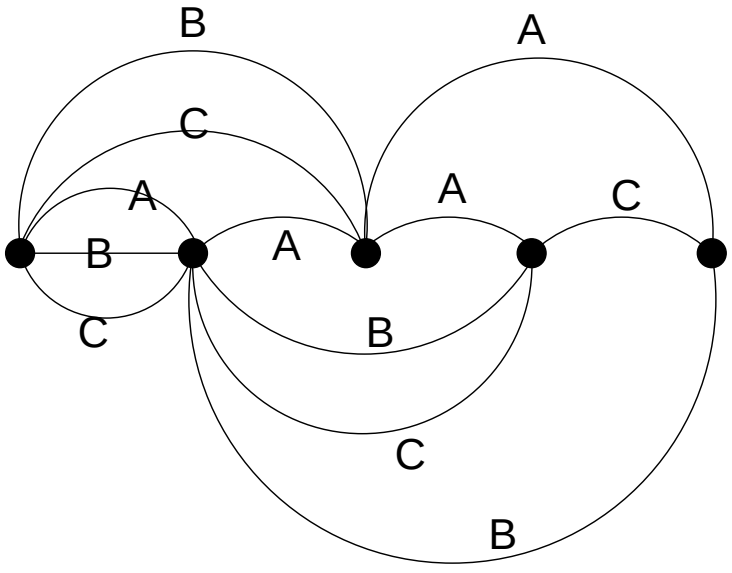
This is a simple modification to the basic token-passing Viterbi search that just alters the `Record Decisions` function. At each frame a WLR is created for the N-Best tokens entering a word instance in the network. Within a word only the single best token is propagated and no extra computation is required, however this can cause search errors in finding N-Best alternatives.

```
for each entry state at each t do
    Select the best token Q
    Create a new WLR w containing:
              (1) a copy of Q
              (2) time t
              (3) identity of emitting word
    Q.link = w
    for 2nd to Nth best token, q, do
        Create a new WLR v containing:
                  (1) a copy of q
                  (2) time t
                  (3) identity of emitting word
        chain v to w;
    end;
end;
```

At the end of the sentence the trace-back procedure is used to convert the history information to a *word lattice* which is a network structure containing the acoustic and language model scores. This can be used to find the N-Best solutions.

Example





Sentence-Dependent N-Best

Basic token passing approach is modified so that each state can hold multiple tokens.

1. Each state holds N tokens
2. All N tokens from each connecting state are propagated using the normal token propagation rules
3. Instead of discarding all but the best token in each state do following
   (a) find all sets of tokens with identical word sequences in their histories
   (b) for each such set, add all partial path probabilities and merge into a single token (or find the maximum of the path probabilities)
   (c) discard all but the N-best tokens

On completion, the N-best sentences are available immediately from the N tokens in the final state.

This is guaranteed to find the best N paths through the network. However the computational cost is high.

Sentence-Dependent N-Best (cont)

The following should be noted:

- The sentence N-best algorithm is computationally very expensive. Main problem is cost of identifying sets of tokens with identical histories.
- N typically needs to be $20 - 100$ for a 1000 word vocab and a modest task, and larger for a sizeable task.
- In practice a pruning threshold must be used to reduce the number of tokens entering a state.
- If token merging adds partial path probabilities the algorithm will find the most likely word sequences and not just the most likely state sequences: this is found to make very little difference in practice.

Word-Dependent N-Best

The idea here is to use an intermediate algorithm between the lattice N-best algorithm and the sentence dependent N-best (exact) algorithm.

The word-dependent N-best algorithm uses what is sometimes called the *word-pair* approximation. In this approximation it is assumed that for any word pair, the boundary time only depends on the two words and not anything else in the history (this is a good approx. for fairly long words).

In the example above the position of the boundary between `cat` and `sat` is independent of the surrounding words.

Each state now stores a fairly small number, n, of tokens. This should be reasonable as the number of paths that get merged due to the word-pair approximation is large.

Word-Dependent N-Best (cont)

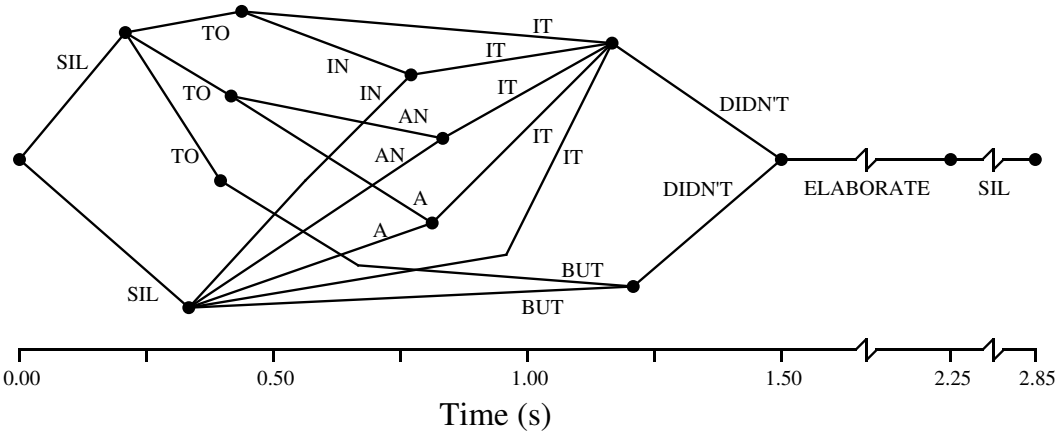
1. Each state holds n tokens
2. All n tokens from each connecting state are propagated using the normal token propagation rules
3. Instead of discarding all but the best token in each state do following
   (a) find all sets of tokens with the same previous word:
   (b) for all such states, merge tokens by adding or finding the max of the partial path probabilities
   (c) keep the top n tokens in each state and discard the others

Typically n is small (e.g. less than 10).

In practice this approximation is very widely used to generate multiple solutions.

Word Lattices

A typical word lattice structure is shown.



A general word lattice structure contains:

- A set of nodes that correspond to points in time (or word-ends)
- A set of arcs that are labelled with the word identifies and the acoustic scores of words between nodes and also the language model scores

Note that if acoustic score information is included then many arcs will be replicated due to slightly different acoustic context / timing information.

Standard Lattice Format

This is an example of the *standard lattice format* SLF that is used in HTK. The latest is generated using a 20,000 word bigram language model. The best hypothesis is `!ENTER IT DIDN'T ELABORATE !EXIT` Score=-20218.25 the !ENTER and !EXIT model initial and final silence.

```
VERSION=1.1
UTTERANCE=4k0c030t
lmscale=16.0
wdpenalty=0.0
N=24   L=39
# Node definitions
I=0    t=0.00
I=1    t=0.25
I=2    t=0.26
I=3    t=0.61
I=4    t=0.62
I=5    t=0.62
I=6    t=0.71
I=7    t=0.72
I=8    t=0.72
I=9    t=0.72
I=10   t=0.72
I=11   t=0.72
I=12   t=0.72
I=13   t=0.73
I=14   t=0.78
I=15   t=0.78
I=16   t=0.80
I=17   t=0.80
I=18   t=0.81
I=19   t=0.81
I=20   t=1.33
I=21   t=2.09
I=22   t=2.09
I=23   t=2.85
```

```
# Link definitions.
#
J=0     S=0    E=1    W=!ENTER      v=0  a=-1432.27  l=0.00
J=1     S=0    E=2    W=!ENTER      v=0  a=-1500.93  l=0.00
J=2     S=0    E=3    W=!ENTER      v=0  a=-3759.32  l=0.00
J=3     S=0    E=4    W=!ENTER      v=0  a=-3829.60  l=0.00
J=4     S=1    E=5    W=TO          v=3  a=-2434.05  l=-87.29
J=5     S=2    E=5    W=TO          v=1  a=-2431.55  l=-87.29
J=6     S=4    E=6    W=AND         v=3  a=-798.30   l=-69.71
J=7     S=4    E=7    W=IT          v=0  a=-791.79   l=-62.05
J=8     S=4    E=8    W=AND         v=2  a=-836.88   l=-69.71
J=9     S=3    E=9    W=BUT         v=0  a=-965.47   l=-51.14
J=10    S=4    E=10   W=A.          v=0  a=-783.36   l=-105.95
J=11    S=4    E=11   W=IN          v=0  a=-835.98   l=-49.01
J=12    S=4    E=12   W=A           v=0  a=-783.36   l=-59.66
J=13    S=4    E=13   W=AT          v=0  a=-923.59   l=-77.95
J=14    S=4    E=14   W=THE         v=0  a=-1326.40  l=-27.96
J=15    S=4    E=15   W=E.          v=0  a=-1321.67  l=-121.96
J=16    S=4    E=16   W=A           v=2  a=-1451.38  l=-59.66
J=17    S=4    E=17   W=THE         v=2  a=-1490.78  l=-27.96
J=18    S=4    E=18   W=IT          v=0  a=-1450.07  l=-62.05
J=19    S=5    E=18   W=IT          v=0  a=-1450.07  l=-110.42
J=20    S=6    E=18   W=IT          v=0  a=-775.76   l=-85.12
J=21    S=7    E=18   W=IT          v=0  a=-687.68   l=-125.32
J=22    S=8    E=18   W=IT          v=0  a=-687.68   l=-85.12
J=23    S=9    E=18   W=IT          v=0  a=-687.68   l=-50.28
J=24    S=10   E=18   W=IT          v=0  a=-689.67   l=-108.91
J=25    S=11   E=18   W=IT          v=0  a=-706.89   l=-113.78
J=26    S=12   E=18   W=IT          v=0  a=-689.67   l=-194.91
J=27    S=13   E=18   W=IT          v=0  a=-619.20   l=-100.24
J=28    S=4    E=19   W=IT          v=1  a=-1567.49  l=-62.05
J=29    S=14   E=20   W=DIDN'T      v=0  a=-4452.87  l=-195.48
J=30    S=15   E=20   W=DIDN'T      v=0  a=-4452.87  l=-118.62
J=31    S=16   E=20   W=DIDN'T      v=0  a=-4303.97  l=-189.88
J=32    S=17   E=20   W=DIDN'T      v=0  a=-4303.97  l=-195.48
J=33    S=18   E=20   W=DIDN'T      v=0  a=-4222.70  l=-78.74
J=34    S=19   E=20   W=DIDN'T      v=0  a=-4235.65  l=-78.74
J=35    S=20   E=21   W=ELABORATE   v=2  a=-5847.54  l=-62.72
J=36    S=20   E=22   W=ELABORATE   v=0  a=-5859.59  l=-62.72
J=37    S=21   E=23   W=!EXIT       v=0  a=-4651.00  l=-13.83
J=38    S=22   E=23   W=!EXIT       v=0  a=-4651.00  l=-13.83
```

Best-First Search

When using a structure such as a word lattice it is often necessary to find the best path through the lattice or generate the N-best alternatives.

The Viterbi style searches used previously are *breadth first* searches and only compare paths that have covered the same amount of input speech and extend all paths in parallel.

An alternative extends the (current) best hypothesis word by word until it reaches the end of the sentence. A set of paths is maintained in general of different lengths. The best path has been extended by one word and the resulting path(s) are then re-inserted into the path stack or heap in order of (estimated) likelihood. This normally involves comparing paths of different lengths and requires an approximate *look-ahead* computation so that paths of different lengths can be stored.

Best-first searches (which are often implemented via the A* algorithm) are very useful for word lattices since the scores in the lattice give an easy to compute look ahead value.

Lattice Accuracy

The quality of the lattices can be measured by finding the *minimum possible* error rate of the lattices with respect to the actual sentence spoken. The most accurate path through the lattice is found using a best first dynamic programming search that minimises the total number of word errors. When the correct sentence exists in the lattice, it will be found otherwise the total number of insertion, deletion and substitution errors will be calculated. The number of word errors and number of sentence errors can both be used as performance figures.

These error rates are often referred to as the *oracle* word and sentence error rates. If alternatives are stored as a list then simply the lowest error rate alternative gives the N-Best error rate.

If the lattice/list are to be successfully used as a constraint for applying further acoustic/language models, want the solution that a full search would have produced in the lattice to avoid search errors.

[Beware overly constrained lattices can under-estimate the word error rate if poorer models are applied or the data is more difficult e.g. noise is added and lattices are from clean data]

Lattice Pruning

The lattices generated can be very large (depends on the beamwidths and the number of tokens per state), but a significant proportion of each one will be relatively unlikely.

The idea of lattice pruning is to find the best possible path through the complete lattice and compare this to the most likely path through any lattice node/arc. If the lattice nodes / arcs have a poorer log likelihood by more than a threshold the arcs / nodes can be discarded. The likelihood of each arc can be found by best-first searches that start either from the start or end of the lattice:

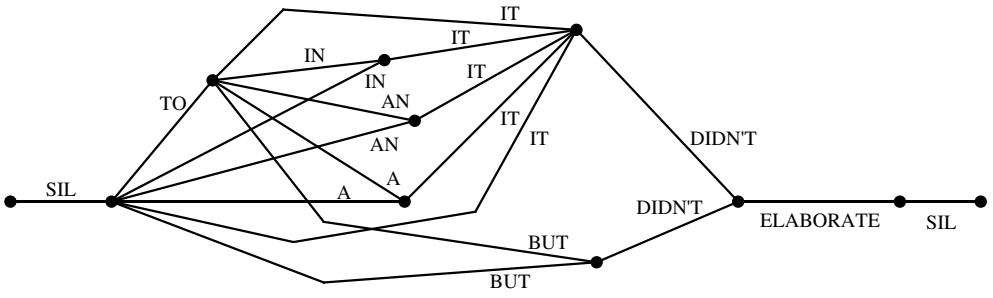
- The forward log likelihood of each node is found. This is the likelihood of the most likely path from the start of the lattice to the node.
- The backward log likelihood of each node is found. This is the likelihood of the most likely path from the end of the lattice to the node.
- The log likelihood of the most likely path through each arc is the sum of the arc log likelihoods added to the forward log likelihood of its start node and the backward log likelihood of its end node.

It is found that since entire utterance likelihoods are used in pruning, the beam-width can be quite tight without damaging lattice accuracy too much.

Acoustic Rescoring

A lattice can be used as a finite-state constraint grammar for acoustic rescoring with more complex models. For instance if the lattice is generated with word-internal context dependent models then the lattice could be used to represent a much reduced search space for cross-word context dependent models.

In this case the duplication of arcs to represent different times/phonetic contexts is not required and a much smaller lattice can be used: the network connectivity giving possible (likely) word sequences and language model scores are used.



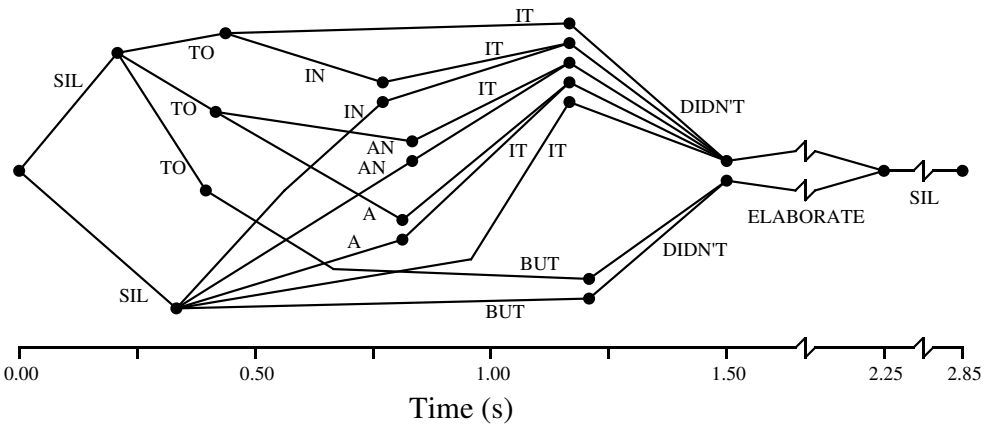
The use of word lattices in this way can be used for multi-pass decoding (also called progressive search) and is also very useful in system development since new acoustic models can be used with a greatly reduced (typically by an order of magnitude) less computation.

Applying Advanced LMs

The basic lattices can also be used to evaluate the effect of new language models without further acoustic processing (using the acoustic scores from the lattice).

A very useful operation is lattice expansion with a new language model. For example a lattice generated with a bigram model could be expanded to a trigram lattice (& is then pruned). This could be used for acoustic rescoring or for finding the best path subject to the new language model information (via a best-first search).

An example trigram lattice generated from the previous bigram lattice is shown below.



An alternative (e.g. if the entire sentence is required for LM probability calculation) is to generate an N-Best list of sentences and then use each sentence *separately* to evaluate a score according to the new language models.

N-Best paradigm

General N-Best approach is

1. Generate N-Best list of alternatives (best-first search from lattice)
2. Rescore each alternative hypotheses with K different knowledge sources and generate a log likelihood L_k for each of the N alternatives
3. Re-rank the alternatives by using a weighted sum of the different knowledge source scores

The optimal weights (to maximise accuracy) of the knowledge sources can be found by using some development data and a grid-search procedure.

This is a very powerful and general framework since very many different knowledge sources can be used that would be hard to integrate directly into the original search, although it requires having a high-quality N-best list to start.

Many different types of knowledge (apart from a range of acoustic and language models) can be used e.g. relative duration, prosodic scores etc.

Confidence Measures

In recent years, it has become increasingly important to supplement to 1-Best recognition output with a "confidence measure" that the word output is in fact correct.

These confidence measures can be used for a range of applications (e.g. in dialogue control, in unsupervised adaptation etc.)

Confidence measures can operate at different levels (e.g. phone, word) etc. Here we concentrate on word-based measures in an LVCSR system.

The "confidence" is usually defined as the probability that a word is correct: therefore the average confidence should equal the average word accuracy rate (ignoring deletions).

Typically there is a two-stage approach in generating such measures: feature selection and confidence generation.

The feature selection stage lists different measures that may help predict the word correctness.

Confidence Scores (cont)

Typical features used include:

- N-Best Homogeneity. This is the most widely used. For each word in the 1-Best list its N-best homogeneity score is the proportion of times that that word appears in that position in a sentence in a (long) N-Best list.
- Acoustic score of word (usually normalised)
- LM score
- Duration score
- Number of training examples of word

Once this set of measures (features) for each word has been determined they are mapped to a confidence score by e.g. a regression tree (asks questions about feature values to try and split words into correctness bands) or a generalised linear model, a neural network etc. This is done using some training data which has the words labelled as correct (or not) and the feature values known.

Note that the N-Best homogeneity feature works well by itself: this shows that recogniser "confusion" about the output word is a good measure of confidence.