PROGRESS IN DYNAMIC NETWORK DECODING

David Nolden [1,2]*, Hagen Soltau* [2]*, Hermann Ney* [1]

[1] RWTH Aachen University, Ahornstr. 55, 52056 Aachen, {nolden,ney}@cs.rwth-aachen.de
[2] IBM T. J. Watson Research Center, Yorktown Heights, NY, 10598, hsoltau@us.ibm.com

ABSTRACT

We show how we boosted the efficiency of the dynamic network decoder in IBM's Attila speech recognition framework, by transforming the underlying concept from token-passing to word-conditioned, and adding speedup methods like sparse LM look-ahead. On several different tasks, we achieve improvements of 30 to 50% in efficiency at equal precision. We compare the efficiency to a state-of-the-art WFST based static decoder, and note that the added methods improve the dynamic decoder under conditions where it was lacking before in comparison, specifically when using a relatively small LM. Overall, the new dynamic decoder performs similarly to the static decoder, with a lead for the dynamic decoder on tasks with a larger LM, and a lead for the static decoder on tasks with a smaller LM.

Index Terms— Decoding, dynamic, static, progress, token-passing, word conditioned

1. INTRODUCTION

Decoders comprising a statically compiled and optimized search network are commonly believed to be the fastest approach to decoding in large vocabulary continuous speech recognition [1, 2]. In such decoders the different knowledge sources are represented by weighted finite state transducers (WFST), which are composed and optimized in a preprocessing step to form a unified static search network. This allows early path recombination, language model (LM) lookup, and LM look-ahead, without inducing any runtime overhead [3]. Unfortunately such a statically composed network may become too large when using a large LM, in which case a pruned LM has to be used for the composition. The static decoder then needs to generate lattices, which can be rescored using the unpruned LM in a succeeding pass. Lattice generation in a WFST based decoder is nontrivial and typically induces a significant runtime overhead.

Dynamic network decoders on the other hand statically compose a search network representing only the single words in the vocabulary, and integrate the LM on-demand during decoding [4].

Later dynamic decoders based on the WFST paradigm using dynamic composition were proposed [5]. The dynamic composition may become a major bottleneck though, especially when using a large vocabulary, and effective pruning methods are more difficult to apply than in classical dynamic network decoders [6].

Classical dynamic network decoders combine the single-word search network and the LM without relying on the WFST composition paradigm. The single-word search network may still be minimized to optimize the efficiency and memory usage [2, 7]. Since the topology of the single-word search network is static, the knowledge about path recombination in the network can be exploited to implement effective pruning methods like word end pruning and LM state pruning [8, 9].

Dynamic network decoders can typically be split into two categories: Token-passing decoders [10, 2] and word-conditioned decoders [4]. Theoretically both have the same search search space, although the search space has a different topology, with different implications regarding efficiency. In token-passing decoders the hypotheses are grouped by the state in the single-word search network which they correspond to. Such grouping allows straight-forward minimization of the search network with early path recombination. The recombination of hypotheses is difficult though, and an additional pruning step is required to constrain the effort of state hypothesis recombination. In word-conditioned decoders on the other hand, hypotheses are grouped by their LM context in virtual tree copies. This allows for very efficient hypothesis recombination, however it complicates the minimization of the search network.

A typical weakness of dynamic network decoders is the LM look-ahead: To apply correct full-order LM look-ahead, one look-ahead table with a size linear to the vocabulary needs to be computed for each observed LM context [11]. Recently we proposed sparse LM look-ahead [12] as a solution to this problem for word conditioned decoders. Sparse LM look-ahead limits the effort to those words which are actually covered by the n-gram LM in the corresponding context, instead of the full vocabulary.

In [7] we presented a detailed comparison of the word conditioned and token-passing approach, and we combined the advantages of both, by introducing search network minimization, and LM state pruning, into a word conditioned decoder. In this work we do the opposite: We start with an effi-

978-1-4799-2893-4/14/$31.00 ©2014 IEEE

cient token-passing decoder, and turn it into a decoder which combines the advantages of both architectures, including the thorough pruning of the token-passing decoder, the efficient hypothesis recombination of a word conditioned decoder, and sparse LM look-ahead.

In Section 2 we briefly compare the token-passing and word conditioned decoding concept, in Section 3 we describe the used token passing decoder, in Section 4 we describe how we turned it into a word conditioned decoder, and in Section 5 we evaluate the improvements experimentally against an efficient WFST based static decoder.

2. THEORETICAL COMPARISON

As described in [7], the fundamental difference between a token passing decoder and a word conditioned decoder is the organization of the active search space. Let (h, s, q) be an active state hypothesis with LM context h, network state s, and probability q. In a token-passing decoder the state hypotheses are grouped by network state s, whereas in a word conditioned decoder they are grouped by LM context h.

For state hypothesis recombination according the the Viterbi approximation, different state hypotheses (h, s, q) and (h', s', q'), which share the same LM context $h = h'$ and the same network state $s = s'$, need to be identified, and only the one with the higher probability retained. In a word conditioned decoder, the hypotheses are grouped by their LM context h, and thus only the state s needs to be matched during recombination, which can be done in linear time using a simple table of size S (eg. the number of states in the network). In a token-passing decoder hypotheses are grouped by their state s, and during recombination, equal histories h need to be matched. This matching can not be performed using a simple table, because the number of different histories can be arbitrarily large when using a large LM. In our basic token-passing decoder a simple linear search is performed to do the matching, which is more efficient than complex data structures, given that the number of different LM contexts sufficiently is low [2]. Due to the more efficient hypothesis recombination, we expect a word conditioned decoder to be more efficient than a token passing decoder, at least if it is based on an equally optimized search network and exploits the same pruning methods.

3. TOKEN PASSING DECODER

Our basic token passing decoder was introduced in [2]. It is part of IBM's Attila speech recognition toolkit [13], and uses a single-word search network fully minimized up to HMM state level. Additionally the network is factorized, eg. linear HMM state sequences are combined into a single node. Within these nodes a more efficient specialized dynamic programming algorithm, with trivial hypothesis recombination, can be applied. Only loop and forward transitions are supported by the models, and the last state of each node is a virtual final state without an actual model, only used as origin for the outgoing cross-node transition. Usually this factorization reduces the size of the network and the costs of cross-node transitions by a large factor. Transition across nodes are implemented by an ϵ transition from the final state of each node into the initial state of all successor nodes. The recombination of LM contexts is done during the cross-node transitions, thus the cross-node expansion is one of the most expensive phases during decoding. States within a factorized node share many important attributes like the LM look-ahead table entry, thus the factorization also helps reducing the amount of repetitive costly lookup of redundant information. The decoder stores a heap of LM conditioned instances in each node, and the full sequence of HMM state hypotheses is allocated for each active node instance, whether the actual hypothesis is active or not.

3.1. Pruning

The score of the best HMM state hypotheses in each node instance is recorded on-the-fly, and many operations which would normally require looking at all HMM state hypotheses, like for example LM state pruning, can be performed more efficiently on the level of the factorized node instances, because there is much less of them. As a positive side-effect, treating factorized HMM state sequences as a unit may increase the effectiveness of the pruning, because all states of a factorized state sequence share an equal set of reachable successor paths. As shown in [9], this allows us to relate the corresponding hypotheses to each other during pruning, because all their successor hypotheses will be recombined within a short interval.

Overall three pruning methods are used: Global beam pruning is applied to all state hypotheses, word end pruning is applied at cross-node transitions corresponding to physical word-ends, and LM state pruning is applied locally to the active instances of each factorized node. Additionally their top-N equivalents are used. The top-N LM state pruning plays a special role, because it is required by the token passing decoder, to keep the effort of the LM state recombination during cross-node expansion tractable. Usually the beams are not tuned per task, but rather there is a set of pre-chosen combinations for different efficiency levels out of which one is selected based on a master beam.

4. WORD CONDITIONED DECODER

To transform the token-passing decoder into a word conditioned decoder, we had to re-design the core algorithm of the decoder, however we use the same underlying minimized search network.

While in the token-passing decoder a heap of instances is managed for each node in the network, we can simplify this in the word conditioned search (WCS) decoder, by using just one global array of instances, together with a local array of entry hypotheses, for each active tree instance. State hypotheses are grouped into tree instances, where each tree instance corresponds to one specific LM history. We enforce a strict order of the tree instances, according to their LM histories. We keep the global node instance array in the same order, thus each tree instance only needs to store the index of its first node instance. The dynamic programming procedure applied at each timeframe looks as follows:

1. Apply local dynamic programming within active node instances.

2. Prune HMM state hypotheses and instances, set probability of unwanted hypotheses to zero.

3. For each successor node of an active final state, add a new entry hypothesis to the corresponding successor tree instance.

4. For each tree instance, recombine active hypotheses with entry hypotheses, and remove zero-probability node instances, writing a new node instance array on-the-fly.

Transitions are more expensive when leaving nodes with a word label, because the LM context is extended, and the entry needs to be inserted into the corresponding successor tree instance. To find the target tree instance quickly, we maintain a simple approximate hash within each tree, which maps from a word label to the corresponding successor tree instance.

We added sparse LM look-ahead as described in [12]: Tree instances are conditioned on their n-gram level additionally to ther LM context, and a transition to the backoff level tree instance is performed, whenever there is no entry in the sparse LM look-ahead table on the current n-gram level.

4.1. Pruning

The word conditioned decoder supports exactly the same pruning methods as the token passing decoder (see Subsection 3.1), thus the search space is nearly equal to the token-passing decoder, when using the same pruning configuration (slight discrepancies arise from minor differences in the order of applied pruning during the expansion of cross-node transitions). The top-N LM state pruning, which limits the number of active LM contexts for each node, is less natural to apply in a WCS decoder than in a token-passing decoder, because it does not match the way hypotheses are organized. Nevertheless, it can be applied efficiently, by exploiting the fact that usually only very few states are affected by this pruning. We simply count the number of surviving instances for each node during pruning, and only invest additional effort for those nodes which exceed the limit.

Since the dynamic programming within a factorized node is cheaper than the transition across a node boundary, and since the cross-boundary transition within a tree is cheaper than the transition across trees, we added two additional pragmatic pruning methods: *Final-pruning* applies a sharper global beam to any final HMM states, and *label-pruning* applies an even sharper global beam to final HMM states of nodes which have a word label attached. We define final-pruning as a factor relative to the global beam, and label-pruning as a factor relative to final-pruning.

5. EXPERIMENTAL RESULTS

The static graph Viterbi decoder we use for comparison was described in [14] and [2]. Both decoders are implemented in C++ and integrated in IBM's Attila speech recognition toolkit [13]. We evaluate efficiency in terms of RTF (real time factor) as measured on an Intel Xeon X5680 processor with 3.33GHz. We use two basic systems for comparison, which only slightly differ from those used in [2]:

1. English BN with 90k pronunciations, SA models, 150k Gaussians, speaker adaptive and discriminatively trained, with 3.3M (small) or 200M (large) 4-gram LM.

2. Vowelized Arabic BN with 2.5M pronunciations for 754k LM words, 400k Gaussians, speaker adaptive and discriminatively trained, 500M 4-gram LM (2.1M for static graph building).

Figure 1 compares the efficiency on the English task using the small 3.3M n-gram LM. The token-passing and WCS decoders perform similarly, and adding the label- and final-pruning improves efficiency only slightly. Adding sparse LM look-ahead reduces the RTF by around 50% at equal precision, which is to be expected due to the sparseness of the small 4-gram LM. The static decoder without lattice generation is faster by another 30%. When adding lattice generation to the static decoder, then the dynamic with sparse LM look-ahead and the static decoder end up on a similar efficiency level.

Figure 2 compares the efficiency on the English task using the large 200M 4-gram LM. Token-passing and WCS perform similarly, and adding final- and label pruning slightly improves efficiency. adding sparse LM look-ahead improves efficiency by around 30%. The static decoder can not use the 200M n-gram LM directly in graph building, thus it first generates lattices based on a graph generated with the smaller 3.3M n-gram LM, and then rescores those lattices using the full LM. Overall, the dynamic decoder with sparse LM look-ahead is slightly more efficient than the static decoder on this task.

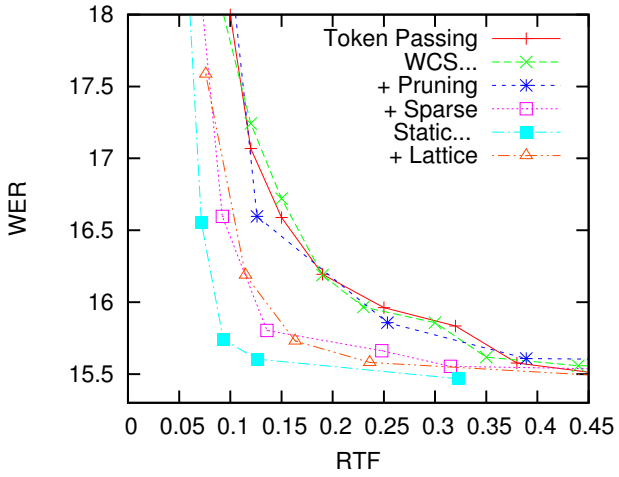

Fig. 1. English BN, 3.3M 4-gram LM, 90k voc.

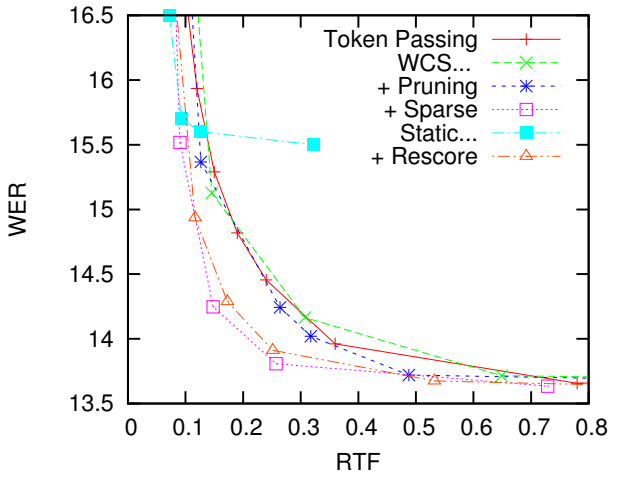

Fig. 2. English BN, 200M 4-gram LM, 90k voc.

Figure 3 compares the results on the Arabic task using the 500M 4-gram LM. The WCS decoder is around 20% faster at equal precision than the token-passing decoder on this task which requires a much larger search space than the previous tasks. Whether the added final- and label pruning improves efficiency is unclear from the graph. However adding sparse LM look-ahead gives another 15 to 20%, despite the large LM which actually isn't that sparse any more in practice (measurements have shown that for the actually requested LM contexts, the LM has an n-gram entry for around 40% of all successor words). The WCS based dynamic network decoder with sparse LM look-ahead is considerably more efficient than the static decoder with LM rescoring on this task. A disadvantage of the static decoder, which uses the small LM during decoding, is that it's search is less focussed than when using the full LM, and thus it has to evaluate more Gaussian densities to cover an equally good set of hypotheses.

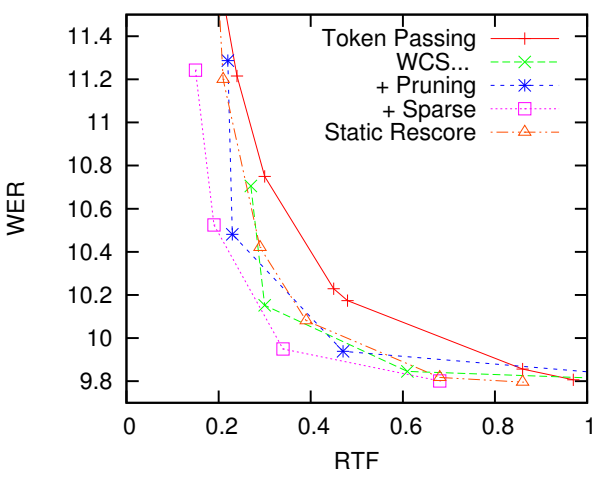

Fig. 3. Gale Arabic, 500M 4-gram LM, 2.5M pronunciations, 754k LM words.

6. CONCLUSIONS

We have shown that the decoder based on hybrid word conditioned search is usually a bit more efficient than the previous token-passing decoder. Sparse LM look-ahead highly improves the efficiency of dynamic network decoders, unless the lexicon is so small that LM look-ahead is not a concern regarding efficiency. A dynamic network decoder has the advantage of very effective pruning methods, and it can use the full LM right during decoding to focus the search, both of which allow a well-crafted dynamic network decoder to beat a rescoring-based static decoder in efficiency, especially when using a large LM.

7. RELATION TO PRIOR WORK

In [7] a similar experiment was conducted, by modifying a word conditioned dynamic network decoder to integrate the advantages of a token-passing decoder. In this work we go further by also exploiting factorization, and comparing to a WFST based static decoder. A comparison between the basic token-passing decoder and the static decoder was previously performed in [2].

8. ACKNOWLEDGEMENTS

Supported by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Defense U.S. Army Research Laboratory (DoD / ARL) contract number W911NF-12-C-0012.[1]

[1]The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoD/ARL, or the U.S. Government.

9. REFERENCES

[1] S. Kanthak, H. Ney, M. Riley, and M. Mohri, "A Comparison of two LVR Search Optimization Techniques," in *ICSLP*, Denver, CO, USA, Sept. 2002, pp. 1309–1312.

[2] H. Soltau and G. Saon, "Dynamic Network Decoding Revisited," in *ASRU*, 2009.

[3] M. Mohri, F. Pereira, and M. Riley, "Speech Recognition with Weighted Finite State Transducers," in *Handbook of Speech Processing*. 2008, pp. 559–582, Springer.

[4] 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.

[5] C. Allauzen, M. Mohri, M. Riley, and B. Roark, "A Generalized Construction of Integrated Speech Recognition Transducers," in *ICASSP*, Merano, Italy, December 2009.

[6] D. Nolden, R. Schluter, and H. Ney, "Advanced Search Space Pruning with Acoustic Look-Ahead for WFST Based LVCSR," in *ICASSP*, 2013.

[7] D. Nolden, D. Rybach, R. Schlüter, and H. Ney, "Joining Advantages of Word-Conditioned and Token-Passing Decoding," in *ICASSP*, 2012.

[8] D. Nolden, R. Schlüter, and H. Ney, "Extended Search Space Pruning in LVCSR," in *ICASSP*, 2012.

[9] D. Nolden, R. Schlüter, and H. Ney, "Search Space Pruning Based on Anticipated Path Recombination in LVCSR," in *Interspeech*, 2012.

[10] 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.

[11] S. Ortmanns, H. Ney, and A. Eiden, "Language-Model Look-Ahead for Large Vocabulary Speech Recognition," in *ICSLP*, Sydney, Australia, Oct. 1998, pp. 2095–2098.

[12] 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, 2011.

[13] H. Soltau, G. Saon, and B. Kingsbury, "The IBM Attila Speech Recognition Toolkit," in *Spoken Language Technology Workshop (SLT), 2010 IEEE*.

[14] G. Saon, D. Povey, and G. Zweig, "Anatomy of an Extremely Fast LVCSR Decoder," in *Interspeech*, 2005.