Phoneme Lattice Based A* Search Algorithm for Speech Recognition

Pascal NOCERA, Georges LINARES, Dominique MASSONIÉ, and Loïc LEFORT

Laboratoire Informatique d'Avignon, LIA, Avignon, France
{pascal.nocera, georges.linares, dominique.massonie, loic.lefort}@lia.univ-avignon.fr

Abstract. This paper presents the Speeral continuous speech recognition system developed in the LIA. Speeral uses a modified A* algorithm to find in the search graph the best path taking into account acoustic and linguistic constraints. Rather than words by words, the A* used in Speeral is based on a phoneme lattice previously generated. To avoid the backtraking problems, the system keeps for each frame the deepest nodes of the partially explored lexical tree starting at this frame. If a new hypothesis to explore is ended by a word and the lexicon starting where this word finishes has already been developed, then the next hypothesis will "jump" directly to the deepest nodes. Decoding performances of Speeral are evaluated on the test set of the ARC B1 campaign of AUPELF'97. The experiments on this French database show the efficiency of the search strategy described in this paper.

1 Introduction

The goal of continuous speech recognition systems is to find the best sentence (list of words) corresponding to an acoustic signal taking into account acoustic and linguistic constraints [1]. This is achieved on some systems with a stack decoder based on the A* search algorithm [2]. As the best path has to be found in the list of all possible paths, the search graph structure is a tree built by concatenation of lexical trees. Each leaf of the tree (an hypothetical word) is connected to a new full lexical tree.

The A* algorithm is a time-asynchronous algorithm. The exploration rank of node x is given by the result of the evaluation function $F(x)$ representing an estimation of the best path involving x. This means that it's possible to backtrack to an hypothesis (theory) very earlier than the deepest theory, because it has then the best evaluated score. Applied to a continuous speech recognition graph, this algorithm will explore the same word many times, with different previous paths (history).

This is why the A* algorithm is almost always used on high level graphs like word lattices. The theory progression is made word after word. It is used on multi-pass systems to find the best hypothesis from the word lattice [3], or in some single-pass systems after a fast-match algorithm to obtain a short list of candidate word extensions of a theory [4]. However, the size of the lattice for both methods has to be sufficient to obtain good results. In one-pass systems, the fast-match algorithm has to be redone each time a new theory is explored (even if another theory was already explored for that frame), because linguistic constraints changes with theory, and the list of candidates can be different.

To avoid this problem, we propose to base our search on phonemes rather than words. The progression is done phoneme after phoneme. To optimize the search when a backtrack occurs, we store for each lexical tree explored, even partially, the deepest nodes corresponding to a complete word or to a "part of word". In case of backtracking, the lexicon already computed won't be explored again. The algorithm will jump directly to the stored nodes, which will be appended to the current theory.

In the first part, we will present the standard A* algorithm and in the second part the enhanced A* algorithm for phoneme lattice. We will then present the LIA speech recognition system (Speeral) and present results obtained on AUPELF'97 evaluation campaign for French.

2 The standard A* algorithm

The A* algorithm is a search algorithm used to find the best path in a graph. It uses an evaluation function $F(x,y)$ for each explored node x. This estimation is computed by the sum of the cost of the path from the starting node of the graph to the node x ($g(x)$), of the current transition from node x to a next node y ($c(x,y)$), and of the estimated cost ($h(y)$) of the remaining path (from y to the final node) (figure 1) .

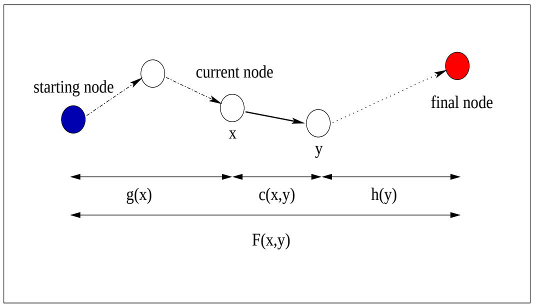

Fig. 1. The evaluation function $F(x,y)$ is the sum of the cost of the optimal path from the starting node to the current one ($g(x)$), of the current transition from current to the next node ($c(x,y)$) and of the sounding function h(y) estimating the cost of the remaining path from next node to the final node (h(y)).

The algorithm uses an ordered list called *Open* which contains all the nodes to be explored in decreasing order of their F value. For each iteration of the search algorithm, the first node x in *Open* is removed from the list and for each node y (successor of the node x in the graph) the estimation function $F(x,y) = g(x) + c(x,y) + h(y)$ is computed (figure 1) and the new hypothesis y is added into *Open*.

The algorithm stops when the top node in the *Open* list is a goal node. It was proven that if the evaluation function F is always better than the optimal cost, this search algorithm always terminates with an optimal path from the start to a goal node. The optimality of the evaluation function is given by an optimal estimated cost function h.

3 The phoneme lattice based A* algorithm

3.1 Lexicon coding

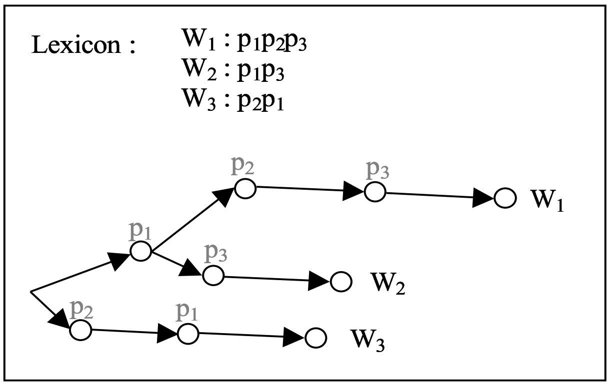

Fig. 2. The lexical tree structure

The lexicon is expressed by the list of words in the vocabulary followed by the list of phonemes for each word. Several lists of phonemes for a same word will express different phonological variations. The shortcuts or the liaisons between words will be expressed as phonological variations. The lexicon is represented by a tree in which words share common beginning phonemes and each leaf corresponds to a word (figure 2).

The search graph is a concatenation of lexical trees. A new lexical tree starts each time a word of a previous lexicon ends.

3.2 Linguistic scoring

In order to have a better language model flexibility, the computation of the linguistic score is made outside the search core. Each time the algorithm needs a linguistic scoring of a theory, it calls an external function with the list of words or nodes.

Linguistic scores of new hypothesis are computed with two functions depending on current state:

- the LM_Word function is used when theory ends with a word. It processes the whole list of words of this hypothesis,
- the LM_Part function processes the whole list of words of this hypothesis including the last "pending word" (internal node n of a lexical tree). This allows a finer grained hypothesis scoring with anticipation of upcoming words through this node.

$$LM\_Part(w_1..w_k, x) = \max_{w_n} LM\_Word(w_1..w_k w_n)$$
$$\textit{where } w_n \textit{ is any leaf (i.e. word) of the sub-tree starting at } x.$$

Moreover, anticipating the linguistic constraints allows an earlier cut of paths leading to improbable words.

3.3 The sounding function (*Hacoust*)

The estimated cost function h retained (*Hacoust*) is an optimal probe representing for each frame the cost of a path to the end. *Hacoust* is only constrained by acoustic values computed by a backward execution of the Viterbi algorithm on specific models. Indead, the Viterbi-back algorithm applied to the full set of contextual models would be an expensive process. In order to speed-up the computation of the *Hacoust* function, we use composite models (figure 3). Composite models are built by grouping all contextual states of models coding a phoneme into a single HMM. Right, central and left states are placed on right, center and left part of the composite ones. Acoustic decoding based on these specific units is an aproximation of the best path (phoneme path) between a frame and the end of the sentence. The corresponding sounding function respects the A* constraint of *Hacoust* optimality : composite units allow lower cost paths than contextual ones, since neither lexical nor linguistic constraints are taken into account.

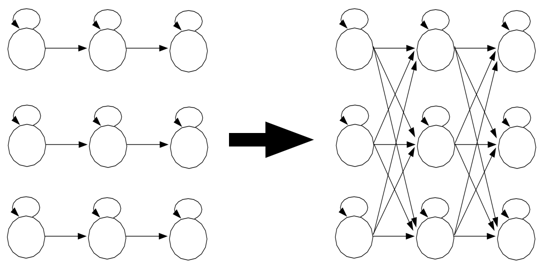

Fig. 3. Specific units used by the evaluation function: all HMMs representing a contextual unit are grouped as a larger one, composed of all contextual states.

The accuracy of the estimated cost function is very important for the searching speed so we first tried to improve this function before or during the search. However, the time saved for the search was negligible compared to the time needed to compute of a better $Hacoust$ function.

3.4 A* search algorithm enhancement

Outline To avoid the re-estimation of already explored parts of the search graph, the system keeps for each frame the deepest nodes of the partially explored lexical tree starting at this frame. If the currently explored hypothesis is ended by word on a frame t and there is an already developed lexical tree starting at t, then the next hypothesis will directly "jump" to the deepest nodes.

Manipulated data

- $HypLex$ represents a node in the lexical tree. Each $HypLex$ contains a pointer to the node in the tree and its final frame.
- Tab_Lex: $Tab\_Lex[t]$ contains the list of the $HypLex$ for the lexicon starting at the frame t.
- $TabEnd$: $TabEnd[t]$ contains the list of word sequences already explored ending at frame t. These sequences constitute the different theories of "whole words" already processed.

Description As explained before, only the deepest nodes are kept in Tab_Lex. These nodes correspond to "pending words" or to "whole words" (i.e. leaves). If the algorithm backtracks, this storage prevents reexploring an even partially processed lexicon.

The algorithm keeps producing new hypothesis until the top of the $Open$ list (i.e. current best theory) is a goal. At each iteration, the current best hypothesis Hyp (an $HypLex$) is taken out of the top of the $Open$ list and:

- if Hyp is a phoneme (a node of a lexical tree), for each $New = Successor(Hyp)$:
  - if New is a phoneme, New is put in Tab_Lex and the best hypothesis from the start of the sentence to New is added to $Open$.
  - if New is a word, all the "whole word" hypothesis ended by New are stored in $TabEnd$ and the best one is added to $Open$ (if there was no better one before).
- if Hyp is a word ending at frame t (leaf of a lexical tree),
  - if the lexicon beginning at the end of Hyp was already explored, all the theories with Hyp followed by $Tab\_Lex[t]$ are generated (fig. 5).
  - otherwise, a new lexical tree is started and $Successor(Lexicon\_Start)$ is stored in $Tab\_Lex[t]$ (fig. 6).

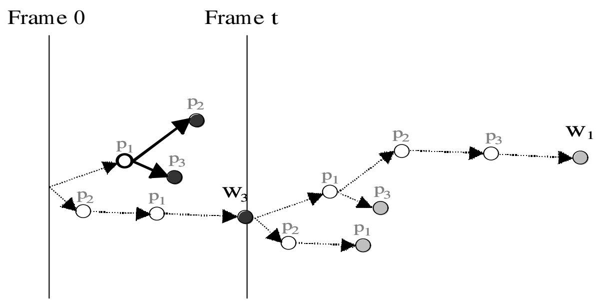

Fig. 4. Initial state of the search graph for the samples below.

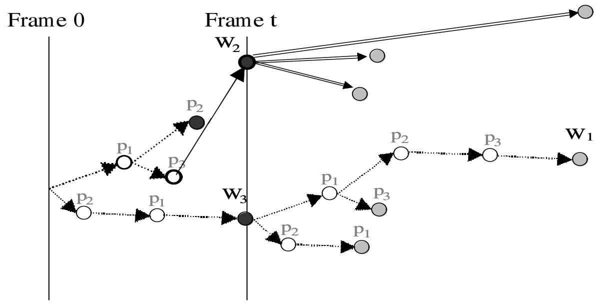

Fig. 5. The word W_2 inherits previous search buffered in $Tab\_Lex[t]$.

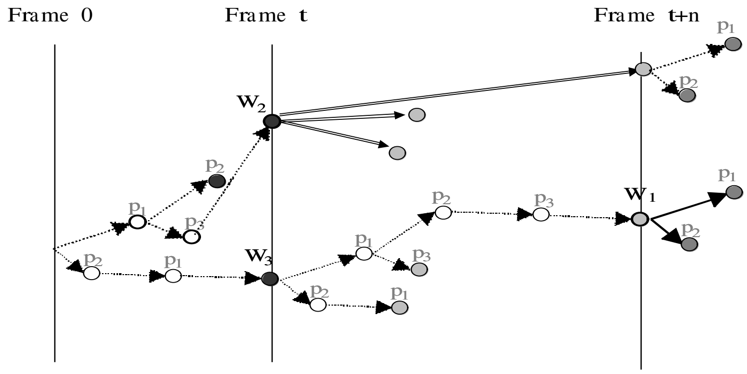

Fig. 6. The extending nodes p_1 and p_2 of the lexical tree starting at frame t are stored in $Tab\_Lex[t+n]$. Thus they will be available (without computation) to extend further hypothesis ending by a whole word at frame $t+n$.

3.5 Limiting the Open-list size

To use the A* algorithm with a phoneme lattice in the Speeral system, we had to define a cut function to prevent backtracking too early. The sounding function *Hacoust* does not take into account lexical and linguistic constraints. So the longer the theory is, the stronger it is constrained. Even if a theory started very far from the top of *Open*, it may become the best when all the others theories in *Open* getting longer are thus more constrained. To prevent this problem, a theory is dropped when it is too short compared to the deepest one.

4 The Speeral system

The Speeral decoding process relies on the A* algorithm exposed in section 3.4. The acoustic models are HMM models and the lattice is constituted by the n-best phonemes for each frame. This lattice is computed using an acoustico-phonetic decoding using the backward Viterbi algorithm. We also obtain at the same time the *Hacoust* function estimations needed for the A* execution. Acoustic models are classical 3-states HMMs with Male/Female specialization and about 600 right contextual units are used for each set. States are modeled by a mixture of 32 gaussians. Acoustic models were trained on the two French databases BREF80 and BREF.

The lexicon contains $20,000$ words and we used a trigram language model computed on the text of the newspaper “Le Monde” from 1987 to 1996. For the calculus of the *LM_Part* function, we defined the “*Best_Tri_Node*” function. This function has a low memory usage.

$$\begin{aligned}
LM_Word(w_i..w_3w_2w_1) &= P(w_1/w_3w_2) \\
LM_Part(w_i..w_2w_1,x) &= Best_Tri_Node(w_2w_1,x) \\
&= \max_{w_n} P(w_n/w_2w_1)
\end{aligned}$$

where w_n is a leaf of the sub-tree starting at x.

This system was tested on the database of the evaluation campain ARC B1 of AUPELF'97 [5]. This database constitutes the only French corpus on which several systems were tested. Table 1 shows Speeral and other systems performances. Nevertheless, Speeral results are obtained several years after the campaign and must be considered only as reference. Currently, we have obtained a word error rate of 19.0% on the baseline system (noted Speeral in table 1). This result is obtained with a phoneme lattice of 75 phonetical hypothesis for each frame.

It is worth noting that this system explores a very low number of word hypothesis at each frame: $200,000$ of the $300,000$ test frames generate no word

Table 1. Word Error Rates of the systems for the task ARC B1 of the AUPELF'97 speech recognition system evaluation campaign. P0-1, P0-2 and P0-3 are CRIM, CRIN, LAFORIA systems. P0-3, P0-4, P0-5 are 3 alternatives of LIMSI base system. Speeral is the actual system of LIA.

System	P0-1	PO-2	P0-3	P0-4	P0-5	P0-6	Speeral
WER	39.6	32.8	39.4	12.2	11.1	13.1	19.0

hypothesis at all. The average number of word hypothesis per frame is 44 which is a very low number compared to several hundreds of generated word hypothesis per frame in classical search algorithms such as the fast-match or word lattice based ones.

5 Conclusion

We have presented an original application of the A* algorithm on a phoneme lattice rather than on a word lattice. To find a solution any speech recognition system has to over-produce hypothesis. The cost of a lattice generation is far less expensive for phonemes than for words. Exploring such a lattice would have been more time consuming without the storing process of the partially explored lexical trees allowing a large reduction of the evaluated paths. According to our first experiments, the results are encouraging. Nevertheless, better performances should be obtained by adapting acoustic models to speakers and by improving acoustic and linguistic models. Moreover, the use of such an A* algorithm allows integration of various sources of information during decoding stage by adding specific terms to the path cost evaluation function. We are working now on the exploitation of this potentiallity.

References

1. R. De Mori, "Spoken dialogues with computers," 1997.
2. J. Pearl, "Heuristics : Intelligent search strategies for computer problem solving," 1984.
3. H.-W. Hon M.-Y. Hwang K.-F. Lee R. Rosenfeld X. Huang F. Alleva, "The SPHINX II speech recognition system: An overview," *Computer Speech and Language*, vol. Vol 7 n°2, pp. 137–148, 1993.
4. D.B. Paul, "Algorithms for an optimal A* search and linearizing the search in the stack decoder," *ICASSP 91*, pp. 693–696, 1991.
5. J. Dolmazon F. Bimbot G. Adda J. Caerou J. Zeiliger M. Adda-Decker, "Première campagne AUPELF d'évaluation des systèmes de Dictée Vocale," "Ressources et évaluation en ingénierie des langues,", pp. 279–307, 2000.
6. Matrouf, O. Bellot, P. Nocera, J.-F. Bonastre, G. Linares, "A posteriori and a priori transformations for speaker adaptation in large vocabulary speech recognition systems," *Eurospeech 2001*, Aalborg