# NOZOMI – A FAST, MEMORY-EFFICIENT STACK DECODER FOR LVCSR

*Mike Schuster*

ATR, Interpreting Telecommunications Research Lab.
2-2 Hikari-dai, Seika-cho, Soraku-gun, Kyoto 619-0288
gustl@itl.atr.co.jp http://www.itl.atr.co.jp/

## ABSTRACT

This paper describes some of the implementation details of the "Nozomi" [1] stack decoder for LVCSR. The decoder was tested on a Japanese Newspaper Dictation Task using a 5000 word vocabulary. Using continuous density acoustic models with 2000 and 3000 states trained on the JNAS/ASJ corpora and a 3-gram LM trained on the RWC text corpus, both models provided by the IPA group [7], it was possible to reach more than 95% word accuracy on the standard test set. With computationally cheap acoustic models we could achieve around 89% accuracy in nearly realtime on a 300 Mhz Pentium II. Using a disk-based LM the memory usage could be optimized to 4 MB in total.

## 1. INTRODUCTION

LVCSR is currently limited to workstations and fast high-end laptops with a lot of memory. To make LVCSR work on PDAs, cellular phones, user-interfaces, wrist watches etc., it is necessary find time- and memory-efficient algorithms. The goal for implementation of any search engine must be to minimize **time** and **memory requirements** as well as the overall **complexity** of the system while maximizing its **flexibility** using all available knowledge sources (pronunciation dictionary, N-gram LM etc.) to search for the desired output.

There are several approaches to decoding, which can be distinguished by their basic search strategy: a) the time-synchronous *transition network decoders* and the usually time-asynchronous *stack decoders*. Stack decoders [3, 5] can be defined as decoders that use during decoding some kind of a *stack* of partial sentence hypotheses each consisting of a certain number of words. In general the partial hypotheses on a stack are expanded by complete words time-synchronously using the pronunciation dictionary to create new partial hypotheses which are inserted into other stacks. When all stacks but the last (result stack) are empty, the result stack will contain the first best hypothesis, the N-best hypotheses or the respective lattices depending on the search mode. Stack decoders operate at least on two levels of search: a) the outer level, which loops over the stacks (*word-level search*), and b) the inner level, which loops over time and states to search for complete words, starting from the end-time of the hypothesis to expand, which is called *state-level search* or word-within search. Every time a word-end is found during the time-synchronous word-within search, its language model score is looked up using the found word plus its history using the hypotheses which are to be expanded. Because the dynamic LM score lookup can take any word history into account, stack decoders can easily make use of any kind of N-th order Markov language model and also of non-Markov language models like link grammars etc. Especially N-gram models of any order are simple to implement, which is one of the major advantages over the transition network decoders.

In this paper, based on a time-asynchronous stack decoder framework, it is shown how it is possible to handle arbitrary order N-grams, how to generate N-best lists or lattices next to the first best hypothesis at almost no computational overhead, how to handle efficiently cross-word acoustic models of any context order, how to efficiently constrain the search with word graphs or word pair grammars, and how to use a fast match with delay to speed up the search, all in one left-to-right search pass. The details of a disk-based representation of an N-gram language model are given, which make it possible to use LMs of arbitrary (file) size in only a few hundred kB of memory.

## 2. A ONE-PASS STACK DECODER

The decoder described here is in its basic implementation similar to the approach described in [5] and [6], which should be consulted for the basic search strategy. Because of space limitations this paper concentrates on the description of some of the decoder modules and issues, which were found to be important for time- and memory-efficient performance.

### 2.1. Stack module

The collection of *stacks* for each time $t$ are accessed by PUSH() and POP() operations taking partial hypotheses as arguments. Because they are used frequently and usually contain a few to several hundred entries in a typical application, the *stacks* (or more precisely lists, because access to their elements is random and not based on a LIFO concept) have to be set up efficiently. The container types used in other decoders are often special tree-structured lists, which are ordered by score and limited in the number of entries. Here a different method is described which was found to be most efficient and simple to implement.

Pushing a hypothesis on a stack involves a check whether a hypothesis being in the same LM state is already on that stack. If yes, the scores of the two hypotheses are compared

[1] "Nozomi" is the name of the fastest, most comfortable and most expensive bullet train in Japan, and also means "hope" in Japanese

and the better one is inserted into the stack, the other one discarded. In case of an N-gram LM the LM state check means to compare the last $MAX(N \Leftrightarrow 1, 1)$ history word IDs. One word has to be compared as a minimum to not violate the at least first order Markov assumption for the complete speech model. Although checking for LM state equivalence for N-gram LMs can theoretically be done in $O(1)$ using a hash table with the $N \Leftrightarrow 1$ words history as the key, it was found that it is in practice not more efficient than a simple non-ordered unlimited list that is searched through linearly up to an average stack size of a few hundred hypotheses. Pushing a hypothesis on a stack can also improve the upper bound for the score at this time, which has to be checked for. Popping a hypothesis from a stack is an $O(1)$ process, since it doesn't matter in what order the hypotheses in beam are extended for the implementation described here.

#### *2.1.1. Lattice generation*

Stack decoders can easily generate lattices at little computational overhead in the first pass by slightly modifying the LM state check procedure. Instead of discarding the worse hypothesis in case of LM state equivalence it can be linked into the lattice. A pointer on the best arc back has to be updated to not loose the best hypothesis for the current LM state and future reference. Compared to the generation of the first best hypothesis there is only little overall increase in memory for the storage of the additional arcs in the lattices (section 3.).

#### *2.1.2. N-best list generation*

The hypotheses in an N-best list differ by at least one word ID. This can directly be checked for by extending the LM state check to the complete history instead just the $MAX(N \Leftrightarrow 1, 1)$ history word IDs like necessary for obtaining the first best hypothesis. It can be done either exactly by checking each word, or approximately by using a hash function for the history. A lattice within the N-best list, referred to as N-best lattice, which includes all possible alignments and pronunciation variants for the same word ID sequence in the possible paths taken backwards from a lattice node, can be produced by merging hypotheses instead of replacing them like discussed above for the first-best lattices. Compared to the lattice generation this procedure uses only little additional memory for the extra nodes of the hypotheses, which are needed because of the increased LM state space, and only little additional time. Since for the generation of N-best lists only the LM state check procedure was modified, they can be generated in the first pass like lattices.

### 2.2. N-gram module

A efficient format for the LM was found to be the following: For a back-off $N$-gram store all $n$-grams with $n = 1, 2, \ldots, N$ in a table for each $n$. Each entry in a table has a word-ID, its LM probability and back-off probability, and a pointer to the beginning of the list of extension word entries in the table holding the $(n+1)$-grams. For the table with the $N$-grams the pointers are not necessary, since no higher order $(N+1)$-grams are following. Each part of an entry table holding a particular set of extension words is ordered by its word-IDs to allow fast access using a binary search. The number of a set of extension words on any level $n$ doesn't have to be stored because it can be calculated by subtracting the pointer (on level $n \Leftrightarrow 1$) on the current set from the next pointer (also on level $n \Leftrightarrow 1$) on the next set. If the next set on level $n$ doesn't happen to have any extension words, indicated by a NULL pointer on level $n \Leftrightarrow 1$, the next non-NULL pointer on level $n \Leftrightarrow 1$ has to be searched for, which is on average not more than a few entries away.

The memory requirements for this $N$-gram representation are 8 bytes per entry for all $\{n < N\}$-grams, and 4 bytes for all $N$-grams, assuming 4-byte pointers, 2-byte word IDs and 1-byte representations for the LM probability and the back-off probability, uniformly distributed across their log-scores, which was found to be a sufficient accuracy to not cause any errors. Access time for this storage format is of $O(1)$ for the unigrams and of $O((n \Leftrightarrow 1) \cdot log_2(K))$ for the $\{n > 1\}$-grams using a binary search, with $K$ being the average number of words following any $n$-gram entry. The average access time can be slightly improved by caching LM states and their scores in a hash table for all $\{n > 1\}$-grams that have been accessed before. This improves average access time to $O(1)$ for already used $\{n > 1\}$-grams, but requires an additional check whether a certain LM state is already in the hash table or not.

A disk-based representation of the N-gram can limit memory requirements to a few hundred kB for N-grams of *any* size [8]. The search for the N-gram scores on disk during the search is of course very time-consuming and has to be minimized using an efficient caching scheme. An efficient implementation was found to be the following: Unigrams are stored in memory and all $\{n > 1\}$-grams are stored on disk in the exact same format that was used for the representation in memory, such that looking up an $n$-gram can be done using the same algorithm. A set of extension words following an $n$-gram is loaded into temporary memory to run the binary search for the correct word ID in memory and not on disk. The LM states that have been used once are cached in a memory-based hash table to minimize disk access.

### 2.3. Cross-word models

A procedure to deal with cross-word models of *any* order (triphones, quintphones, etc.) incorporating cross-word effects in a delayed manner was found to be very efficient in time and memory requirements, and is especially well suited for a stack decoder:

- Run the state-level search for any set of hypotheses to expand with only word-internal context-dependent models.
- When popping the hypotheses from a stack to expand, realign the last $M$ words using cross-word models at the word boundaries before entering the state-level search to find the extension words.
- Because cross-word effects are incorporated with a one-word delay, it is also necessary to realign the last $M$ words for all hypotheses on the final result stack.

This procedure as illustrated in Fig. 1 incorporates all cross-word effects within the last $M$ words, and is optimal for cross-word triphones with $M = 2$ for most cases and possibly $M = 3$, if the word before the last word is

a one-phone word. To capture all cross-word effects with quintphones theoretically $M = 5$ is necessary, if all words in the dictionary would be one-phone words.

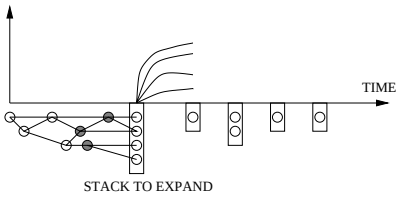


**Figure 1. Visualization of the method to incorporate cross-word models of any context order. Circles denote hyp-nodes, filled circles are the word boundaries that are corrected by the procedure using cross-word models *before* the stack (box) is expanded. In this example only two words are realigned, but there could be more like discussed in the text. The same procedure is used for the fast-match.**

The realignment for each hypothesis to extend is in detail done as follows: Take the last $M$ words and find the correct (cross-word) HMMs for each phone at the word boundaries which don't already cover the maximum available context given the acoustic model set. Use a local Viterbi search to find $M$ new acoustic scores and possibly $M \Leftrightarrow 1$ new word boundaries. Generate $M$ new arcs and $M \Leftrightarrow 1$ new hyp-nodes and replace the old hypothesis end-hyp-node by the new one.

The *correct* cross-word HMM model is defined as the model which covers the most context around the current center-phone. This definition is also used for finding the correct context-dependent HMM within words during construction of the tree lexicon containing context-dependent models given only a monophone pronunciation dictionary.

Compared to the procedure described in [2], which locally rescores every word that is found during the state-level search, the method described here rescores only words that have been found to be considerably likely being part of stacks to expand. The average number of hypotheses to expand per frame is in general between five and one-hundred and cross-word rescoring is only applied to those few. This requires only very little temporary memory and is fast, because of the low number of hypotheses and because of the fact, that most of the states to be evaluated during rescoring for their observation likelihood are already in cache.

A potential drawback of this method is, that because cross-word effects are incorporated delayed, scores might vary more during the lookahead, which might require larger beams than if this delay wouldn't be used.

### 2.4. Fast-match with delay

The method to handle arbitrary cross-word effects is easily extended to allow an efficient acoustic fast-match with a one-word delay, which in a similar form without delay is described in [1, 4]. The basic idea of a fast-match in a stack decoder is to use simple acoustic models to find possible extension words, and rescore them locally with better, but computationally more expensive models. This avoids the use of expensive models for the initial state-level search and can speed up the complete search.

The fast-match procedure described here keeps the use of the expensive models at a minimum and is almost identical with the method to incorporate cross-word models. Instead of using word-within context-dependent (CD) models for the state-level search, simple monophones with a low number of mixtures or small neural-network based models are used in a context-independent tree-lexicon, and the found words are inserted in the corresponding stacks. Rescoring of the last $M$ words including all cross-word effects is done later using the accurate, but expensive CD models, but only when a stack is expanded, such that many of the previously found words will be out of the beam. The difference to the cross-word procedure from section 2.3. is, that *all* phones of the last $M$ words have to be mapped to their correct CD HMM model, and not only the ones at the word boundaries. This can be interpreted as local rescoring with a one-word delay, which limits the number of necessary rescoring turns per frame to less than ten to one-hundred for most applications, and requires very little additional memory.

## 3. EXPERIMENTS

All experiments were conducted using the described one-pass stack decoder for the recognition of read sentences from a Japanese newspaper using a 5000 word pronunciation dictionary with on average 1.5 pronunciations per word. The acoustic models are gender-dependent decision tree state-clustered Gaussian mixture models trained on 20k sentences per gender from the ASJ and JNAS database. Acoustic preprocessing is standard 12-dimensional MFCCs plus log energy, with applied cepstrum mean subtraction per sentence and first derivatives every 10 ms. A trigram LM was trained on around 45 million words from the RWC corpus containing four years of newspaper articles from the Mainichi Shinbun, a daily newspaper in Japan. The standard test data are the first ten sentences from the speakers 006, 014, 017, 021, 026, 089, 102, 115, 122 from the JNAS database. All acoustic models, initial language models and the pronunciation dictionary were kindly provided by the IPA group, who also defined the test set [7].

Tab. 3. shows the results, for which the search parameter settings were optimized to reach a low word error rate. The experiments of this task were run in two modes, a

| MODEL | MALE Kat/Kan | FEMALE Kat/Kan | RTF |
|---|---|---|---|
| 129 x 16 | 88.7/87.5 | 91.8/90.8 | 9 |
| 2000 x 16 | 95.2/93.3 | **96.9/95.2** | 22 |
| 3000 x 16 | **96.4/94.8** | 95.9/94.5 | 23 |
| 129 x 16 | 87.9/86.7 | 91.0/90.0 | 9 |
| 2000 x 16 | 94.4/92.6 | **96.1/94.4** | 22 |
| 3000 x 16 | **95.6/94.0** | 95.0/93.6 | 23 |

**Table 1. Recognition results for high accuracy, cleaned of errors that shouldn't be counted in Japanese (upper) and not cleaned (lower), for Katakana (Kat) and Kanji (Kan) recognition mode. Cross-word modeling was used.**

*Katakana* mode, where all word-IDs and all transcriptions

are written only in Katakana, and in a *Kanji* mode, where all word IDs and transcriptions are written like they occur in a newspaper. Best recognition results in Kanji recognition mode are 5.2% word error rate (WER) for the male using 3000 state models and 4.8% WER for the female speakers using 2000 state models, if the results are cleaned from errors that shouldn't be counted as errors in Japanese, which can be classified into two types. Type I errors are due to the fact that there are no spaces in a regular Japanese text, which were artificially introduced to define words to build a pronunciation dictionary and a LM. This leads to ambiguous word definitions and many errors of the kind: 'a' 'while' ⇔ 'awhile'. Also, in Japanese it is common and correct to write many words with the exact same pronunciation and meaning using different symbols, which occurs in English only for numbers (Type II errors). The raw outputs from the recognizer are about 15% relative (1% absolute) worse, showing that these errors, which are specific to Japanese, shouldn't be neglected. The Katakana results, which hide misrecognition of homonyms occurring in Japanese more frequently than for example in English, overestimate the score of interest on average by about 1% absolute.

Tab. 3. shows results for experiments that were run to maximize decoding speed at a low (around 1%) search error and minimize memory requirements, with (a) a regular memory-based trigram LM and (b) a disk-based LM. Almost realtime performance including all observation likelihood calculations is possible with around 11% word error rate using 10 MB of total memory. The disk-based LM slows down the search by about a factor of three for the monophones. The realtime factor and memory requirements for all results are for a 300 MHz Pentium II.

| MODEL | M Kat | F Kat | MEM (MB) | RTF |
|---|---|---|---|---|
| 129 x 16 | 87.0 | 90.2 | 10 | 1.3 |
| 2000 x 16 | 93.0 | 94.6 | 20 | 9 |
| 2000 x 16 (fast-match) | 93.0 | 94.6 | 20 | 7 |
| 129 x 16 | 87.0 | 90.2 | 4 | 3.9 |
| 2000 x 16 | 93.0 | 94.6 | 14 | 14 |

**Table 2. Results for high speed and low memory, with memory-based LM (upper) and disk-based LM (lower), not cleaned of errors that shouldn't be counted as errors in Japanese. Cross-word models were used. Fast-match models were 3-state monophones with four mixtures each.**

The results shown in Tab. 3. compare the time and memory requirements for generating the first best hypothesis with the time for generating lattices or N-best lists in the first pass. It can be seen that the more complicated LM state check for the N-best lists creates only little overhead, and is almost independent of the length of the N-best lists.

## 4. CONCLUSIONS

It can be concluded that a time-asynchronous stack decoder is a conceptually attractive framework for integrating many often needed procedures for speech recognition tasks. Although efficient in memory and faster than the decoder mentioned in [7] for the same task, it should be noted that the speed of a time-asynchronous stack decoder like implemented here is probably not optimal for the specific task of generating a first-best hypothesis or a lattice from a feature vector sequence, because the globally time-asynchronous search over the state space results in the generation of many (about 90%) later not expanded partial hypotheses. This could be avoided by using a time-synchronous stack decoder with multiple trees, which hasn't been tried here.

| SEARCH MODE | RTF | MEMORY |
|---|---|---|
| first best (absolute) | 9 | 20 MB |
| first best | 100% | 100% |
| lattice | 107% | 106% |
| N-best list, N = 10 | 113% | 100.4% |
| N-best list, N = 50 | 116% | 100.4% |
| N-best list, N = 100 | 117% | 100.5% |

**Table 3. Relative time and memory (as measured by the UNIX `top` command) for several search modes with beams leading to lattices of about 2500 arcs and 500 hyp-nodes, and an average N-best list length of 90 hypotheses.**

## 5. ACKNOWLEDGMENTS

This work wouldn't have been possible without the support from the IPA group [7]. Prof. Shikano from NAIST pointed out the specific importance of cross-word modeling for Japanese.

## REFERENCES

[1] L.R. Bahl, P.V. de Souza, P.S. Gopalakrishnan, D. Nahamoo, M. Picheny, "A fast match for continuous speech recognition using allophonic models", in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, pp. I-17 - I-20, 1992.

[2] L.R. Bahl, P.V. de Souza, P.S. Gopalakrishnan, D. Nahamoo, M. Picheny, "Word lookahead scheme for cross-word right context models in a stack decoder", in *Proc. Eurospeech*, pp. 851-854, Berlin, Germany, 1993.

[3] P.S. Gopalakrishnan, "A tree search strategy for large vocabulary continuous speech recognition", in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, pp. 572-575, 1995.

[4] P.S. Gopalakrishnan, L.R. Bahl, "Fast matching techniques", in *Automatic Speech Recognition: Advanced Topics* Eds. Boston: Kluwer Academic Publishers, 1996.

[5] S. Renals and M. Hochberg, "Decoder technology for connectionist large vocabulary speech recognition", Technical Report CUED/ F-INFENG/ TR.186, Cambridge University, England, 1995.

[6] M. Schuster, "Nozomi - a fast, memory efficient one-pass stack decoder", ASJ spring meeting 1997, pp. 155-156, Yokohama, Japan, 1997.

[7] T. Kawahara, et al, "Common platform of Japanese large vocabulary continuous speech recognizer assessment – proposal and initial results", Proc. EALREW-98, pp. 117-122, Tsukuba, Japan, 1998.

[8] M.K. Ravishankar, "Efficient Algorithms for Speech Recognition", Doctor Thesis, Technial Report CMU-CS-96-143, Pittsburgh, USA, 1996.