TRANSDUCER COMPOSITION FOR "ON-THE-FLY" LEXICON AND LANGUAGE MODEL INTEGRATION

Diamantino Caseiro, Isabel Trancoso

L^2F Spoken Language Systems Lab.
INESC-ID/IST
Rua Alves Redol 9, 1000-029 Lisbon, Portugal
{dcaseiro, Isabel.Trancoso}@l2f.inesc-id.pt

ABSTRACT

In this work we present the use of a specialized composition algorithm that allows the generation of a determinized search network for ASR in a single step. The algorithm is exact in the sense that the result is determinized when the lexicon and the language model are represented as determinized transducers. The composition and determinization are performed simultaneously, which is of great importance for "on-the-fly" operation. The algorithm pushes the language model weights towards the initial state of the network. Our results show that it is advantageous to use the maximum amount of information as early as possible in the decoding procedure.

1. INTRODUCTION

Our long term motivation is to work towards the development of specialized composition algorithms that will allow the use of knowledge sources of higher level than the language model. When our task moves from simple transcription to understanding, other knowledge sources are available that are not usually taken into account early in the decoding process. We believe that in order to fully exploit that knowledge some "on-the-fly" processing will be required, as they are very often dynamic and complex. The usual approach consists on interleaving "on-the-fly" composition, with "on-the-fly" determinization [1]. We think that a specialized "on-the-fly" composition that generates a determinized result can be more efficient.

Our first step towards this long term goal was described in [2], where we formally presented an algorithm for the composition of the lexicon with the language model. There we showed that our method was able to directly generate similar sized networks as those obtained with explicit determinization [3].

The current paper builds on that work and shows how the algorithm can be extended to compress "on-the-fly" the search network. Here, we also investigate the applicability of the developed algorithm to real recognition tasks, comparing the impact of various refinements on the performance of the decoder.

In the next section we will present our composition algorithm. Then, in section 3, we will show how we can reduce the size of the composition network by sharing some equivalent states, and also how this network can be factorized to reduce the memory requirements of the decoder. In section 4, we present recognition experiments that show the impact on the performance of the decoder of various refinements of the algorithm. Finally, in section 5, we summarize the main conclusions and present out plans for future work.

The present work is part of Diamantino Caseiro's PhD thesis, initially sponsored by a FCT scholarship (PRAXIS XXI/BD/15836/98). This works was also partially funded by the POSI program of the "Quadro Comunitario de Apoio III".

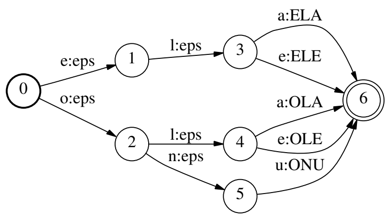

Fig. 1. Lexicon.

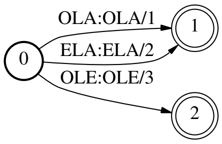

Fig. 2. Language model.

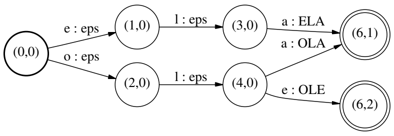

Fig. 3. Composition of the lexicon with the language model.

2. LEXICON AND LANGUAGE MODEL COMPOSITION

Our algorithm requires the use of a lexicon and a language model represented as two weighted finite state transducers (*WFST*), L and G respectively. The lexica and language models commonly used in ASR are easily representable as *WFSTs*, as shown in [2],

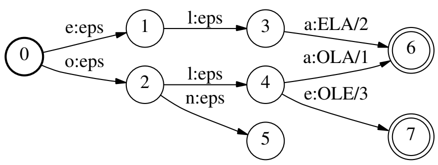

Fig. 4. Dead-ends generated by the composition algorithm.

although some approximations are used in the case of n-gram language models.

Figure 3 shows the composition of the toy lexicon and language model *WFSTs* of figures 1 and 2. In the *WFST* composition algorithm [4], each state of the composition *WFST* represents a pair with a state from each of the argument *WFSTs*. For example, state $(0, 1)$ in figure 3 consists of the state 0 from the lexicon and of state 1 from the language model. Each edge leaving a state in the composition is obtained from one pair of compatible edges, one leaving the respective lexicon state and the other one leaving the language model state. The edges are deemed compatible if the output of the lexicon edge is equal to the input label of the language model edge. The resulting edge is assigned the input label of the lexicon edge and the output label of the language model edge. The epsilon edges can be easily dealt with by adding extra "self-loop" edges to each state labelled with auxiliary epsilon symbols [4].

It is well known that when both transducers are deterministic on the input side their composition will also be deterministic, as proved by Mohri in theorem 1 of [1]. But the composition algorithm, as described, is very unpractical for the case of the composition of a lexicon *WFST* with a language model *WFST*. The problem is that it will mostly generate "dead-end-paths", that is, paths that end in a non-final state with no successors. Figure 4 illustrates the problem by showing all the paths generated by the composition algorithm, including the dead-end of state 5.

In our algorithm [2], we avoid the generation of "dead-end-paths" by pre-calculating the sets of output words reachable from each edge of the lexicon *WFST*; each set is then associated with the respective edge. Using that information, our algorithm will not match epsilon-output edges from the lexicon with edges from the language model when the input of the language model edge is not in the set associated with the edge of the lexicon.

The computation of the set of edges leaving a state $s = (s^l, s^g)$ in the composition can be easily performed "on-the-fly" as no look-ahead is required. In fact, the algorithm only needs to know the identity of the lexicon and language models states (s^l and s^g respectively), in order to access the set of lexicon or language model edges that leave those states. With that information we can compute more than a simple composition and it is straightforward to perform language model label and weight pushing.

2.1. Pushing

As the first step in the direction of an approximation to "on-the-fly" minimization, we implemented an approximation of pushing. It is useful to push the output labels and the output weights as much towards the initial state as possible. Pushing the output labels will allow further sharing of suffixes by our network compression algorithm, resulting in smaller composition *WFSTs*. Pushing the weights allows an earlier use of the language model during search, in what is sometimes called "language model look-ahead". Our "on-the-fly" implementation of weight pushing only spreads the language model weights. Our purpose is to spread the weights of the language model throughout the path from the initial state of the word until its identity is known.

The pushing of output labels is done outputting the label as soon as only one of the edges leaving the language model state (s^g) matches with a given lexicon edge. The pushing of weights towards the initial state is done by first selecting the weight of the best matching language model edge, and then outputting the difference between that weight and the partial sum of the previously produced weights in that path. Each state of the composition keeps its own partial sum.

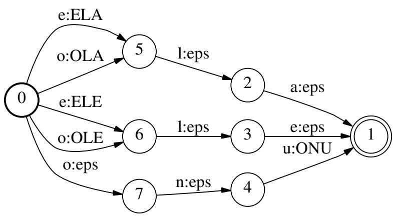

Fig. 5. Suffix sharing lexicon.

3. "ON-THE-FLY" NETWORK COMPRESSION

Although the composition of deterministic transducers will yield a deterministic transducer, the composition of minimal transducers will not, in general, generate a minimal one. In order to obtain a true minimal transducer, the result will have to be explicitly minimized using a minimization algorithm such as Mohri's minimization algorithm for *WFSTs*. It consists of pushing the output labels and the weights as much as possible towards the initial state, and then performing a classical minimization of the underlying automaton[1] .

Observing the results of the determinization algorithm, we notice that it essentially shares the prefixes of the pronunciation of words that leave the same language model state, by replicating the structure of the lexicon *WFST*. We want our compression algorithm to also allow the sharing of suffixes of words that arrive at the same language model state, and we want to compute this "on-the-fly". Our approach consists of selectively replacing the lexicon with a "suffix sharing" equivalent lexicon R, like the one illustrated in figure 5. It consists of the reverse of a deterministic *WFST* (so that is shares all possible suffixes). We call this *WFST* "suffix lexicon", and we use it to guess the structure of the composition *WFST* only for states (s^l, s^g) in a path after the output of the label and before reaching the end of the pronunciation. When the label pushing version of the composition outputs a label, the destination of that edge will be a state (d^l, d^g), ***here we replace the destination state d^l with a state d^r of the suffix lexicon that starts an equivalent path to a final state. In general, if multiple pronunciations are used, then there can be more than one equivalent d^r state, and a small amount of non-determinancy will be created by outputting multiple edges.

To find the d^r states compatible with d^l, we start by computing the set of pairs of states (s^l, s^r) such that there exists a path from i^l (the initial state of L) to s^l equal to a path from i^r to s^r, and

[1]The underlying automaton is obtained by treating each tuple (input, output, weight) as a single label.

also a path from s^l to a final state of L equal to a path from s^r to a final state of R. Those pairs are just the ones that label each state of the composition of the inverse[2] of the lexicon with the reverse lexicon ($L^i \circ R$). In a process similar to the one used to resolve the epsilon edges in the composition, the states of R are assigned the set of words that have the state in its pronunciation path. All this information is precomputed. In runtime, the table is consulted to find candidate states d^r, that are used if the label just traversed is in their sets of words.

In our tests, while producing bigger networks than the true minimization algorithm, this algorithm still managed to reduce the number of states to about 40% and the number of edges to 60%. There was no visible impact on the recognition time.

3.1. Network Factorization

The *WFST* that results from the composition of the acoustic model *WFST* H [3] with the composition of the lexicon with the language model, has very long sequences of states forming "linear paths", where each state has only one sucessor and eventually a self-loop edge. In [3], Mohri and Riley factorized the network by replacing such paths with an identifying edge and thus were able to achieve a reduction of its size of about 4 times. Those special edges were conceptually expanded in run time by the decoder. That particular factorization method cannot be applied "on-the-fly" due to its global nature. But we can still associate the composition cascade as $(H \circ L) \circ G$ instead of $(H \circ (L \circ G))$ and factorize $(H \circ L)$ before composing it with the language model. Alternatively, we can just factorize H, in which case each factorized path will resemble an acoustic unit *HMM* in a tradicional decoder. In our experiments we opted for this last approach and still managed a reduction of about 2.5 times in the number of states.

3.2. Decoder Modification

In order to evaluate the quality of the combined lexicon and language model networks, we had to change our initial decoder [5]. The previous version was based in the propagation of tokens, where each token represents sets of hypotheses that share the same state in the search network. By working with sets of hypotheses it was possible to decouple the language model from the search network in a time-synchronous mode of operation.

As the decoder was implemented using the principles of data abstraction, it was easy to reuse most of the search algorithms, and adapt the decoder to the efficient use of one single integrated search network, by using a more suitable representation of the tokens that store only one hypothesis. The decoder was also adapted to directly work with macro arcs that directly correspond to the factorized paths described in section 3.1, so that it can simultaneously work with macro arcs and with regular edges.

4. RECOGNITION EXPERIMENTS

4.1. Experimental Setup

In this section we describe the recognition experiments performed to evaluate the networks produced by our composition algorithm. All the experiments were based on the BDPÚBLICO corpus[6] framework. It consists of European Portuguese read speech collected in a sound-proof room with a high quality microphone, and of texts from the online edition of the PÚBLICO newspaper. It is equivalent in size and purpose to WSJ0. The experiments were performed using standard 600MHz pentium III PCs with 512MB or 1GB of RAM, running Linux.

We used an European Portuguese lexicon with 27k words. The lexicon was converted to a linear lexicon *WFST* and disambiguating labels were added at the end of the pronunciations. This *WFST* originally had 281,815 states and 313,962 edges. It was then determinized and minimized for use with our composition algorithm resulting in an equivalent *WFST* with 29,620 states and 59,366 edges.

We used various trigram backoff language models, trained from 46 million words from the online edition of the PÚBLICO newspaper, corresponding to the years from 1995 to 1998. The generation of language model of various sizes was done by applying different cutoffs (we applied the same cutoff to bigrams and trigrams). The language models were approximated by *WFSTs* by representing each context by a different state and each n-gram in the model by an edge between contexts. The backoffs were represented by epsilon edges from specific contexts to more general ones.

The best acoustic models available in our research group are based on the combination of the output of various neural networks [7]. In our recognition experiments, we extracted 3 different sets of features from the speech signal: 12 plp coefficients + log energy + deltas; 12 log-rasta coefficients + log energy + deltas; 28 modulation spectrogram features.

We used 3 separate multilayer perceptrons (*MLP*), one for each set of features. The input of each *MLP* was a window of 7 vectors centered on the vector being analyzed. The *MLPs* had a 3-layer architecture with 500 units in the hidden layer, and the output consisted of 39 softmax units corresponding to 38 context independent phones plus silence. The output of the 3 *MLPs* was combined using the average of the logarithm of the probability estimated for each phone.

The acoustic model topology consisted of a sequence of states with no self-loop to enforce the minimal duration of the model, and one final state with a self-loop. The acoustic models were encoded in a single acoustic model *WFST*.

4.2. Results

In figure 6 we compare the search WER, shown as a function of recognition time, using integrated composition networks obtained with three versions of our algorithm: with pushed labels; with pushed labels & weights; and with pushed labels & weights but using a minimized language model. It also shows the performance of using an integrated network that was offline minimized [4]. All versions used the same language model model (with cutoffs of 10 trigrams and bigrams). We can very clearly see the advantages of pushing the weights. Is is also obvious that full minimization post-processing of the network outperforms the other approaches. The language model *WFST* used was determinized but not minimized in the 2 first versions. We see, in the third version, that the use of a minimized language model drastically reduces the difference to the full minimized network. We think that there is an acceptable

[2]The inverse of a *WFST* is obtained by swapping its input and output labels, it has that name because it computes the inverse relation.

[3]This *WFST* represents the structure of the subword *HMMs*, and maps distributions to subword units.

[4]The minimization standardizes the network, thus all versions of the algorithm show the same performance after the offline minimization.

loss in not performing the offline minimization. This minimization cannot be applied "on-the-fly" during recognition.

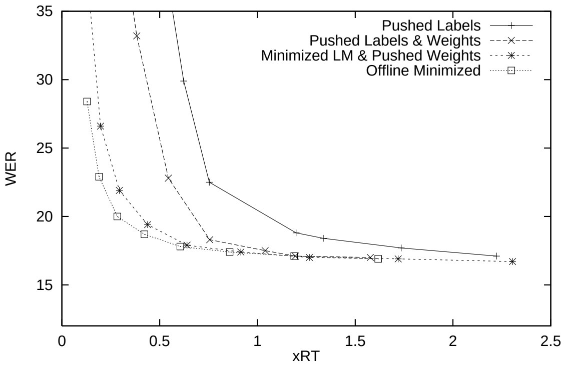

Fig. 6. Performance of various versions of the algorithm.

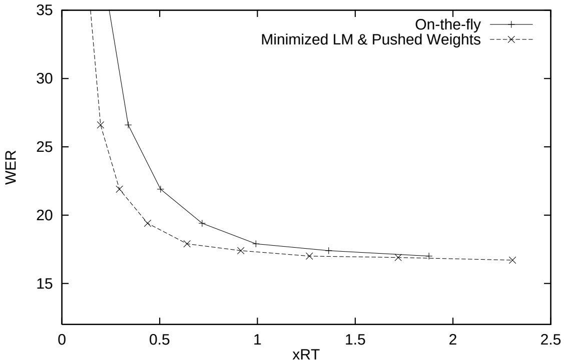

Fig. 7. "On-the-fly" vs precomputed network.

All the previous experiments were run with a pre-composed integrated network. To check if the composition is efficient enough for "on-the-fly" use, we also performed the experiments shown in figure 7, comparing the use of the static network with its "on-the-fly" construction. It was clear, from early experiments, that a pure "on-the-fly" use of the network was too slow and that a cache had to be used. Besides the use of this cache, we also precomputed some parts of the network that we expected to be frequently used. The criterion was to precompute the most dense parts of the network, computing the first few arcs (up to a depth of 4) starting from the start of only those pronunciations which leave a language model state with more than 30 arcs. We further enforced that only states of the network with an outdegree over 5 were precomputed. During recognition, the cache was flushed regularly. We see from the results that the network computation still has a very appreciable impact on performance.

In order to investigate the scalability of the algorithm to larger language models, some experiments were performed using cutoffs of 10, 5 and 2. As can be seen in figure 8, although there is a negative impact of the larger language model at lower beams, the better information available is used to easily outperform the smaller model. The figure shows that, if allowed by the available memory, there is much to be gained by using a detailed language model in a single pass decoder.

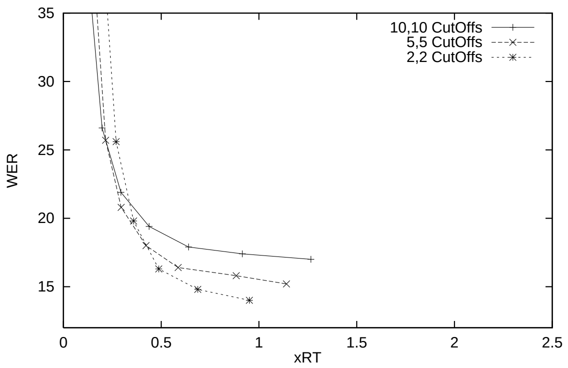

Fig. 8. Performance variation with the increase of the language model.

5. CONCLUSIONS

We have shown how the deterministic integration of the lexicon with the language model can be performed "on-the-fly" in a finite-state framework. We have also shown how to build very efficient networks for real-time decoding in one pass. The results show that pushing is a very important factor, and that it should be taken into account by "on-the-fly" networks expansion algorithms. Our current implementation of the algorithm is feasible for "on-the-fly" use, but at the cost of some time or accuracy penalty.

In the future we plan to investigate better ways of selecting which parts of the network should be precalculated for a given allowed memory space, and also better ways of managing the cache during recognition. We also plan to investigate, with more detail, why the full minimization outperforms the preminimization of the language model, hoping to find ways of improving the approximation.

6. REFERENCES

[1] M. Mohri, "Finite-state transducers in language and speech processing," *Computational Linguistics*, vol. 23, no. 2, pp. 269–311, June 1997.

[2] D. Caseiro and I. Trancoso, "On integrating the lexicon with the language model," in *Proc. Eurospeech '2001*, Aalborg, Denmark, Sept. 2001.

[3] M. Mohri and M. Riley, "Integrated context-dependent networks in very large vocabulary speech recognition," in *Proc. Eurospeech '99*, Budapest, Hungary, Sept. 1999.

[4] M. Mohri, F. Pereira, and M. Riley, "Weighted automata in text and speech processing," in *ECAI 96 Workshop*. Aug. 1996.

[5] D. Caseiro and I. Trancoso, "A decoder for finite-state structured search spaces," in *ASR 2000 Workshop*, Sept. 2000.

[6] H. Meinedo L. Almeida J. Neto, C. Martins, "The design of a large vocabulary speech corpus for portuguese," in *Proc. Eurospeech '97*, Sept. 1997.

[7] J. P. Neto H. Meinedo, "Combination of acoustic models in continuous speech recognition hybrid systems," in *Proc. ICSLP '2000*, Beijing, China, Oct. 2000.