A One-Pass Real-Time Decoder Using Memory-Efficient State Network

Jian SHAO\(^{(a)}\), Student Member, Ta LI\(^{(b)}\), Qingqing ZHANG\(^{(a)}\), Qingwei ZHAO\(^{(b)}\), and Yonghong YAN\(^{(c)}\), Nonmembers

SUMMARY This paper presents our developed decoder which adopts the idea of statically optimizing part of the knowledge sources while handling the others dynamically. The lexicon, phonetic contexts and acoustic model are statically integrated to form a memory-efficient state network, while the language model (LM) is dynamically incorporated on the fly by means of extended tokens. The novelties of our approach for constructing the state network are (1) introducing two layers of dummy nodes to cluster the cross-word (CW) context dependent fan-in and fan-out triphones, (2) introducing a so-called "WI layer" to store the word identities and putting the nodes of this layer in the non-shared mid-part of the network, (3) optimizing the network at state level by a sufficient forward and backward node-merge process. The state network is organized as a multi-layer structure for distinct token propagation at each layer. By exploiting the characteristics of the state network, several techniques including LM look-ahead, LM cache and beam pruning are specially designed for search efficiency. Especially in beam pruning, a layer-dependent pruning method is proposed to further reduce the search space. The layer-dependent pruning takes account of the neck-like characteristics of WI layer and the reduced variety of word endings, which enables tighter beam without introducing much search errors. In addition, other techniques including LM compression, lattice-based bookkeeping and lattice garbage collection are also employed to reduce the memory requirements. Experiments are carried out on a Mandarin spontaneous speech recognition task where the decoder involves a tri-gram LM and CW triphone models. A comparison with HDecode of HTK toolkit shows that, within 1% performance deviation, our decoder can run 5 times faster with half of the memory footprint.

key words: real-time, memory-efficient, layer-dependent beam pruning

1. Introduction

There are several knowledge sources involved in large vocabulary continuous speech recognition (LVCSR), namely, the language models (LM), the pronunciation lexicon, the context dependent phones and the hidden Markov models (HMM). The key features of a successful one-pass decoding scheme include the ability to incorporate them whilst keeping the time and memory requirements within acceptable bounds\[3\]. To obtain the target, it is desirable to organize the search network compactly and to apply efficient search space pruning.

From the viewpoint of search space representation, the search network can be expanded either statically or dynamically\[14\]. For static expansion, the search network is compiled completely beforehand and kept statically in memory. The most popular approach using this design is the weighted finite-state transducers (WFST) method\[13\]. Determinization and minimization\[7\] proved to be useful approaches for reducing the size of the search network. However, the feasibility of this method relies strongly on the inherent redundancies of the knowledge sources. If very large models are considered, it may be incapable of compiling the search network beforehand. For dynamic expansion, such large models could still be applied, since only part of the search space has to be kept in memory. Usually the pronunciation lexicon is statically compiled into a phonetic prefix tree, whereas the LM information is dynamically incorporated by means of tree copies\[2\]–\[4\]. If cross-word (CW) contexts are considered, the fan-out arcs are also treated dynamically\[16\].

In contrast to above WFST based and tree copy based decoding scheme, this paper presents a "hybrid" scheme which compiles and optimizes the lexicon, phonetic contexts and acoustic model statically while dynamically incorporates the LM information by means of extended tokens. By storing both the prefixes and suffixes in a tree structure and optimizing it at state level, the lexicon is structured as a state network. However, a problem inherent to the use of network structure is to find a position in a path to locate the word identity with the guarantee that each path through the network represents exactly one word. Our solution is to introduce one type of dummy nodes to store the word identities and to put them in non-shared mid-part of the network. The network is organized as a multi-layer structure for search. All the dummy nodes with word identities form a so-called "WI layer" which is positioned in the middle of the network. By exploiting the bottle-neck characteristics of WI layer and considering the reduced variety of word endings, it is possible to use tighter beam widths for tokens at WI layer and his following layers for more efficient search space pruning. In order to further speed up the search process, LM look-ahead\[6\] and LM cache\[5\],\[9\] are specially designed. Moreover, for memory efficiency, other techniques including LM compression\[9\],\[11\], lattice-based bookkeeping and lattice garbage collection\[12\] are also employed.

The organization of this paper is as follows. Section 2 introduces the state network and its construction. Section 3 describes the one-pass DP search algorithm for decoding the state network. The experimental results produced by the decoder are discussed in Sect. 4. Finally, conclusions are given in Sect. 5.
2. The State Network

2.1 Structure of the State Network

The state network is pre-compiled from a linear lexicon and phonetic decision trees. The linear lexicon defines a list of words with their monophone sequence. A three-word linear lexicon is given in Fig. 1. If CW triphone models are used, the monophone sequence is converted into a triphone sequence. According to their positions, the triphone sequence can be separated into three parts: the initial fan-in triphone, the middle triphones and the final fan-out triphone. Here, the initial fan-in triphones are context dependent variants of the initial phone which are expanded according to any possible left CW contexts. Similarly, the final fan-out triphones are variants of the final phone according to any possible right CW contexts. Sharing both the fan-in and fan-out triphones lead to a phonetic network. Together with phonetic decision trees which give a clustering of the HMM states, the phonetic network can be expanded into an original state network. Figure 2 shows the original state network for the three-word lexicon (the special words SENT_START, SENT_END and SENT_PAUSE used for denoting the silence at sentence borders or shorter pause inside a sentence are omitted in the network for clarity). In Fig. 2, a triphone instance with central phone b, left context a and right context c is denoted by “a – b + c” and expanded with its corresponding three states. For efficiency, the original state network can be further optimized at state level. Figure 3 shows the final optimized state network for the three-word lexicon.

The state network is a left-to-right graph and structured in three parts: individual mid-part, shared initial part (fan-in triphones) and final part (fan-out triphones). There are two types of nodes in the network: state nodes and dummy nodes. Each state node is associated with an HMM state index, while dummy node is not related to any acoustic event and used to compress network or mark special situations. In Figs. 2 and 3, symbol “O” denotes state node and symbol “□” denotes dummy node. The state index is marked around the node, while the number inside the node is the node identity which is only used for description.

Five types of dummy node are introduced: ROOT, WE, FI, FO and WI.

- ROOT node is the root of the state network.
- WE node is used to mark the end of a word.
- FI node is used to gather the fan-in triphones with identical initial phone and left intra-word context.
- FO node is used to gather the fan-out triphones with identical final phone and left inter-word context.
- WI node is used to store the word identity.

Three types of state nodes are defined: SP, AW and DW.

- SP node is tied with a sp state for allowing an optional period of silence between word instances.
- AW nodes are nodes between FI and WI nodes.
- DW nodes are nodes between WI and SP nodes.

2.2 Constructing the State Network

Three steps are taken to construct the state network.

Step 1: Build an original state network by clustering the fan-in and fan-out triphones. An example is shown in
Fig. 4 The concept of forward-backward node-merge process.

Fig. 2. All words in linear lexicon are added to the network one by one. The procedure to add one word to an existing network is described as follows:

- Create the mid-part and put WI node at the end.
- Create the fan-in triphones if not existed in the initial part and link them to corresponding FI nodes and the first state node of the mid-part.
- Create the fan-out triphones and append the SP and WE nodes if not existed in the final part, and link them to corresponding FO and FI nodes.

Step 2: Optimize the network by a forward node-merge process, which is implemented recursively starting from FI nodes and FO nodes in left-to-right direction. Figure 4 shows the concept of the forward node-merge process: (1) Two sister nodes with the same predecessors and the same state are forward-merged. In Fig. 4, sister nodes N5 and N6 with the same predecessors N3 and N4 and the same state s are merged into N9. (2) All successors of two sister nodes become successors of the merged node. It is from Fig. 2 that node N17 and N22 with the same parent node N16 and state index 2298 are merged into node N64 in Fig. 3. Other nodes N61-N63 in Fig. 3 are also created in this step. It should be noted that putting WI node at the end of the mid-part in the first step makes the forward merge process sufficient.

Step 3: Optimize the network by a backward node-merge process. First, put each WI node forward up to an ancestor node with multiple successors. Then, implement the backward node-merge process to merge the state nodes recursively starting from FI nodes and FO nodes in reverse direction. Figure 4 also shows the concept of the backward node-merge process: (1) Two sister nodes with the same successors and the same state are backward-merged. In Fig. 4, sister nodes N3 and N4 with the same successor N9 and the same state σ are merged into N10. (2) All predecessors of two sister nodes become predecessors of the merged node. Finally, put the WI nodes forward again to ensure that they are the first non-shared nodes. From Fig. 2, WI node N33 and N51 are moved forward, thus Node N32 and N50 with the same state index 3731 and the same successor node N34 can be merged as N68 in Fig. 3. Other Nodes N65-N67, N69-N71 in Fig. 3 are also created in this step.

2.3 Organizing the Network in Multi-Layer Structure

A depth is assigned to each node in the network. All dummy nodes with the same type are identified by a unique depth. The depth of a state node is defined as the depth of its last ancestor dummy node plus the number of consecutive nodes between them. All nodes at the same depth form a layer of the network. In this way, FI nodes and WE nodes form the first and last layer of the network, while WI nodes form an individual layer in the middle of the network. The AW nodes are positioned between FI layer and WI layer. The DW nodes are positioned between WI layer and SP layer. Nodes at the same layer may have some common properties, which enables the use of distinct token propagation strategy and different pruning beam during search.

3. One-Pass DP Search

3.1 Token Structure and Recombination

The traditional token structure [1] representing a partial path is extended in this paper. Besides the path probability and traceback information, each token also contains LM history and current word identity. Current word identity equals "unknown" ("unk" in short) at the beginning of the network and is filled in whenever a WI node is traversed. The additional information has to be considered for token recombination. Three requirements must be met for two tokens to be considered equivalent:

- Reside in the same node of the network.
- Contain the same current word identity.
- Have the same immediate LM history.

3.2 LM Look-Ahead, Compression and Cache

The traditional token structure [1] representing a partial path is extended in this paper. Besides the path probability and traceback information, each token also contains LM history and current word identity. Current word identity equals "unknown" ("unk" in short) at the beginning of the network and is filled in whenever a WI node is traversed. The additional information has to be considered for token recombination. Three requirements must be met for two tokens to be considered equivalent:

- Reside in the same node of the network.
- Contain the same current word identity.
- Have the same immediate LM history.

The LM probability is incorporated into the accumulated score when a token is propagated into a WI node. Since the WI node in our state network is moved as forward as possible, this naturally offers some forms of LM forwarding. Furthermore, the LM look-ahead, which is achieved by factoring the LM probabilities over the nodes of the phonetic lexical tree [6], is extended to be applied in our state network. For a bigram LM, the factored LM probability \( \pi_v(s_{aw}) \) for AW node \( s_{aw} \) and predecessor word \( v \) is defined as:

\[
\pi_v(s_{aw}) = \max_{w \in W(s_{aw})} p(w|v),
\]

where \( W(s_{aw}) \) is the possible current words that can be found in successor WI nodes, the terms \( p(w|v) \) denotes the conditioned bigram probability. Likewise, the factored LM probability \( \pi_w(s_{dw}) \) for DW node \( s_{dw} \) and current word \( w \) is defined as:

\[
\pi_w(s_{dw}) = \max_{z \in W(s_{dw})} p(z|w),
\]

where \( W(s_{dw}) \) is the possible following words that can be found by re-entering back to the network. All entries of \( \pi_v(s_{aw}) \) and \( \pi_w(s_{dw}) \) are pre-computed and stored as a LM look-ahead model.

Normally, both the size of N-gram model and the size
of LM look-ahead model are very large. One solution is to reduce the model size by quantizing the 4-byte floating probabilities into 1-byte indices. During search, the true probabilities are looked up from their appropriate tables. Another is to keep the models on disk and only load the required parts on demand. However, to avoid excessive delays due to disk access, we must resort to some caching strategy [5], [9]. For N-gram model, when a bigram \( p(w|v) \) is needed and it is not in memory, all the bigrams follow the same word \( v \) are read and cached into memory. For LM look-ahead model, several cache arrays (say 200) are allocated for the three fan-in triphone layers which hold the most tokens. When a factored LM probability \( \pi_v(s_{aw}) \) with predecessor word \( v \) is needed and it is not in cache array, all the LM factored probabilities \( \pi_v(s_{aw}) \) for each node \( s_{aw} \) in the fan-in triphone layers are read and cached in cache array.

3.3 DP Recursions

For a quantitative specification of search procedure on such a network, the notations defined in [10] are slightly modified:

- \( Q_{aw}(t, s_{type}, w) := \text{score of the best partial path that ends at time } t \text{ in state node } s_{type} \text{ for LM history } (u, v) \text{ and current word } w \text{ (}w\text{ may be unknown)}. \)
- \( \tilde{Q}_{aw}(t, s_{type}, w) := \text{modified score that incorporates } Q_{aw}(t, s_{type}, w) \text{ with its anticipated LM probability}. \)
- \( H(v, w, r(w); t) := \text{the joint probability of generating the acoustic vectors } x_1 \cdots x_t \text{ and a word sequence } w_1 \cdots w_n \text{ with ending words } (v, w), \text{right context } r(w) \text{ and ending time } t. \)

To avoid abrupt change in score, a token in AW node holds the modified score \( \tilde{Q}_{aw}(t, s_{type}, w) \), while a token in DW node holds score \( Q_{aw}(t, s_{type}, w) \). The DP search recursions over different types of nodes are defined as follows:

- Recursions at AW, DW, SP nodes:
  \[ Q_{aw}(t, s_{aw}, unk) = \max_{\sigma} \left\{ p(x_t, s_{aw}|\sigma), \frac{Q_{aw}(t-1, s_{aw}, unk)}{\pi_v(\sigma)} \right\} \]
  \[ \tilde{Q}_{aw}(t, s_{aw}, w) = \max_{\sigma} \left\{ p(x_t, s_{aw}|\sigma), Q_{aw}(t-1, s_{aw}, w) \right\} \]
- Recursions at WI, WE nodes:
  \[ Q_{aw}(t, s_{we}, w) = \max_{\sigma} \left\{ Q_{aw}(t-1, s_{we}, unk), p(x_t|s_{we}) \right\} \]
  \[ H(v, w, r(w); t) = \max_{\sigma} \left\{ \frac{Q_{aw}(t, s_{we}, w)}{\pi_v(\sigma)}, p(w|u, v) \right\} \]
  \[ Q_{aw}(t, s_{fi}, unk) = H(v, w, r(w); t), \]

where \( \sigma \) is one of the predecessor node of state node \( s_{type}, p(x_t, s_{type}|\sigma) \) denotes the product of the probability for the transition from the state of \( \sigma \) to the state of \( s_{type} \) and the emission probability of state of \( s_{type} \) at time \( t, r(w) \), the right context of word \( w \), which is implicitly determined by WE node \( s_{we} \) in the network.

The above recursions can be implemented by moving tokens along the network. In order to avoid recombination between the old tokens and the newly created tokens which have observed one additional frame of acoustic data, tokens are propagated in reverse depth order (higher depth first). If there are multiple successors, special operations are performed in terms of its node type. Details are described in Table 1.

<table>
<thead>
<tr>
<th>Table 1 One-pass DP search on the state network.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proceed over frame ( t ) from left to right</td>
</tr>
<tr>
<td>ACOUSTIC LEVEL: propagate tokens to successors</td>
</tr>
<tr>
<td>For each SP node, operations are performed according to the node type of successors.</td>
</tr>
<tr>
<td>- SP: using DP ( Q_{aw}(t, s_{sp}, w) ).</td>
</tr>
<tr>
<td>- WE: just copying.</td>
</tr>
<tr>
<td>For each DW node in reverse depth order</td>
</tr>
<tr>
<td>- DW: using DP ( Q_{aw}(t, s_{aw}, w) ).</td>
</tr>
<tr>
<td>- SP: using DP ( Q_{aw}(t, s_{sp}, w) ) or skipping SP node to WE nodes.</td>
</tr>
<tr>
<td>- WE: just copying.</td>
</tr>
<tr>
<td>For each AW node in reverse depth order</td>
</tr>
<tr>
<td>- AW: using DP ( Q_{aw}(t, s_{aw}, unk) ) and updating anticipated probability.</td>
</tr>
<tr>
<td>- WI: using DP ( Q_{aw}(t, s_{we}, w) ), updating the word identity and LM probability.</td>
</tr>
<tr>
<td>WORD LEVEL.</td>
</tr>
<tr>
<td>For each WI node</td>
</tr>
<tr>
<td>- DW: using DP ( Q_{aw}(t, s_{aw}, w) ).</td>
</tr>
<tr>
<td>For each WE node</td>
</tr>
<tr>
<td>- For each triple ((v, w, t))</td>
</tr>
<tr>
<td>- Token recombination at word level:</td>
</tr>
<tr>
<td>- Create a trace node filled with the word identity ( w ) , end time ( t ) and joint probability ( H(v, w, r(w); t) ) and attach the trace node to all predecessor trace nodes ( H(u, v, r(v); t) )</td>
</tr>
<tr>
<td>- Update token context: ( Q_{aw}(t, s_{we}, unk) = H(v, w, r(w); t) )</td>
</tr>
<tr>
<td>- Propagate recombined token over FL node to its AW successors: Using DP ( Q_{aw}(t, s_{aw}, unk) )</td>
</tr>
</tbody>
</table>

3.4 Beam Pruning

Beam pruning is used to reduce the search space and computation cost. They are implemented at each time step by keeping a record of the best token and de-activating all tokens whose log probabilities fall more than a beam-width below the best. Normally, three forms of pruning were used: acoustic pruning, word end pruning and histogram pruning [4], [10]. For word end pruning, since the range of the language model likelihood is much smaller over different histories than over different words, this, coupled with the greater degree of certainty of word identity at word end, leads to the use of a tighter beam width for word end tokens [4]. By extending the idea behind the word end pruning, it is possible to enable the use of separate pruning
thresholds and reference likelihood scores for the nodes with common properties. In our hierarchical network, the nodes with identical type or depth have some common properties. Therefore, a layer-dependent pruning method is proposed.

Before introducing the pruning approach, the best score over AW, WI, DW, and WE nodes are firstly defined respectively:

\[
\begin{align*}
Q_{AW}(t) & = \max_{(w,W,w,\text{type})} (\tilde{Q}_w(t, s_{dw}, \text{unk})) \\
Q_{WI}(t) & = \max_{(w,W,w,\text{type})} (Q_w(t, s_{wi}, w)) \\
Q_{DW}(t) & = \max_{(w,W,w,\text{type})} (\tilde{Q}_w(t, s_{dw}, w)) \\
Q_{WE}(t) & = \max_{(w,W,w,\text{type})} (Q_w(t, s_{we}, w)).
\end{align*}
\]

The best score over all nodes is defined as

\[
Q_{AC}(t) = \max_{\text{type}(AW,W,W,DW,WE)} (Q_{\text{type}}(t)).
\]

According to above definitions, the three-step pruning approach [4], [10] is re-formulated as follows:

- Acoustic pruning is applied over all type of nodes. Tokens are removed if \(Q_{\text{type}}(t) < f_{AC} \cdot Q_{AC}(t)\). Here, \(f_{AC}\) is the so-called acoustic pruning beam.
- Layer-dependent pruning is only applied to the nodes with identical type or depth. Tokens in WI, DW and WE nodes are removed respectively if
  \[
  \begin{align*}
  Q_{AW}(t, s_{wi}, w) & < f_{WI} \cdot Q_{WI}(t) \\
  Q_{AW}(t, s_{dw}, w) & < f_{DW} \cdot Q_{DW}(t) \\
  Q_{AW}(t, s_{we}, w) & < f_{WE} \cdot Q_{WE}(t).
  \end{align*}
  \]

Here, \(f_{WI}\), \(f_{DW}\), and \(f_{WE}\) are the so-called layer-dependent pruning beams. Tighter beam can be used for them.
- Histogram pruning method [4] is employed to limit the number of survived tokens to a maximum number (MaxToken).

3.5 Word Graph Generation

In order to extend the single-best decoding algorithm for word graph construction, what has to be added is the bookkeeping of all the word sequence hypotheses that are recombined into just one hypothesis to start up next word models [8]. Normally, bookkeeping list is used to store the trace back information. For efficiency reasons, we use a trace lattice structure instead. The trace lattice is an acyclic directed graph consisting of nodes and arcs. Each node contains a word identity \(w\), the word end time \(t\) and the joint probability \(H(w, v, r(w); \tau, t)\). Each arc contains LM likelihood \(p(w|u, v)\). The lattice is expanding during search. As shown in Table 1, the trace node is not created at WI nodes but delayed until a token reaches to a WE node. It is needed to note that the trace node is created after WE pruning. In this way, the WE pruning can be used to control the size of trace lattice. In addition, to limit the memory needs for storing the trace lattice, a lattice garbage collection strategy [12] is applied. When a token is discarded, trace back the lattice node pointed by this token, if the lattice node is not pointed by any active tokens, the node is marked as dead, the memory they consume can be reused as the lattice continuously grows in time.

At the end of utterance, the trace lattice is converted to the HTK standard lattice format (SLF) word lattice. The acoustic likelihood \(h(l, w, r; \tau, t)\) attached to an arc of SLF lattice is defined as

\[
h(l, w, r; \tau, t) = \frac{H(v, w, r(v); \tau)}{H(u, v, r(v); \tau). p(w|u, v)}.
\]

4. Experimental Results

4.1 Training Data, Development Set and Test Set

Experiments were carried out on Mandarin spontaneous speech corpus. The acoustic training data consists of three parts: CallHome&CallFriend (45.9 hours), HTr04 (150 hours) and 863Tr05 (5 hours), yielding a total of about 200 hours of data.

HTr04 was collected by Hong Kong University of Science and Technology (HKUST) in 2004. A development set, comprising of 4 hours of data and 24 phone calls, was also released. We separated it into two parts: HDev04-I and HDev04-II. HDev04-I with two and a half hours data is used as a holdout set to optimize the LM perplexity. HDev04-II with the residual one and a half hour data is used as our development set.

863Tr05 is the training set for the keyword spotting task in China 2005 HTRDP (HTRDP, i.e., the "863" Program) automatic speech recognition evaluation. All the data were recorded through the landline telephone with local service in real world with environmental noise. All utterances are in Chinese Mandarin and in spontaneous style. More details about the corpora are described in [18]. The test set of the evaluation (863Test05) with one hour data is used as our test set.

4.2 Acoustic Model Training

There were three steps for the front-end process. First, a reduced bandwidth analysis, 60-3400Hz, was used to generate 52-dimensional PLP feature vectors which consist of 12 PLP Cepstra along with the zeroth Cepstra and their first, second and third-order differences. Next, utterance-based Cepstra mean and variance normalization (CMS/CVN) was applied. Finally, a heteroscedastic linear discriminant analysis (HLDA) was directly applied to project 52-dimensional feature vectors into 39-dimensions.

The acoustic models were trained by minimum phone error (MPE) training [15], which was seeded by maximum mutual information estimated (MMIE) models. All the triphone HMMs are 3-state left-to-right topology. A robust
Table 2  Size and perplexity of the bigram and trigram.

<table>
<thead>
<tr>
<th>Property</th>
<th>Bigram</th>
<th>Trigram</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size of LM in ARPA format</td>
<td>464 MB</td>
<td>1143 MB</td>
</tr>
<tr>
<td>Size of LM in binary format</td>
<td>147 MB</td>
<td>409 MB</td>
</tr>
<tr>
<td>Size of LM in compact format</td>
<td>78 MB</td>
<td>191 MB</td>
</tr>
<tr>
<td>PP on HDev04-II</td>
<td>197</td>
<td>169</td>
</tr>
<tr>
<td>PP on 863Test05</td>
<td>258</td>
<td>233</td>
</tr>
</tbody>
</table>

Table 3  Size of resulted network for 40k word set.

<table>
<thead>
<tr>
<th>Network</th>
<th>Nodes</th>
<th>Arcs</th>
<th>Mem.(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>WW</td>
<td>526 382</td>
<td>605 591</td>
<td>8.54</td>
</tr>
<tr>
<td>forward</td>
<td>213 342</td>
<td>292 551</td>
<td>3.64</td>
</tr>
<tr>
<td>final</td>
<td>129 072</td>
<td>174 018</td>
<td>2.19</td>
</tr>
<tr>
<td>CW</td>
<td>1 102 082</td>
<td>7 915 241</td>
<td>43.84</td>
</tr>
<tr>
<td>forward</td>
<td>419 954</td>
<td>936 463</td>
<td>12.49</td>
</tr>
<tr>
<td>final</td>
<td>145 233</td>
<td>243 805</td>
<td>2.66</td>
</tr>
</tbody>
</table>

state clustering with two-level phonetic decision trees [17] was used. 5210 shared states were empirically determined with 48 Gaussians per state.

4.3 Language Modeling and Compression

The SRILM tools [20] were used to build the N-gram language models. The LM training data consists of four parts: the transcriptions of the acoustic training data, general web data provided by the University of Washington [19], self-collected data using Google with N-gram queries from transcriptions and news corpora of 2005. The HDev04-I was used as a holdout set to optimize LM perplexity. The interpolating LM models were primarily in ARPA format. They were converted and quantized into a binary compact format. As Table 2 shows, the quantization-based LM compression technique can reduce the size of LM by a factor of two. The LM perplexities (PP) of the bigram and trigram models on HDev04-II and 863Test05 are also given.

4.4 State Network Building

Table 3 gives detail information of the resulting networks by using 179 mono phones for a size of 44 902 word set. Performing forward-backward node-merge process reduces the memory by a factor of 4 for within-word (WW) context and by a factor of 16 for CW context. By contrast, using CW context only increases the memory size of final network from 2.19 MB to 2.66 MB. These results clearly indicate the efficiency of the proposed forward-backward node-merge process, especially under CW context constrains. The CW final network were used in the following experiments. It is organized as a 69-layer network.

4.5 LM Look-Ahead

In the first set of experiments on HDev04-II, the effect of LM look-ahead on the size of search space and Chinese character error rate (CER) are investigated. Figure 5 shows the results by varying the acoustic beam width $f_{AC}$ (in log likelihood) while keeping the other parameters fixed ($logf_{WE} = 200$, $logf_{WI} = 200$, $logf_{DW} = 200$, $MaxToken = 1000k$). Without LM look-ahead, using a beam width of 150 consumes 226k average active tokens per frame but only reaches a CER of 47.9%. Under unigram look-ahead, using a beam width of 140 achieves a CER of 46.8% with 110k average tokens. Under bigram look-ahead, using 130 achieves the same CER with only 44k average tokens. In total, using bigram look-ahead against no look-ahead reduces the search space by a factor of more than five without introducing any search errors.

4.6 Beam Pruning

In the second set of experiments on HDev04-II, the effect of several pruning techniques on the size of search space and CER are investigated.

First, the acoustic pruning is investigated. For the bigram look-ahead case in Fig. 5, using a beam width of 110 compared to 130 results in a factor of four reductions in the number of average tokens and only increases the error rate from 46.8% to 47.4%.

Next, the layer-dependent pruning is investigated. Figure 6 shows the results. By using WI pruning the search space is reduced by a factor of 2.6 (from 11.04k for $f_{WI} = 200$ to 4.27k for $f_{WI} = 50$) at a loss of absolute 0.2% error rate (from 47.4% to 47.6%). By using DW pruning, the search space is further reduced by a factor of 1.71 (from
4.27 k for $f_{DW} = 200$ to 2.49 k for $f_{DW} = 60$) without introducing any additional search errors. These results clearly demonstrate that the power of WI pruning and DW pruning is great. One reason for this is that the reduced variety of word ending enables tighter beam width. Another reason is that the bottleneck and word-end-like characteristics of WI node offer some inherent ability to control the search space. On the other hand, the power of WE pruning is very limited. The degradation of WE pruning is mainly because of the introduction of WI pruning which partly substitutes the effect of WE pruning. However, the WE pruning is still indispensable. As described in Subsection 3.5, the WE pruning is also used to control the size of trace lattice. For some application related to word graph, a tradeoff between the size and the quality of word graph is necessary.

Finally, histogram pruning is investigated. As Fig. 6 shows, by limit the number of active tokens, the search space is further reduced by about 12% (from 2.49 k to 2.18 k) without introducing any search errors.

In total, a combination of above pruning techniques reduces the search space by a factor of twenty and increases the error rate by absolute 0.8% (from 46.8% to 47.6%).

### 4.7 Comparison with HDecode

In the last set of experiments on 863Test05, a comparison with HDecode [21] is made. The current version of HDecode supports trigram and bigram full decoding with cross-word triphone models and LM look-ahead. For convenience, our decoder is referred to as “TDecode” in the following. Two decoders used the same acoustic models (Our acoustic model was converted into HTK format for HDecode) and the same language models (HDecode used the uncompressed ARPA format model). However, the used recognition networks for two decoders are different. For HDecode, a word level network is first built and then expanded into a 9-layer triphone-model-based network. While for TDecode, as described in Sect. 2 and Subsection 4.4, the recognition network is a 69-layer state network. In addition, the used pruning techniques are also different. For HDecode, only the conventional three forms of pruning were used. For TDecode, the layer-dependent pruning (WI and DW pruning) was added. In order to compare the performances of the two decoders at different real time factors (RTFs), we properly adjust the pruning parameters including $f_{AC}$, $f_{WI}$, $f_{WE}$, $f_{DW}$ and MaxToken at different pruning levels. Table 4 shows the RTF and CER as a function of the average number of active tokens per time frame. Here, the RTF is the number of seconds needed to process one second of speech on a Pentium-D 3.40 Ghz PC with 2.00 GB RAM. In addition, the amount of peak memory usage (PMU) is also given. When loosing pruning parameters, the memory requirement of tokens and trace lattice will increase. However, the amount of increased memory compared to the size of the used language model and acoustic model is small. In Table 4, only the highest value that is observed among different pruning parameters is given.

<table>
<thead>
<tr>
<th>Table 4</th>
<th>Comparison between HDecode and TDecode on 863Test05.</th>
</tr>
</thead>
<tbody>
<tr>
<td>LM</td>
<td>Decoder</td>
</tr>
<tr>
<td>HDecode</td>
<td>bigram</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>trigram</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>HDecode</td>
<td>bigram</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>trigram</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In bigram case, HDecode can not reach real-time performance for this spontaneous 40 k word set task. At a RTF of 1.01, the CER is only 51.5%. Compared to the best CER of 46.1% at a RTF of 5.66, search errors (i.e, errors that are caused by loosing the most probable state sequences due to pruning) account for 10.5% of total error rate. In contrast for TDecode, by using the additional layer-dependent pruning and LM cache technique, at a RTF of 0.98 the CER is 46.2%. Compared to the best CER of 45.8% at a RTF of 1.71, search errors only account for 1% of total error rate. Furthermore, compared to HDecode with a best CER of 46.1%, TDecode achieves a best CER of 45.8%. This improvement is made due to the more refined processes for long or short silence in TDecode.

In trigram case, some improvements in CER are obtained for two decoders. However, the runtime and memory requirements increase, especially for HDecode. With the guarantee that search errors account for less than 1% of total error rates, HDecode requires five times real-time, while TDecode is still capable of reaching real-time performance with half of memory usage.

### 5. Conclusions

This paper presents a time-synchronous decoder that is able to integrate all the knowledge sources in single pass. The pronunciation lexicon together with cross-word triphone contexts and phonetic decision trees are pre-compiled into the novel compact state network. To obtain the real time performance even with good accuracy, two main ways are proposed. One is to dynamically incorporate the LM information as early as possible by means of word identity forwarding, LM look-ahead and LM cache techniques. Another is to reduce the search space as much as possible by means of compact state network and efficient pruning techniques especially the additional layer-depend pruning.

Three sets of experiments are carried out on Mandarin spontaneous speech corpus. The first two sets of experiments on the development set HDev04-II show that using bigram look-ahead leads to a reduction of search space by
a factor of more than five without introducing any search errors. The combination of several pruning techniques, especially the layer-dependent pruning, is able to further reduce the search space by a factor of twenty at a lost of absolute 0.8% error rate. The last comparative experiments with HDecode on the test set 863Test05 show that, with trigram language model and cross-word triphone contexts, the proposed decoder can achieve real-time performance (5 times faster than HDecode) and consume half of memory of HDecode even with slightly better recognition accuracy.

Acknowledgments

This work is partially supported by MOST (973 program, 2004CB318106), National Natural Science Foundation of China (10574140, 60535030), The National High Technology Research and Development Program of China (863 program, 2006AA010102, 2006AA01Z195).

References


Jian Shao received his B.S. degree in electrical science and engineering from Nanjing University in 2003. Now he is a Ph. D candidate of ThinkIT Speech Lab, Institute of Acoustics (IOA), Chinese Academy of Sciences (CAS). His research is focused on spontaneous speech recognition and spoken document retrieval.

Ta Li received his B.S. degree in electrical engineering from Nanjing University in 2003. He now is a Ph. D candidate of ThinkIT Speech Lab, IOA, CAS. His research interests include decoding algorithms for speech recognition and embedded speech recognition systems.

Qingqing Zhang graduated from Beijing University of Posts and Telecommunications (BUPT) in July 2005 with Bachelor’s degree from School of Telecommunication Engineering. Currently she is a Ph. D candidate of ThinkIT Speech Laboratory, IOA, CAS. Her research interests include speech signal processing and speech recognition.

Qingwei Zhao received his PhD in electronic engineering from Tsinghua University in 1999. Now he is an Associate Professor at ThinkIT Speech Lab, IOA, CAS. Before joining CAS, he was with Intel as a Senior Researcher. His research interests include spontaneous speech recognition, keyword spotting and automatic pronunciation evaluation.
Yonghong Yan received his BE from Tsinghua University in 1990, and his PhD from Oregon Graduate Institute (OGI). He worked in OGI as Assistant Professor (1995), Associate Professor (1998) and Associate Director (1997) of Center for Spoken Language Understanding. He worked in Intel from 1998–2001, chaired Human Computer Interface Research Council, worked as Principal Engineer of Microprocessor Research Lab and Director of Intel China Research Center. Currently he is a professor and director of ThinkIT Lab. His research interests include speech processing and recognition, language/speaker recognition, and human computer interface. He has published more than 100 papers and holds 40 patents.