An Efficient A* Stack Decoder Algorithm for Continuous Speech Recognition with a Stochastic Language Model*

Douglas B. Paul

Lincoln Laboratory, MIT
Lexington, Ma. 02173

ABSTRACT

The stack decoder is an attractive algorithm for controlling the acoustic and language model matching in a continuous speech recognizer. A previous paper described a near-optimal admissible Viterbi A* search algorithm for use with non-cross-word acoustic models and no-grammar language models [16]. This paper extends this algorithm to include unigram language models and describes a modified version of the algorithm which includes the full (forward) decoder, cross-word acoustic models and longer-span language models. The resultant algorithm is not admissible, but has been demonstrated to have a low probability of search error and to be very efficient.

INTRODUCTION

Speech recognition may be treated as a tree network search problem. As one proceeds from the root toward the leaves, the branches leaving each junction represent the set of words which may be appended to the current partial sentence. Each of the branches leaving a junction has a probability and each word has a likelihood of being produced by the observed acoustic data. The recognition problem is to identify the most likely path (word sequence, W^*) from the root (beginning of the sentence) to a leaf (end of the sentence) taking into account the junction probabilities (the stochastic language model, $p(W)$) and the acoustic match (including time alignment, $p(O|W)$) given that path [2]:

$$W^* = \underset{\{W\}}{\text{argmax}}\, p(O|W)p(W) \qquad (1)$$

where O is the acoustic observation sequence and W is a word sequence.

This paper is concerned with the network search problem and therefore correct recognition is defined as outputting the most likely sentence W^* given the language model, the acoustic models, and the observed acoustic data. If the most likely sentence is not the one spoken, it is a modeling error—not a search error. This paper will assume for simplicity that an isolated sentence is the object to be recognized. (The algorithm extends trivially to recognize continuous input.)

*This work was sponsored by the Defense Advanced Research Projects Agency. The views expressed are those of the author and do not reflect the official policy or position of the U.S. Government.

THE BASIC STACK DECODER

The stack decoder [8], as used in speech, is an implementation of a best-first tree search. The basic operation of a sentence decoder is as follows [2,5]:

1. Initialize the stack with a null theory.
2. Pop the best (highest scoring) theory off the stack.
3. if(end-of-sentence) output the sentence and terminate.
4. Perform acoustic and language-model fast matches to obtain a short list of candidate word extensions of the theory.
5. For each word on the candidate list:
   (a) Perform acoustic and language-model detailed matches to compute the new theory output log-likelihood.
      i. if(not end-of-sentence) insert into the stack.
      ii. if(end-of-sentence) insert into the stack with end-of-sentence flag = TRUE.
6. Go to 2.

The fast matches [4,5,7] are computationally cheap methods for reducing the number of word extensions which must be checked by the more accurate, but computationally expensive detailed matches.[1] (The fast matches may also be considered a predictive component for the detailed matches.) Top-N (N-best) mode is achieved by delaying termination until N sentences have been output.

[1]The following discussion concerns the basic stack decoder and therefore it will be assumed that the correct word will always be on the fast match list. This can be guaranteed by the scheme outlined in reference [5].

The stack itself is just a sorted list which supports the following operations: pop the best entry and insert new entries according to their scores. The following items must be contained in the ith stack entry:

1. a stack score: $StSc_i$
2. a reference time: t_ref_i
3. a word history i: (path or theory identification)
4. an output log-likelihood distribution: $L_i(t)$
5. an end-of-sentence flag

THE A* STACK CRITERION

A key issue in the stack decoder is deciding which theory should be popped from the stack to be extended. This is decided by the stack score and the reference time. (All scores used here are log-likelihoods or log-probabilities.)

The near-optimal A* criterion [11] used here is the difference between the actual log-likelihood of reaching a point in time on a path and a least upper bound on the log-likelihood of any path reaching that point in time:

$$\Lambda_i(t) = L_i(t) - \text{lub}L(t) \tag{2}$$

where $\Lambda_i(t)$ is the A* scoring function, $L_i(t)$ is the output log-likelihood, t denotes time, i denotes the path (tree branch or left sentence fragment) and $\text{lub}L(t)$ is the least upper bound on $L_i(t)$. (This criterion is derived in the appendix.) In order to sort the stack entries, it is necessary to reduce the $\Lambda_i(t)$ to a single number (the stack score):

$$StSc_i = \max_t \Lambda_i(t). \tag{3}$$

It is also convenient at this point to define the *minimum* time which satisfies equation 3:

$$t\_min_i = \operatorname{argmin}_t (StSc_i = \Lambda_i(t)). \tag{4}$$

It is also possible to estimate the most likely theory exit time as

$$\hat{t}\_exit_i = \operatorname{argmax}_t L_i(t) - \alpha t \tag{5}$$

for an appropriately chosen value for α.

A STACK DECODER FOR CSR WITH A UNIGRAM LANGUAGE MODEL

It is not possible to compute the exact least upper bound on the theory likelihoods without first performing the recognition. It is, however, possible to compute the least-upper-bound-so-far (lubsf) on the likelihoods that have already been computed, which requires negligible computation and is sufficient to perform the near-optimal A* search. This creates two difficulties:

1. Since $\hat{\text{lub}}L(t) = \text{lubsf}L(t)$ can change as the theories are evaluated, the stack order can also change.
2. A degeneracy in determining the best path by $StSc$ alone can occur since $\text{lubsf}L(t)$ can equal $L_i(t)$ for more than one i (path) at different times.

Problem 1 is easily cured by reevaluating the stack scores $StSc$ every time $\text{lubsf}L(t)$ is updated and reorganizing the stack. This is easily accomplished if the stack is stored as a heap [10].

Problem 2 occurs because different theories may dominate different parts of the current upper bound. Thus all of these theories will have a score of zero. The cure is to extend the shortest theory (minimum t_min) which has a stack score equal to the best. If $t\_ref_i = t\_min_i$, this can be accomplished by performing a major sort on the stack score $StSc$ and a minor sort on the reference time t_ref.

This guarantees that $\text{lubsf}L(t) = \text{lub}L(t)$ for $t \leq t\_ref_p$ (where p denotes the theory which is about to be popped) and therefore the relevant part of the least-upper-bound has been computed by the time that it is needed. Since the bound, at the time that it is needed, is the least-upper-bound, the search is admissible and near-optimal. Furthermore, when the first sentence is output, the least-upper-bound-so-far will be the exact least-upper-bound.

A stack pruning threshold can be used to limit the stack size [16]. Any theory whose $StSc$ falls below the threshold can be deleted from the stack. This can be applied on stack insertions and any time the stack is reorganized. This stack pruning threshold has little effect on the computational requirements and can therefore be set very conservatively to essentially eliminate any chance that the correct theory will be pruned.

In a time-synchronous (TS) no-grammar/unigram language model Viterbi decoder, all word output likelihoods are compared and only the maximum is passed on as input to the word models. Thus by comparison, only theories that dominate the lubsf need be retained on the stack and the stack pruning threshold can be set to zero for top-1 recognition. Since all stack scores, $StSc$, of all theories popped from the stack will be zero until the first sentence is output, all theories popped from the stack will be in reference time t_min order. (Of course, the stack pruning threshold must be non-zero if a top-N list of sentences is desired.) For top-N recognition, this algorithm adaptively raises the effective computational pruning threshold (which equals the current best $StSc$) by the minimum required to produce N output sentences,

subject to the limit placed by the stack pruning threshold.

This algorithm is near-optimal and admissible only for a Viterbi decode using non-cross word acoustic models and a no-grammar or unigram language model.

A STACK DECODER FOR CSR WITH A LONG-SPAN LANGUAGE MODEL

The above algorithm fails with a long span language model because the overall best theory can have a less-than-best intermediate score. This less-than-best intermediate score can be locally "shadowed" by the best score and thus will not be popped from the stack [6].

An efficient stack decoder algorithm which can be used with cross-word acoustic models, the full (forward) decoder, and longer-span (≥ 2) language models can be produced by two simple changes:

1. change the stack ordering to be a major sort on the reference time t_ref (favoring the lesser times) and a minor sort on the stack score $StSc$ and
2. use a non-zero stack pruning threshold.

The reference time t_ref may also be changed from the minimum time which satisfies equation 3 used in the no-grammar/unigram language-model version to $\hat{t}\_exit$ as defined in equation 5. (Either will work and both required similar amounts of computation in tests.) This algorithm appears to be a simplification of one developed at IBM [3].

This algorithm is not admissible because the correct theory can be pruned from the stack. The stack-pruning threshold now becomes the computational pruning threshold which controls the trade-off between the amount of computation and the probability of pruning the the correct theory by controlling the likelihood "depth" that will be searched. Unlike the previous algorithm, an (unpruned) theory cannot be shadowed because it will be extended when its reference time is reached. This algorithm is quasi-time-synchronous because it, in effect, moves a time bound forward and whenever this time bound becomes equal to the reference time of a theory, the theory is expanded.

Note that the stack pruning threshold can also be set to zero for no-grammar/unigram language model top-1 recognition with this algorithm. With a zero stack pruning threshold and $t\_ref_i = t\_min_i$, it becomes equivalent to the near-optimal, admissible no-grammar/unigram language model algorithm described above for top-1 recognition. (While this algorithm can also perform top-N recognition with or without a language model, it cannot be made equivalent to the no-grammar/unigram language model version for top-N. Its pruning threshold is fixed and it will only output theories whose relative likelihoods do not fall below the threshold.)

DISCUSSION AND CONCLUSIONS

The above stack-search algorithms have been implemented in a prototype implementation which uses real speech input, but does not yet have all of the features of the Lincoln TS CSR [13,14,15]. (The primary missing feature is cross-word phonetic modeling.) The prototype runs faster than does the TS system on the corresponding recognition task, frequently by a significant factor. (In fairness, the TS system does not include a fast match.) Current experience using the DARPA Resource Management Database [17] shows the required number of stack pops and the stack size to be surprisingly small. In addition, the prototype includes a proposed CSR-NL interface [12] and has been run with unigram, word-pair, bigram, and trigram language models accessed through the interface without difficulty. (It has also been run using a no-grammar language model, which, of course, does not require the interface.) This prototype implementation has also been tested with vocabulary sizes up to 64K words. The CSR computation, which is dominated by the fast match, scales approximately as the square root of the vocabulary size.

Methods for joining the acoustic matching of separate theories and caching of acoustic computations to reduce the acoustic match computation were described in reference [16]. These algorithms were tested in a stack-decoder simulator (real stack decoder with simulated input data). The path join accelerator is used in the prototype stack decoder to remove copies of theories which are identical except for non-grammatical items such as optional intermediate silences.

A* search using the scoring function described by Nilsson [11] (equation 6) requires computing the likelihood of the future data ($h^*(t)$ in equation 7). The optimal A* decoder requires exact evaluation of $h^*(t)$ which requires solving the top-1 recognition problem by some other means, such as a reverse direction TS decoder [19], before the A* search can begin. The alternative described here substitutes a near-optimal scoring function which is derived from the A* search and requires negligible additional computation over that required by the search itself. Since, as noted above, the Lincoln top-1 TS decoder takes more CPU time than does the near-optimal stack decoder, the near-optimal stack decoder algorithm appears to be the most efficient of the

three approaches for top-1 recognition. In addition, the long-span language model version of the stack decoder can very easily integrate long-span language models into the search. However, if top-N recognition is the goal, the optimal A* search may be preferred because, once the price is paid for computing $h^*(t)$, the A* search can find the additional N-1 sentences very efficiently for no-grammar/unigram language models [19].

Recently, several other algorithms have been proposed for top-N recognition using A* search [9,19,22] which use the Nilsson formulation of the scoring function. All of these approaches use a reverse direction TS decoder to compute $h^*(t)$. (A reverse direction top-1 stack decoder could also be used to compute $h^*(t)$.) (There are also some proposed non-A* methods for recognizing the top-N sentences [1,18,21]. In general, the bidirectional approaches appear to be more efficient than the unidirectional approaches.) These bidirectional A* methods must wait for the end of data (or a pseudo-end-of-data [9]) to begin the A* (or the reverse direction) pass. In contrast, because they do not need data beyond that necessary to extend the current theory (this includes data up to *t_ref* required to choose the current theory), the two stack decoder formulations proposed here can proceed totally left-to-right as the input data becomes available from the front end. The long-span language-model version of the stack search will output all top-N theories with minimal delay following the end-of-data because all theories are pursued in quasi-parallel or, in top-1 mode, it can output the partial sentence as soon as all unpruned theories have a common partial history (initial word sequence). (A similar technique for continuous output after a short delay from continuous input exists for TS decoders [20].)

One of the motivations for some of these other A* (and top-N) algorithms is as a method for using weaker and cheaper initial acoustic and language models to produce a top-N sentence list for later refinement by more detailed and expensive acoustic and/or language models, which now need only consider a few theories. In contrast the algorithm proposed here integrates both the detailed acoustic and language models directly in the stack search and therefore need only produce a top-1 output. It attempts to minimize the computation by applying all available information to constrain the search. (The stack decoder as described here can, of course, also be used with weak and cheap acoustic and/or language models to produce a top-N list for later processing.) The ultimate choice between the two methods may be determined by the number of sentences required by the top-N approaches and the relative computational costs of the various modules in each system. The architectural simplicity of each system may also have some bearing.

The stack decoder has long shown promise for integrating long-span language models and acoustic models into a single effective search which applies information from both sources into controlling the search. It has not been used at many sites, primarily due to the difficulty of making the search efficient. The algorithms described above will hopefully remove this barrier.

APPENDIX: DERIVATION OF THE A* CRITERION USED IN EQUATION 2

Nilsson [11] states the optimal A* criterion (slightly rewritten to match the speech recognition problem) as

$$f_i(t) = g_i(t) + h^*(t) \tag{6}$$

where $f_i(t)$ is the log-likelihood of a sentence with the partial theory i ending at time t, $g_i(t)$ is the log-likelihood of partial theory i, and $h^*(t)$ is the log-likelihood of the best extension of any theory from time t to the end of the data. (Nilsson uses costs which are interpreted here as negative log-likelihoods. All descriptions here will use sign conventions appropriate for log-likelihoods to be consistent with the rest of the paper.) The theory $\underset{i}{\text{argmax}}\,(\max_t f_i(t))$ is chosen as the next to be popped from the stack and expanded.

Equation 6 requires that the computation of the total likelihood of a sentence must be separable into a beginning part and an end part separated by a single time, which disallows this derivation for the full (forward) decoder because the full decoder does not have a unique transition time between two words. Thus, the derivation is limited to a decoder which is Viterbi between words. It also limits the derivation to non-cross-word acoustic models and no-grammar or unigram language model recognition tasks.

Define

$$f^*(t) = g^*(t) + h^*(t). \tag{7}$$

for the best theory with a word transition at time t. The function $f^*(t)$ is slowly varying with global maxima at the word transition points of the correct theory, at which points it equals the likelihood of the correct theory. Specifically, it is maximum at $t = 0$ and $t = T$. (T is the end of data.) Since $g_i(t)$ is an exact value (rather than a bound or estimate) for a tree search, $g^*(t) = \text{lub} g_i(t)$ and since $h^*(t)$ is not a function of i, $f^*(t) = \text{lub} f_i(t)$.

Subtract equation 7 from equation 6 and define $\hat{f}_i(t)$

$$\hat{f}_i(t) = f_i(t) - f^*(t) = g_i(t) - g^*(t). \tag{8}$$

This is just equation 2 in a different notation: $g_i(t) = L_i(t)$ and $g^*(t) = \text{ub}L(t)$ (specifically $\text{lub}L(t)$) and therefore $\hat{f}_i(t) = \Lambda_i(t)$. Thus, if $f^*(t)$ were a constant, $\hat{f}_i(t)$ would just be an offset from $f_i(t)$ and the search would be optimum because $\underset{i}{\text{argmax}}\,(\max_t \hat{f}_i(t))$ would always be equal to $\underset{i}{\text{argmax}}\,(\max_t f_i(t))$. As noted earlier, $f^*(t)$ has maxima at word transition times of the correct theory. Thus $\hat{f}_i(t)$ is zero at word transition times on the correct theory and ≤ 0 for all other i and t. Thus the search is admissible because it can never block the correct theory by giving a better score to an incorrect theory, but sub-optimal because it can cause incorrect theories to be popped from the stack and be evaluated. The evaluation function "error" $f^*(t) - f^*(0)$ is slowly varying and small, therefore the search is near-optimal.

Since the stack decoder treats each theory and all points on the likelihood distribution $L_i(t))$ as a unit, each theory is evaluated at its optimum point: the $\max_t \Lambda_i(t)$ as defined in equation 3, to give it its "best" chance and then, for efficiency, the likelihood of all points on the distribution $L_i(t)$ are extended in one operation.

The fact that all $StSc_i$ are zero until the first sentence is output and the tie is broken by choosing the theory with the minimum reference time t_min, insures that all candidate theories which might alter $\text{lubsf}L_i(t \leq t\_min_{pop})$ have already been computed. Thus the $\text{lubsf}L(t) = \text{lub}L(t)$ for $t \leq t\_min_{pop}$.

This derivation shows the stack criterion $\max StSc_i$ with a minimum t_min_i tie-breaker to be adequate to perform a near-optimal admissible A*-search Viterbi-recognition with non-cross word acoustic models and a no-grammar/unigram language-model using the stack decoder algorithm.

REFERENCES

1. S. Austin, R. Schwartz, and P. Placeway "The Forward-Backward Search Algorithm," ICASSP 91, Toronto, May 1991.
2. L. R. Bahl, F. Jelinek, and R. L. Mercer, "A Maximum Likelihood Approach to Continuous Speech Recognition," IEEE Trans. Pattern Analysis and Machine Intelligence, PAMI-5, March 1983.
3. L. R. Bahl and F. Jelinek, "Apparatus and Method for Determining a Likely Word Sequence from Labels Generated by an Acoustic Processor," US Patent 4,748,670, May 31, 1988.
4. L. Bahl, S. V. De Gennaro, P. S. Gopalakrishnam, R. L. Mercer, "A Fast Approximate Acoustic Match for Large Vocabulary Speech Recognition," submitted to ASSP.
5. L. Bahl, P. S. Gopalakrishnam, D. Kanevsky, D. Nahamoo, "Matrix Fast Match: A Fast Method for Identifying a Short List of Candidate Words for Decoding," ICASSP 89, Glasgow, May 1989.
6. J. K. Baker, personal communication, June 1990.
7. L. S. Gillick and R. Roth, "A Rapid Match Algorithm for Continuous Speech Recognition," Proceedings June 1990 Speech and Natural Language Workshop, Morgan Kaufmann Publishers, June, 1990.
8. F. Jelinek, "A Fast Sequential Decoding Algorithm Using a Stack," IBM J. Res. Develop., vol. 13, November 1969.
9. P. Kenny, R. Hollan, V. Gupta, M. Lennig, P. Mermelstein, and D. O'Shaughnessy, "A* - Admissible Heuristics for Rapid Lexical Access," ICASSP 91, Toronto, May 1991.
10. D. E. Knuth, "The Art of Computer Programming: Sorting and Searching,", Vol. 3., Addison-Wesly, Menlo Park, California, 1973.
11. N. J. Nilsson, "Problem-Solving Methods of Artificial Intelligence," McGraw-Hill, New York, 1971.
12. D. B. Paul, "A CSR-NL Interface Specification," Proceedings October, 1989 DARPA Speech and Natural Language Workshop, Morgan Kaufmann Publishers, October, 1989.
13. D. B. Paul, "Speech Recognition using Hidden Markov Models," Lincoln Laboratory Journal, Vol. 3, no. 1, Spring 1990.
14. D. B. Paul, "New Results with the Lincoln Tied-Mixture HMM CSR System," Proceedings Fourth DARPA Speech and Natural Language Workshop, Morgan Kaufmann Publishers, February, 1991.
15. D. B. Paul, "The Lincoln Tied-Mixture HMM Continuous Speech Recognizer," ICASSP 91, Toronto, May 1991.
16. D. B. Paul, "Algorithms for an Optimal A* Search and Linearizing the Search in the Stack Decoder," ICASSP 91, Toronto, May 1991.
    also
    D. B. Paul, "Algorithms for an Optimal A* Search and Linearizing the Search in the Stack Decoder," Proceedings June 1990 Speech and Natural Language Workshop, Morgan Kaufmann Publishers, June, 1990.
17. P. Price, W. Fisher, J. Bernstein, and D. Pallett, "The DARPA 1000-Word Resource Management Database for Continuous Speech Recognition," ICASSP 88, New York, April 1988.
18. R. Schwartz and S. Austin, "A Comparison of Several Approximate Algorithms for Finding Multiple (N-Best) Sentence Hypotheses," ICASSP 91, Toronto, May 1991.
19. F. K. Soong and E. F. Huang, "A Tree-Trellis Fast Search for Finding the N Best Sentence Hypotheses in Continuous Speech Recognition," ICASSP 91, Toronto, May 1991.
20. J. C. Spohrer, P. F. Brown, P. H. Hochschild, and J. K. Baker, "Partial Backtrace in Continuous Speech Recognition," Proc. Int. Conf. on Systems, Man, and Cybernetics, 1980.
21. V. Steinbiss, "Sentence-Hypothesis Generation in a Continuous Speech Recognition System," EUROSPEECH 89, Paris, Sept 1989.
22. V. Zue, J. Glass, D. Goodine, H. Leung, M. Phillips, J. Polifroni, and S. Seneff, "Integration of Speech Recognition and Natural Language Processing in the MIT Voyager System," ICASSP 91, Toronto, May 1991.