Efficient, High-Performance Algorithms for N-Best Search

Richard Schwartz, Steve Austin

BBN Systems and Technologies Inc.
10 Moulton St.
Cambridge, MA, 02138

Abstract

We present two efficient search algorithms for real-time spoken language systems. The first called the Word-Dependent N-Best algorithm is an improved algorithm for finding the top N sentence hypotheses. The new algorithm is shown to perform as well as the Exact Sentence-Dependent algorithm presented previously but with an order of magnitude less computation. The second algorithm is a fast match scheme for continuous speech recognition called the Forward-Backward Search. This algorithm, which is directly motivated by the Baum-Welch Forward-Backward training algorithm, has been shown to reduce the computation of a time-synchronous beam search by a factor of 40 with no additional search errors.

1. Introduction

In a Spoken Language System (SLS) we must use all available knowledge sources (KSs) to decide on the spoken sentence. While there are many knowledge sources, they are often grouped together into speech models, statistical language model, and natural language understanding models. To optimize accuracy we must choose the sentence that has the highest score (probability) given all of the KSs. This potentially requires a very large search space. The N-Best paradigm for integrating several diverse KSs has been described previously [2, 10] . First, we use a subset of the KSs to choose a small number of likely sentences. Then these sentences are scored using the remainder of the KSs.

In Chow et. al., we also presented an efficient speech recognition search algorithm that was capable of computing the N most likely sentence hypotheses for an utterance, given the speech models and statistical language models. However, this algorithm greatly increases the needed computation over that needed for finding the best single sentence. In this paper we introduce two techniques that dramatically decrease the computation needed for the N-Best search. These algorithms are being used in a real-time SLS [1]. In the remainder of the introduction we review the exact N-Best search briefly and describe its problems. In Section 2 we describe two approximations to the exact algorithm and compare their accuracy with that of the exact algorithm.

The resulting algorithm is still not fast enough for real-time implementation. In Section 3 we present a new sentence-level fast match scheme for continuous speech recognition. The algorithm is motivated by the mathematics of the Baum-Welch Forward-Backward training algorithm.

The N-Best Paradigm

The basic notion of the n-best paradigm is that, while we must ultimately use all the available KSs to improve recognition accuracy, the sources vary greatly in terms of perplexity reduction and required complexity. For example, a first-order statistical language model can reduce perplexity by at least a factor of 10 with little computation, while applying complete natural language (NL) models of syntax and semantics to all partial hypotheses typically requires more computation for less perplexity reduction. (Murveit [6] has shown that the use of an efficiently implemented syntax within a recognition search actually slowed down the search unless it was used very sparingly.) Therefore it is advantageous to use a strategy in which we use the most powerful, efficient KSs first to produce a scored list of all the likely sentences. This list is then filtered and reordered using the remaining KSs to arrive at the best single sentence. Figure 1 contains a block diagram that illustrates this basic idea. In addition to reducing total computation the resulting systems would be more modular if we could separate radically different KSs.

The Exact Sentence-Dependent Algorithm

We have previously presented an efficient time-synchronous algorithm for finding the N most likely sentence hypotheses. This algorithm was unique in that it computed the correct forward probability score for each hypothesis found. The way this is accomplished is that, at each state, we keep an independent score for each different preceding sequence of words. That is, the scores for two theories are added only if the preceding word sequences are identical. We preserve up to N different theories at each state, as long as they are above the pruning beamwidth. This algorithm guarantees finding the N best hypotheses within a threshold of the best hypothesis. The algorithm was optimized to avoid expensive sorting operations so that it required computation that was less than linear with the number of sentence hypotheses found. It is easy to show that the inaccuracy in the scores computed is bounded by the product of the sentence length

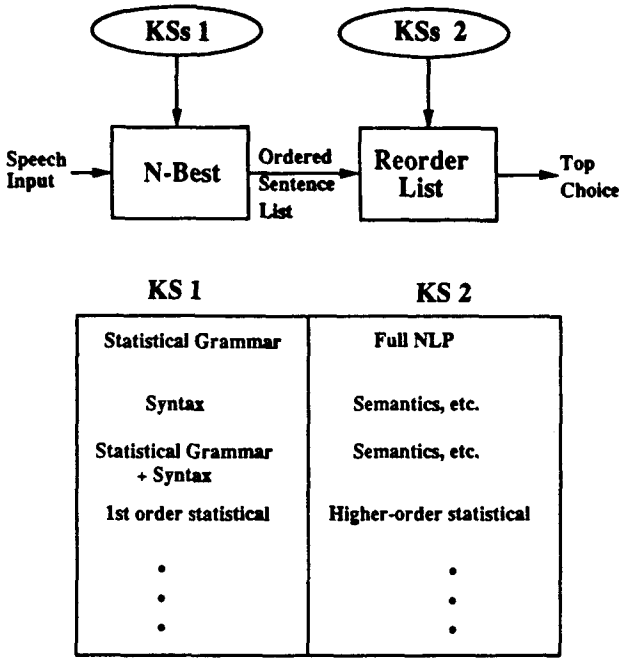

Figure 1: The N-best Search Paradigm. The most efficient knowledge sources, KS1, are used to find the N Best sentences. Then the remaining knowledge sources, KS2 are used to reorder the sentences and pick the most likely one.

and the pruning beamwidth. For example, if a sentence is 1000 frames long and a relative pruning beamwidth of 10^{-15} is maintained throughout the sentence, then all scores are guaranteed to be accurate to within 10^{-12} of the maximum score. The proof is not given here, since it is not the subject of this paper. In the remainder of the paper we will refer to this particular algorithm as the Exact algorithm or the Sentence-Dependent algorithm.

There is a problem associated with the use of this exact algorithm. If we assume that the probability of a single word being misrecognized is roughly independent of the position within a sentence, then we would expect that a longer sentence will have more errors. Consequently the typical rank of the correct answer will be lower (further from the top) on longer sentences. Therefore if we wanted the algorithm to find the correct answer within the list of hypotheses some fixed percentage of the time, the value of N will have to increase significantly for longer sentences.

When we examine the different answers found we notice that, many of the different answers are simple one-word variations of each other. This is likely to result in much duplicated computation. One might imagine that if the difference between two hypothesized word sequences were several words in the past then any difference in score due to that past word would remain constant. In the next section we present two algorithms that attempt to avoid these problems.

2. Two Approximate N-Best Algorithms

While the exact N-Best algorithm is theoretically interesting, we can generate lists of sentences with much less computation if we are willing to allow for some approximations. As long as the correct sentence can be guaranteed to be within the list, the list can always be reordered by rescoring each hypothesis individually at the end. We present two such approximate algorithms with reduced computation.

Lattice N-Best

The first algorithm will derive an approximate list of the N Best sentences with no more computation than the usual 1-Best search. Figure 2 illustrates the algorithm. Within words we use the time-synchronous forward-pass search algorithm [8], with only one theory at each state. We add the probabilities of all paths that come to each state. At each grammar node (for each frame) we simply store all of the theories that arrive at that node along with their respective scores in a traceback list. This requires no extra computation above the 1-Best algorithm. The score for the best hypothesis at the grammar node is sent on as in the normal time-synchronous forward-pass search. A pointer to the saved list is also sent on. At the end of the sentence we simply search (recursively) through the saved traceback lists for all of the complete sentence hypotheses that are above some threshold below the best theory. This recursive traceback can be performed very quickly. (We typically extract the 100 best answers, which causes no noticeable delay.) We call this algorithm the Lattice N-Best algorithm since we essentially have a dense word lattice represented by the traceback information. Another advantage of this algorithm is that it naturally produces more answers for longer sentences.

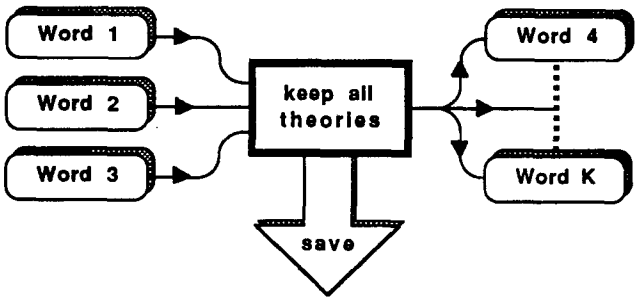

Figure 2: The Lattice N-best Algorithm. We save all theories at grammar nodes. Then we recursively trace back all sequences.

This algorithm is similar to the one suggested by Steinbiss [9], with a few differences. First, he uses the standard Viterbi algorithm rather than the time-synchronous algorithm within words. That is he takes the maximum of the path probabilities at a state rather than the sum. We have

observed a 20% higher error rate when using the maximum rather than the sum. The second difference is that when several word hypotheses come together at a common grammar node at the same time, he traces back each of the choices and keeps the N (typically 10) best sentence hypotheses up to that time and node. This step unnecessarily limits the number of sentence hypotheses that are produced to N. As above the score of the best hypothesis is sent on to all words following the grammar node. At the end of the sentence he then has an approximation to the N best sentences. He reports that one third of the errors made by the 1-Best search are corrected in this way. However, as with a word lattice, many of the words are constrained to end at the same time - which leads to the main problem with this algorithm.

The Lattice N-Best algorithm, while very fast, underestimates or misses high scoring hypotheses. Figure 3 shows an example in which two different words (words 1 and 2) can each be followed by the same word (word 3). Since there is only one theory at each state within a word, there is only one best beginning time. This best beginning time is determined by the best boundary between the best previous word (word 2 in the example) and the current word. But, as shown in Figure 3, the second-best theory involving a different previous word (word 1 in the example), would naturally end at a slightly different time. Thus the best score for the second-best theory would be severely underestimated or lost altogether.

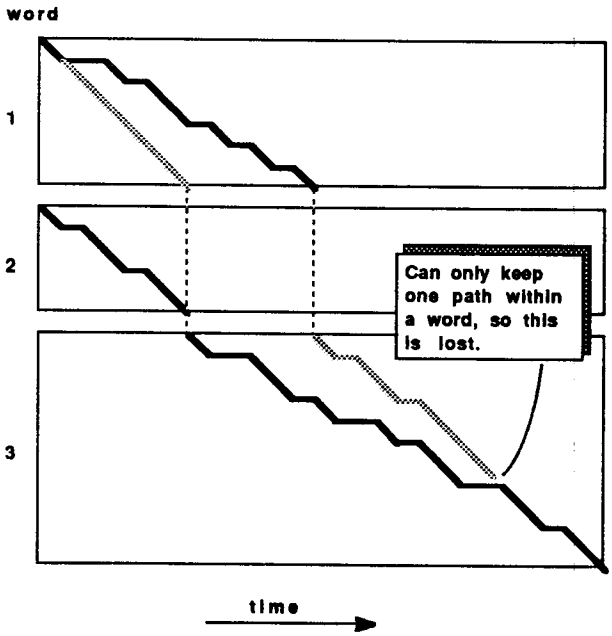

Figure 3: Alternate paths in the Lattice algorithm. The best path for words 2-3 overrides the best path for words 1-3.

Word-Dependent N-Best

As a compromise between the exact sentence-dependent algorithm and the lattice algorithm we devised a Word-Dependent N-Best algorithm. We reason that while the best starting time for a word does depend on the preceding word, it probably does not depend on any word before that. Therefore instead of separating theories based on the whole preceding sequence, we separate them only if previous word is different. At each state within the word we preserve the total probability for each of $n(<< N)$ different preceding words. At the end of each word we record the score for each hypothesis along with the name of the previous word. Then we proceed on with a single theory with the name of the word that just ended. At the end of the sentence we perform a recursive traceback to derive a large list of the most likely sentences. The resulting theory paths are illustrated schematically in Figure 4. Like the lattice algorithm the

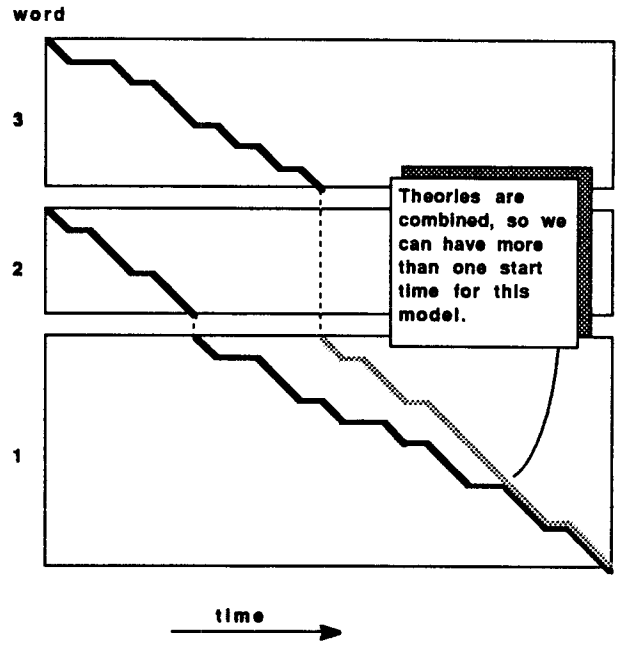

Figure 4: Alternate paths in the Word-Dependent algorithm. Best path for words 1-3 is preserved along with path for words 2-3.

word-dependent algorithm naturally produces more answers for longer sentences. However, since we keep multiple theories within the word, we correctly identify the second best path. While the computation needed is greater than for the lattice algorithm it is less than for the sentence-dependent algorithm, since the number of theories only needs to account for number of possible previous words - not all possible preceding sequences. Therefore the number n, of theories kept locally only needs to be 3 to 6 instead of 20 to 100.

Comparison of N-Best Algorithms

We performed experiments to compare the behavior of the three N-Best algorithms. In all three cases we used the Class Grammar [3], a first-order statistical grammar based on 100 word classes. All words within a class are assumed equally likely. The test set perplexity is approximately 100. The test set used was the June '88 speaker-dependent test set of 300 sentences. To enable direct comparison with previous results we did not use models of triphones across word boundaries, and the models were not smoothed. We expect all three algorithms to improve significantly when the latest modeling methods are used.

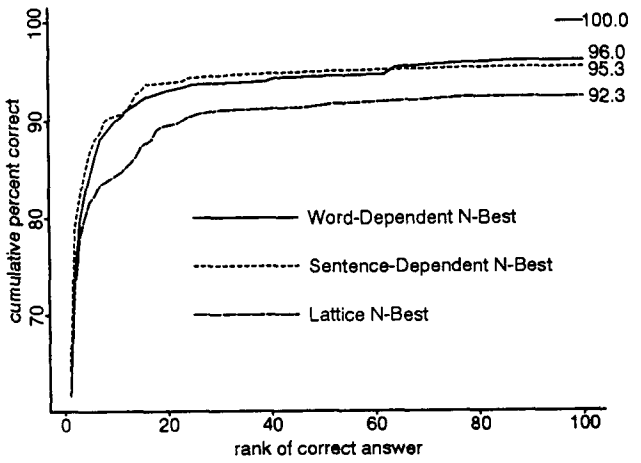

Figure 5: Comparison of the Rank of the Correct Sentence for the Sentence-Dependent, Word-Dependent, and Lattice N-Best Algorithms.

Figure 5 shows the cumulative distribution of the rank of the correct answer for the three algorithms. As can be seen, all three algorithms get the sentence correct on the first choice about 62% of the time. All three cumulative distributions increase substantially with more choices. However, we observe that the Word-Dependent algorithm yields accuracies quite close to that of the Exact Sentence-Dependent algorithm, while the Lattice N-Best is substantially worse. In particular, the sentence error rate at rank 100 (8%) is double that of the Word-Dependent algorithm (4%). Therefore, if we can afford the computation of the Word-Dependent algorithm it is clearly preferred.

We also observe in Figure 5 that the Word-Dependent algorithm is actually better than the Sentence-Dependent algorithm for very high ranks. This is because the score of the correct word sequence fell outside the pruning beamwidth. However, in the Word-Dependent algorithm each hypothesis gets the benefit of the best theory two words back. Therefore the correct answer was preserved in the traceback. This is another advantage that both of the approximate algorithms have over the Sentence-Dependent algorithm.

In the next section we describe a technique that can be used to speed up all of these time-synchronous search algorithms by a large factor.

3. Forward-Backward Search

The time-synchronous beam search follows a large number of theories on the off chance that they will get better during the remainder of the sentence. Typically, we must keep over 1000 theories to guarantee finding the highest answer. In some sense the computation for all but one answer will have been wasted.

We need a way to speed up the beam search without causing search errors. We could prune out most of the choices if we only knew the correct answer ahead of time or if we could look ahead at the remainder of the sentence. Several papers have described fast match schemes that look ahead (incurring a delay) to determine which words are likely (e.g. [4]). The basic idea is to perform some approximate match that can be used to eliminate most of the possible following words. However, since we cannot tell when words end in continuous speech, the predictions of the score for each word is quite approximate. In addition, even if a word matches well we cannot tell whether the remainder of the sentence will be consistent with that word without looking further ahead and incurring a longer delay.

Let us consider the time-synchronous forward pass. The score at any given state and time $\alpha_t(s)$ is the probability of the input up to time t, summed over all of the paths that get to state s at t. When these scores are normalized they give the relative probability of paths ending at this state as opposed to paths ending at any other state. These forward pass probabilities are the ideal measure to predict which theories in a backward search are expected to score well. Figure 6 illustrates several paths from the beginning of an utterance to different states at time t, and several theories from the end of the utterance T backward to time t. From the Baum-Welch Forward-Backward training algorithm we have

$$\gamma_t(s) = \frac{\alpha_t(s)\beta_t(s)}{\alpha_T}$$

where $\gamma_t(s)$ is the probability of the data given all paths through state s, divided by the probability of the data for all paths, which is the probability that state s is appropriate at time t. α_T is derived from the forward pass. Of course if we have already gone through the whole utterance in the forward direction we already know the most likely sentence.

Now let us consider a practical Forward-Backward Search algorithm. First we perform a forward pass over the whole utterance using a simplified acoustics or language model. In each frame we save the highest forward probability and the probabilities of all words that have ending scores above the pruning beamwidth. Typically this includes about 20 words in each frame. Then we perform a search in the backward direction. This search uses the normal beam search within words. However, whenever a score is about to be transfered backwards through the language model into the end of a word we first check whether that word had an ending score for that frame in the forward pass. That is we ask, "Was there a reasonable path from the beginning of the utterance to this time ending with this word?" Again, referring to Figure 6, the backward theory that is looking for word

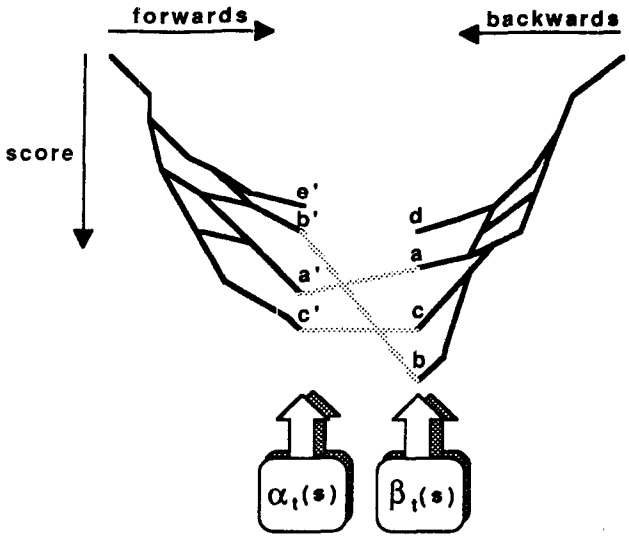

Figure 6: Forward-Backward Search. Forward and backward scores for the same state and time are added to predict final score for each theory extension.

d cannot find any corresponding forward score, and so is aborted. When there is a score, as in the cases for words a,b,c, then we multiply the present backward score of the theory, $\beta_t(s)$ by the forward pass score for this word, $\alpha_t(s)$, divided by the whole sentence score, α_T. Only if this ratio is greater than the pruning beamwidth do we extend the theory backwards by this word. For example, although the backward theory looking for word c has a good score, the corresponding forward score c' is not good, and the product may be pruned out.

The Forward-Backward search is only useful if the forward pass is faster than the backward would have been. This can be true if we use a different grammar, or a less expensive acoustic model. If the forward acoustic models or language model is different than in the backward pass, then we must reestimate α_T before using it in the algorithm above. For simplicity we estimate α_T at each time t as

$$\alpha_T(t) = \max_s \alpha_t(s) \max_s \beta_t(s)$$

the product of the maximum state scores in each direction. (Note that since the two maxima are not necessarily on the same state it would be more accurate to use

$$\alpha_T(t) = \max_s \alpha_t(s)\beta_t(s)$$

forcing the two states to be the same. However, since most of the active states are internal to words, this would require a large computation and also require that we had stored all of the state scores in the forward direction for every time.)

We observe that the average number of active phoneme arcs in the backward direction is reduced by a factor of 40 (e.g. from 800 to 20) - with a corresonding reduction in computation and with no increase in search errors.

Uses of Forward-Backward Search

As stated above, this algorithm is only useful when the forward pass can be computed differently (much more quickly) than the backward (real) search. For example, we could use a null grammar in the forward direction and a more complex grammar in the backward search. We have used this extensively in our past work with very large RTN grammars or high-order statistical grammars [7]. When no grammar is used in the forward pass we can compact the entire dictionary into a phonetic tree, thereby greatly reducing the computation for large dictionaries.

A variation on the above use is to use a simpler acoustic model in the forward direction. For example restricting the model to triphones within words, using simpler HMM topologies, etc.

A second use is for real-time computation of the N Best sentences [1]. First we perform a normal 1-Best search forward. The best answer can be processed by NL immediately (on another processor) while we perform the N-Best search backwards. We find that the backward N-Best search is sped up by a factor of 40 when using the forward pass scores for pruning. Thus the delay until we have the remainder of the answers is usually quite short. If the delay is less than the time required to process the first answer through NL, then we have lost no time.

Finally, we can use the Forward-Backward Search to greatly reduce the time needed for experiments. Experiments involving expensive decoding conditions can be reduced from days to hours. For example all of the experiments with the Word-Dependent and Lattice N-Best algorithms were performed using the Forward-Backward Search.

4. Conclusion

We have considered several approximations to the exact Sentence-Dependent N-Best algorithm, and evaluated them thoroughly. We show that an approximation that only separates theories when the previous words are different allows a significant reduction in computation, makes the algorithm scalable to long sentences and less susceptable to pruning errors, and does not increase the search errors measurably. In contrast, the Lattice N-Best algorithm, which is still less expensive, appears to miss twice as many sentences within the N-Best choices.

We have introduced a new two-pass search strategy called the Forward-Backward Search, which is generally applicable to a wide range of problems. This strategy increases the speed of the recognition search by a factor of 40 with no additional pruning errors observed.

Acknowledgement

This work was supported by the Defense Advanced Research Projects Agency and monitored by the Office of Naval Research under Contract No. N00014-89-C-0008.

References

[1] Austin, S., Peterson, P., Placeway, P., Schwartz, R, and Vandergrift, J., "Toward a Real-Time Commercial System Using Commercial Hardware". *Proceedings of the DARPA Speech and Natural Language Workshop* Hidden Valley, June 1990 (1990).

[2] Chow, Y-L. and Schwartz, R.M., "The N-Best Algorithm: An Efficient Procedure for Finding Top N Sentence Hypotheses". *Proceedings of the DARPA Speech and Natural Language Workshop* Cape Cod, October 1989 (1989).

[3] Derr, A., and Schwartz, R.M., "A Simple Statistical Class Grammar for Measuring Speech Recognition Performance". *Proceedings of the DARPA Speech and Natural Language Workshop* Cape Cod, October 1989 (1989).

[4] Bahl, L.R., de Souza, P., Gopalakrishnan, P.S., Kanevsky, D., and Nahamoo, D. "Constructing Groups of Acoustically Confusable Words". *Proceedings of the ICASSP 90*, April, 1990.

[5] Fissore, L., Micca, G., and Pieraccini, R., "Very Large Vocabulary Isolated Utterance Recognition: A Comparison Between One Pass and Two Pass Strategies". *Proceedings of the ICASSP 88*, pp. 267-270, April, 1988.

[6] Murveit, H., "Integrating Natural Language Constraints into HMM-based Speech Recognition". *Proceedings of the ICASSP 90*, April, 1990.

[7] Rohlicek, J.A., Chow, Y-L., and Roucos, S., "Statistical Language Modeling Using a Small Corpus from an Application Domain". *Proceedings of the DARPA Speech and Natural Language Workshop* Cambridge, October 1987 (1987). Also in *Proceedings of the ICASSP 88*, pp. 267-270, April, 1988.

[8] Schwartz, R.M., Chow, Y., Kimball, O., Roucos, S., Krasner, M., and Makhoul, J. "Context-Dependent Modeling for Acoustic-Phonetic Recognition of Continuous Speech". *Proceedings of the ICASSP 85*, pp. 1205-1208, March, 1985.

[9] V. Steinbiss (1989) "Sentence-Hypotheses Generation in a Continuous-Speech Recognition System," *Proc. of the European Conf. on Speech Communciation and Technology, Paris, Sept. 1989, Vol. 2, pp. 51-54*

[10] Young, S. (1984) "Generating Multiple Solutions from Connected Word DP Recognition Algorithms". *Proc. of the Institute of Acoustics, 1984, Vol. 6 Part 4, pp. 351-354*