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.Moorl[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.

3 FST-BASED SEARCH ALGORITHM

3.1 FST and WFST

An FST $\mathcal{A}$ can be represented as a six-tuple: $< Q, \Sigma, F, \delta, \lambda, \mu >$. $\mathcal{Q}$ is the set of all states in $\mathcal{A}$, while $n \in \mathcal{Q}$ is the initial state and $F \subseteq \mathcal{Q}$ is the set of all final states. $\Sigma$ is the alphabet set of input strings, while $\Delta$ is the alphabet set of out-
put strings. Finally, $\delta$ is the set of all possible transitions $(Q \times \Sigma \to Q \times \Delta)$. In speech recognition task, a weight is further attached to each transition. A weighted FST (WFST) is exactly the same as FST, except that $\delta$ is now the set of all possible transitions $(Q \times \Sigma \to Q \times \Delta \times K)$, where $K$ is the set of all possible weights. A transition $t = ([d[t]], [a(t)], [d(t)], [o(t)], w(t))$ 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 $a(t)$ 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 $t_1...t_n$ with $d[t_i] = t_{i-1}$, $i = 1,...,n-1$. Transitions with an empty string $\epsilon$ as the input label consumes no input. A successful path $\pi = t_1...t_n$ is the path from initial state $i$ to a final state $f \in F$. The input labels of the path $\pi$ is the result of the concatenation of the input labels of its constituent transitions: $[i\pi] = [i]a[t]...a[t_n]$ while the output labels is $o[\pi] = o[t]...o[t_n]$ similarly. The weight associated to $\pi$ is the sum of the initial weight, the weights of its constituent transitions and the final weight of the state reached by $\pi$.

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.

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.

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 form our HMM acoustic model $H$. At each final state of $H$, we additionally include an additional arc, with both input and output label $\epsilon$, returning to the initial state.

### 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:

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

where $w_i$ is the $i$-th word in the sequence, $w_{i-1}^{i-1} = (w_{i-1}, w_{i-2},..., w_1)$, and each probability $p(w_i \mid w_{i-1}^{i-1})$ can be calculated from training data. However, the quantity of the corpus needed grows exponentially with the increase of $n$. The back-off smoothing method is usually applied to compensate for the missing probability $p(w_i \mid w_{i-1}^{i-1})$ if the pattern $w_{i-1}^{i-1}$ does not appear in the limited training corpus, i.e., $p(w_i \mid w_{i-1}^{i-1}) = p(w_i \mid w_{i-1}^{i-2}) \cdot \delta_{w_i,w_{i-1}^{i-2}}$, where $\delta_{w_i,w_{i-1}^{i-2}}$ is a back-off weight (this back-off process can be repeated if $w_{i-1}^{i-2}$ 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, $0 \leq m \leq n-1$, as the word history. Also, we use $\epsilon$ transition to apply the back-off method mentioned above. After the introduction of $\epsilon$ 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 $\epsilon$ b, and their respective weightings $p(a) \cdot p(b \mid a)$, $p(a) \cdot \epsilon$, $p(\epsilon \mid a) \cdot b$ with $\epsilon$ 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 $p(a) \cdot \epsilon$, $p(\epsilon \mid a) \cdot b$.

### 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 $i[p]=i[p_A] \cdot o[p_B]=i[p_A] \cdot o[p_B]$ and $w[p]=w[p_A]+w[p_B]$. Table.1 is the complete composition algorithm. We need to especially note that if there are $\epsilon$ 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. $<1,1> \rightarrow <1,2> \rightarrow <2,2> \rightarrow <3,2>$
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.

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

\[
GLH \Delta D = \sum \frac{\sigma}{B} \frac{\Delta}{x},
\]

where \( \sigma \) is the transition weight from \( a \) to \( b \).

When \( t = 0 \),

\[
S(q,0) = \begin{cases} 1 \text{ if } q = n, \\ 0 \text{ otherwise} \end{cases}
\]

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-dimensional, 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 Fig 5 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.
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. Allauzen [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