Improved Semi-dynamic Network Decoding Using WFSTs

Dong-Hoon Ahn[1], Su-Byeong Oh[1], and Minhwa Chung[2]

[1]Telecommunication Network Business Division
Samsung Electronics Co., Ltd.
Maetan-3-dong, Yeongtong-ku, Suwon 442-600, Korea
{donghoon.ahn,subyeong.oh}@samsung.com

[2]Department of Linguistics
Seoul National University
Sillim-dong, Kwanak-ku, Seoul 151-742, Korea
mchung@snu.ac.kr

Abstract

In this paper, we present an improved semi-dynamic network decoding strategy by incorporating weighted finite-state transducer (WFST)-based search network. In our approach, a static search network is first optimized by applying WFST algorithms (determinization and minimization) to the composition of a lexicon and a language model. Then the WFST is partitioned into a set of subnetworks according to language model (LM) histories, and transformed into a subnetwork-based search network with exploiting structural differences where a WFST is a Mealy machine and our representation is essentially a Moore machine. This new strategy, which is opposite to our previous approach where each subnetwork depending on a LM history is first constructed and aggregates into a complete network, can let any static network compatible to WFST enjoy the run-time efficiency from the subnetwork-caching operation as well as the static efficiency from the WFST algorithms. The experimental results using Korean read speech dictation task are presented to show its efficiency.

1. Introduction

Decoding continuous speech over large vocabulary is computationally complex process since a decoder should find the most probable answer from huge potential search space consisting of various knowledge sources under limited computational resources. Due to this reason, many decoding techniques to cope with such complexities have been developed so far [1]. Investigating such techniques, they are broadly categorized into two classes: *static* and *dynamic* representation of search space. Static representation (static network decoding) is intuitively simple and easy to construct, but suffers from huge memory requirements. On the other hand, dynamic representation (dynamic network decoding) relieves the memory requirement with on-demand network constructions and tight beam pruning. Although the on-demand operations pose computational overheads and are limited to local and approximate ones [2][3], this dynamic network decoding has been adopted in many LVCSR systems until recently due to much less memory requirements.

One prominent approach recently introduced is the use of weighted finite-state transducers (WFSTs) [4], which is theoretically founded and therefore facilitates statically optimal (minimal in terms of the number of states) organization of search networks. This optimality results from the applications of determinization and minimization algorithms. However, while theoretically optimal, there are several practical issues when we are to perform decoding with WFSTs. Firstly, the two algorithms are computationally expensive. Since they require much proportional to the size of a network, their applications to very large network that is not optimized yet, may often fail. Secondly, although WFSTs are statically optimal, it seems rather inefficient to keep a complete network in memory during decoding regardless of pruning size. Thirdly, although the number of *states* is minimal, that of arcs may not be [5]. Considering the computations during decoding depend on identities of HMMs and words labeled on *arcs*, such an organization is suboptimal in terms of memory requirement for keeping labels and the amount of computations, unless carefully treated. Lastly, while applying the determinization algorithm, output labels (word identities) are typically located on arcs where word identities are resolved (at the beginning of *linear tails*). This has the effect of word labels being pushed toward the initial state and is helpful to obtain more size reductions and improve pruning behaviors (via word-end pruning). However, this is not always desirable since once they are pushed, word boundaries and their posterior probabilities are hard to be correctly computed and may produce wrong lattices.

In this paper, we resolve these problems with our subnetwork-based semi-dynamic network decoder [6][7]. In case of the first problem, we have already addressed it by introducing subnetwork-based aligned tail-sharing algorithm [7]. For the rest of the problems, we improve our semi-dynamic network decoder by partitioning and converting a WFST into subnetwork-based search networks with exploiting structural differences where a WFST is a Mealy machine and our representation is essentially a Moore machine. The final search network can now enjoy the run-time efficiency from the subnetwork-caching operation as well as the static efficiency from the WFST algorithms. We present the experimental results on this strategy using Korean read speech dictation task to show its efficiency.

This paper is organized as follows. In the next section, structural differences are investigated between a WFST and our representation. Learning their differences, we describe our method to convert a WFST into a subnetwork-based search network in Section 3. In Section 4, the method is evaluated with the Korean read speech dictation task and its results are analyzed, followed by discussions and conclusions in Section 5.

2. Structural comparisons

Figure 1 shows the two types of network structures, namely WFST and our network representation in [7]. In this figure, weights distributed over arcs are omitted for simplicity, and the w_i's indicate each word and the others (a_i's, b_i's and c_i's) indicate individual HMMs. Both of the structures are part of a search network using a bigram language model (LM) and show the equivalent situation where there are four words (w_1~w_4) outgoing a (possibly initial) state, and additionally the

word w_4 can appear after w_1, w_2 and w_3. In case of the WFST (Figure 1(a)), it is obtained by composing a lexicon and a language model followed by the determinization and the minimization algorithms. Figure 1(b) is constructed by applying the subnetwork-based network construction method using a language model network [6], followed by the application of the subnetwork-based tail-sharing algorithm [7]. The states consisting of word w_4 after w_1, w_2 and w_3 can be shared since the network uses a bigram LM which means the LM history is of 1-word length.

At a glance, Figure 1(b) seems more complicated than Figure 1(a) since the former has more states and arcs. There are two factors to induce such a situation.

Comparing the two network structures, we can learn that their essential differences lie in their types of finite-state machines. The WFST in Figure 1(a) is a Mealy machine that associates input (HMM) and output (word) with a transition over an arc, which means that output depends on the previous state and the input while our FSN (finite state network) representation in Figure 1(b) can be considered as a Moore machine since output depends on the current state only. It is well known that there is equivalence between them with typically more states being introduced to a Moore machine.

Second difference lies in the locations of output symbols, that is, in where a word is identified. In case of the WFST, the identity of a word can be known before its last HMM since once it is determinized, the output label is located on the first arc that belongs exclusively to the word, or at the beginning arcs of linear tails while the representation in Figure 1(b) forces word identities (which we call are word-end nodes) to be put behind the last HMMs. Due to this organization of the WFST, the subsequent application of the minimization algorithm can find more equivalent states without having to match output labels until the next outputs are met. In addition, it can also reduce computations with the pruning strategy similar to the word-end pruning [2][3]. For example, state 4 in Figure 1(a) can be shared since word labels have been *pushed* toward the initial state. On the other hand, two states labeled as b_3 in Figure 1(b) cannot be shared since different words (w_2 and w_3) are followed.

Although the paragraphs in the above give explanations in favor of WFSTs, however, there are several practical issues specific to WFSTs when we are to perform decoding with them. Firstly, although the number of *states* is minimal, that of arcs may not be [5]. Furthermore, considering the computations during decoding depend on identities of HMMs and words labeled on *arcs*, such an organization is suboptimal in terms of memory requirement for keeping labels and the amount of observation computations, unless carefully treated. For example, if we count HMMs and words existing in each representation, we have the following result in Table 1.

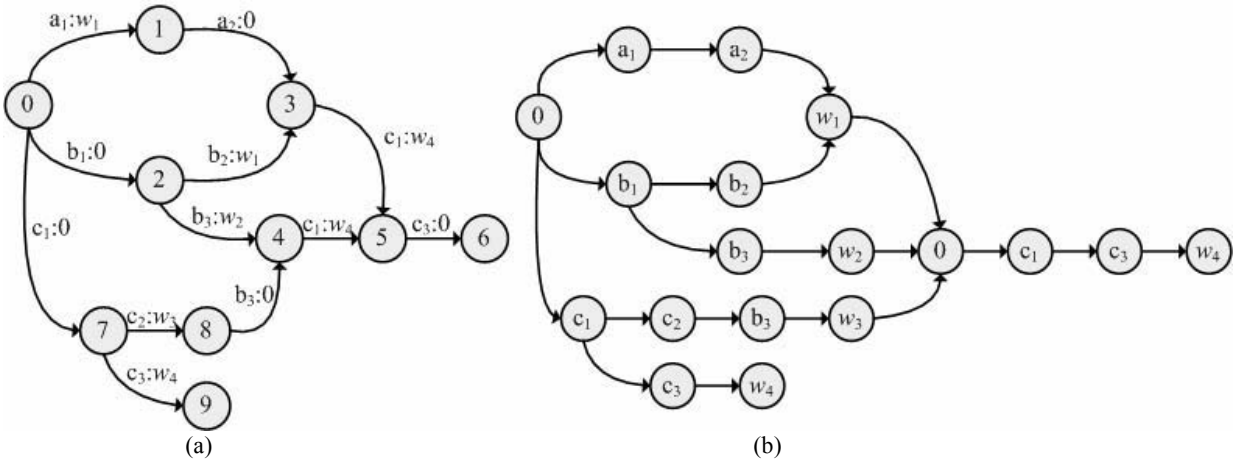
Table 1: The number of HMMs/words

	Figure 1(a)	Figure 1(b)
#states	10	18
#arcs	11	20
#HMMs	**12**	**11**
#words	**6**	**5**

It is interesting that Table 1 shows the numbers of HMMs and words in Figure 1(a) are greater than those in Figure 1(b) although the numbers of states and arcs are less. The differences in counts come from the arcs reaching state 5 of Figure 1(a) where both of the incoming arcs have the same HMM/word labels (c_1:w_4) while they appear only once in Figure 1(b). If multiple incoming arcs have the same labels, it will be more efficient to share them, e.g., by keeping them in the reaching state, in which case, of course, different label cases (e.g., state 3 in Figure 1(a)) are also to be resolved.

Second issue is related to the "pushed" word labels. Although it gives much more structural optimization with the minimization algorithm, it is not always desirable since once they are pushed, word boundaries and their posterior probabilities are hard to be correctly computed and may not produce right lattices that are quite significant in many spoken applications.

To summarize, we basically prefer the statically optimal WFST as a search network while we wish to resolve those issues mentioned in the above. Our approach is based on the subnetwork-based semi-dynamic network decoding with converting a WFST into a subnetwork-based representation while exploiting structural differences. This strategy lets any static network compatible to WFST enjoy the run-time efficiency from the subnetwork-caching [6] as well.



3. Converting a WFST into a subnetwork-based search network

To convert a WFST into a subnetwork-based representation, we should take two steps: partition the WFST into sub-WFSTs and then convert each sub-WFST into a subnetwork.

3.1. Partitioning a WFST

To partition a WFST, we followed the original idea of partitioning a network using LM histories or contexts [6]: a subnetwork is constructed for each state in an LM network. In addition, once linear tails in a search network are shared into a shared tail, it belongs exclusively to a specific word, which can initiate a transition to the next state in an LM network. Since *pushed* word labels on arcs in a WFST correspond to equivalent situations, we have a sub-WFST include states and arcs in the path from the (entry) states reached with word label to the (exit) states before arcs with word labels. In case of a boundary arc connecting an entry state and an exit state in two different partitions, weights and destinations are kept in the exit state while input/output labels are kept in the entry state. One thing to note is when two paths from different LM contexts are met or *collided* in a state (e.g., state 4 in Figure 1(a)). If such cases occur, they are kept in the same partition for non-null incoming arcs. If there are null arcs reaching different LM contexts such as back-off arcs, their destination states should go into different partitions. This procedure is repeated for all arcs in a breadth-first order with keeping the maximum accumulated path probability as an *activation frequency* [6] of each sub-WFST for subnetwork caching. Figure 2 shows the partitions of a WFST in Figure 1(a).

3.2. Converting into a subnetwork-based search network

Once a WFST is partitioned into sub-WFSTs, they are converted into subnetwork-based representation. Although this can be done via a generic Mealy-to-Moore conversion algorithm, there is a more efficient way in this case that also considers those problems mentioned in the previous section. The basic rule is to move the input (HMM) labels to its destination state and the output (word) labels to the word end while duplicating states with context collisions. For this purpose, we utilize the destination states of arcs with auxiliary symbols ($\#_0$,$\#_1$,…) introduced to disambiguate homonyms [4] as word-end states indicating the end of words, rather than replacing them with epsilons. Thus, while traversing a WFST in the partition step, an LM context is determined with the latest word label (or its function) that has been met, and a word-end state can identify itself as its LM context and resolve the misaligned word boundary problem.

When moving labels to appropriate states, if all the incoming arcs have the same HMMs *and* the same LM contexts, it is done with the move (e.g., state 5). However, if any of the two is different, a state (in case of different input, but the same LM context, e.g., state 3) or a path to a word-end state (in case of different LM contexts, e.g., state 4) should be duplicated to keep its context. Though these duplications may seem to increase the number of states and arcs significantly, they do not in real situations; most collisions only occur at different pronunciations of a word and when different word histories share the same successor word sequences, both of which are not so common cases. Figure 3 illustrates this process at state 4. To more compactly keep the duplicated path in a subnetwork, the last duplicated states (e.g., state 5 and 11) share their arc index pairs (first arc index, arc count) rather than duplicate since their destinations are all the same. Additionally, if the source state (state 5) have l arcs (l>1) that are indexed from k+1 in the arc-set, an intermediate state (state 12) with an arc (at k+l+1) is created, their index pairs are first swapped and then shared rather than copying all the outgoing arcs (Figure 3(d)).

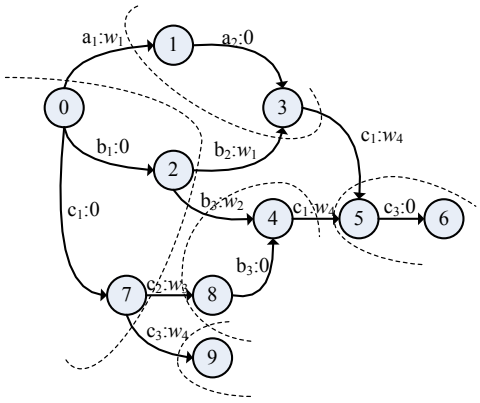


Figure 2: Partitions of a WFST in Figure 1(a) using LM histories. Partitions are marked as dashed.

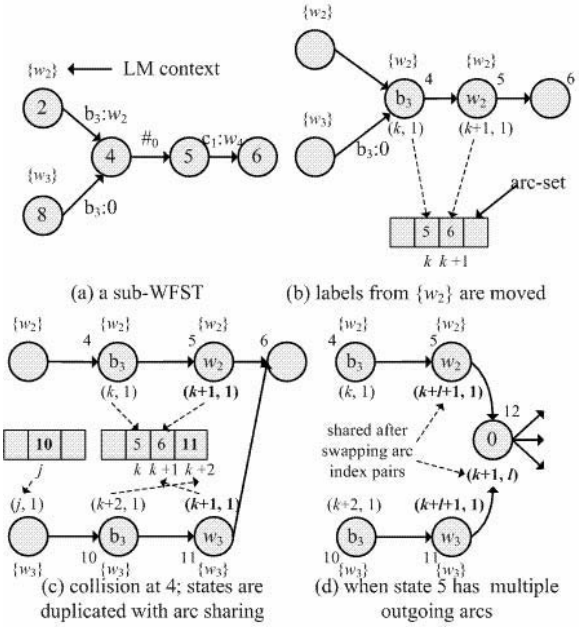


Figure 3: State duplications and arc sharing.

4. Experimental Results

4.1. Conditions

The new decoding method is tested on a 20k-word dictation task for Korean. The acoustic models are trained using 70 hours of Korean read speech and have a set of tied-state CHMMs consisting of 7k tied states with each state of 12 Gaussian mixtures. An LM is trained using 7M morphemes

Table 2: Statistics and word error rates for each network structure.

network type	network size (MB)	#subnetworks	#states	#HMMs	#null states (#words)	#arcs (after sharing)	WER (%)
baseline	374.14	280,214	13,944,058	13,383,633	560,425 (280,214)	17,338,957	14.06
tail-shared	137.35	280,214	3,552,044	2,991,620	560,424 (280,214)	6,950,773	14.06
converted WFST	105.99	239,401	2,449,784	2,207,398	242,386 (239,401)	5,201,334 (4,875,261)	14.02
original WFST	-	-	2,255,066	4,437,391 (on arcs)	2,580,262 (on arcs)	4,994,764	-

collected from newspaper articles, broadcast news and novels, with a modified Kneser-Ney estimation method for trigram model. Also, most frequent 20,659 words from the corpus are used in the lexicon, with 30,908 pronunciations (1.5 prons per word). A test set consists of 608 sentences without OOVs and the perplexity is 99.5 (trigram).

4.2. Comparisons of network structures

Table 2 summarizes various statistics and word error rates (WER) for each network structure: baseline (a network introduced in [6]), tail-shared (a network applied to by the tail-sharing algorithm in [7]), and a converted WFST in this paper. For comparison, we also include the statistics on the original WFST. As with the WERs, both the tail-shared network and the converted network give the similar WERs. In case of the network size, while the size of the baseline network is 374.14Mbytes with over 13M states, the overall size is shown to significantly be reduced with the network optimization methods; by about 63% with the tail-sharing algorithm and further reduced by about 72% with the converted WFST. Comparing the original WFST with the converted WFST, we can see that the number of states and arcs are slightly increased in the latter. This is due to the state duplication process to resolve context collisions. However, in case of the number of HMMs and words existing in those networks, the original WFST is defeated by the converted one, which is undoubtedly due to their structural differences. The amount of memory to keep them and computations during decoding may be significantly affected by these differences. The reduction from the arc sharing during the duplication process is also noticeable with 6.3% reduction in arcs.

Finally, it should be noted that the number of subnetworks which are derived for each LM context or each state in an LM network, is reduced by about 40K. This is basically the result of the minimization of an LM network where the number of its states is reduced from 280,214 to 239,401. We did not minimize the LM network itself in advance to keep the identity of each word.

4.3. Caching performance

Table 3 shows the performance of subnetwork caching applied to the converted WFST. In the table, n is the number of statically preloaded subnetworks before decoding and k is the frame during which a subnetwork with no tokens (active HMMs) can reside in memory, and after which it is withdrawn from memory. We can see that the subnetwork caching is quite effective by reducing the maximum memory required while it does not affect the RTFs noticeably.

Table 3: Performance of the subnetwork caching.

(n, k)	max mem (MB)	RTF
as a static net	105.99	2.12
no caching	26.05	2.23
(10,000, 10)	38.66	2.17
(50,000, 100)	58.06	2.15

※WER: 14.02%, avg. act. HMMs: 4245.88

5. Conclusions

In this paper, we presented an improved semi-dynamic network decoding strategy by incorporating the optimal WFSTs. By effectively converting a WFST into a subnetwork-based representation, a decoding could be more efficient in both of memory and run-time aspects.

Acknowledgements

This work has been supported by Ministry of Science and Technology's Brain Neuroinformatics Research Program.

References

[1] X.L. Aubert, "An overview of decoding techniques for large vocabulary continuous speech recognition," Computer Speech and Language, vol.16, no.1, pp.89-114, January 2002.

[2] H. Ney and S. Ortmanns, "Progress in dynamic programming search for LVCSR," Proceedings of the IEEE, vol.88, no.8, pp.1224-1240, August 2000.

[3] J. Odell, The use of context in large vocabulary speech recognition, Ph.D. thesis, University of Cambridge, 1995.

[4] M. Mohri, F. Pereira, and M Riley, "Weighted finite-state transducers in speech recognition," Computer Speech and Language, vol.16, no.1, pp.69-88, January 2002.

[5] G. Zweig, G. Saon, and F. Yvon, "Arc minimization in finite state decoding graphs with cross-word acoustic context," in Proc. of 7th ICSLP '02, Denver CO, USA, September 2002.

[6] D.-H. Ahn, M. Chung, "A one pass semi-dynamic network decoder based on language model network," in Proc. of 7th EUROSPEECH '01, Aalborg, Denmark, September 2001.

[7] D.-H. Ahn and M. Chung, "Compact subnetwork-based large vocabulary continuous speech recognition," in Proc. of 7th ICSLP '02, Denver CO, USA, September 2002.