LARGE VOCABULARY CONTINUOUS MANDARIN SPEECH RECOGNITION USING FINITE STATE MACHINE

Yi-Cheng Pan, Chia-Hsing Yu, and Lin-Shan Lee
Dept. of Computer Science and Information Engineering, National Taiwan Univ., Taipei
{thomas, davidyu, lsl}@speech.csie.ntu.edu.tw

ABSTRACT

Finite state transducer (FST), popularly used in natural language processing (NLP) area to represent the grammar rules and the characteristics of a language, has been extensively used as the core in large vocabulary continuous speech recognition (LVCSR) in recent years. By means of FST, we can effectively compose the acoustic model, pronunciation lexicon, and language model to form a compact search space. In this paper, we present our approaches of developing a LVCSR decoder using FST as the core. In addition, the traditional one-pass tree-copy search algorithm is also described for comparison in terms of speed, memory requirements and achieved character accuracy.

1 INTRODUCTION

Automata theory has been developed with a long history, and relevant research is still ongoing due to its elegant framework and high efficiency. It has been widely used in natural language processing (NLP) area to model the grammar rules and characteristics of the language. The application of finite state automata on large vocabulary continuous speech recognition (LVCSR) was first introduced by M.Mohri[1], and a new concept of weighted-finite-state machine was introduced, including approaches of transforming the popularly used models such as HMM models and N-gram language models to finite state machine[1][2]. By means of the weighting scheme introduced, we can effectively integrate several probability likelihood functions in a finite state machine in a unified approach. We can then incorporate different sources of knowledge easier and reduce the complexity of the search space. Finite-state-transducer (FST) is an extension of finite state automata. A finite state machine can accept specific input strings (the set of strings accepted by a finite-state-machine is referred to as a "language") and a FST further has a string as output when it accepts strings. In the LVCSR system using FST as the core, we first compile three basic knowledge sources (HMM acoustic models, pronunciation lexicon, n-gram language model) into three respective FSTs. By means of the composition algorithm, we can further integrate the three FSTs into a vast search network. A Viterbi search is then performed on this network and a best-matched path along with the recognized word sequence will be returned.

In the rest of this paper, we first review traditional one-pass tree-copy search algorithm in Section 2. We then introduce our decoder based on FST in Section 3. The experiments and comparison between the two different decoders in terms of speed, memory requirement and character accuracy are presented in Section 4. Finally, in Section 5 some discussions are given.

2 ONE-PASS TREE-COPY SEARCH

In this section, we briefly review the traditional one-pass tree copy search algorithm.

In this algorithm, the search is implemented in a left-to-right, frame-synchronous fashion. We first compile a lexicon tree as shown in Fig.1, in which each arc represents a sub-syllabic HMM model and each path from the root to a leaf represents one or several word(s) with the same pronunciation. The arcs visited by each path are the respective HMM models, of which each word is composed. In the process of searching, each active node may have several copies, where each copy represents a different language model history. The path from the root to an active node forms a partial path. If the active node is a leaf node, a new language model history is generated, all the nodes having the same history are recombined, and only the node with the highest score will survive while others pruned. Then from the new node having survived, we further generate a new tree copy or replace the respective existing tree root node, if the tree with specific history is already generated and the score of the new node having survived is higher than the existing one. It should be noted that we don't need to actually make several tree-copies at run-time, and the tree is just the data structure taken as a reference. In practice, only the active nodes need to be kept in the memory, where each active node represents a partial path. It should be also noted that the number of the nodes activated at each frame may increase rapidly, thus the total number of active nodes may increase exponentially. With limited memory and computing power, we therefore need to further prune those active nodes with lowest scores. During pruning, we need to take the language model scores into account. At each node, there are one or several paths to go to different leaf node(s). We pick up the leaf among them which has the highest uni-gram language model score, and this score is the look-ahead language model score for the node. Each active node is then judged by the sum of their decoding score and the look-ahead language model score during pruning.

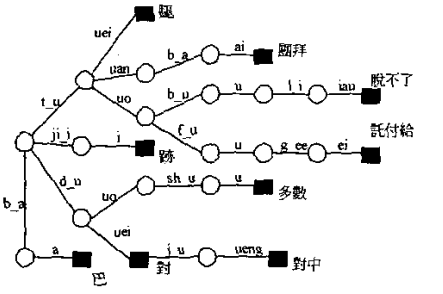

Fig.1 An example of a tree lexicon

3 FST-BASED SEARCH ALGORITHM

3.1 FST and WFST

An FST A can be represented as a six-tuple: $< Q, n, F, \Sigma, \Delta, \delta >$. Q is the set of all states in A, while $n \in Q$ is the initial state and $F \subseteq Q$ is the set of all final states. Σ is the alphabet set of input strings, while Δ is the alphabet set of out-

put strings. Finally, δ is the set of all possible transitions $(Q\times\Sigma\rightarrow Q\times\Delta)$. In speech recognition task, a weight is further attached to each transition. A weighted FST (WFST) is exactly the same as FST, except that δ is now the set of all possible transitions $(Q\times\Sigma\rightarrow Q\times\Delta\times K)$,where K is the set of all possible weights .A transition $t = (s[t],i[t],d[t],o[t],w[t])$ can be represented by an arc from the source state *s[t]* to the destination state *d[t]* with input label *i[t]* and output label *o[t]* and a weight *w[t]*.In our task, *w[t]* is the log probability score.

A path in A is a sequence of consecutive transitions $t_1 \ldots t_n$ with $d[t_i] = i[t_{i+1}], i=1, \ldots, n-1$.Transitions with an empty string ε as the input label consumes no input. A *successful path* $\pi = t_1 \ldots t_n$ is the path from *initial state i* to a *final state* $f\in F$. The input labels of the path π is the result of the concatenation of the input labels of its constituent transitions: $i[\pi]=i[t_1] \ldots i[t_n]$,while the output labels is $o[\pi]=o[t_1] \ldots o[t_n]$ similarly. The weight associated to π is the sum of the initial weight, the weights of its constituent transitions and the final weight of the state reached by π.

Union operation plays an important role in the process of building the WFST for speech recognition. We can easily build small WFST pieces first, and through *union operation* we then bind them together to build the whole WFST.

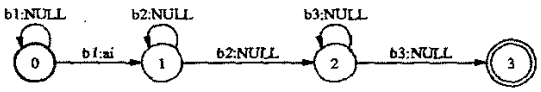

Fig.2 An HMM model for the sub-syllabic unit /ai/ represented by FST

There are three main WFST models used in the speech recognition task. They are the HMM acoustic model H, the pronunciation lexicon model L, and the n-gram language model G. In the following sections, we will give brief descriptions about the construction of these WFST models.

3.2 HMM Acoustic Models

HMM model is widely used in speech recognition. There is a specific probability density function for each state to model the statistics of the acoustic feature vectors for that state. There are also transition probabilities between states.

Remember that the search space encoded in a WFST is the language accepted by the WFST, and the language is the set of all *successful paths*. When transforming a HMM model into a WFST, we have to shift the probability density function onto the arcs. That is, an HMM WFST is a mapping from the state probability density function to the sub-syllabic acoustic model. Fig.2 is a WFST to describe the sub-syllable HMM model /ai/, in which there are three left-to-right HMM states. The regular expression language accepted by the WFST is $b_1^+b_2^+b_3^+$. The output label "ai" can be arbitrarily attached on either arc among the arcs between states.

We can build a WFST for each sub-syllabic acoustic model in the same way. Then through the *union operation*, we can combine them together to from our HMM acoustic model H. At each final state of H, we additionally include an additional arc, with both input and output label ε, returning to the initial state.

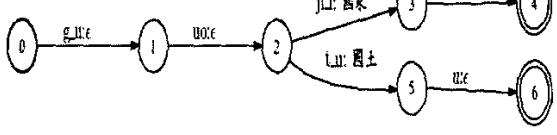

Fig.3 An example partial list of the tree lexicon in FST

3.3 Pronunciation Lexicon

The pronunciation lexicon is used to describe the pronunciation sequence of sub-syllabic units for each word. We can follow the same steps as we build HMM acoustic models to build a linear pronunciation lexicon. We can also further compile the linear lexicon into a tree lexicon to make it more compact and save the memory. Fig.3 is an example partial list of a FST for tree pronunciation lexicon.

3.4 N-gram Language Model

The most popular language model so far is the stochastic *n-gram* language model. In *n-gram* language model, every word is assumed to be dependent only on its previous *n-1* words. Thus the probability of a word sequence with length N can be approximated as:

$$P_{w_1,\ldots,w_N} = \prod_{i=1}^{N} p(w_i \mid w_{i-n+1}^{i-1}),$$

where w_i is the *i-th* word in the sequence, $w_{i-n+1}^{i-1} = (w_{i-n+1}, w_{i-n+2}, \ldots, w_{i-1})$, and each probability $p(w_i \mid w_{i-n+1}^{i-1})$ can be calculated from training data. However, the quantity of the corpus needed grow exponentially with the increase of n. The back-off smoothing method is usually applied to compensate for the missing probability $p(w_i \mid w_{i-n+1}^{i-1})$ if the pattern w_{i-n+1}^{i} does not appear in the limited training corpus, i.e., $p(w_i \mid w_{i-n+1}^{i-1}) = p(w_i \mid w_{i-n+2}^{i-1}) \cdot \partial_{w_{i-n+1}^{i-2}}$, where $\partial_{w_{i-n+1}^{i-2}}$ is a back-off weight (this back-off process can be repeated if w_{i-n+2}^{i-1} still does not appear in the corpus). In our system, $n = 3$, so it is a *trigram* language model.

Fig.4 is a simplified WFST example with $n = 2$. We can note that in the WFST for the n-gram language model, every state is annotated by m words, $0 \le m \le n-1$, as the word history. Also, we use ε transition to apply the back-off method mentioned above. After the introduction of ε transitions, an input sequence of words might have more than one *successful paths*. For example, in Fig.4 if the input string is ab, then we will have two successful paths: ab, a ε b, and their respective weightings $p(a)\cdot p(b \mid a)$, $p(a)\cdot\partial_a \cdot p(b \mid a)$, with ∂_a a back-off weight. Since in the search process, Viterbi search is applied, we thus can further assume that the path with back-off weight should always have lower score than the one without back-off weight. Therefore, in the search process we can only get $p(a)\cdot p(b \mid a)$ but never $p(a)\cdot\partial_a \cdot p(b \mid a)$.

3.5 Composition Operation

Given two WFSTs A and B, the composition operation takes the output labels of A as the input labels of B and construct a new WFST C, denoted as $C = A \circ B$. Each state of C is composed of a state of A and a state of B, and each *successful path p* of C is composed of a path p_a of A and a path p_b of B, with $i[p]=i[p_a], o[p_a]=i[p_b], o[p_b]=o[p]$, and $w[p]=w[p_a]+w[p_a]$. Table.1 is the complete composition algorithm. We need to especially note that if there are ε transitions existing in both A and B, as illustrated in Fig.5, there will be several possible concatenation sequence of p_a and p_b to form the path p of C. For example, in Fig.5 if we try to go from state <1,1> of C to state <3,2> of C, there are 5 different routes:

1. $<1,1>\rightarrow<1,2>\rightarrow<2,2>\rightarrow<3,2>$
2. $<1,1>\rightarrow<2,1>\rightarrow<2,2>\rightarrow<3,2>$
3. $<1,1>\rightarrow<2,1>\rightarrow<3,1>\rightarrow<3,2>$
4. $<1,1>\rightarrow<2,2>\rightarrow<3,2>$
5. $<1,1>\rightarrow<2,1>\rightarrow<3,2>$

These 5 different routes all have exactly the same score, and we need to have further mechanism in the algorithm to pick up one of them.

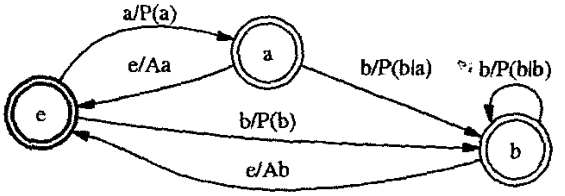

Fig.4 An example of a bi-gram FSA language model

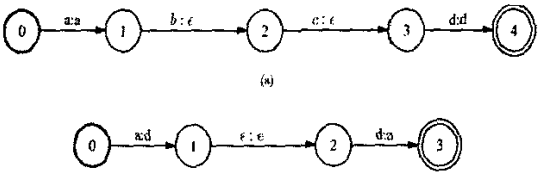

Fig.5 WFST A and WFST B with null transitions

```
COMPOSE(A, B)
 1: Let A = ⟨Q_a, i_a, F_a, Σ_a, Δ_a, δ_a⟩
 2: Let B = ⟨Q_b, i_b, F_b, Σ_b, Δ_b, δ_b⟩
 3: i ⇐ ⟨i_a, i_b⟩
 4: Q ⇐ {i}
 5: F ⇐ ∅
 6: Σ ⇐ Σ_a
 7: Δ ⇐ Δ_b
 8: δ ⇐ ∅
 9:
10: S ⇐ {i}
11: while S ≠ ∅ do
12:     ⟨q_a, q_b⟩ ⇐ an element in S
13:     S ⇐ S \ ⟨q_a, q_b⟩
14:     Q ⇐ Q ∪ ⟨q_a, q_b⟩
15:     if q_a ∈ F_a and q_b ∈ F_b then
16:         F ⇐ F ∪ ⟨q_a, q_b⟩
17:     end if
18:     T ⇐ TRANS(q_a, q_b, δ_a, δ_b)
19:     δ ⇐ δ ∪ T
20:     for all ⟨q'_a, q'_b⟩ ∈ T do
21:         if ⟨q'_a, q'_b⟩ ∉ Q then
22:             S ⇐ S ∪ ⟨q'_a, q'_b⟩
23:         end if
24:     end for
25: end while
26: return ⟨Q, i, F, Σ, Δ, δ⟩
```

Table.1 Algorithm for Composition operation

3.6 Viterbi Beam Search

After we have composed the three WFSTs together, the HMM model H, the pronunciation Lexicon L and the language model G, we have $S := H \circ L \circ G$, we can implement viterbi beam search on S. Given S = $< Q, n, F, \Sigma, \Delta, \delta >$, state q' $\in Q$, input label $\sigma \in \Sigma$, output label $x \in \Delta$, respective pdf B_σ for σ, a feature vector o_t at time t, we can compute the best score of state q at time t, $S(q,t)$, as:

$$S(q,t) = \min_{\delta(q',\sigma)=(q,x,w)} S(q',t-1) + B_\sigma(o_t) + w,$$

where w is the transition weight from q' to q.

when t = 0,

$$S(q,0) = \begin{cases} \bar{1} \ldots if & q = n \\ \bar{0} \ldots otherwise \end{cases},$$

where $\bar{1}$ and $\bar{0}$ represent *active* and *inactive* respectively. At each time t, we call a state q with $S(q,t) \neq \bar{0}$ an *active* state. At time 0, only the initial state is activated, and along with the transition within the WFST, the number of activated states increases rapidly. With limited computing power, we can't keep all the activated states when the number of them becomes too big. We can apply the same beam pruning strategy as we did in the one-pass search to keep those states with highest $S(q,t)$ only.

4 EXPERIMENTS

In this section, we compare the efficiencies of the two decoders, the one with one-pass tree copy search and the one with WFST, in terms of speed and memory requirements along with achieved character accuracies.

The acoustic models consist of 151 initial-final sub-syllabic units, including 112 Initials, 38 Finals, and a silence. The acoustic features we used is 39-dimentional, including 13 MFCCs and delta and delta delta MFCCs. The 60K-word trigram language model is estimated on a 40M-character corpus of news from the Central News Agency at Taipei for year 1997-1999, smoothed with Good-Turing discounting by SRI Language Modeling Toolkit. The test set consists of 100 Mandarin broadcast news collected from News98 Radio Station at Taipei in September 2002. The total length is 0.7 hours. The experiments are implemented on a computer with AMD Opteron 246 CPU and 8 GB RAM.

Table 2 is the number of states and arcs of the WFST, and Table 3 is the comparison of the running time, character accuracy, and memory usage of the two search programs with different Viterbi beam widths. The comparison between them may be better examined by the two curves in Fig5.and Fig.6. We can see that at the same running time, the character accuracy of the WFST approach is always higher than the traditional one, while the growth rate of the memory usage is lower.

	# of states	# of transitions
HMM model H	343	589
Pronunciation lexicon L	287,920	349,432
Trigram language model G	1,529,442	11,430,683
$H \circ L \circ G$	19,696,373	29,700,249

Table.2 Number of states and transitions used in WFST in the experiment

Beam width (10^4)		10	11	12	13	14	15	16
One-Pass Tree-Copy Search	Real-time factor	1.4	1.82	2.43	3.29	4.56	6.36	8.82
	Character accuracy	81.6	82.7	83.2	83.4	83.6	83.8	83.8
	Memory (MB)	662	725	856	1049	1281	1558	1737
FST-based search	Real-time factor	0.74	1.04	1.45	2.03	2.74	3.69	4.80
	Character accuracy	78.0	81.3	83.1	84.2	84.8	85.2	85.3
	Memory (MB)	752	779	802	824	838	850	868

Table.3 Comparison of the two decoders

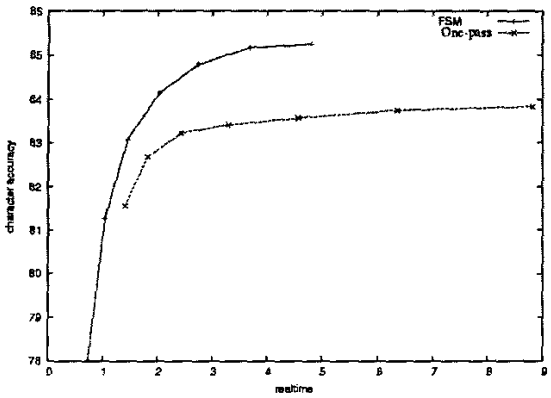

Fig.5 Comparisons on accuracies between the two decoders

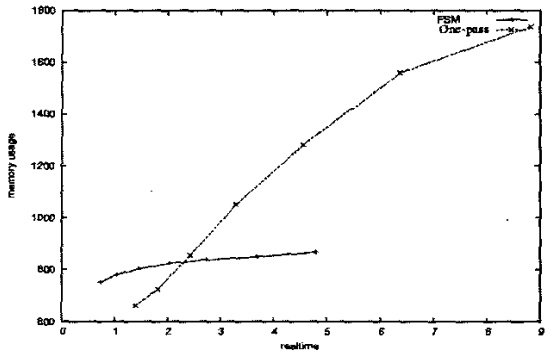

Fig.6 Comparison on memory usages between the two decoders

5 DISCUSSIONS

In this paper, we propose a miniature system to integrate different element models, H, L, G, for large vocabulary speech recognition, and we found it has comparable or better performance. M.Mohri further proposed some methods to promote FST's efficiency, such as the algorithms for determination [3], minimization [4], ε-removal [4], and weight pushing [6]. Given a non-deterministic WFST A, we can find an equivalent deterministic WFST B after the determination process. A deterministic WFST has fewer active states than non-deterministic one. After minimization, we can further find an equivalent C with minimal states. These algorithms all can help us further reduce the vast search space and have the decoder find the best path in a shorter time. Since not all non-deterministic WFSTs can be determined, C.Alluzen [7] proposed an optimization algorithm to transfer a non-determinable WFST A to an equivalent B but determinable. D.Caseiro [8] also proposed a method to lower the large number of memories required in the process of determination. With these algorithms implemented, it is believed the system performances can be further improved.

REFERENCES

[1] Mehryar Mohri, Fernando Pereira, and Michael Riley, "Weighted automata in text and speech processing," in *Extended Finite State Models of Language: Proceedings of the ECAI'96 Workshop*, Andras Kornai.Ed.,1996,pp. 46-50.

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

[3] Mehryar Mohri, "On some applications of finite-state sutomata theory to natural language processing," *Journal of National Language Engineering*, vol.2 pp. 1-20, 1996.

[4] Mehryar Mohri, "Minimization of sequential transducers," in *Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching*, M. Crochemore and D. Gusfield, Eds., Asilomar, CA, 1994, number 807, pp. 151-163, Springer-Verlag, Berlin.

[5] Mehryar Mohri, "Generic epsilon-removal and input epsilon-normalization alogorithms for weighted transducers," in *International Jounal of Foundations of Computer Science*, vol. 13, no. 1, pp. 129-143, 2002.

[6] Mehryar Mohri and Michael Riley, "A weighted pushing algorithm for large vocabulary speech recognition," in Proceedings of Eurospeech, 2001.

[7] Cyril Allauzen and Mehryar Mohri, "Generalized optimization algorithm for speech recognition transducers," in *Proceedings of ICASSP*, 2003.

[8] Diamantina Caseiro and Isabel Trancoso, "Using dynamic WFST composition for recognizing broadcast news," in *Proceedings of the ICSLP*, 2002.