# A WORD LATTICE PARSING ALGORITHM FOR NATURALLY SPOKEN ENGLISH

Russell J. Collingham and Roberto Garigliano

Artificial Intelligence Systems Research Group
School of Engineering and Computer Science
University of Durham, England

ABSTRACT A system is described which provides a prototype speech recognition aid for deaf students in university lectures. A dynamic programming algorithm builds a lattice of likely spoken words from the phoneme form of naturally spoken English. A mixture of breadth first search and best first search incorporating a novel set of "anti-grammar" rules to penalise grammatically incorrect or grammatically bad sequences of words is used to parse the word lattice and produce the best recognised word sequence.

## INTRODUCTION

This paper provides an introduction to the syntactic sub-system of AURAID: a speech recognition aid for use by deaf students in lectures. This sub-system produces a useful word recognition level from a continuous sequence of phonemes as could be provided by a continuous speech phoneme recognition system. Currently, this unit consists of a two stage process: dynamic programming and trace back (parsing). The dynamic programming stage matches a continuous sequence of corrupted phonemes with a dictionary to produce a word lattice. This stage also determines suitable points in the phoneme input that indicate a likely end of a word, this effectively breaks the word lattice into a series of chunks each spanning several spoken words. The trace back stage performs a reverse (and occasionally a forward) parse using a mixture of breadth first searching and best first searching incorporating a novel set of "anti-grammar" rules to penalise grammatically incorrect or grammatically bad sequences of words. AURAID has a vocabulary of 2200 words and currently works in real-time using a simulated continuous speech phoneme recognition system (modelled on the performance of the DRA (UK) Speech Research Unit's Armada system). The phoneme error rate provided by this simulation is approximately 26%. Word recognition rates of approximately 85% have been achieved on sections of the simulated data using unrestricted speech. The simulated data is taken from real university lectures on the subject of software engineering.

In order to make the recognition task simpler, and to achieve higher rates of recognition, many speech recognition systems have taken the approach of restricting what can be spoken, not only in terms of the *different* words that can be spoken but also the *order* that the words may be spoken. While this approach is appropriate for specific applications, for example interrogating a database where the number of queries is limited, imposing a restrictive grammar on what can be spoken is certainly not appropriate for the kind of system we are developing. Neither is it possible to define a complete grammar for spoken English (unrestricted, naturally). Further, the chunks of words present in the word lattice are not necessarily divided into complete sentences, but more likely, sentence fragments. We have taken the opposite approach by developing a set of rules that can be used to check the syntactic *incorrectness* of sequences of words. We call these syntactic rules an "anti-grammar" as the rules are used to penalise certain syntactic constructs rather than identify syntactically correct sequences. An anti-grammar rule, for example, may penalise the occurrence of two articles in succession, or the word "an" being followed by a word that doesn't start with a vowel.

This paper will discuss both the dynamic programming and trace back stages of AURAID and present the latest results of the current implementation.

## DATA PREPARATION

Four lectures from each of two lecture courses in different aspects of software engineering were recorded on to audio cassette. The text of these lectures was typed into a computer as accurately as possible to include partial words, "ums", "errs" and the like, and an indication of the location of short and long pauses. The phoneme representation for each word in each lecture was obtained from the *Oxford Advanced Learner's Dictionary*. The phoneme representation of each lecture was then corrupted in order to accurately reproduce the performance of a continuous phoneme recognition system by using real data figures obtained during the assessment of Armada (Browning *et al.*, 1990). These corrupted phoneme lecture files form the basis of the simulation.

## DYNAMIC PROGRAMMING

Dynamic programming is a mathematical concept that has been used for many years for multistage optimal decision calculation. In the field of speech recognition (where it is also know as dynamic time warping) it was used initially in isolated word recognition systems for comparison of segments of speech with stored word templates. This was extended to continuous word recognition by storing each template as a series of frames which were then compared to the segments of speech. A detailed description of dynamic programming for speech recognition can be found in (Silverman and Morgan, 1990). There are several dynamic programming algorithms in existence differing in memory use and execution speed. The most significant difference is between early decision (or one stage/level/pass) and deferred decision (or two stage/level/pass) algorithms. In a deferred decision dynamic programming algorithm, no output is forthcoming until the talker either pauses or finishes speaking. Early decision algorithms, however, produce their output as the talker is speaking. In theory, deferred decision algorithms should perform better than early decision algorithms as they have more information available. This, however, has not yet been proven.

By assuming a continuous stream of phonemes as its input, AURAID does not deal with frames or segments of the speech signal. However, dynamic programming can be used to match stored template words, made up of a series of phonemes, with the input phonemes. From the above description of the available dynamic programming algorithms, it can be seen that AURAID requires a fast, early decision algorithm.

Each dictionary word is composed of one or more phonemes. A typical dynamic programming algorithm uses a matching routine to measure the distance or dissimilarity between the $p$'th phoneme of the $w$'th word and the $t$'th input phoneme. The dynamic programming score $S(w,p,t)$ is defined for any word phoneme and input phoneme as the sum of the phoneme distances for the best way of aligning the first $t$ phonemes of the input with any possible sequence of words followed by the first $p$ phonemes of the $w$'th word, in other words, the best path to $(w,p,t)$. Thus, the score is not only a function of the position in the input and also of the position in a word, but also depends on the other words and the way they might explain parts of the input. Transitions between words are always from the end of one word to the beginning of other words. Thus the score for the start of each word depends on the score of the end phoneme of the best word of the previous input phoneme. The distance or similarity score between phonemes can depend of a variety of things, and varies from algorithm to algorithm. Most algorithms group phonemes into classes according to their confusability. The phoneme classes used by AURAID are based on manner of articulation and are shown in Table 1. The distance between phonemes within the same class is then less than that between phonemes from different classes. This can be measured, for example, by absolute values or logarithms of the probability of confusing one phoneme for another based on experimental data.

The Bridle dynamic programming algorithm (Bridle *et al.*, 1983) uses the following equations:

$$S(w,1,t) = dist(w,1,t) + \min_{r \in R(w)} \{S(r,N(r),t-1)\} \tag{1}$$

$$S(w,p,t) = \min_{a=0,1,2} \{S(w,p-a,t-1) + P(a) + dist(w,p,t)\} \tag{2}$$

where $R(w)$ is the set of words that are allowed to precede the $w$'th word, $N(r)$ is the length in phonemes of the $r$'th word, and $P(a)$ is a time distortion penalty that allows, for example, phonemes to be repeated or single phonemes to be skipped. In this case $R(w)$ will be the whole dictionary as AURAID does not use a grammar to restrict word order.

| *Class* | *Name* | *Phonemes* |
|---|---|---|
| 0 | Plosive | p b t d k g |
| 1 | Affricative | tS dZ |
| 2 | Strong Fricative | s z S Z |
| 3 | Weak Fricative | f v T D h |
| 4 | Liquid/Glide | l r w j |
| 5 | Nasal | n m N |
| 6 | Vowel | i I E { A Q O U u 3 V @ aI eI oI aU @U I@ e@ U@ |

Table 1: Phoneme Classes used by AURAID

The Ney dynamic programming algorithm (Ney, 1984) uses the following equations:

$$S(w,1,t) = dist(w,1,t) + \min\{S(w,1,t-1);\quad S(r,N(r),t-1) : r \in R(w)\} \tag{3}$$

$$\begin{array}{rcccl} S(w,p,t) = \min\{\ 0.5 & \cdot & dist(w,p,t) & + & S(w,p-1,t); \\ 2 & \cdot & dist(w,p,t) & + & S(w,p,t-1); \\ & & dist(w,p,t) & + & S(w,p-1,t-1); \end{array} \tag{4}$$

These two dynamic programming algorithms do not perform as well as our own algorithm. They fall down because they do not attempt to model the kinds of errors that actually occur. In the algorithm that we use, we explicitly model the kinds of errors that may occur, both within words and between words. That is, we consider inserted phonemes, deleted phonemes, substituted phonemes and word final phoneme deletion. We also found that long words are unduly penalised because of their length and are not recognised as well as they should be. To overcome this inadequacy we normalise the distance scores according to the length of the word being considered. The equations for our dynamic programming algorithm are:

$$\begin{array}{rcccl} S(w,1,t) = \min\{\dfrac{ins\_pen}{L(w)} & + & \dfrac{sub\_pen(w,1,t)}{L(w)} & + & \min\limits_{r\in R(w)}\{S(r,N(r),t-2)\}; \\ & & \dfrac{sub\_pen(w,1,t)}{L(w)} & + & \min\limits_{r\in R(w)}\{S(r,N(r),t-1)\}; \\ \dfrac{del\_pen}{L(w)} & + & \dfrac{sub\_pen(w,1,t)}{L(w)} & + & \min\limits_{r\in R(w)}\{S(r,N(r)-1,t-1)\}\} \end{array} \tag{5}$$

$$\begin{array}{rcccl} S(w,p,t) = \min\{\dfrac{ins\_pen}{L(w)} & + & \dfrac{sub\_pen(w,p,t)}{L(w)} & + & S(w,p-1,t-2); \\ & & \dfrac{sub\_pen(w,p,t)}{L(w)} & + & S(w,p-1,t-1); \\ \dfrac{del\_pen}{L(w)} & + & \dfrac{sub\_pen(w,p,t)}{L(w)} & + & S(w,p-2,t-1)\} \end{array} \tag{6}$$

$$where \qquad L(w) = \begin{cases} 3 & \text{if } (phoneme\_length(w) = 1) \\ 4 & \text{if } (phoneme\_length(w) = 2) \\ phoneme\_length(w) & \text{otherwise} \end{cases}$$

The three penalties, *ins_pen*, *del_pen* and *sub_pen* each return absolute values independent of the particular phoneme being considered, with the exception that the *sub_pen* for two phonemes of the same class is less than that for two phonemes of different classes. It was found that this approach to phoneme distance calculation produced better results than using logarithms of the probability of confusing one phoneme for another.

In both Equations 5 and 6 it can be seen that a minimum score choice is to be taken between either the last input phoneme being an insertion error, the current input phoneme being correct or a substitution error, or the previous phoneme of the current word being matched (or the final phoneme of the previous best scoring word) being deleted. This truly represents the events that could possibly occur.

The three dynamic programming algorithms outlined above have been evaluated against each other. The word lattices produced by executing each algorithm on a small portion of each of the corrupted lecture files have been compared to determine the best algorithm. For each input phoneme a word lattice contains a list of the best scoring words that end at that point. This list could have an absolute length, for example the 25 top scoring words ending at that particular point in the input, or it could be determined by the closeness of the word scores, for example a list made up of all words having a score within 10% of the top scoring word at that particular point. The ranks of each of the real spoken words was determined at the point where they should have been recognised in each word lattice produced by each algorithm on each lecture file. The algorithms were then evaluated against each other by determining the percentage of real spoken words within the, say, top ten candidates of their particular word lists. The results of this evaluation are summarised in Table 2.

| *Dynamic Programming Algorithm* | *Lecture 1* *57 words* | | | *Lecture 2* *72 words* | | | *Lecture 3* *58 words* | | | *Lecture 4* *52 words* | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *% ranks within top* | | | *% ranks within top* | | | *% ranks within top* | | | *% ranks within top* | | |
| | *10* | *20* | *70* | *10* | *20* | *70* | *10* | *20* | *70* | *10* | *20* | *70* |
| AURAID | 91.2 | 98.2 | 100.0 | 90.3 | 95.8 | 100.0 | 93.1 | 96.6 | 100.0 | 88.5 | 96.2 | 96.2 |
| Ney | 91.2 | 93.0 | 96.5 | 87.5 | 90.3 | 95.8 | 86.2 | 89.7 | 93.1 | 92.3 | 92.3 | 96.2 |
| Bridle | 56.1 | 71.9 | 78.9 | 56.9 | 73.6 | 76.4 | 56.9 | 72.4 | 86.2 | 51.9 | 65.4 | 76.9 |

Table 2: A Comparison of Three Dynamic Programming Algorithms

## TRACE BACK (PARSING)

The word lattice produced by the dynamic programming stage needs to be broken into chunks of manageable size for the traceback stage. Each chunk must finish at the end of a word. At certain points during the processing, ends of words can be identified, either by pauses in the speech, or by "common consent" of best words at different input phonemes. This latter category of word ends can be determined by keeping a tally (or vote) of the previous word ends suggested by the best word at each input phoneme. When this vote count reaches a specified level it can be deduced, with a certain amount of accuracy, that a definite end of word exists. At this point a trace back over *previous* word ends beginning at that point can take place either to the start of the recognition, or to the previous trace back, and the memory used to store this information freed.

A standard trace back often experiences difficulty because of the large phoneme error rate present. Votes by sequences of short words can often incorrectly identify the location of word ends, we therefore ignore those votes for word ends that are close by (i.e. neighbours). Using a standard trace back algorithm from (Bridle *et al.*, 1983) or (Ney, 1984) on the word lattices produced even by AURAID's dynamic programming algorithm on an example corrupted phoneme sequence would produce the following word sequence as output, a word recognition rate of approximately 57%. This figure is lower if one of the other two dynamic programming algorithms is used to generate the word lattices.

```
that just cow to suite the courses in case europe confused course on software main to a suite
maintainability every much i'm to say very much british subject air eye briefly the syllabus
is add falls eye may or may not stick exactly it this

this lecture is cow go arm boy be an introductory scenario
```

The correct transcription is as follows.

```
that just to say what the course is in case you're confused course on software maintenance
we've got nine lectures it's not very much time to say very much about this subject very
briefly the syllabus is as follows i may or may not stick exactly to this
```

```
this lecture is going to be an introductory scenario
```

As was said in the introduction, many speech recognition systems restrict what may be spoken by use of a grammar. We have taken the opposite approach by developing a set of rules that can be used to check the syntactic *incorrectness* of sequences of words. We call these syntactic rules an "anti-grammar" as the rules are used to penalise certain syntactic constructs rather than identify syntactically correct sequences. Currently, the anti-grammar is made up of four parts.

- simple rules concerning sequences of particular syntactic categories, for example:

  `ADJECTIVE ARTICLE ADJECTIVE`

- more complicated rules concerning sequences of syntactic categories in addition to particular forms of words, for example: `ARTICLE VERB(not 'ing' form)`
- rules concerning words that behave in a strange manner, for example, `not` and `very`
- common constructs of spoken English, for example, `to VERB` or `very ADJ` and common words are given an advantage

If a particular chunk of words from the word lattice is of less than ten phonemes in length, parsing is delayed until the next chunk of words is received. The two chunks are then concatenated before parsing. A chunk of less than ten phonemes is only likely to contain one, two or maybe three words and is too short for the anti-grammar to have an effect. The parsing takes place in reverse, that is, from the end of a chunk of words in the word lattice, to the beginning. This is for speed and simplicity. As the dynamic programming stage determines which words are likely to end at particular phonemes, a data structure is constructed that contains a list of phonemes, each phoneme having associated with it a list of words that are thought to end at that point in the input. This means that it is easier to work from the end of a chunk of words.

The search algorithm used to parse a sentence is a mixture of breadth first and best first methods. Initially a breadth first search is used, this is to avoid early commitment along an incorrect parse path. Then a best first search takes over. This part of the search involves evaluating an underestimate of the penalty likely to be incurred between the current point of each path and the goal. This step is one of the two improvements that distinguish a normal best first search from an A* search. The other step involves removing all but the best scoring path that reach the same point in a parse. However, an A* search would not be appropriate during our trace back because the anti-grammar may reduce as well as increase the scores of alternative paths through the word lattice chunk, a path cannot be removed as the next word to be parsed may reduce that path's score, making it the current best path. If the last word (or words) of a chunk of words is not very recognisable the parser may spend unnecessary time looking for the correct words at the end of the chunk before proceeding to find the words at the beginning of the chunk. If this occurs, the parse algorithm switches to a *forward* combination of breadth first and best first searches. The time involved in constructing a new data structure, that associates a list of phonemes with the list of words that *start* at that point, is less than the time spent unsuccessfully searching for words at the end of the chunk. It is also worth using the technique for the greater accuracy of recognition that is produced.

The parsing algorithm also handles phonemes that have been inserted into and deleted from the input by allowing a particular path to ignore (skip) a phoneme, or allowing a phoneme to be "shared" by two different words (co-articulation). This would incur a small penalty. In the first example below, a phoneme has been incorrectly inserted between the two words, and in the second example a phoneme has been incorrectly deleted between the two words.

```
just          to                                    just   to
dZ V s t   k  t @                                   dZ V s t @
```

In the first example, the word lattice would contain the word just spanning the first set of phonemes, and the word to spanning the last set of phonemes, and probably a word like stick spanning the "s t k" phonemes. This is handled in AURAID by allowing the parsing algorithm to skip over the inserted phoneme. In the second example, the word lattice would contain the word just spanning the "dZ V s t" phonemes, and the word to spanning the "t @" phonemes. This is handled in AURAID by allowing the parsing algorithm to parse the "t" phoneme twice, enabling both words to span it.

The output produced by the current version of the trace back mechanism of AURAID is reproduced below. This gives a word recognition rate of approximately 85%. Incorrect words and deletions are marked with an asterisk.

```
that just to say what the course is in case you are confused course on software maintenance
why* got in* by* no* lectures it's not very much I'm* to say very much but* is* subject very
briefly the syllabus is as flows* i may or me* not stick exactly * this

this lecture is going to be an introductory scenario
```

## CONCLUSIONS

This research has produced an efficient one-pass dynamic programming algorithm for use in recognising words from a continuous stream of corrupted phonemes. This algorithm produces several word lattices that span the spoken input. The trace back (parsing) stage makes use of a set of anti-grammar rules to effectively parse these word lattices of naturally spoken English. This anti-grammar does not define the form of legal sentences in the usual sense of grammar but penalises grammatically incorrect or grammatically bad sequences of words.

The next stage of research is to analyse a large corpus of annotated (i.e. words are tagged with their syntactic categories) speech and try to determine empirically a set of reliable anti-grammar rules. We will then incorporate a semantic stage during the trace back, which would use knowledge about the topic of the lecture to distinguish between certain paths in the word lattice. This semantic stage would also make use of cliches specific to the topic of the lecture. The system will soon move on from being a simulation to being a prototype system when we use a real continuous phoneme recognition system.

## ACKNOWLEDGEMENTS

This research was funded for two years by a Science and Engineering Research Council (UK) studentship. The Royal National Institute for the Deaf also supported this research with a grant. A Special Equipment Grant was provided by the University of Durham and financial assistance with the purchase of the transputers used in this research was provided by Transtech Devices Ltd. The research is currently funded by a grant from the Leverhulme Trust.

## REFERENCES

Bridle, J. S., Brown, M. D. & Chamberlain, R. M. (1983), *Continuous Connected Word Recognition using Whole Word Templates*, The Radio and Electronic Engineer, 53(4), pages 167–175.

Browning, S. R., Moore, R. K., Ponting, K. M. & Russell, M. J. (1990), *A Phonetically Motivated Analysis of the Performance of the ARM Continuous Speech Recognition System*, Proceedings of the Institute of Acoustics Autumn Conference, Windermere.

Ney, H. (1984), *The Use of a One-Stage Dynamic Programming Algorithm for Connected Word Recognition*, IEEE Transactions on Acoustics, Speech, and Signal Processing, 32(2), pages 336–340.

Silverman, H. F. & Morgan, D. P. (1990), *The Application of Dynamic Programming to Connected Speech Recognition*, IEEE ASSP Magazine, pages 6–25, July.