ISCA Archive
http://www.isca-speech.org/archive

6th European Conference on
Speech Communication and Technology
(EUROSPEECH'99)
Budapest, Hungary, September 5-9, 1999

Efficient General Lattice Generation and Rescoring

Andrej Ljolje, Fernando Pereira and Michael Riley
{alj,pereira,riley}@research.att.com
AT&T Labs – Research
180 Park Avenue, Florham Park, NJ 07932-0971, USA

ABSTRACT

We describe a lattice generation method that produces high-quality lattices with less than 10% increased computation over standard Viterbi decoding. Using the North American Business News (NAB) task, we show our method is within 0.2% in lattice word-error rate of 'full lattices', which are those that contain all the recognition hypotheses within the search beam. Our method is closely related to previous lattice generation methods, but applies to more general network topologies.

We also give real-time results on the NAB task, in which we generate lattices in a first pass and then rescore them with stronger acoustic and language models in a second pass. We are able to achieve at 3x real-time a word error rate of 11.2% on the Eval '95 test set, which is only 1.7% worse than AT&T's official benchmark result that year using what was then a 1000x real-time system.

1. Introduction

A speech-recognition *lattice* is a labeled, weighted, directed acyclic graph in which each complete path represents an alternative transcription hypothesis, weighted by its recognition score for a given utterance. Lattices are useful when more than one alternative is desired from a recognition pass. For example, it is often desirable to divide speech decoding into two passes, a first pass that creates a lattice using relatively simple acoustic and language models, and a second 'rescoring' pass that uses more elaborate models, which would be too expensive to use in the first pass, to select a best hypothesis from the lattice.

We describe a lattice generation method that was added to a standard Viterbi decoder to produce high-quality lattices with little added computational overhead. We will present the conditions under which our lattices are equivalent to *full lattices*, which consist of all the recognition hypotheses within the Viterbi search beam. Full lattices are desirable in the sense that only 'implausible' paths outside the beam have been discarded.

Our method is closely related to previously reported efficient lattice generation methods [9, 10, 15], but applies under less restrictive conditions on the ASR network topology. The earlier methods assume that 1) the time boundary between two words found by the recognizer does not depend on the word history before those words, and require that 2) all the predecessor words of a given word occurrence in the ASR network should be identical (a full bigram model satisfies this condition), and that 3) phones from different words should not be merged into the same decoding state (a 'tree lexicon' satisfies this condition). Lattices generated under these conditions should be full lattices.

Unfortunately, conditions (2) and (3) are rather restrictive, and incompatible with our approach to building very compact and easy-to-search phone-level networks for very large tasks [5]. In particular, compact approximate finite-state representations of n-gram language models [12] violate (2), while (3) is violated by the determinization and minimization algorithms we use to optimize our statically constructed phone-level networks [4]. As such, we sought a lattice construction method that worked well on these more general network topologies.

2. Methods

2.1. One-Best Decoding

In our work, we represent ASR networks as *finite-state transducers* [11, 14, 4, 5, 8]. These are labeled, weighted, directed graphs in which each *arc* a has a source state $S(a)$, destination state $D(a)$, an input label $I(a)$, output label $O(a)$, and a weight $P(a)$. For a typical transducer T described here, each arc's input label is a context-dependent phone symbol (corresponding to a distinct HMM) or ϵ (the empty or null phone), each arc's output label is a word label or ϵ (the empty or null word), and each arc's weight is the language model probability[1] for that arc. A complete path through such a transducer from the distinguished start state to a distinguished final state, gives a legal sequence of words in the language model as specified by the path's output labels, a corresponding pronunciation in terms of context-dependent phones as specified by the input labels, and a language model probability as specified by the product of the arc weights. The construction of these transducers from a context-dependency specification, a pronunciation lexicon, and a language model, and their optimization for recognition time and space performance are described in detail elsewhere [14, 4, 5].

[1] This weight can include pronunciation probability alternatives as well when there are multiple pronunciations for a given word (in context) and a stochastic model for them.

The recognition task can be characterized as finding that path of arcs $\mathbf{a} = a_1 ..., a_n$ from the initial state to a final state in T that maximizes the joint probability of the acoustic and language models when applied to a given utterance [11]. In other words, we seek the path $\mathbf{a}$ that satisfies:

$$\max_{\mathbf{a}} P(\vec{x}[0,\tau], \mathbf{a}) = \qquad (1)$$
$$\max_{\mathbf{a}, t_1, \cdots, t_{n-1}} \prod_{i=1}^{n} P(\vec{x}[t_{i-1}, t_i] | I(a_i)) \, P(a_i).$$

The first factor represents the acoustic model likelihood for the context-dependent phone $I(a_i)$ when applied to the speech feature vectors $\vec{x}$ beginining at time t_{i-1} and ending at time t_i. The second factor, as indicated above, represents the language model probability for that arc in T. This formulation contains the usual approximations of conditional independence of the context-dependent phone models and the Viterbi condition by using the maximum rather than the sum over the possible segmentations $t_0 (= 0), t_1, \ldots, t_{n-1}, t_n (= \tau)$. The recognized word sequence is given by $O(a_1) \cdots O(a_n)$.

We now express (1) in terms of the best scoring path to each state of T for a given time, since this is central to the Viterbi algorithm. Let $B(s)$ be the set of all paths $a_1 \cdots a_k$ in T from the initial state to state s and let:

$$\alpha(s,t) = \max_{\mathbf{a} \in B(s), t_1, \cdots, t_{k-1}} \prod_{i=1}^{k} P(\vec{x}[t_{i-1}, t_i] | a_i) P(a_i). \qquad (2)$$

Then:

$$\begin{aligned} & \max_{\mathbf{a}} P(\vec{x}[0,t], \mathbf{a}) \\ & = \quad \max_{s} \alpha(s,t) && (3) \\ & = \quad \max_{D(a)=s} p(a) \left[\max_{t' < t} P(\vec{x}[t', t] | I(a)) \alpha(S(a), t') \right] && (4) \end{aligned}$$

In (4) we factor the familiar Viterbi recursion [3] into two nested $(\max)$ loops: the outer loop considers each possible active arc a ending in s, with LM probability $P(a)$, and the inner loop picks out the optimal start time t' for a by combining the acoustic likelihood of a between t' and t with the best path likelihood from the start to state $S(a)$ at time t'. Typically, the inner maximization is itself evaluated using dynamic programming internal to the HMM associated with arc a, building on previously tabulated values of $\alpha(s', t')$ for appropriate states s' and times t'. We omit the description of this step here, since it is not needed for our lattice generation presentation.

Since the full set of paths expressed in (4) could be very large and computationally burdensome, states s for which $\alpha(s,t) < \kappa * \max_{s'} \alpha(s', t)$ are commonly pruned out at each t, where κ, the *beam* factor, is empirically chosen to be small enough that the arcs that are left "active" are likely to contain the best scoring path that would be chosen if no pruning were done.

2.2. Lattice Generation

Our lattice generation algorithm consists of three main steps. The first step creates a context-dependent phone-to-word transducer lattice, the second step converts this to a word lattice, and a third step prunes the lattice relative to the best scoring path through the *entire* lattice.

The first step involves no extra computation over the normal Viterbi algorithm other than the negligible time needed to add states and arcs into the transducer lattice as it is being constructed. Each state of a phone-to-word transducer lattice L corresponds to a pair (t, s) of a time frame in the recognition and a state from the recognition transducer T. The initial state is the pair of utterance start time 0 and the start state of T. The final states are the pairs consisting the utterance end time τ and a final state from T. If, during the Viterbi recursion of (4), we have identified the optimal start time t' for arc a ending in state $D(a)$ at time t, then a corresponding arc is added to L from state $(t', S(a))$ to state $(t, D(a))$. If necessary, state $(t, D(a))$ was first created ; state $(t', S(a))$ must already have been in L by induction. The new arc has input label $I(a)$, output label $O(a)$, and weight $P(\vec{x}[t_{i-1}, t_i]|a) P(a)$. All this information is readily available from the Viterbi recursion. It is not difficult to see that this step is a phone-level analog of the prior word-level methods [9, 10, 15].

An implementation detail is that since we represent states in a transducer M as integers from 0 to $|M| - 1$, we need a way in the above steps to map from the (t, s) pairs to their corresponding integer state indices in L. While a hash table is a possible solution, a more efficient one is as follows. We store the index for the pair $(t, D(a))$ in the decoder active state data structure for $D(a)$ at the current time, and we store the index for the pair $(t', S(a))$ in the decoder active arc data structure for a.

To convert these phone-to-word lattices to word lattices, we rely on our efficient implementations of the general finite-state operations found in our *FSM library* [7, 6] [2]. The first step is the trivial *projection* on to the word labels by, in effect, deleting in the input labels from the lattice arcs and leaving only the output labels. Since in the transducer representation of T only one arc per word can contain the word label on the output side, paired with one of its phones on the input side, all the other arcs of that word have a phone on the input side and an ϵ on the output side [11]. Therefore, projection creates a lattice with many ϵ arcs, in addition to the usual word arcs.

To reduce the size of the lattice, we would like to remove the ϵ arcs while preserving the overall cost of any word path. If arcs were not weighted, we could use the classical ϵ-removal algorithm for finite automata [2]. As described elsewhere [7, 6], we

[2] The FSM library can be downloaded for non-commercial use from the cited URL.

have extended this algorithm to the weighted case. In the special case where condition (3) above holds, ϵ-arc removal is trivial because of the 'tree' structure of the network, and in fact can be done on-the-fly during decoding. More general topologies of T can require our more general weighted ϵ-removal method, for which no efficient on-the-fly version is yet available. However, a typical T has many tree-like subnetworks (states with indegree of one), to which we apply the trivial on-the-fly ϵ-removal method. In either case, ϵ-removal requires a negligible part of the overall computation time.

The final step is to prune the lattice relative to the best scoring path through the lattice. While the normal recognition beam prunes the search space, that pruning is based only on the best scoring path up to the current time t. Subsequent beam pruning of the lattice relative to complete path scores through the lattice proves to be much more effective [10]. The method is simple: the best score $\alpha(s)$ among paths from the initial state to state s is found as well as the best score $\beta(s)$ among paths from state s to a final state; both can be solved by single-source shortest distance algorithms [1]. [3] States for which $\alpha(s) + \beta(s) < \kappa * \beta(s_0)$ are then pruned, where s_0 is the initial state of the lattice. A similar procedure can be used for arcs.

The same pruning algorithm can be used to garbage-collect the lattice under construction periodically during decoding. For this purpose, for each active arc a we add to the lattice a temporary ϵ-arc with weight $1/\alpha(S(a))$ to a temporary lattice final state. This construction ensures that any pruned path would have been pruned anyway by the previously described pruning of the complete lattice.

2.3. Lattice Optimality

In general, the lattice generation method we have described may not contain all the recognition hypotheses within Viterbi beam of the best scoring path. This is because the chosen start time t' of the active arc a in (4) may be the optimal boundary between a and the preceding phone history only for the path $\mathbf{a}'$ that maximizes $\alpha(S(a), t')$. Another distinct phone history might require the boundary with a to be placed elsewhere.

However, if we assume A) the time boundary between two (context-dependent) phones found by the recognizer does not depend on the phone history before those phones, and require that B) all predecessor (context-dependent) phones of a given phone instance in the ASR network be identical, then a lattice generated by our method will be a full lattice.

Our method can be modified to generate full lattices by computing and saving explicitly the values of t' for a (and corresponding arc costs) in (4) that survive the beam. However, this precludes the efficient dynamic programming internal to arc a, thus incurring significant extra costs in recognition time and space.

Compared with the conditions imposed by the previous lattice generation methods described in the introduction, we have replaced word-level conditions (1)-(3) by two phone-level conditions (A) and (B). As we have claimed our condition (B) is a less restrictive on the ASR network topology than conditions (2) and (3), since (2) and (3) imply (B) but not conversely. For example, (B) holds for any lexicon and language model which is used with (fully) triphonic context-dependent phone models, so there are no restrictions at the context-independent phone or word-level in that case. Since our ASR networks are typically built for triphonic (or larger) acoustic models, condition (B) is generally met without network modification [5].[4]

However, it is also possible to impose condition (B) on a network with less than triphonic context by means of a finite-state transduction. One simply uses the triphonic construction presented in [14] to build a triphonic context-dependent transducer, which is then composed with the cascade of the lexicon and language model. Even in this case, the network optimizations that are applied to the combination of lexicon and language model presented in [5] can still be applied without violating condition (B), but optimizations to the full context-dependent network (or lower) as presented in [8] can not.

Condition (A) is *more* restrictive than condition (1). But this is an assumption about acoustic phonetics, not ASR network topology. Whether it is a reasonable assumption is an empirical question. A key result from the next section is that our lattices are empirically very similar to full lattices, thus we conclude condition (A) is reasonable.

3. Results

3.1. Lattice Quality

We compared our efficient lattice generation method with full lattice generation on the NAB task using the Eval '95 test set. We used a 160,000 word vocabulary, a bigram language model with 3.6 million n-grams, and a triphonic gaussian mixture acoustic model with 5000 mixtures and four components per mixture.

We measured lattice quality by computing the *lattice word accuracy*, which is highest word accuracy rate of all paths in the lattice. Table gives the results of this experiment at various recognition beams. We see that our lattice WACC for lattices generated by our efficient method differs little from lattices that contain the full set of recognition hypotheses.

[3] A 'superfinal' state reached by ϵ-arcs from the final states must be added for $\beta(s)$ to be a single-source problem.

[4] Strictly speaking, our acoustic models would have to have all possible triphonic models for condition (B) to be completely valid. Our typical acoustic models for NAB, the number of distinct HMMs is over half the number of all possible triphones (i.e., $nphones^3$), which appears to be sufficient to give high-quality lattices as shown in the next section.

Beam	Lattice WACC	
	Efficient	*Full*
12.0	93.3	93.4
13.0	95.3	95.3
14.0	96.4	96.6
15.0	97.2	97.4

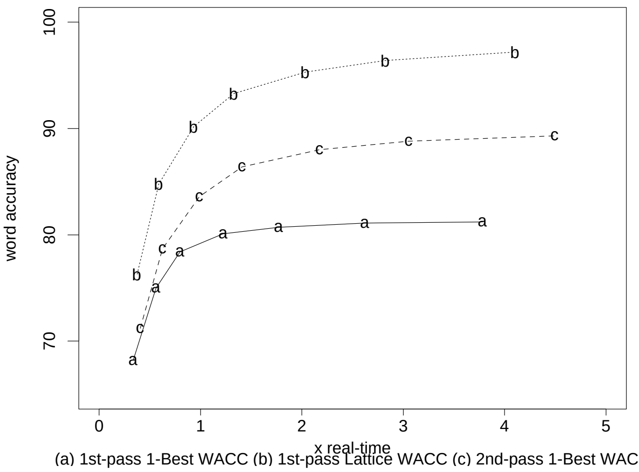
Table 1: Efficient Lattice versus Full Lattice Quality

3.2. Lattice Generation Time

Using the same NAB task and models as above, we compared the time to produce lattices versus the time to produce just the one-best response at various beams. Figure 1a,b shows that lattice generation requires typically less than 10% additional computation time above one-best decoding. [5] Curve (b) represents an upper bound on the accuracy that can be attained with rescoring, since any form of rescoring can only find answers that are already present (possibly with incorrect scores) in the first-pass lattice.

3.3. Lattice Rescoring

Finally, we demonstrate how this lattice generation method can be used to create a two-pass system. For the first pass, we used the same NAB models as described above to generate lattices at various beams. For the second lattice-rescoring pass, we used a a 6-gram language model with 20 million n-grams, a stochastic, TIMIT-trained, 463331 word multiple-pronunciation lexicon, and a triphonic gaussian mixture acoustic model with 7000 mixtures and twelve components per mixture. The acoustic models were adapted to each speaker using a single full-matrix MLLR transform. Fig. 1c shows total recognition time versus word accuracy for the two-pass system at different beams.



(a) 1st-pass 1-Best WACC (b) 1st-pass Lattice WACC (c) 2nd-pass 1-Best WACC

Figure 1: Comparison of (1st and 2nd Pass) 1-best and lattice word accuracies: each letter on a curve corresponds to beams 9 through 15 in unit steps.

With this second pass we were able to achieve a 3x real-time word error rate of 11.2% on the Eval '95 test set with the two-pass system, which is only 1.7% worse than AT&T's official benchmark result that year using what was then a 1000x real-time system [13]. [6]

4. REFERENCES

1. T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*. 1992. The MIT Press: Cambridge, MA.
2. J. Hopcroft and J. Ullman, *Introduction to Automata Theory, Languages, and Computation*. 1979. Addison Wesley: Reading, MA.
3. C. Lee and L. Rabiner, "A Frame-Synchronous Network Search Algorithm for Connected Word Recognition", *IEEE Trans. on ASSP*. Nov. 1989. pp. 1649-1658.
4. M. Mohri and M. Riley, "Weighted Determinization and Minimization for Large Vocabulary Recognition". *Proc. Eurospeech*. Sept. 1997. Rhodes, Greece.
5. M. Mohri, M. Riley, D. Hindle, A. Ljolje, and F. Pereira, "Full Expansion of Context-Dependent Networks in Large Vocabulary Speech Recognition", *Proc. ICASSP*. May 1998. Seattle, WA.
6. M. Mohri, F. Pereira, and M. Riley, "FSM Library – General-Purpose Finite-State Machine Software Tools", *http://www.research.att.com/sw/tools/fsm* 1998.
7. M. Mohri, F. Pereira, and M. Riley, "A Rational Design for a Weighted Finite-State Transducer Library ", *Lecture Notes in Computer Science*. 1436, 1998.
8. M. Mohri and M. Riley, "Integrated Context-Dependent Networks in Very Large Vocabulary", *Proc. Eurospeech*. Sept. 1999. Budapest, Hungary.
9. H. Ney and X. Aubert, "A Word Graph Algorithm for Large Vocabulary, Continuous Speech Recognition", *Proc. ICSLP*. 1994. Yokohama, Japan.
10. J. Odell, *The Use of Context in Large Vocabulary Speech Recognition*. Ph.D thesis, 1995. Cambridge U. Eng. Dept. Cambridge, U.K..
11. F. Pereira and M. Riley. "Speech Recognition by Composition of Weighted Finite Automata" . In E. Roche and Y. Schabes, editors, *Finite-State Language Processing*. 1997. MIT Press, Cambridge, Massachusetts.
12. G. Riccardi, E. Bocchieri, and R. Pieraccini, "Non-deterministic stochastic language models for speech recognition" *Proc. ICASSP*. 1995.
13. M. Riley, A. Ljolje, D. Hindle, and F. Pereira, "The AT&T 60,000 word speech-to-text system." *Proc. Eurospeech*. Sept. 1995. Madrid, Spain.
14. M. Riley, F. Pereira, and M. Mohri, "Transducer Composition for Context-Dependent Network Expansion". *Proc. Eurospeech.* Sept. 1997. Rhodes, Greece.
15. R. Schwartz and S. Austin, "A Comparison of Several Approximate Algorithms for Finding Multiple (N-BEST) Sentence Hypotheses", *Proc. ICASSP*. 1993. Minneapolis, MN.

[5] These experiments were run on a Compaq Alpha 21164.

[6] We estimate that the machine on which the present experiments were conducted is around 10 times faster than that used for our 1995 experiments, thus these results suggest a speed-up of at least 50 times for a relative increase in error rate of less than 20%.