JOINING ADVANTAGES OF WORD-CONDITIONED AND TOKEN-PASSING DECODING

David Nolden, David Rybach, Ralf Schlüter, Hermann Ney
Chair of Computer Science 6, RWTH Aachen University
Ahornstr. 55
D-52056 Aachen, Germany
{nolden, rybach, schlueter, ney}@cs.rwth-aachen.de

ABSTRACT

We compare the families of token-passing and word conditioned decoders, and derive a more efficient dynamic decoder. The advantage of the word conditioned approach is a trivially simple hypothesis recombination, while the advantage of the token-passing approach is a straight-forward minimization of the search network with compressed word tails. We derive a dynamic decoder which joins the advantages of both decoding architectures by minimizing the search network of a word conditioned decoder. We describe the decoder and analyze its efficiency regarding acoustic look-ahead, network minimization, and facilitated pruning methods. Finally, we compare the new decoder with a WFST based decoder extended by acoustic look-ahead.

Index Terms— LVCSR, search, decoding, word conditioned, tree-search, token-passing, WFST, pruning, minimization, acoustic look-ahead

1. INTRODUCTION

In dynamic network decoders, the language model (LM) and acoustic model (AM) are combined dynamically. The acoustic model is used to build a compact HMM search network representing all words in the vocabulary, and the LM dependencies are maintained by appropriate management of state hypotheses [1].

Static decoders on the other hand, usually based on the weighted finite state transducer (WFST) approach [2], combine the AM and LM statically by constructing one huge search network integrating both models.

WFST based decoders are popular because the static integration of knowledge sources allows shifting some effort from the runtime to a preprocessing step, most importantly the calculation of LM- and look-ahead scores. However the preprocessing is very expensive for large models, and the resulting static search network takes a huge amount of memory. Alternatively only the encountered parts of the search network can be expanded on-demand [3], which neutralizes a part of the runtime efficiency advantage though. The search network of WFST decoders is usually only expanded up to the allophone level, therefore an additional expansion from allophones to HMM states is required while decoding, and the state recombination is suboptimal.

For these reasons, dynamic network decoding stays an interesting approach. The authors of [4] have shown that a well-crafted dynamic network decoder based on token-passing is competitive with WFST decoders regarding runtime when large LMs are used. Meanwhile the weakness of dynamic decoders regarding LM look-ahead was further compensated [5] and extended acoustic look-ahead was introduced [6], both of which further boost the efficiency of dynamic network decoders.

State-of-the-art dynamic decoders are typically divided into two families with slight differences: Token-passing decoders [7] [4] and word conditioned decoders [1]. Both are theoretically equivalent regarding the search space, but have different implications regarding the efficiency.

In token-passing decoders state hypotheses are grouped by their network-state. This allows straight-forward minimization of the search network with compressed word tails, which leads to a much smaller search network, an earlier recombination of state hypotheses, and less effort handling word-ends, all of which increases the efficiency. The recombination of hypotheses is difficult though and an additional pruning is required to make the recombination efficient.

In word-conditioned decoders state hypotheses are grouped by their LM context. This allows very efficient hypothesis recombination, but the minimization of the search network is difficult.

In this work we try to join the advantages of both approaches into one decoder. In Section 2 we compare both decoding architectures, in Section 3 we describe the new decoder, and in Section 4 we analyze it experimentally, comparing it with the WFST approach.

2. COMPARISON OF TOKEN-PASSING AND WORD CONDITIONED SEARCH

In token-passing decoders, a list of state hypotheses (tokens) with equal network state but different LM contexts is attached to each state of the search network.

In word conditioned decoders, virtual copies of the search network are created for each encountered LM context, and a list of state hypotheses with equal LM context but different network states is attached to each copy.

Let (h, s, q) be an active state hypothesis with LM context h, network state s and probability q. The fundamental difference between token-passing and word conditioned decoders is: In a token-passing decoder, state hypotheses are grouped by equal network state s, while in a word conditioned decoder, state hypotheses are grouped by equal LM context h. This storage layout leads to different implications regarding efficiency.

Table 1 shows a quick comparison of the practical advantages and disadvantages, explained in the following sections.

2.1. State Hypothesis Recombination

During decoding, when two state hypotheses (h, s, q) and (h', s', q') share the same LM context $h = h'$ and network state $s = s'$, then the state hypothesis with the lower probability q is discarded, according to the Viterbi approximation. Hence, the decoder needs to match hypotheses with equal LM context and state.

This work was partly realized under the Quaero Programme, funded by OSEO, French State agency for innovation.

Table 1. Overview of advantages (+) and disadvantages (-).

Word Conditioned	Token-Passing
State hypothesis recombination	
+Trivial	−Tricky
Search network minimization	
− No	+Yes
LM state pruning (requires minimization)	
+Not required	−Required
−Not effective	+Effective

In word conditioned decoders all state hypotheses with equal LM context are grouped together within one virtual network copy. To perform recombination, all state hypotheses within the network copy which share the same state s need to be identified. Since the number of states S is typically not too high, the matching can be performed very efficiently network-copy wise using a simple table of size S. The costs of table lookups are negligible if the state indices are sorted topologically, due to the CPU cache.

In token-passing decoders hypotheses are grouped by their state s, and during recombination, the hypotheses assigned to one state s which share the same LM context h need to be matched. If the LM and vocabulary are large, the number of possible contexts h is huge, and sophisticated data-structures or algorithms are required to perform the matching of equal contexts, for example hash-structures or heaps. Therefore it is critical to keep the number of hypotheses assigned to one state low using *LM state pruning*. In [4] a simple linear search is used to perform the matching, which is more efficient than sophisticated structures if the number of hypotheses on states is very low.

2.2. Search Network Minimization

The search network of token-passing decoders is typically minimized by pushing word labels from the end into the body of the network and by compressing the suffixes. Thereby the size of the network and the number of word-end labels attached to the network is greatly reduced, the runtime effort of word-end handling is reduced, and full LM scores are applied earlier leading to more precise pruning. Furthermore different search paths start overlapping already before the physical word ends are reached, which leads to earlier recombination and effective pruning of overlapping paths using LM state pruning.

Classical word conditioned decoders do not minimize the search network.

3. DECODER

A decoder which joins the advantages of token-passing decoders and word conditioned decoders (see Table 1) needs the following attributes: Trivial recombination of state hypotheses, a minimized search network, and LM state pruning is possible, efficient and effective but not mandatory.

We start with our word conditioned decoder to leverage the efficient hypothesis recombination. Full order LM look-ahead is integrated efficiently by exploiting the sparseness of backing-off n-gram LMs [5], and acoustic look-ahead is used to focus the search regarding future acoustic observations [6].

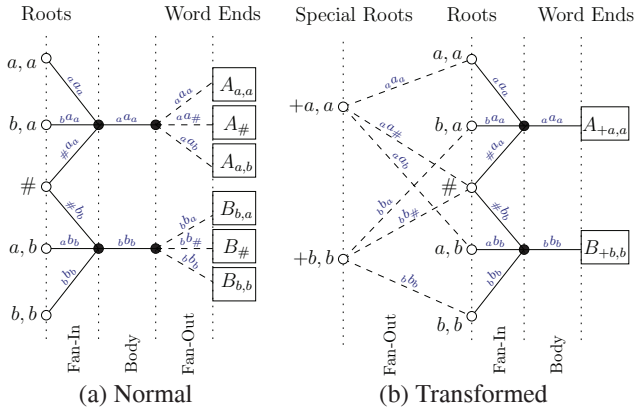
3.1. Search Network Structure

The normal tree-like HMM search network can be split into three parts: A minimized fan-in which models the across-word coarticulation at word starts, a tree-like body following the fan-in, and a coarticulated tree-like fan-out before the word ends. If the phonetical context-dependency is limited to two neighbor phonemes (triphones) as in our decoder, then the fan-in ends after the first phoneme, and the fan-out starts one phoneme before the word ends (see Figure 1a). The network is expanded and compressed up to the HMM state level (Figure 1 shows only the allophone arcs as a simplification).

At the end of the fan-out, there is one word end label for each word and for each possible successor phoneme. Each word end label is annotated with the matching coarticulated root state where the acoustic search path continues. Any number of labels may be attached to one HMM state. The overall size of the fan-out is linear in the size of the vocabulary and the number of phonemes, thus the fan-out can dominate the whole search network when large vocabularies are used.

3.2. Search Network Minimization

The fan-out can be minimized by pushing the word end labels before the fan-out, moving their fan-out suffix into a new minimized fan-out structure prepended before the fan-in, and inserting new special roots before the minimized fan-out structure (see Figure 1b). The resulting search network is equivalent, but contains only one word end label for each word, and the whole fan-out is minimized. For true minimization of the whole search network, the word end labels could be pushed further into the body, however we expect no further significant improvement (according to [4] only the minimization of the fan-out is worthy regarding runtime efficiency).



(a) Normal (b) Transformed

Fig. 1. Suffix-sharing transformation of the search network demonstrated with 2 words A and B and 2 phonemes a and b (with pronunciations $A = aaa$ and $B = bbb$).

Such a minimization leads to all advantages of token-passing decoders enumerated in Subsection 2.2, but also introduces some problems which need to be worked around.

The transformed network is equivalent regarding the probabilities assigned to whole search paths, but when lattices are created, exact partial probabilities and time-marks may be required for individual words in the lattice. Word end labels are now encountered one phoneme before the physical word end, thus the tracebacks are annotated with time-marks and probabilities shifted by one recognized phoneme. We correct the lattices by checking when exactly each traceback crosses from the fan-out into the fan-in for the first time, and updating the time- and score marks accordingly (this works without measurable overhead).

A further problem arises from the earlier emission of word ends: More different LM contexts are encountered, and thus more LM look-ahead tables are required. We circumvent this problem by using only unigram look-ahead as long as the fan-in of a network copy was not entered (this measure improves the efficiency by approximately 4%).

3.3. Pruning

The most important pruning method is the global acoustic pruning: At each timeframe, all state hypotheses which have a lower probability than the best one multiplied by a specific pruning threshold are discarded.

Since very effective look-ahead knowledge sources are available (regarding LM as well as AM), an additional acoustic pruning step using the same threshold is perfomed after expanding the HMM states, but *before* computing acoustic scores (we call this *early acoustic pruning*), to prevent expensive acoustic score calculations.

Whenever a word end label is encountered during decoding, a word end hypothesis is created and the matching coarticulated root state with extended LM context is activated. Since new LM look-ahead tables need to be initialized for each new unique LM context, it is very important to keep the number of word end hypotheses low. Therefore, word end hypotheses are pruned at each timeframe by discarding all word end hypotheses which have a probability lower than the best one multiplied by a specific pruning threshold [1] (we call this *word end pruning*).

Additionally to acoustic pruning and word end pruning, *histogram pruning* is used to limit the absolute number of state- and word-end hypotheses that appear at each timeframe. This pruning variant is mainly required to cut off peaks in situations of high uncertainty, and does not play a significant role regarding the runtime efficiency.

Typically a significant amount of time is spent handling word end hypotheses (looking up LM scores, extending tracebacks, LM recombination, pruning, etc). We showed in [8] that preventing transitions into word starts at a specific fraction of all timeframes does not increase the WER. We transfer this concept into the word conditioned decoder by only handling word ends at each ith timeframe (we call this *word end interval*).

3.4. LM State Pruning

LM state pruning as known from token-passing decoders can be integrated efficiently by using a single table of size S, and two runs over all active state hypotheses: In the first pass, the table is used to store the highest probability for each network state. In the second run, all state hypotheses are discarded which have a lower probability than the stored best one in the same network state multiplied by a specific pruning threshold.

4. EXPERIMENTAL RESULTS

We perform our experiments on the first speaker-independent pass of the RWTH Aachen Quaero English ASR system [9]. The lexicon comprises 158k words with 180k pronunciations, modeled by 45 phonemes and 6 non-speech phones, and the 4-gram LM is composed of 50M n-grams. The acoustic model comprises 4501 Gaussian mixture models with a globally tied covariance matrix and 1M mixture densities. The test corpus consists of 1482 segments with a duration of 3.4h and about 36k spoken words.

The acoustic scores are computed efficiently using quantized features, temporal batching, and Gaussian preselection. The Gaussian densities are clustered into 256 clusters and at each timeframe only the closest 32 clusters are considered (batching and preselection together reduce the effort of acoustic scoring to approximately one third at equal error rate).

The WFST decoder used for comparison was introduced in [10] and significantly outperformed the old dynamic network decoder on smaller tasks. It is based on OpenFST and combines the $C \circ L$ and G transducers dynamically, and is well-tuned for the Quaero English system.

Real time factors (RTF) were measured on a 8-core AMD Opteron 4130 machine with 2.6Ghz and 32GB of memory (without parallelization).

4.1. Minimization

The classical search network consists of 7.8M HMM states and 8.1M word-end labels, and takes about 160MB of memory. By minimization, the search network is reduced to 1M HMM states and 267k word-end labels, taking about 12MB of memory.

4.2. LM State Pruning

We intensively tested the potential effect of LM state pruning in conjunction with a minimized search network. If word end pruning is fixed to some suboptimal value, then the LM state pruning can achieve a slight efficiency improvement. However if we choose the optimal word end pruning threshold, then we achieve no significant improvement by the additional LM state pruning.

Since word ends are placed right before the fan-out in our decoder, only few paths actually reach the fan-out due to word end pruning. LM state pruning however shows most of its effect in the fan-out, where many different acoustic paths join.

4.3. Pruning Comparison

Table 2 shows the efficiency of the decoder in different configurations, with full acoustic look-ahead. The minimization of the search network leads to an improvement of about 5 to 10% in RTF at equal WER in comparison to the tree-like search network. When adding early acoustic pruning, another 10 to 20% are achieved. By further adding a word end interval of 2, the efficiency is again improved, however mainly for high pruning thresholds and low error rates.

For each acoustic pruning threshold, the optimal word end pruning threshold was chosen beforehand by testing all relevant combinations on a similar dev-corpus and selecting a flattened pareto-frontier regarding RTF and WER.

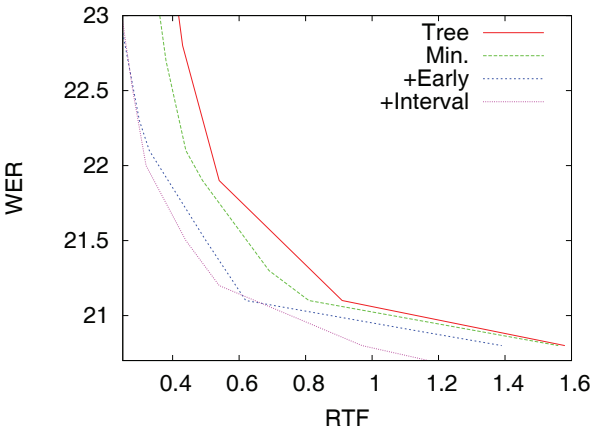


Fig. 2. Varied acoustic pruning and word end pruning.

Table 2 shows the decoder statistics for fixed pruning thresholds very close to the optimum. Less word ends are encountered in the minimized search network, but still more new LM look-ahead tables need to be computed, because the word ends were pushed by one phoneme towards the roots, and thus potentially different words are hypothesized more frequently.

Table 2. Decoding statistics (per timeframe after pruning).

	Tree	Min.	+Early	+Interval
RTF	1.52	1.34	1.13	1.03
WER	20.8%	20.9%	20.9%	20.8 %
States	9.7k	8k	7.8k	7.3k
Word Ends	1300	843	838	402
Look-Ahead Tables	1.8	1.98	1.98	1.82

4.4. Profiling

Table 3 shows the profiling of the decoder for the same pruning thresholds used for the decoder statistics. By applying minimization, basically every single component becomes more efficient. The most significant reduction is achieved in the computation of LM look-ahead tables, probably because the size of the compressed LM look-ahead network is reduced from 304k to 221k nodes. The costs of computing LM look-ahead tables are relatively low, which shows that the limits of the sparse LM look-ahead regarding vocabulary- and LM size are far from being reached. The *Apply Look-Ahead* component accounts for look-up of acoustic look-ahead scores, LM look-ahead scores, and eventually transferring state hypotheses into back-off trees according to sparse LM look-ahead [5]. The costs of the Viterbi search include expansion and recombination of state hypotheses. The costs of reading LM scores are contained in the *Word Ends* component.

Table 3. Profiling.

	Tree	Min.	+Early	+Interval
RTF	1.52	1.34	1.13	1.03
Acoustic Scorer	0.7	0.64	0.43	0.42
Look-Ahead Tables	0.19	0.12	0.12	0.12
Apply Look-Ahead	0.2	0.19	0.19	0.17
Viterbi	0.23	0.2	0.2	0.18
Word Ends	0.16	0.15	0.15	0.08
Other	0.14	0.04	0.04	0.06

4.5. Acoustic Look-Ahead and WFST Search

We have integrated the temporally approximated acoustic look-ahead as proposed in [6] into our WFST based dynamic decoder, the model-based approximation is more difficult to integrate efficiently.

Figure 3 shows a comparison between the best performing configuration of the dynamic decoder and the WFST decoder regarding RTF and WER with and without acoustic look-ahead. Both decoders perform similarly without acoustic look-ahead, and achieve an improvement of 10 to 20% in RTF at equal WER through temporal acoustic look-ahead. The dynamic decoder profits more from the look-ahead than the WFST decoder. The model-approximated acoustic look-ahead gives the dynamic decoder a significant lead, further improving the efficiency by approximately 10%.

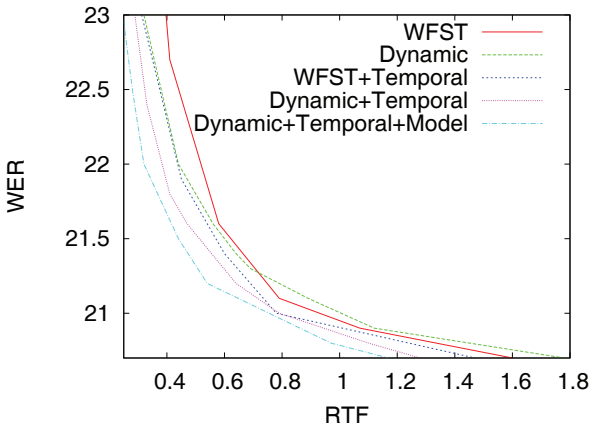


Fig. 3. Efficiency of the dynamic decoder vs. WFST decoder with and without acoustic look-ahead.

5. CONCLUSIONS

Our new decoder joins the advantages of token-passing and word conditioned decoders. The decoder exploits minimization of the search network, but it does not require LM state pruning. The recombination of state hypotheses using a table is simpler and more efficient than the LM context matching required in token-passing decoders.

Without acoustic look-ahead, our new dynamic decoder and our WFST based decoder perform very similarly on our test task regarding runtime and precision. The efficiency of the WFST decoder was improved by temporal acoustic look-ahead, but the dynamic network decoder gained more, especially when adding the model-based look-ahead.

WFST decoders and modern dynamic decoders are similarly efficient. The disadvantage of the WFST decoder is that it potentially requires more memory to manage the search network, while the disadvantage of the dynamic decoder is that there are more tunable decoder parameters affecting the efficiency.

6. REFERENCES

[1] H. Ney and S. Ortmanns, "Progress in dynamic programming search for lvcsr," in *Proceedings of the IEEE*, Barcelona, Spain, August 2000, vol. 88, pp. 1224 – 1240.

[2] M. Mohri, F. Pereira, and M. Riley, "Speech recognition with weighted finite state transducers," in *Handbook of Speech Processing*. 2008, pp. 559–582, Springer.

[3] C. Allauzen, M. Riley, and M. Mohri, "A generalized composition algorithm for weighted finite-state transducers," in *Interspeech*, Brighton, U.K., September 2009, pp. 1203 – 1206.

[4] H. Soltau and G. Saon, "Dynamic network decoding revisited," in *ASRU*, 2009.

[5] D. Nolden, H. Ney, and R. Schlüter, "Exploiting sparseness of backing-off language models for efficient look-ahead in lvcsr," in *ICASSP*, Prague, Czech Republic, May 2011.

[6] D. Nolden, R. Schlüter, and H. Ney, "Acoustic look-ahead for more efficient decoding in lvcsr," in *Interspeech*, Florence, Italy, August 2011.

[7] S. J. Young, N. H. Russell, and J. H. S. Thornton, "Token passing: a simple conceptual model for connected speech recognition," in *Tech. Report*, Cambridge University Engineering Department, 1989.

[8] D. Nolden, H. Ney, and R. Schlüter, "Time conditioned search in automatic speech recognition reconsidered," in *Interspeech*, Makuhari, Japan, September 2010.

[9] M. Sundermeyer, M. Nußbaum-Thom, S. Wiesler, C. Plahl, A. El-Desoky Mousa, S. Hahn, D. Nolden, R. Schlüter, and H. Ney, "The rwth 2010 quaero asr evaluation system for english, french, and german," in *ICASSP*, Prague, Czech Republic, May 2011, pp. 2212–2215.

[10] David Rybach, Ralf Schlüter, and Hermann Ney, "A comparative analysis of dynamic network decoding," in *ICASSP*.