FAST LM LOOK-AHEAD FOR LARGE VOCABULARY CONTINUOUS SPEECH RECOGNITION USING PERFECT HASHING

Antonio Cardenal-López, F. Javier Diéguez-Tirado, Carmen García-Mateo
E.T.S.I.Telecomunicacion
Dpto. Teoría de la Señal y Comunicaciones
Universidad de Vigo
36200 Vigo-Spain
e-mail: cardenal,jdieguez,carmen@gts.tsc.uvigo.es

ABSTRACT
In this paper we present a fast method to implement a language model (LM) look-ahead algorithm in a Viterbi-based, single-lexical-tree speech recognizer. We have used three different mechanisms to speed up the calculation: a cache memory attached to each node of the network, a pre-calculation of the probabilities of the active contexts, and an organization of the LM using perfect hash. These enhancements make it possible to use the full trigram LM to compute the look-ahead with better overall results, both in terms of recognition rate and computation time, than using a unigram or bigram based approximation.

1. INTRODUCTION
Language model (LM) look-ahead techniques are incorporated in most modern continuous speech recognizers with a tree organized lexicon. The main idea is to use the maximum probability for all possible words that may be reached from a network node, given a LM context, as an approximation of the unknown LM score [1]. It is widely known that with this approximation the overall search effort is reduced, but at some additional computational cost. Due to the number of active paths per frame in a typical recognition task, the access to LM probabilities and the computation of the prediction must be performed millions of times in the recognition of several seconds of speech. This has an important impact in the computational load. The extra effort may be partially reduced using a cache based strategy, but there is still an overload that may compensate the reduction of the complexity.

For this reason many systems incorporate simplified methods, like unigram [2] or bigram maximization. In [3] it is shown that the use of trigrams in the prediction reduces the acoustic search effort in almost 20% compared with bigram case. However, due to the extra cost of the LM look-ahead, the overall recognition time is increased.

In the recognizer developed in the University of Vigo, we have combined three mechanisms that are applied sequentially to reduce the cost of the LM prediction. We use a cache memory in each node of the tree to store the values of the calculated look-ahead values. The probabilities of the new active LM contexts are pre-computed and stored to speed up the maximization. Finally an organization of the LM using perfect hash techniques allows an efficient access to a single LM probability. With these improvements a trigram based look-ahead can be employed with no extra computational cost.

In the remainder of this paper, we will describe in detail the mentioned mechanisms. First of all we give an overview of the decoder of the University of Vigo and a description of the experiments used to obtain the results presented in this paper. In section 3 the three mechanisms used for the fast calculation of the LM look-ahead are described. Finally we present some preliminary results and make some comments about further work.

2. FRAMEWORK DESCRIPTION

2.1. Decoder description
In this section a brief description of the decoder of the University of Vigo is given.

The system uses a two-pass strategy. First pass is basically a Viterbi synchronous beam search, implemented using the token passing paradigm. We use a single lexical-tree, with the LM separated from the network as in [4]. With this kind of organization, all LM scores must be obtained by query, so the access to the LM probabilities must be carefully designed.

The first pass includes all available knowledge sources: context-dependent Hidden Markov Models (HMMs), cross word-models, and trigram based LM. In the experiments presented in this paper we have employed demiphones [5], but as the network is constructed off-line, other context-dependent models may also be employed. Search complexity is reduced using several pruning thresholds applied to global, acoustic and LM scores. An additional word level pruning is applied to restrict the number of new words starting in each frame. The unpropagated words are stored to be used later.

In the second pass all words extracted in the beam search are organized in a word-graph, which is re-scored using an A* algorithm. The goal of this pass is to include in the search the words not propagated in the previous pass.

2.2. Test-bed
We have performed several experiments using a multi-speaker telephone database in Galician language. The test data consists of 364

This work has been partially supported by the Spanish project TIC2000-1005-C03-02 “Tecnologías del habla multilingüe”.

0-7803-7402-9/02/$17.00 ©2002 IEEE 1 - 705
files from 59 male and female speakers, with a total of 15891 uttered words. Each speaker was asked to read a short journalistic text.

The LM was extracted from a newspaper macrotext and contains 21,216 unigrams, 267,806 bigrams and 218,050 trigrams. The perplexity of the task is 149.68. There are no out-of-vocabulary words. For the acoustic modeling, we have used 497 demiphones with 2 states per model and 4 mixtures per state trained from the SpeechDat database for Galician.

The experiments were performed using a 930 MHz Pentium III processor.

3. EFFICIENT LM LOOK-AHEAD

To obtain the LM look-ahead, given a tree node $N$ and a LM context $\{w_1, w_2\}$, the set of words (unigrams) that may be reached from $N$, $U_N = \{w_1, \ldots, u_M\}$, must be identified. If $N$ is an initial node of the network this may be an expensive task. In our decoder, the lexical tree is constructed off-line using a list of transcriptions sorted by ASCII code. Each of them is identified by its position in the sort. Since we use a single ASCII symbol to represent each phoneme in the inventory, the transcriptions $T_N = \{t_1, \ldots, t_O\}$ that can be reached from $N$ have correlative keys. In this way, the set $T_N$ may be defined using the index of $t_1$ and the length $O$. Both values are combined in a single integer which is stored in the lexical node. To obtain the set $U_N$ from $T_N$, (i.e. the unigrams from the phonetic transcriptions), a simple array is used. The values $O$ and $M$ may be different if several alternative transcriptions per unigram are employed.

The next, and most expensive part of the algorithm is to maximize the probabilities $Pr(u_i/w_1, w_2)$, with $u_i \in U_N$. This task must be performed millions of times in the recognition of a short speech segment, so the number of queries of LM probabilities are thousands of millions. For example, in the recognition of a five second long (500 frames) speech segment, with a global pruning threshold of 160, a total number of 20 millions of LM predictions are calculated. A more detailed study shows that only 100,000 of these queries have different arguments (i.e. different node $N$ and LM context $\{w_1, w_2\}$). Moreover, the number of active contexts is only 500 with only 7000 active nodes. These results expressed per frame are shown in table 1.

In order to avoid these repeated calculations, many recognizers use LM caching. In this way the number of look-ahead computations may be reduced by a factor of 10. Nevertheless the remaining 10% may turn out quite expensive to compute. If a back-off LM is employed, the calculation of a single trigram probability may comprise three or more accesses to the LM data.

To achieve additional computation reductions we have combined the classic cache strategy with two additional mechanisms that are applied sequentially (Fig. 1). First, we take advantage of the reduced number of active contexts per frame by pre-computing and storing the probabilities of each new active context. Second, the extraction of a single LM score is accelerated using perfect hash.

### 3.1. LM Look-Ahead cache

The results in table 1 suggest the use of a cache memory to store the calculated look-ahead values. If a global cache is chosen, the back-off identifier of the tree node $N$, and the words in the context must be used as the index of the table. Therefore the search in the cache requires a maximum of three comparisons per key. According to the results shown in table 1, a cache of more than 200 would be necessary to store the average number of different values per frame in this experiment.

The use of a single lexical tree allows a more interesting approach. In this case multiple tokens with different LM contexts may reach the same lexical node, so a separate small cache may be attached to each node. This strategy has several advantages. On one hand, the size of each cache may be fixed to a very small value (10 to 20), so that a very fast search can be performed. In addition, the comparison with the node identifier is avoided. As we will see in the following section, each active context can be identified with a single integer, so the search is reduced to a single comparison.

The performance of the node cache as a function of its size...
is shown in figure 2. Best results are achieved for a size of 15. For greater values improvements in percentage of hits are partially compensated by the increase in the cost of the search in the cache memory.

3.2. Pre-computation of LM context probabilities

LM look-ahead probabilities that are not available in the node cache must be calculated in a more complex way.

To make this calculation as fast as possible, it has been taken into account that most of the tokens in a given frame share only a few n-gram histories, as it was illustrated with the previous example at the beginning of the section (Table 1). Also, as tokens traverse the network, LM look-ahead probabilities are refined by maximizing a subset of previously applied n-gram probabilities.

It has proven useful to assign a correlative number to each n-gram history found in a frame, and to calculate in advance each of the 20K possible n-grams beginning with that history, storing these probabilities in an array. All of these arrays (LM contexts) are kept in a list to be able to erase them if they are not used within a frame time. For each token fed onto the network, the appropriate context is searched in this list or created if it’s not found, and is then linked to the token. LM look-ahead is in this way reduced to a simple maximization of items in this array, for which a piece of highly optimized program code has been written.

One of the disadvantages of this method is that not all the n-grams that have been evaluated will be actually used. However, it is remarkable that this method allows the node cache (Sec.3.1) to be searched by a single context identifier instead of by several history words, saving time and space. Another major issue about using LM contexts is that they require a considerable amount of memory by themselves.

As we have used 4 bytes to store each probability, each context takes up about 80KB, so it is desirable that the number of contexts per frame is kept reasonably low. This has been achieved by the use of word pruning at the first nodes, so that only the most likely transcriptions are let into the network. Using word pruning, the number of required iterations can be reduced to a simple maximization of items in an array, for which a piece of highly optimized program code has been written.

One of the disadvantages of this method is that not all the n-grams that have been evaluated will be actually used. However, it is remarkable that this method allows the node cache (Sec.3.1) to be searched by a single context identifier instead of by several history words, saving time and space. Another major issue about using LM contexts is that they require a considerable amount of memory by themselves.

As we have used 4 bytes to store each probability, each context takes up about 80KB, so it is desirable that the number of contexts per frame is kept reasonably low. This has been achieved by the use of word pruning at the first nodes, so that only the most likely transcriptions are let into the network. Using word pruning, the number of required iterations can be reduced to a simple maximization of items in an array, for which a piece of highly optimized program code has been written.

One of the disadvantages of this method is that not all the n-grams that have been evaluated will be actually used. However, it is remarkable that this method allows the node cache (Sec.3.1) to be searched by a single context identifier instead of by several history words, saving time and space. Another major issue about using LM contexts is that they require a considerable amount of memory by themselves.

As we have used 4 bytes to store each probability, each context takes up about 80KB, so it is desirable that the number of contexts per frame is kept reasonably low. This has been achieved by the use of word pruning at the first nodes, so that only the most likely transcriptions are let into the network. Using word pruning, the number of required iterations can be reduced to a simple maximization of items in an array, for which a piece of highly optimized program code has been written.

One of the disadvantages of this method is that not all the n-grams that have been evaluated will be actually used. However, it is remarkable that this method allows the node cache (Sec.3.1) to be searched by a single context identifier instead of by several history words, saving time and space. Another major issue about using LM contexts is that they require a considerable amount of memory by themselves.

As we have used 4 bytes to store each probability, each context takes up about 80KB, so it is desirable that the number of contexts per frame is kept reasonably low. This has been achieved by the use of word pruning at the first nodes, so that only the most likely transcriptions are let into the network. Using word pruning, the number of required iterations can be reduced to a simple maximization of items in an array, for which a piece of highly optimized program code has been written.

One of the disadvantages of this method is that not all the n-grams that have been evaluated will be actually used. However, it is remarkable that this method allows the node cache (Sec.3.1) to be searched by a single context identifier instead of by several history words, saving time and space. Another major issue about using LM contexts is that they require a considerable amount of memory by themselves.

As we have used 4 bytes to store each probability, each context takes up about 80KB, so it is desirable that the number of contexts per frame is kept reasonably low. This has been achieved by the use of word pruning at the first nodes, so that only the most likely transcriptions are let into the network. Using word pruning, the number of required iterations can be reduced to a simple maximization of items in an array, for which a piece of highly optimized program code has been written.

One of the disadvantages of this method is that not all the n-grams that have been evaluated will be actually used. However, it is remarkable that this method allows the node cache (Sec.3.1) to be searched by a single context identifier instead of by several history words, saving time and space. Another major issue about using LM contexts is that they require a considerable amount of memory by themselves.

As we have used 4 bytes to store each probability, each context takes up about 80KB, so it is desirable that the number of contexts per frame is kept reasonably low. This has been achieved by the use of word pruning at the first nodes, so that only the most likely transcriptions are let into the network. Using word pruning, the number of required iterations can be reduced to a simple maximization of items in an array, for which a piece of highly optimized program code has been written.

One of the disadvantages of this method is that not all the n-grams that have been evaluated will be actually used. However, it is remarkable that this method allows the node cache (Sec.3.1) to be searched by a single context identifier instead of by several history words, saving time and space. Another major issue about using LM contexts is that they require a considerable amount of memory by themselves.

As we have used 4 bytes to store each probability, each context takes up about 80KB, so it is desirable that the number of contexts per frame is kept reasonably low. This has been achieved by the use of word pruning at the first nodes, so that only the most likely transcriptions are let into the network. Using word pruning, the number of required iterations can be reduced to a simple maximization of items in an array, for which a piece of highly optimized program code has been written.

One of the disadvantages of this method is that not all the n-grams that have been evaluated will be actually used. However, it is remarkable that this method allows the node cache (Sec.3.1) to be searched by a single context identifier instead of by several history words, saving time and space. Another major issue about using LM contexts is that they require a considerable amount of memory by themselves.

As we have used 4 bytes to store each probability, each context takes up about 80KB, so it is desirable that the number of contexts per frame is kept reasonably low. This has been achieved by the use of word pruning at the first nodes, so that only the most likely transcriptions are let into the network. Using word pruning, the number of required iterations can be reduced to a simple maximization of items in an array, for which a piece of highly optimized program code has been written.

One of the disadvantages of this method is that not all the n-grams that have been evaluated will be actually used. However, it is remarkable that this method allows the node cache (Sec.3.1) to be searched by a single context identifier instead of by several history words, saving time and space. Another major issue about using LM contexts is that they require a considerable amount of memory by themselves.

As we have used 4 bytes to store each probability, each context takes up about 80KB, so it is desirable that the number of contexts per frame is kept reasonably low. This has been achieved by the use of word pruning at the first nodes, so that only the most likely transcriptions are let into the network. Using word pruning, the number of required iterations can be reduced to a simple maximization of items in an array, for which a piece of highly optimized program code has been written.

One of the disadvantages of this method is that not all the n-grams that have been evaluated will be actually used. However, it is remarkable that this method allows the node cache (Sec.3.1) to be searched by a single context identifier instead of by several history words, saving time and space. Another major issue about using LM contexts is that they require a considerable amount of memory by themselves.

As we have used 4 bytes to store each probability, each context takes up about 80KB, so it is desirable that the number of contexts per frame is kept reasonably low. This has been achieved by the use of word pruning at the first nodes, so that only the most likely transcriptions are let into the network. Using word pruning, the number of required iterations can be reduced to a simple maximization of items in an array, for which a piece of highly optimized program code has been written.

One of the disadvantages of this method is that not all the n-grams that have been evaluated will be actually used. However, it is remarkable that this method allows the node cache (Sec.3.1) to be searched by a single context identifier instead of by several history words, saving time and space. Another major issue about using LM contexts is that they require a considerable amount of memory by themselves.

As we have used 4 bytes to store each probability, each context takes up about 80KB, so it is desirable that the number of contexts per frame is kept reasonably low. This has been achieved by the use of word pruning at the first nodes, so that only the most likely transcriptions are let into the network. Using word pruning, the number of required iterations can be reduced to a simple maximization of items in an array, for which a piece of highly optimized program code has been written.

One of the disadvantages of this method is that not all the n-grams that have been evaluated will be actually used. However, it is remarkable that this method allows the node cache (Sec.3.1) to be searched by a single context identifier instead of by several history words, saving time and space. Another major issue about using LM contexts is that they require a considerable amount of memory by themselves.

As we have used 4 bytes to store each probability, each context takes up about 80KB, so it is desirable that the number of contexts per frame is kept reasonably low. This has been achieved by the use of word pruning at the first nodes, so that only the most likely transcriptions are let into the network. Using word pruning, the number of required iterations can be reduced to a simple maximization of items in an array, for which a piece of highly optimized program code has been written.

One of the disadvantages of this method is that not all the n-grams that have been evaluated will be actually used. However, it is remarkable that this method allows the node cache (Sec.3.1) to be searched by a single context identifier instead of by several history words, saving time and space. Another major issue about using LM contexts is that they require a considerable amount of memory by themselves.

As we have used 4 bytes to store each probability, each context takes up about 80KB, so it is desirable that the number of contexts per frame is kept reasonably low. This has been achieved by the use of word pruning at the first nodes, so that only the most likely transcriptions are let into the network. Using word pruning, the number of required iterations can be reduced to a simple maximization of items in an array, for which a piece of highly optimized program code has been written.
scheme and a binary tree organized LM even for single accesses outside a loop. Taking advantage of the loop structure, big improvements have been achieved.

4. EXPERIMENTAL RESULTS

In this section we present some results obtained using the 20K word dictation task described in section 2.2. They are given in terms of real time factor, mean number of elements per frame and word error rate (WER). In order to describe the time consumption of the different parts of the recognizer, routines have been classified into the following groups: LM look-ahead calculations (LM-LA); computation of exact LM probabilities (LM); computation of acoustic probabilities; token propagation and other routines (second pass and initializations). The results for unigram, bigram and trigram based LM look-ahead are shown in Table 5. The exact LM probabilities must be obtained for every new word that reaches the end of the network if bigram and unigram based look-ahead is used. In the case of trigrams the computation is only necessary if the word is different from the prediction. This is the reason why the computation of final LM probabilities is more expensive for unigram and bigram than for trigram LM look-ahead.

As shown in Table 5, trigram based LM look-ahead has better performance in terms of computational time than bigram based maximization. The WER is also improved, but the difference is very small and should not be considered relevant. Due to the efficient calculation of the look-ahead the extra time expended in the computation of the trigram LM based prediction is compensated by the decrease in the cost of the token propagation. For unigram based prediction the real time factor and word error rate are, as expected, worse than in the other cases.

However, the difference in real time factor between bigram and trigram based prediction is quite small for this particular experiment. This is a consequence of the limited reduction in search complexity that has been obtained. Using trigram based look-ahead only a 6% reduction of active tokens per frame is achieved. As a result, the faster token propagation achieved is solely compensated by the extra cost of the LM look-ahead. Therefore the global reduction in real time factor observed in Table 5 may be attributed to the speed-up in the calculation of the final LM probabilities. We suspect that this result is mainly related with the poor quality of the used LM. In other experiments with a higher decrease in the number of active tokens per frame, greater improvements may be obtained.

5. CONCLUSIONS AND FURTHER WORK

In this paper we have described an efficient method to speed-up the computation of the LM look-ahead in a single lexical tree decoder. We have shown that combining a node cache strategy with the pre-computation of the probabilities of active LM contexts and a highly optimized LM access using perfect hash techniques, the use of trigram based LM prediction is allowed at no extra cost.

In order to obtain more advantageous results, further work should be done to speed up the probability array calculation, as the other two layers presented in this paper (cache access and probability maximization) already perform fast and consistently regardless of the n-gram order (Tab. 2). This could be done by exploiting the order preserving property of perfect hashing to seek into the first bigram and trigram needed and to get the following ones from consecutive positions without hashing, or by devising a way so that the probabilities are calculated and stored dynamically as they are needed.

It could also be attempted to extend the LM context creation so that it also pre-calculates and stores the LM maximums for each node. This would eliminate the need for both cache and maximization at the cost of a slower context creation and higher memory usage for them.

6. REFERENCES