GENERALIZATION OF SPECIALIZED ON-THE-FLY COMPOSITION

Tasuku Oonishi, Paul R. Dixon, Koji Iwano, Sadaoki Furui

Department of Computer Science, Tokyo Institute of Technology
2-12-1, Ookayama, Meguro-ku, Tokyo, Japan, 152-8552

ABSTRACT

In the Weighted Finite State Transducer (WFST) framework for speech recognition, we can reduce memory usage and increase flexibility by using on-the-fly composition which generates the search network dynamically during decoding. Methods have also been proposed for optimizing WFSTs in on-the-fly composition, however, these operations place restrictions on the structure of the component WFSTs. We propose extended on-the-fly optimization operations which can operate on WFSTs of arbitrary structure by utilizing a filter composition. The evaluations illustrate the proposed method is able to generate more efficient WFSTs.

Index Terms— WFST, on-the-fly composition, optimization operation

1. INTRODUCTION

We are currently developing the Tokyo Tech Transducer-based decoder($\mathrm{T^3\ Decoder}$) based on the Weighted Finite State Transducer (WFST) framework with the aim of achieving high performance and flexibility [1]. In the WFST framework, the models used for speech recognition are all expressed as WFSTs. We can then use WFST operations to compose and optimize the knowledge sources together and this can give a final highly efficient network that can achieve high recognition performance. Drawbacks with the approach are that large models require large amounts of memory during optimization and decoding, and access to the original knowledge sources is lost so that making on-line changes is very difficult.

To reduce memory usage and increase flexibility during decoding, on-the-fly dynamic composition techniques have been proposed [2]. In on-the-fly composition states are generated as needed, which reduces memory requirements because the size of the sum of the component WFSTs is smaller than the size of the fully composed network. On-the-fly methods also give direct access to the original knowledge sources, therefore they allow models to be replaced or changed more easily and quickly.

However, when using on-the-fly composition decoding speed decreases mainly due to two reasons. Firstly, there is overhead associated when performing the composition operation during decoding. Secondly, it is no longer possible to optimize the fully composed WFST. Hori has proposed re-scoring technique [3] to solve first issue and Caseiro has proposed on-line approximate optimization operations [4] to solve the second one. However, the Caseiro operations placed restrictions on the topology of the component of WFSTs.

We propose a generalized version of the Caseiro operations by utilizing a filter technique [5]. We have evaluated the speed of these operations within the $\mathrm{T^3\ Decoder}$ on various WFST combinations [6]. In this paper we describe in detail the improved algorithms for on-the-fly optimization derived from the Caseiro approach and show new experimental results.

Cheng also proposed on-the-fly composition techniques to optimize WFSTs with arbitrary structure [7]. This approach simulates composition and optimization operation by modifications to the tokens inside the decoder. As a result, the clean abstraction between decoding engine and the knowledge sources is lost.

One of the advantages of our approach is that we can separate the decoding and search network composition parts thus maintaining maximum system flexibility. Moreover, it is may be possible to use our operations in other tasks because no restrictions are placed on the WFSTs. In the next section, we explain the composition and optimization operations.

2. COMPOSITION

Let L be a WFST which maps from x to y with weight w_l and R be a WFST which maps from y to z with weight w_r. The composition transducer $L \circ R$ will map from x to z with $w_l + w_r$ under the tropical semiring [5]. The composition state (q_l, q_r) corresponds to state q_l in L and state q_r in R. A transition is denoted as the tuple $(q, i, o, w, q^{'})$ with source state q, input symbol i, output symbol o, weight w and destination state $q^{'}$. Composition arcs are generated according to:

1. If $o_l = \epsilon$ then generate $((q_l, q_r), i_l, \epsilon, w_l, (q_l^{'}, q_r))$. We call the destination "state by ϵ-output"
2. If $i_r = \epsilon$ then generate $((q_l, q_r), \epsilon, o_r, w_r, (q_l, q_r^{'}))$. We call the destination "state by ϵ-input"
3. If $o_l = i_r$ then generate $((q_l, q_r), i_l, o_r, w_l + w_r, (q_l^{'}, q_r^{'}))$ We call the destination "state by symbol matching"

3. OPTIMIZATION IN ON-THE-FLY COMPOSITION

Caseiro proposed the following operations for on-line optimization during on-the-fly composition: *avoiding dead-end states*, *dynamic pushing* and *state sharing*. We have implemented generalized versions of these optimization operations which can accept transducers of arbitrary structure. In this section we describe *avoiding dead-end state* and *dynamic pushing* operations proposed by Caseiro then present our extended operations.

3.1. Avoiding dead-end states

States which do not have a path to a final state are known as non-coaccessible or dead-end states. Such states impact search efficiency and therefore they are removed after off-line composition, however, on-line removal is more difficult. Fig. 1 shows an example of dead-end states in composition transducer $L_1 \circ R_1$. One example is state $(4, 2)$ which was generated because the previous epsilon transition

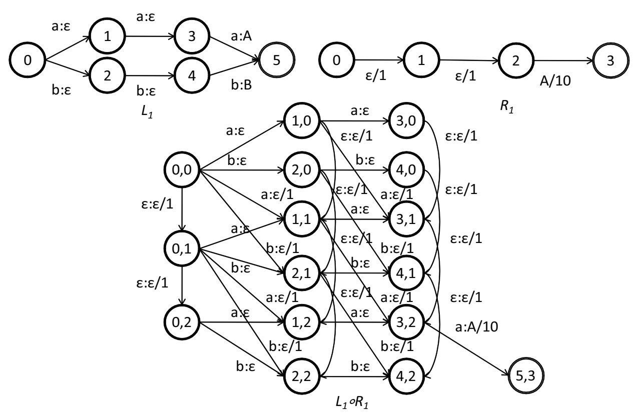
Fig. 1. Example of a WFST with dead-end states.

could not be followed by a matching rule (section 2) that would allow for an exit transition.

3.1.1. Method of Caseiro

Caseiro proposed a method for avoiding dead-end state generation when performing on-the-fly composition [4]. First, during a preprocessing step for each state in the L transducer an *anticipated label set* is constructed. The set contains all the outputs which can be reached from a particular state in L transducer.

During composition of (q_l, q_r) we take the intersection between the anticipated label set of q_l and the input label set from q_r. If the intersection is empty, no future matches are possible so no arcs will be generated. For example by using this technique the transition from states $(2, 2)$ to $(4, 2)$ would not be generated because there would be no arcs in the anticipated label set of state 2 in L_1 matching an arcs input label from state 2 in R_1.

We observe problems when q_r has ϵ input transitions. For example when expanding the arcs to $(0, 1)$, the input epsilon in R has no input labels to match against so the intersection is null. If we avoid to generate such states, an incorrect WFST is produced. While, if we omit dead-end checking for ϵ input transitions in R, we generate inefficient WFST which has many dead-end states.

To solve these problems, Caseiro placed the following restriction in composition:

- Only allow ϵ input transitions (rule 1. in section 2) when q_l is in the initial state of L. Dead-end state is not checked.
- Otherwise use only transition of rules 2. and 3. to omit generating a transition that will lead to dead-end states using the above algorithm.

The composed WFST with these rules is shown in Fig. 2. The following restrictions are also placed on the L transducer:

- The transducer must loop through the initial state.
- Each path between the initial and final state must output only one label.

3.1.2. Proposed Method

In this paper we propose a generalized extension which accepts transducers with arbitrary structure.

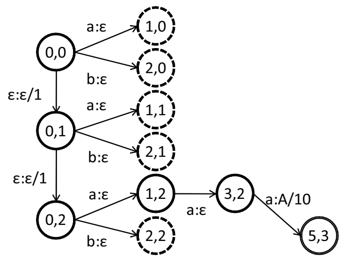
Fig. 2. WFST with Caseiro method [4].

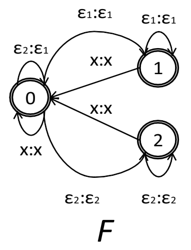
Fig. 3. Filter for composition F.

The idea is to use the filter [5] shown in Fig. 3. In filter composition the ϵ outputs in L are substituted with ϵ_2 symbols and ϵ inputs in R are substituted with ϵ_1. A self loop with output ϵ_1 is added to every state in L and a self loop with input ϵ_2 is added to every state in R. These transducers are denoted L' and R' respectively. The x symbol in filter F of Fig. 3 represents a match with any non ϵ label. Performing the composition $L' \circ F \circ R'$ will give a WFST with the redundant paths removed. With the introduction of a filter the following three way composition of L, F and R gives state identified with the following tuple (q_l, q_f, q_r). The additional parameter q_f is one of the three filter states which we can enter as follows: State zero is generation by symbol matching. State one is generation by ϵ input and state two is generation by ϵ output. The filter does not permit a transition between states one and two, which means the composition WFST will not have the paths which contain an ϵ output (input) transition followed by an ϵ input (output) transition before a transition by symbol matching.

The proposed method exploits these transition restrictions to perform dead-end state avoidance. Given a composed state (q_l, q_f, q_r) the below rules are used:

1. If $q_f = 0$ and q_r has no ϵ input transition, then perform standard dead-end checking with intersection of q_l and q_r.
2. If $q_f = 1$ and q_l has only ϵ output transitions, do not expand.
3. If $q_f = 2$ then perform standard dead-end checking with intersection of q_l and q_r.

With this technique no restrictions are placed on the component transducer. After applying these rules the WFST in Fig. 4 is generated. We can avoid most dead-end states, however, all dead-end states cannot be avoided completely like $(2, 0, 1)$.

3.2. Dynamic pushing

The pushing operation attempts to move weight as close to the initial state as possible and spread weight throughout the path. It is a gen-

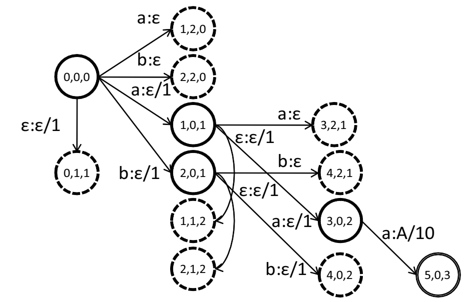

Fig. 4. WFST with filter composition and avoiding dead-end state operation.

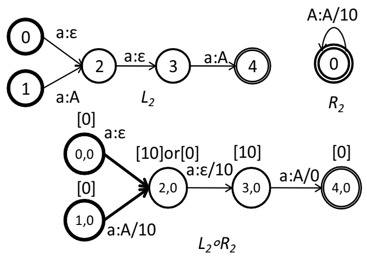

Fig. 5. WFST where the lookahead score cannot be set uniquely.

eralization of language model look-ahead [8] and can help improve decoding speed. For each state the pushing operation first calculates the best path cost to a final state. Next each transition is re-weighted by accumulating the difference between lookahead score of destination state and source state. However, pushing during on-the-fly composition is difficult because of the complexity in obtaining the shortest distance to a final state. Caseiro proposed an approximate on-the-fly operation known as dynamic pushing [4].

3.2.1. Method of Caseiro

In Caseiro's dynamic pushing a lookahead score is set in (q_l, q_r) according to the following rules:

- If state by ϵ output, the lookahead score is set as the smallest arc weight from q_r whose input is also contained in the anticipated label set of q_l.
- Otherwise, the lookahead score is set to zero.

However, in certain topologies the above rules attempt to set a conflicting lookahead score. Fig. 5 illustrates the problem, where the square bracket [ ] indicates the lookahead score set in composition state. In the WFST, the lookahead score in (2,0) is simultaneously set to 10 by the incoming $a : \epsilon$ arc and zero by incoming $a : A$ arc. To avoid this problem Caseiro restricts the structure of L so that the lookahead score in such ambiguous states are set to zero.

3.2.2. Proposed Method

Fortunately, when performing composition with an additional filter, a separate state will be generated for the ϵ output, ϵ input and symbol matching rules. Therefore, lookahead can be set in these states

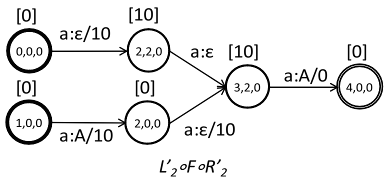

Fig. 6. Dynamic pushing with filter composition.

without any conflict. Fig. 6 illustrates the results of the previous composition after the addition of the filter. The corresponding state (2,0) in Fig. 5 has now become unique states (2,2,0) and (2,0,0), and therefore the lookahead score can be set uniquely.

4. EXPERIMENT

We evaluated our proposed on-the-fly algorithms on a large vocabulary recognition using the Corpus of Spontaneous Japanese (CSJ). The recognition cascade $(H \circ C) \circ L \circ G$ consisted of three WFSTs ($(H \circ C)$, L and G) that were composed and optimized on-the-fly during decoding. We compared the decoding speed against two other composition methods. A pre-composed and optimized cascade is $(H \circ C \circ L \circ G)$. Caseiro method where the L and G WFST is composed with on-the-fly optimizations [4] and $(H \circ C)$ is composed with $L \circ G$ without optimizations. The latter restriction occurs because $(H \circ C)$ does not match the criteria in 3.1.1. In our implementation the decoder holds a cache of states which are composed on-the-fly, the cache is cleared for each utterance. Pushing was performed in the tropical semiring.

4.1. Experimental Setup

The speech waveforms were first converted to sequences of 39 dimensional feature vectors with 10 ms frame rate and 25 ms window size. Each feature vector was composed of 12 Mel-frequency cepstral coefficients (MFCCs) with delta and delta-deltas, augmented with log energy, log delta and log delta-delta energy terms. The acoustic models were three state left-to-right HMM tri-phone models where each state output density was a 32 component Gaussian mixture model with diagonal covariance. EM training was performed utilizing the data from 967 lectures. The language model was back-off tri-gram with a vocabulary of 65k words trained on 2,682 lectures of data. The test set used for evaluations was composed of 2338 utterances which spanned 10 lectures. This yielded a total of 116 minutes of speech. The experiments were conducted on a 2.4 GHz Intel Core2 machines with 2GB of memory.

4.2. Results

We first consider the size of the WFSTs generated by various methods. The number of states and arcs for the $L \circ G$ and $(H \circ C) \circ L \circ G$ WFSTs composed by static, Caseiro and our proposed method is shown in Table 1.

When considering the $L \circ G$ combinations, all methods produce networks of similar sizes due to application of optimization operations. As expected our method produces a slightly larger transducer because of the extra information inserted by the filter. In the case of $(H \circ C) \circ L \circ G$ WFST we are able to produce a substantially

Table 1. The WFST parameter count for the various composition method.

Network	Method	#states	#arcs
$L \circ G$	Static	1,031,229	2,207,982
$L \circ G$	Caseiro	1,040,267	2,251,670
$L \circ G$	Proposed	1,217,744	2,865,283
$(H \circ C \circ L \circ G)$	Static	1,091,075	2,437,298
$(H \circ C) \circ L \circ G$	Caseiro	25,357,146	34,054,721
$(H \circ C) \circ L \circ G$	Proposed	5,342,672	11,533,358

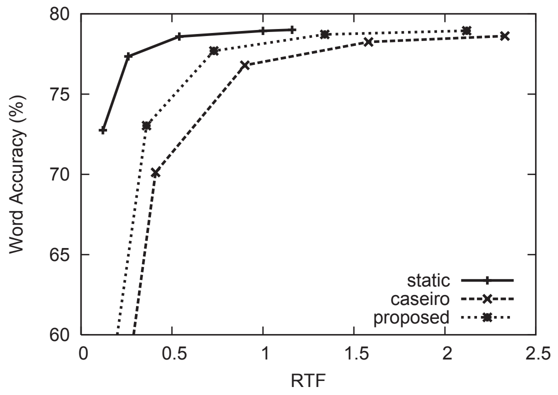

Fig. 7. The decoding speed without overhead of composition.

smaller network than the Caseiro approach. This is because we also optimize the composition which takes the $H \circ C$ WFST, while in the Caseiro approach we use a standard composition at this stage, which will generate many more useless states and arcs. The standard static approach achieves much smaller networks. This is because we can remove useless states completely and minimize the fully composed WFST, further size reductions are achieved by factoring the phone arcs.

Fig. 7 shows the speed of the various WFST techniques when the composition and optimization operations are performed off-line. The vertical axis expresses word accuracy and horizontal axis expresses real time factor (RTF). This is to illustrate the efficiency of the resulting networks without any of the composition overhead. The result shows that the proposed method produces networks that achieve better accuracy and decodes faster than the standard Caseiro method. The slowdown when compared to the standard fully static approach is partly because of larger state and arc count observed in our's and Caseiro's approach and the approximated pushing.

Fig. 8 shows the decoding speed with on-the-fly composition also enabled. In this figure the static network is fully pre-composing and pre-optimizing. The other techniques contain two on-the-fly operations. The result shows that proposed method decodes slightly faster than the standard Caseiro method. The decoding speed of the proposed method is closer to Caseiro one in Fig. 8 than in Fig. 7. This indicates that the proposed method has larger composition overhead.

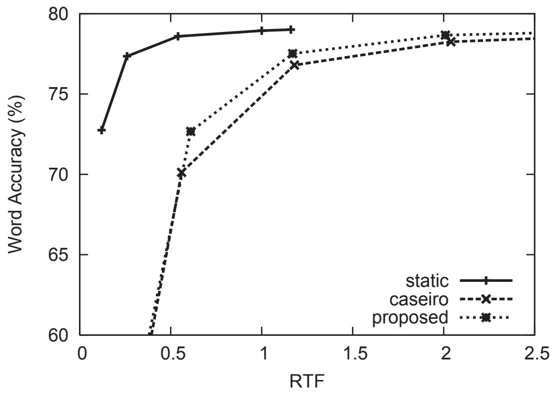

Fig. 8. The decoding speed of on-the-fly composition.

5. SUMMARY

In this paper we have described our generalized method for on-the-fly composition. On the evaluation task the results show the method can perform better than the original operation proposed in [4] in terms of the network size and the accuracy. Furthermore, our method is less restrictive on the type of topology of the component transducers. In future work we will investigate ways to reduce the computational overhead of the on-the-fly operations.

6. REFERENCES

[1] Paul R. Dixon, Diamantino A. Caseiro, Tasuku Oonishi, and Sadaoki Furui, "The TITECH large vocabulary WFST speech recognition system," *Proc. IEEE Workshop on ASRU*, pp. 443–448, 2007.

[2] Hans J.G.A. Dolfing and I.Lee Hetherington, "Incremental language models for speech recognition using finite-state transducers," *Proc. IEEE Workshop on ASRU*, pp. 194–197, 2001.

[3] Takaaki Hori, Chiori Hori, Yasuhiro Minami, and Atsushi Nakamura, "Efficient wfst-based one-pass decoding with on-the-fly hypothesis rescoring in extremely large vocabulary continuous speech recognition," *IEEE Transactions on Audio, Speech and Language Processing*, vol. 15, pp. 1352–1365, 2007.

[4] Diamantino A. Caseiro and Isabel Trancoso, "A specialized on-the-fly algorithm for lexicon and language model composition," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 14, no. 4, pp. 1281–1291, 2006.

[5] Mehryar Mohri, Fernando Pereira, and Michael Riley, "Weighted finite-state transducers in speech recognition," *Computer Speech and Language*, vol. 16, no. 1, pp. 69–88, 2002.

[6] Tasuku Oonishi, Paul R. Dixon, Koji Iwano, and Sadaoki Furui, "Implementation and evaluation of fast on-the-fly composition algorithms," *Proc. Interspeech*, pp. 2110–2113, 2008.

[7] Octavian Cheng, John Dines, and Mathew Magimai Doss, "A generalized dynamic composition algorithm of weighted finite state transducers for large vocabulary speech recognition," *Proc. ICASSP*, pp. 345–348, 2007.

[8] S. Ortmanns, H. Ney, and A. Eiden, "Language-model look-ahead for large vocabulary speech recognition," *Proc. Spoken Language Processing*, pp. 2095–2098, 1996.