ON LATTICE GENERATION FOR LARGE VOCABULARY SPEECH RECOGNITION

David Rybach, Michael Riley, Johan Schalkwyk

Google Inc., USA

ABSTRACT

Lattice generation is an essential feature of the decoder for many speech recognition applications. In this paper, we first review lattice generation methods for WFST-based decoding and describe in a uniform formalism two established approaches for state-of-the-art speech recognition systems: the *phone pair* and the *N-best histories* approaches. We then present a novel optimization method, *pruned determinization* followed by minimization, that produces a deterministic minimal lattice that retains all paths within specified weight and lattice size thresholds. Experimentally, we show that before optimization, the phone-pair and the N-best histories approaches each have conditions where they perform better when evaluated on video transcription and mixed voice search and dictation tasks. However, once this lattice optimization procedure is applied, the phone pair approach has the lowest oracle WER for a given lattice density by a significant margin. We further show that the pruned determinization presented here is efficient to use during decoding unlike classical weighted determinization from which it is derived. Finally, we consider *on-the-fly lattice rescoring* in which the lattice generation and combination with the secondary LM are done in one step. We compare the phone pair and N-best histories approaches for this scenario and find the former superior in our experiments.

Index Terms: decoder, lattice, WFST

1. INTRODUCTION

The speech recognition decoder computes the most likely word sequence for a given (audio) input signal. For many applications it is required to generate more than just the best scoring hypothesis. For example, to offer alternative results to the user, to rerank the hypotheses using a model that cannot be applied during decoding [1], to narrow the search space for multi-pass decoding, or for downstream applications that take competing hypotheses as input (e.g. confidence score estimation, acoustic model sequence training).

Sentence alternatives can be generated as an N-best list. That is, a list of the N most likely sentence hypotheses. The straightforward method to generate an N-best list is to separate the search hypotheses by their distinct predecessor word sequences [2, 3, 4].

An efficient way to construct as well as compactly encode multiple sentence alternatives is a word lattice (or word graph). The word lattice is conveniently represented as weighted finite-state transducer (WFST). Each path through this FST is a (weighted) sentence alternative. An N-best list can be generated efficiently as the N best paths in the lattice [5]. The lattice can be generated on various levels of detail, depending on the downstream application. Here, we focus on word lattices: acyclic weighted automata with word labels.

Various lattice generation methods have been developed and described in the literature. Lattice generation with lexical prefix tree decoding is simple and effective but has constraints on the search space structure [6, 7, 8]. The lattice generation method using *N-best histories* described in [9, 10] works for arbitrary search networks and somewhat resembles the N-best list approach. In [11], the lattice is generated on the context-dependent phone level and then converted to a word lattice, we refer to this method as the *phone pair approach*. The method described in [12] is based on an HMM state level lattice which is converted to a word lattice using determinization over a special semiring. A lattice generation method for small grammars was presented in [13]. A comparison of lattice construction using a prefix tree-based decoder and the phone pair approach for a WFST-based decoder is shown in [14]. We look at the N-best histories and the phone pair approaches in more detail in the next section, presented in a uniform formalism.

In a two-pass decoding architecture, the lattice is generated first and then rescored with another language model (LM). This second-pass model cannot be integrated in the search network, because it is too complex, too costly to apply or dynamic and hence not available during search network construction. One example of models often applied via lattice rescoring are long-span neural network LMs [15, 16].

In the two-pass scenario, it is in principle possible to apply classical optimizations such as weighted determinization and minimization to improve the lattice density prior to rescoring [17, 18]. However it has often been observed anecdotally that weighted determization can blow up on speech lattices; we will quantify that in a later section. To deal with this issue, we adapt the determinization algorithm to admit an integrated pruning that retains all paths within specified weight and lattice size thresholds. This can produce more compact lattices for a given oracle lattice WER and thus facilitating any subsequent rescoring steps. We describe this algorithm in Section 3.

The two-pass approach has the advantages that the number of hypotheses to be evaluated by the second-pass model is only a small fraction of the search space of the first pass, is easily applied to complex second-pass LMs, and allows the lattices to be globally optimized prior to rescoring. A drawback is, though, that the supposedly higher quality second-pass model cannot be used for the pruning decisions during decoding. That may result in search errors which cannot be recovered in the rescoring pass if the hypotheses were pruned from the lattice, too. On-the-fly rescoring approaches aim at incorporating a second model as early as possible to avoid search errors without the cost of including it in the search network [19, 20]. In Section 4 we review how the two presented lattice generation methods can be adapted for *on-the-fly* lattice rescoring.

In Section 5 we compare the quality of the two lattice generation methods with and without the proposed lattice optimizations. We further compare how classicial and pruned weighted determinization behave on lattice input. Finally, we compare the two lattice generation methods in the on-the-fly lattice rescoring setting. We conclude with Section 6 with a discussion of these results.

2. LATTICE GENERATION

The decoder computes the most likely word sequence $\mathbf{w} = w_1 \dots w_M$ for the observed speech signal, represented as sequence of acoustic feature vectors $\mathbf{x} = x_1 \dots x_T$:

$$\mathbf{x} \mapsto \hat{\mathbf{w}}(\mathbf{x}) = \operatorname*{argmax}_{\mathbf{w}} \{p(\mathbf{w}) \cdot p(\mathbf{x}|\mathbf{w})\} \tag{1}$$

Here, we use a WFST representation of the search network which encodes the LM, the lexicon and the phone context-dependency [18]. We denote this search network transducer as $R = (\Sigma, \Delta, Q_R, A_R, i_R, F_R)$, which has context-dependent phone models Σ as input labels, words Δ as output labels, a set of states Q_R, a set of arcs $A_R \subseteq Q_R \times \Sigma \times \Delta \times \mathbb{R} \times Q_R$, initial state $i_R \in Q_R$, and a set of final states $F_R \subseteq Q_R$. For an arc $a \in A_R$, we denote its predecessor, input, output, weight and next state as $p[a] \in Q_R$, $i[a] \in \Sigma$, $o[a] \in \Delta$, $\omega[a] \in \mathbb{R}$, $n[a] \in Q_R$ respectively.

The optimization problem in Eq. 1 is solved by applying a dynamic programming algorithm, which generates and evaluates partial hypotheses in a strictly left-to-right time-synchronous fashion. We use the quantity $D(t, q)$ which is the score of the best partial path $\pi \in A_R^*$ in R that starts at the initial state i_R and ends at state $q \in Q_R$ for time frame $t \in \{0, 1, \dots, T\}$:

$$D(t, q) = \min_{\pi : p[\pi]=i_R \wedge n[\pi]=q} \Big\{ \omega[\pi] - \log p(x[1, t]|i[\pi]) \Big\} \tag{2}$$

where $i[\pi]$ is the input label sequence of π, $p[\pi]$ its predecessor state, $n[\pi]$ the next state, and $\omega[\pi]$ is the path weight (for numerical stability, probabilities are stored as negative log probability weights in the search network). This can be rewritten as a recursive optimization over predecessor arcs $a \in A_R$:

$$D(t, q) = \min_{a:n[a]=q} \min_{t' \le t} \gamma(t', t, a) \tag{3}$$

$$\gamma(t', t, a) = D(t', p[a]) + \omega[a] - \log p(x[t', t]|i[a]) \tag{4}$$

The outer minimization is over all arcs a ending in state q. The inner minimization finds the optimal start time t' of a. The LM is incorporated in the form of the arc weight $\omega[a]$, the acoustic model contribution is the likelihood $p(x[t', t]|i[a])$ for the alignment of $i[a]$ between t' and t. For epsilon transitions in R, i.e. $i[a] = \epsilon$, we assume $t' = t$ and $p(\cdot|\epsilon) = 1$.

The innermost loop is typically the optimization within an HMM or the alignment between "blank" and triphone models in a CTC neural-network acoustic model [21].

If paths merge in the search network (due to the limited context size of the models), further extensions of the hypothesis at that state are independent of which of the path hypotheses scored better. In order to retrace the best path at the end, we record which of the joining partial path hypotheses was best. For that, we save the local decisions, i.e., the best a and t' in Equation 3:

$$\alpha_B(t, q) = \operatorname*{argmin}_{a:n[a]=q} \min_{t' \le t} \gamma(t', t, a) \tag{5}$$

$$\tau_B(t, a) = \operatorname*{argmin}_{t' \le t} \gamma(t', t, a) \tag{6}$$

Then these hypotheses can be represented in an FST $(\Sigma, \Delta, Q_B, A_B, (0, i_R), F_B)$ as follows:

$$\begin{aligned} Q_B &= \{(0, i_R)\} \cup \{(t, q) : D(t, q) < \theta(t)\} \\ A_B &= \big\{ \big((t', p[a]), i[a], o[a], \omega, (t, q)\big) : a = \alpha_B(t, q) \big\} \\ &\quad \forall (t, q) \in Q_B,\ t' = \tau_B(t, a),\ \omega = D(t, q) - D(t', p[a]) \\ F_B &= \{(T, q) \in Q_B : q \in F_R\} \end{aligned}$$

If $\theta(t) = \infty$, the search for the best path is unpruned. In most cases it is necessary to limit the number of evaluated hypotheses by applying beam search with $\theta(t) = \min_{q'} D(t, q') + \kappa$. This keeps at each time frame only state hypotheses within a certain range relative to the currently best hypothesis. This lattice construction produces a tree rooted at $(0, i_R)$.

The general idea for lattice generation is to keep track of more than just the best predecessor hypothesis. That is, in addition to the best predecessor $(\alpha_B(t, q), \tau_B(t, q))$, we store all or a subset of the hypotheses associated with the incoming arcs of a state (t, q).

In general, the optimal time frame of a word boundary depends on the whole utterance. Keeping hypotheses separate in both their predecessor and successor results not in a lattice, but in an N-best list. Therefore, in order to construct a compact lattice with distinct word sequence alternatives, certain approximations have to be made. Ideal is a *full lattice*, which contains all hypotheses that were not pruned during the search [11].

We can assume that the start time of a word end hypothesis of a word depends only on the word and its $(n - 1)$ predecessor words. For $n = 2$, this is known as the "word pair approximation" [4, 7]. The history conditioned lexical prefix tree search strategy requires only slight modifications to efficiently generate full lattices [8], because all search hypotheses have a distinct word context and hypotheses merge only at word ends.

2.1. Phone Pair Approach

If the decoder uses a minimized search network, paths can merge within words. Furthermore, the use of the compact WFST representation of n-gram LMs results in a loss of unique predecessor words for all hypotheses. Therefore, the word pair assumptions described in the previous section do not hold.

In [11], the word-level assumptions are replaced by the assumptions that the boundary between two context-dependent phones does not depend on the preceding phone sequence and that each context-dependent phone model in the search network has a unique predecessor phone. The latter condition is only approximately met for tied triphone models and determinized search networks.

The lattice construction works by creating an arc in the lattice for each hypothesis entering a new state in the search network. We save those arcs in:

$$\alpha_P(t, q) = \big\{ a : n[a] = q, \min_{t' \le t} \gamma(t', t, a) < \infty \big\} \tag{7}$$

These hypotheses can then be represented in an FST $(\Sigma, \Delta, Q_B, A_P, (0, i_R), F_B)$ with:

$$\begin{aligned} A_P &= \big\{ \big((t', p[a]), i[a], o[a], \omega, (t, q)\big) : a \in \alpha_P(t, q) \big\} \\ &\quad \forall (t, q) \in Q_B,\ t' = \tau_B(t, a),\ \omega = \gamma(t', t, a) - D(t', p[a]) \end{aligned}$$

The generated lattice is on the level of the search network input, i.e. context-dependent phone models. Subsequent projection of the lattice FST to output labels followed by application of the weighted epsilon removal algorithm [18] converts the lattice into a word lattice.

In fact, the projection is not required if building the lattice as an acceptor with word labels (ignoring the search network's input labels). Many of the epsilon arcs can be removed already during the construction. Epsilon arcs are only required for lattice states with two or more incoming arcs which don't have an output (word) label. In linear parts of the search networks (tree shaped), the lattice arcs can be extended by updating time and weights of the existing arc instead of adding a successor arc. The final epsilon removal is still required but less expensive due to the smaller lattice.

2.2. N-best Histories Approach

The lattice construction method described in [9, 10] is also suited for arbitrary search networks. Each search hypothesis is associated with a (limited size) list of word-level "tokens". Each token corresponds to a distinct word sequence. The method thereby keeps track of the

N-best distinct word sequences arriving at a state. Each token consists of a pointer to the previous lattice state, the current path score and a hash value of the word sequence on the path.

The token lists are propagated while expanding the search hypotheses through the search network. For search network arcs with a word (output) label, new lattice arcs are generated for all tokens in the list. The arcs connect the respective predecessor states from the token with a new state corresponding to the search network state (and current time frame). The token list is resized to one element and the word sequence hash updated with the new word. Thereby, only the best token's word history is used afterwards. When two paths merge and the corresponding hypotheses are recombined, the two associated token lists are merged. Out of the up to $2N$ unique (w.r.t. history) tokens, the N best tokens are kept.

We can relate the N-best histories approach to the previous one by representing its hypotheses in an FST $(\Sigma, \Delta, Q_H, A_H, (0, i_R, \epsilon), F_H)$ with:

$$\begin{aligned}
&Q_H = Q_B \times \Delta^{0:T} \\
&F_H = \{(t, q, h) : (t, q) \in F_B\} \\
&A_H = \big\{\big((t', p[a], h'), i[a], o[a], \omega, (t, q, h)\big) : a \in \alpha_P(t, q)\big\} \\
&\quad \forall (t, q) \in Q_B,\ t' = \tau_B(t, a),\ \omega = \gamma(t', t, a) - D(t', p[a]), \\
&\quad h = \begin{cases} h' & \text{if } o[a] = \epsilon \\ \hat{h}(t, q)\, o[a] & \text{otherwise} \end{cases}
\end{aligned}$$

where $\hat{h}(t, q)$ is the history h of the best predecessor state:

$$\begin{aligned}
P(t, q) &= \big\{(t', p[a], h) : a \in \alpha_P(t, q), t' = \tau_B(t, a)\big\} \\
\hat{h}(t, q) &= \operatorname*{argmin}_{h : (t', q', h) \in P(t, q)} \big\{\gamma(t', t, a) : a \in \alpha_P(t, q)\big\}
\end{aligned}$$

The local N-best is applied by limiting the number of states (t, q, h) to the N-best distinct histories: $\forall (t, q) : |\{(t, q, h) \in Q_H\}| \leq N$. The word histories $h \in \Delta^*$ are approximated by a hash value $H(h) \in \mathbb{Z}_n$. Note that we don't actually create all of these lattice states and arcs explicitly, but only those arcs with non-epsilon output labels. The "word internal" states are implemented as elements of N-best sets per search state (t, q). This results in dropping arcs for recombined histories: out of arcs $((t_j, q_j, h_j), i_j, \epsilon, \omega_j, (t, q, h))$, $j \geq 1$ only the arc to the best predecessor is kept.

The frequent sort and unique operations on the token lists become expensive for larger N. In practice, the token lists are often limited to 5 elements.

In [22], a faster but less accurate variant of this method is mentioned. It avoids sorting and merging token lists, by adding epsilon arcs to the lattice for merging paths. The epsilon arcs are removed afterwards. This approach resembles the method described in the previous section.

3. LATTICE OPTIMIZATION

To create compact lattices, we apply in sequence pruning, weighted determinization, and minimization. The pruning removes low probability states and arcs from the lattice. The beam search during the decoding prunes hypotheses based only on the partial path score up to the current time (the *forward* score). To prune a lattice state or arc, we use the score of the complete best path (forward and backward scores) through that state or arc compared to the overall best path in the lattice [23, 24]. We can also apply lattice pruning periodically during decoding to keep the lattice under construction small.

The determinization removes redundant paths from the lattice. A state in the lattice can have two outgoing arcs with the same word label (the compact ϵ approximation of backoff arcs in an n-gram FST over the tropical semiring is a common source of lattice non-determinism [25]). For most applications, we are only interested in the arc on the better path. Therefore, we apply weighted determinization over the tropical semiring [18]. Determinization has worst-case exponential complexity and in practice a fraction of lattices will be very slow to determinize and grow very large if unconstrained (see Section 5). However, the weighted determinization algorithm admits an efficient integrated pruning, presented below, which effectively controls the computation. Note that although the lattice is pruned prior to determinization, the determinization changes the automaton's topology and can result in new states and arcs that are candidates for pruning based on their complete best path scores through those states and arcs.

Once an acyclic, determinstic lattice is attained, weighted minimization can be applied to produce the equivalent minimal (in states and arcs) deterministic automaton in linear time [18]. Pruning, weighted determinization (with pruning) and weighted minimization are all available in the open-source OpenFst Library [26].

3.1. Pruned Weighted Determinization

Suppose we have a lattice represented as an acyclic non-deterministic finite state automaton $L = (\Sigma, Q_L, A_L, i_L, F_L)$ from which we wish to produce a (possibly pruned) deterministic automaton $D = (\Sigma, Q_D, A_D, i_D, F_D)$. Our algorithm consists of three steps: (1) finding the shortest distance from each state in Q_L to the final states, (2) applying on-the-fly weighted determinization of the automaton L while (3) searching for the states and arcs that are to kept versus pruned in the result. This algorithm has many similarities in its initial steps to the n-best (unique) strings algorithm on a weighted automata [5].

3.1.1. *Shortest-distances to final states*

The first step consists of computing the shortest distance from each state $q \in Q_L$ to the final states F_L (the *backward score*):

$$\beta_L[q] = \min_{\pi : p[\pi] = q \wedge n[\pi] \in F_L} \omega[\pi]$$

The distances $\beta_L[q]$ can be directly computed in linear time by running a shortest-paths algorithm from the final states F using the reversed automaton [27].

3.1.2. *Weighted Determinization*

Weighted determinization is a generalization of the classical subset construction [28]. We give a brief description of it here; see [17] for a fuller account. The states of D correspond to *weighted subsets* $\{(q_0, \omega_0), \ldots, (q_n, \omega_n)\}$ with $q_i \in Q_L$ and ω_i a remainder weight. The algorithm starts with the subset $i_D = \{(i_L, 0)\}$. From each weighted subset S, it will create an arc labeled with $l \in \Sigma$ and weight ω leaving $\{(q_0, \omega_0), \ldots, (q_n, \omega_n)\}$ when at least one state q_i has an outgoing transition labeled with l; ω is given by:

$$\omega = \min\{\omega_i + \omega[a] : (q_i, \omega_i) \in S, a \in A_L, p[a] = q_i, i[a] = l\}$$

The destination state of that arc corresponds to the subset containing the pairs (q', ω') with $q' \in \{n[a] : p[a] = q_i, i[a] = l\}$ and the remainder weight $\omega' = \min\{\omega_i + \omega[a] - \omega : n[a] = q'\}$.

For simplicity of presentation, the result D will have only a single final state f_D to which final arcs labeled with a special final word (e.g. `</S>`) will be directed. Alternately, we could introduce final weights in our definition of a weighted automaton [17]. A state in Q_D has a final arc if it corresponds to a weighted subset S containing a pair (q, ω) where q is a final state ($q \in F_L$) and has final arc weight $\min\{\omega : (q, \omega) \in S, q \in F_L\}$

The pruned version we introduce in this paper utilizes a feature of the determinization algorithm: it admits a natural on-the-fly implementation. The computation of the transitions leaving a subset S only depends on the states and remainder weights of that subset and on the input automaton, it is independent of the previous subsets visited or constructed. That is we can limit the computation of the result of determinization to just the part that is needed.

Our pruning step also needs the shortest-distance information, $\beta_L[q]$, in L propagated to the determinized result. For this, we assign to each state q' of the result of determinization the quantity:

$$\beta_D[q'] = \min_i \; \omega_i + \beta_L[q_i]$$

where q' corresponds to the subset $\{(q_0, \omega_0), \ldots, (q_n, \omega_n)\}$ [5].

3.1.3. *Pruning Search*

The pruning algorithm uses A^* search to visit that portion of D that is within threshold [29]. We use both a weight threshold θ and a number of states threshold η. The following is the pseudocode for the algorithm.

```
SEARCH($D, \theta, \eta$)
 1  for $q \in Q_D$ do
 2      $\alpha_D[q] \leftarrow \infty$
 3  $S \leftarrow (i_D, \beta_D[i_D])$
 4  $n \leftarrow 1$
 5  while $S \neq \emptyset$ do
 6      $(q, c) \leftarrow$ HEAD($S$)
 7      DEQUEUE($S$)
 8      for $a \in A_D \wedge p[a] = q$ do
 9          $c' \leftarrow \alpha_D[p[a]] + \omega[a] + \beta_D[n[a]]$
10          if $c' > \theta + \beta_D[i_D]$ then
11              DELETE($a$)
12          elseif $\alpha_D[n[a]] > \alpha_D[p[a]] + \omega[a]$ then
13              $\alpha_D[n[a]] \leftarrow \alpha_D[p[a]] + \omega[a]$
14              if $n[a] \in S$ then
15                  UPDATE($S$, $(n[q], c')$)
16              elseif $n \leq \eta$ then
17                  ENQUEUE($S$, $(n[q], c')$)
18                  $n \leftarrow n + 1$
```

The algorithm uses a priority queue S on pairs (q, c) of a state $q \in Q_D$ and the complete best path weight c (so far) through q; the latter is used to order the queue (lines 15,17). The shortest distance from the inital state (*forward score*) is computed and stored in α_D (line 12-13) and combined with the provided β_D to compute the complete best path weights through states and arcs (line 9). No state s is enqueued whose complete best path weight exceeds the weight threshold (line 12) or is above the state count (line 16). No arc from a state within threshold is kept if its complete best path weight exceeds threshold (line 10). Since only states and arcs within threshold are followed and D can be computed on-the-fly, the determinization is limited to that portion. The result consists of the states and arcs visited in the search (less any arcs that have been DELETE'd). The algorithm terminates when all states within threshold have been processed at most once (line 5) since no state is ever reinserted into the queue [29].

4. ON-THE-FLY LATTICE RESCORING

The goal of on-the-fly rescoring is to apply an LM that is not incorporated in the search network during decoding. Thereby, search errors can be avoided which would otherwise result in missing hypotheses in the lattice. These missing hypotheses cannot not be recovered in a downstream lattice rescoring pass.

In this section, we focus on on-the-fly *lattice* rescoring, which applies the rescoring model during lattice generation [19]. That is, we apply the rescoring model to the lattice as it is constructed during decoding. The updated scores of the (partially) rescored lattice can then be used for pruning during decoding (in [19], the authors don't state whether the rescored hypotheses are also used for the search process). On-the-fly lattice rescoring has been applied successfully for salient n-gram biasing [30], incorporating contextual models [31] and for resource constrained on-device speech recognition [32].

This is in contrast to incorporating a larger LM by building the search network with a small LM and then combine it on demand with a modified higher order LM during decoding [33, 34]. A similar approach is the so-called on-the-fly hypothesis rescoring described in [20]. In the WFST framework, the composition of the lexicon FST and the LM FST can be performed on-the-fly during decoding [35, 36, 37, 38].

On-the-fly lattice rescoring is in principle the composition of the lattice with the rescoring model FST. Each arc added to the lattice is immediately rescored. The updated score is then used to update the corresponding search hypothesis. The states in the rescored lattice are tuples of a state in the regular lattice and a state in the rescoring model.

We consider two different types of rescoring model: Full language models that can be applied to any word sequence generated by the decoder. A regular n-gram LM would be a full LM (the LM used to build the search network is often a pruned version of the rescoring model). The other type of rescoring models are partial language models which cover only some n-grams for example n-gram biasing models [30].

In the descriptions below, we assume the rescoring model to be an acceptor $G = (\Delta, Q_G, A_G, i_G, F_G)$ which accepts any $\mathbf{w} \in \Delta^*$. Partial models would need to be augmented with failure transitions [30].

4.1. N-best Histories Approach

The N-best histories approach can be extended for on-the-fly rescoring with full n-gram models by replacing the word sequence (hash) h with a LM history state $q_G \in Q_G$. Thereby, the lattice states are not conditioned on the full word sequence anymore but on at most $(n-1)$ words. The corresponding FST $(\Sigma, \Delta, Q_M, A_M, (0, i_R, i_G), F_M)$ can be defined as follows:

$$\begin{aligned}
Q_M &= Q_B \times Q_G \\
F_M &= \{(t, q, h) : (t, q) \in F_B,\; h \in Q_G\} \\
A_M &= \big\{ \big((t', p[a], h'), i[a], o[a], \omega, (t, q, h)\big) : a \in \alpha_P(t, q), \big\} \\
&\quad \forall (t, q) \in Q_B,\; t' = \tau_B(t, a),\; (h', o[a], \omega_G, h) \in A_G \\
&\quad \omega = f\left(\gamma(t', t, a) - D(t', p[a]), \omega_G\right)
\end{aligned}$$

For epsilon arcs, we assume $(h, \epsilon, \bar{1}, h) \in A_G \, \forall h \in Q_G$. The number of unique histories h per state (t, q) is limited to some fixed N as above. Note that instead of reducing the history list to the best element, here we keep N unique histories. The scores of the two models are combined using a combination function $f : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$, for example log-linear combination. We keep the LM and AM score contributions of the hypothesis cost separate and combine only the LM part, while keeping the AM part unmodified. For simplicity, in the equations above, we ignored the fact that the LM scores might be distributed along the path.

If the rescoring model is a partial model or a subset of the search network LM, the history state would be equal for most of the tokens. That would result in a very small lattice; e.g. just the one-best sequence if $Q_G = \{i_G\}$ (or if none of the n-grams in a biasing grammar was hypothesized). To avoid this, we use both the word sequence hash and the history state to distinguish tokens.

4.2. Phone Pair Approach

The on-the-fly rescoring extension of the phone pair approach is very similar to the N-best history approach described in the previous section. Each search hypothesis (t, q) is associated with a set of lattices states (t, q, h), $h \in Q_G$. For efficiency, the size of the sets is limited to the N-best elements. We can use the same FST as in the previous section to describe the construction. The difference is subtle. As described in Section 2.2, the N-best histories approach keeps only the best arc for recombined histories and thereby drops predecessor states. In contrast, the rescored phone pair approach actually adds the intermediate states and epsilon arcs $((t_j, q_j, h_j), i_j, \epsilon, \omega_j, (t, q, h))$ to all predecessor states (t_j, q_j, h_j). The rescoring can only split states and hence the un-rescored lattice is the lower bound.

5. EXPERIMENTAL RESULTS

We study three speech recognition systems here: voice search/dictation, video transcription, and on-device dictation. The first system is a state-of-the-art server-based recognizer for voice search and dictation. The recognizer has a vocabulary of about 4 M words and uses a 5-gram LM with 95 M n-grams. The video transcription system has a vocabulary of 950 K words and uses a 5-gram LM with 50 M n-grams. For both large vocabulary systems, we look at the lattices generated by the decoder without any rescoring. We use a small vocabulary system with 64 K words, which has been developed for on-device recognition [32], for the analysis of on-the-fly rescoring. The search network is built with an LM of just 70 K n-grams. The rescoring LM is a 5-gram containing 1.2 M n-grams. All systems use Long Short-Term Memory (LSTM) recurrent neural networks trained with connectionist temporal classification (CTC) [39] as acoustic models.

For the voice search/dictation system, we use a test set of 20 K utterances with a total of 170 K words. One third of the data are voice search queries, the rest are dictation utterances. The on-device dictation system is evaluated on 13 K utterances. Both test sets consist of anonymized utterances that were randomly sampled from Google traffic. To analyze the video transcription system, we use a set of 296 manually transcribed YouTube videos (duration: ∼25 h) and another set of 2.8 M untranscribed YouTube videos split into 14.8 M segments (duration: ∼209 K h).

The metrics we use here to assess the generated lattices are lattice density and oracle word error rate (WER). Lattice density is computed as the number of lattice arcs over the number of words in the (reference) transcript. The oracle WER is the lowest WER of all paths in the lattice. In our voice search/dictation system, text normalization rules are applied to the recognized word sequence before computing the edit distance to the reference transcript. These rules cannot be applied to the lattice for technical reasons. Therefore, we approximate the oracle WER by computing the lowest WER of the (up to) 1,000 best unique paths in the lattice. The total number of traversed paths is limited to 2,000. We also measure lattice *redundancy* as number of lattice arcs over number of arcs in the minimized lattice. The evaluation of the video transcription system does not require text normalization; the oracle WER can be computed on the full lattice.

Figure 1 shows the results for the voice search/dictation task. For the N-best histories lattice, the density saturates quickly with increasing lattice pruning threshold, especially for small N. With larger N, both lattice density and oracle WER increase, but the oracle WER saturates at around 3.2%. Even with a large N = 100 (impractical because of increased runtime for sort unique operations), the oracle WER does not decrease, albeit the lattice density

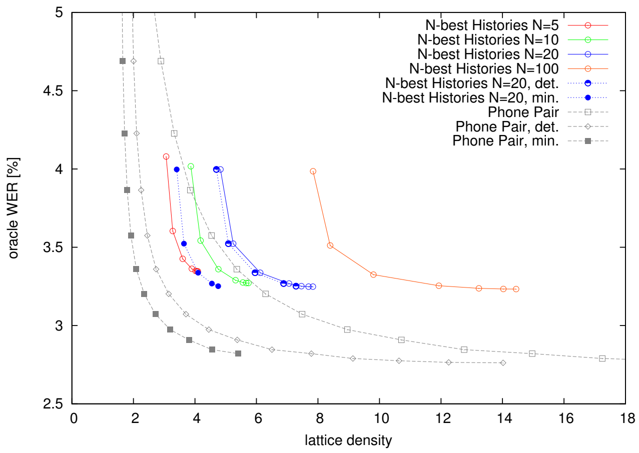

Fig. 1. Lattice quality of the voice search/dictation system using the N-best histories approach and the phone pair approach, with and without optimization. Results are shown for various lattice pruning thresholds. The 1-best WER is 7.1%.

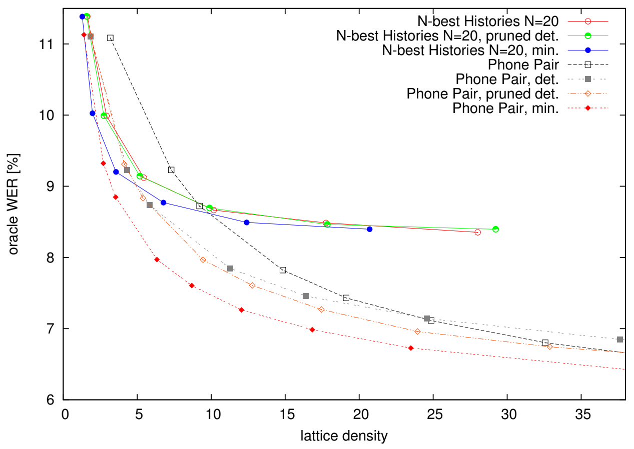

Fig. 2. Lattice quality of the video transcription system, with and without lattice optimization. The 1-best WER is 12.9%.

increases. The larger N-best sets cannot compensate for pruning predecessors during recombination (see Section 2.2). The quality of the lattices generated with the phone pair approach can be increased further – the oracle WER goes down to 2.8%.

Determinization decreases the size of the N-best histories lattices only slightly, but the lattices are more compact after minimization. The average lattice redundancy is 1.4. The lattices generated with the phone pair approach are not deterministic. Their average redundancy is between 1.9 and 4.1 depending on pruning threshold. Hence, both determinization and minimization result in significantly more compact lattices here.

The recognition quality of the voice search/dictation system is quite high and hence even sparse lattices already have a good oracle WER. Therefore, we looked at a more challenging task – video transcription. Here, we want to focus on the effect of optimization using pruned determinization. The quality of lattices is shown in Figure 2. The results obtained with pruned determinization versus regular weighted determinization are essentially indistinguishable at low densities and slightly more compact for wider pruning thresholds on our evaluation sets. We use a weight threshold θ in pruned determinization that matches the threshold used during lattice generation.

Table 1. Change in lattice size by determinization. The size increase δ is measured as number of arcs in the determinized lattice over number of arcs in the original lattice. The table shows the number of lattices below or over a given δ. Due to the small percentages, the absolute number of lattices is shown for $\delta > 10$.

	det.	pruned det., $\theta = 12$	
		$\eta = \infty$	$\eta = 2 \cdot \lvert Q_L \rvert$
$\delta \leq 1$	90.73%	99.70%	99.70%
$\delta \leq 2$	99.85%	100.00%	100.00%
$\delta > 10$	376	13	0
$\delta > 100$	88	0	
$\delta > 1000$	34		

Table 2. Distribution of computation time for det. per input arc.

	CPU time [μs / arc]	
percentile	det.	pruned det.
50	0.9	2.3
90	11.0	8.8
99	25.1	18.4

The main benefit of pruned determinization is seen when we examine larger data sets. To observe the rare but fatal phenomenon of enormous size increase by determinization, we decoded the 14 M segments of the unsupervised data set and compared the lattice size before and after determinization. The results are shown in Table 1. Regular weighted determinization works well in almost all cases. However, in the instances where it doesn't, the size increase is significant. In a production system, these outliers are not acceptable. On the other hand, the pruned determinization based only on a weight threshold θ has only 13 out of 15 M lattices where the lattice size increases significantly. With an additional threshold on the number of states η, those cases can be entirely avoided.

The input lattices blowing up with determinization are not necessarily abnormally large. For example, in this test set we saw a lattice with 400 states, 900 arcs that grew to 2.3 M states, 7.2 M arcs after determinization, which took 16 s. In contrast, pruned determinization processed the lattice in 24 ms and generated 1.2 K states, 1.3 K arcs.

Pruned determinization has an additional computational cost compared to regular determinization. For many cases, the additional computation results in higher CPU time. However, for "problematic" lattices, the pruned determinization reduces CPU time. In our experiments, computation time in the long tail was significantly reduced, as shown in Table 2 (note that running time of weighted determinization is not linear in the input size). Pruned determinization did not add significantly to the overall system latency in the experiments. The small increase in average CPU time achieves lower variability and avoids latency spikes, which is beneficial for production systems.

The results with on-the-fly rescoring are shown in Figure 3. For comparison, without rescoring, the 1-best WER is 15.6%, the lowest oracle WER is 5.5% and 5.1% for the N-best histories and phone pair approaches respectively. Increasing the number of histories from 5 to 10 increases the quality of the lattice generated by the N-best histories approach significantly. Increasing N further to 20 results in smaller gains. The lattices generated with the phone pair method have lower oracle WER for comparable density. The number of history states used has an impact on the achievable lattice quality, too. In this experiment, using more than 20 history states per search hypothesis does not yield a significantly lower oracle WER. The difference in 1-best WER between on-the-fly rescored N-best histories and phone pair lattices is less than 1% relative for this experiment. In contrast to the results above, the on-the-fly rescored N-best histories lattice are significantly more compact after determinization. The redundancy of the lattices is between 5.0 and 8.1.

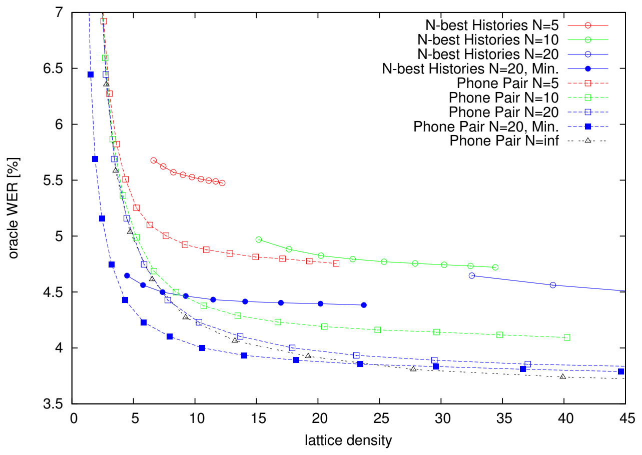

Fig. 3. Lattice quality of the on-device dictation system with on-the-fly rescoring. The 1-best WER is 10.6% for N=5, 10.4% for N=20.

6. DISCUSSION

The experimental results in the previous section show that prior to optimization the phone pair approach generates lattices with a quality sometimes better and sometimes worse than the N-best histories approach depending on the task and parameter settings. However, once (pruned) weighted determinization and minimization are applied, the former is superior.

The phone pair approach consists of essentially just writing down the states and arcs that are visited during search. The unique N-best histories approach, however, adds disambiguation and N-best pruning features. Once determinization and minimization are applied, any benefits of those features are likely lost to the global optimizations and likely hurt compared to the phone pair approach by dropped paths (at smaller N) and increased latency (at larger N). We could not measure a significant increase in latency due to pruned determinization and minimization.

In the on-the-fly lattice rescoring case, a secondary LM is applied prior to any global lattice optimizations. However, any earlier optimization would likely not be particularly effective for this stage since the bulk of the computation is in the simultaneous first-pass decode. On the other hand, optimizations on lattice output of on-the-fly rescoring (e.g. aimed at further rescoring or application-specific use) benefit from weighted determinization and minimization as our experiments show.

In conclusion, we believe the simple-to-implement phone-pair lattice generation approach together with pruned weighted determinization and minimization, available in the open-source OpenFst Library, offer very efficient and high-quality lattice generation. Its extension to on-the-fly rescoring offers added flexibility and potentially reduced search errors.

7. ACKNOWLEDGEMENTS

We thank Mehryar Mohri for contributions to the pruned weighted determinization algorithm and Shankar Kumar for helping in the generation of the lattices used in our experiments.

8. REFERENCES

[1] Y.-L. Chow and R. Schwartz, "The N-best algorithm: An efficient procedure for finding top N sentence hypotheses," in *Proc. DARPA Speech and Natural Language Workshop*, Cape Cod, MA, USA, Oct. 1989, pp. 199–202.

[2] V. Steinbiss, "Sentence-hypotheses generation in a continuous-speech recognition system," in *Eurospeech*, Paris, France, Sep. 1989, pp. 2051–2054.

[3] R. Schwartz and Y.-L. Chow, "The N-best algorithms: An efficient and exact procedure for finding the N most likely sentence hypotheses," in *ICASSP*, Albuquerque, NM, USA, Apr. 1990, pp. 81–84.

[4] R. Schwartz and S. Austin, "A comparison of several approximate algorithms for finding multiple (N-best) sentence hypotheses," in *ICASSP*, Toronto, ON, Canada, Apr. 1991, pp. 701–704.

[5] M. Mohri and M. Riley, "An efficient algorithm for the n-best-strings problem," in *ICSLP*, Denver, CO, USA, Sep. 2002, pp. 1313–1316.

[6] H. Ney and X. Aubert, "A word graph algorithm for large vocabulary continuous speech recognition," in *ICSLP*, Yokohama, Japan, Sep. 1994, pp. 1355–1358.

[7] X. Aubert and H. Ney, "Large vocabulary continuous speech recognition using word graphs," in *ICASSP*, Detroit, MI, USA, May 1995, pp. 49–52.

[8] S. Ortmanns, H. Ney, and X. Aubert, "A word graph algorithm for large vocabulary continuous speech recognition," *Computer Speech and Language*, vol. 11, no. 1, pp. 43–72, Jan. 1997.

[9] G. Saon, D. Povey, and G. Zweig, "Anatomy of an extremely fast LVCSR decoder," in *Eurospeech*, Lisbon, Portugal, Sep. 2005, pp. 549–552.

[10] S. Chen, B. Kingsbury, L. Mangu, D. Povey, G. Saon, H. Soltau, and G. Zweig, "Advances in speech transcription at IBM under the DARPA EARS program," *IEEE Trans. on Audio, Speech and Language Processing*, vol. 14, no. 5, pp. 1596–1608, Sep. 2006.

[11] A. Ljolje, F. Pereira, and M. Riley, "Efficient general lattice generation and rescoring," in *Eurospeech*, Budapest, Hungary, Sep. 1999, pp. 1251–1254.

[12] D. Povey, M. Hannemann, G. Boulianne, L. Burget, A. Ghoshal, M. Janda, M. Karafiát, S. Kombrink, P. Motlíček, Y. Qian, K. Riedhammer, K. Veselý, and N. T. Vu, "Generating exact lattices in the WFST framework," in *ICASSP*, Kyoto, Japan, Mar. 2012, pp. 4213–4216.

[13] M. Novak, "Memory efficient approximative lattice generation for grammar based decoding," in *Interspeech*, Lisbon, Portugal, Sep. 2005, pp. 573–576.

[14] D. Rybach, H. Ney, and R. Schlüter, "Lexical prefix tree and WFST: A comparison of two dynamic search concepts for LVCSR," *IEEE Trans. on Audio, Speech and Language Processing*, vol. 21, no. 6, pp. 1295–1307, Jun. 2013.

[15] X. Liu, Y. Wang, X. Chen, M. Gales, and P. Woodland, "Efficient lattice rescoring using recurrent neural network language models," in *ICASSP*, Florence, Italy, May 2014, pp. 4908–4912.

[16] M. Sundermeyer, Z. Tüske, R. Schlüter, and H. Ney, "Lattice decoding and rescoring with long-span neural network language models," in *Interspeech*, Singapore, Sep. 2014, pp. 661–665.

[17] M. Mohri, "Weighted automata algorithms," in *Handbook of Weighted Automata*, M. Droste, W. Kuich, and H. Vogler, Eds. Springer, Berlin, 2009, ch. 6.

[18] M. Mohri, F. Pereira, and M. Riley, "Speech recognition with weighted finite-state transducers," in *Handbook of Speech Processing*, J. Benesty, M. Sondhi, and Y. Huang, Eds. Springer, 2008, ch. 28, pp. 559–582.

[19] H. Sak, M. Saraçlar, and T. Güngör, "On-the-fly lattice rescoring for real-time automatic speech recognition," in *Interspeech*, Makuhari, Japan, Sep. 2010, pp. 2450–2453.

[20] T. Hori, C. Hori, Y. Minami, and A. Nakamura, "Efficient WFST-based one-pass decoding with on-the-fly hypothesis rescoring in extremely large vocabulary continuous speech recognition," *IEEE Trans. on Audio, Speech and Language Processing*, vol. 15, no. 4, pp. 1352–1365, 2007.

[21] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks," in *ICML*, Pittsburgh, PA, USA, Jun. 2006, pp. 369–376.

[22] H. Soltau and G. Saon, "Dynamic network decoding revisited," in *ASRU*, Merano, Italy, Dec. 2009, pp. 276–281.

[23] J. J. Odell, "The use of context in large vocabulary speech recognition," Ph.D. dissertation, University of Cambridge, Cambridge, United Kingdom, Mar. 1995.

[24] A. Sixtus and S. Ortmanns, "High quality word graphs using forward-backward pruning," in *ICASSP*, Phoenix, AZ, USA, Mar. 1999, pp. 593–596.

[25] G. Riccardi, E. Bocchieri, and R. Pieraccini, "Non-deterministic stochastic language models for speech recognition," in *ICASSP*, Detroit, MI, USA, May 1995, pp. 237–240.

[26] C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, and M. Mohri, "OpenFst: a general and efficient weighted finite-state transducer library," in *Proc. Int. Conf. on Implementation and Application of Automata*, Prague, Czech Republic, Jul. 2007, pp. 11–23.

[27] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*. Cambridge, MA: The MIT Press, 1992.

[28] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers, Principles, Techniques and Tools*. Reading, MA: Addison Wesley, 1986.

[29] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[30] K. Hall, E. Cho, C. Allauzen, F. Beaufays, N. Coccaro, K. Nakajima, M. Riley, B. Roark, D. Rybach, and L. Zhang, "Composition-based on-the-fly rescoring for salient n-gram biasing," in *Interspeech*, Dresden, Germany, Sep. 2015, pp. 1418–1422.

[31] P. Aleksic, M. Ghodsi, A. Michaely, C. Allauzen, K. Hall, B. Roark, D. Rybach, and P. Moreno, "Bringing contextual information to Google speech recognition," in *Interspeech*, Dresden, Germany, Sep. 2015, pp. 468–472.

[32] I. McGraw, R. Prabhavalkar, R. Alvarez, M. G. Arenas, K. Rao, D. Rybach, O. Alsharif, H. Sak, A. Gruenstein, F. Beaufays, and C. Parada, "Personalized speech recognition on mobile devices," in *ICASSP*, Shanghai, China, Apr. 2016, pp. 5955–5959.

[33] H. Dolfing and I. L. Hetherington, "Incremental language models for speech recognition using finite-state transducers," in *ASRU*, Madonna di Campiglio, Italy, Dec. 2001, pp. 194–197.

[34] D. Willett and S. Katagiri, "Recent advances in efficient decoding combining on-line transducer composition and smoothed language model incorporation," in *ICASSP*, Orlando, FL, USA, May 2002, pp. 713–716.

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

[36] C. Allauzen, M. Riley, and J. Schalkwyk, "A generalized composition algorithm for weighted finite-state transducers," in *Interspeech*, Brighton, U.K., Sep. 2009, pp. 1203–1206.

[37] T. Oonishi, P. R. Dixon, K. Iwano, and S. Furui, "Implementation and evaluation of fast on-the-fly WFST composition algorithms," in *Interspeech*, Brisbane, Australia, Sep. 2008, pp. 2110–2113.

[38] P. R. Dixon, C. Hori, and H. Kashioka, "A comparison of dynamic WFST decoding approaches," in *ICASSP*, Kyoto, Japan, Mar. 2012, pp. 4209–4212.

[39] H. Sak, A. Senior, K. Rao, and F. Beaufays, "Fast and accurate recurrent neural network acoustic models for speech recognition," in *Interspeech*, Dresden, Germany, Sep. 2015, pp. 1468–1472.