Report for Automatic Speech Recognition 2019-20 Assignment

S1735938

Abstract

This report discusses a number of experiments on the weighted finite state transducers (WFST) and the Viterbi decoder based speech recognition system. A range of methods are attempted to improve the recognition accuracy and the computational efficiency respectively, with result and conclusion provided following each experiment.

1. Initial systems (Task 1)

In this report, all the experiment are performed with a 12 words lexicon, with each phone in words represented by 3 WFST states. The WFST is built with OpenFst-Python library (Allauzen et al., 2007).

The performance of each model/experiment is measured in the following ways over the 180 utterances made available for the course[1]:

- Run time: The time (in the unit of second) consumed in the experiment to decode and back trace the utterances. The runtime is an indication of speed, and the results are more meaningful compared relatively among the experiments.
- Number of forward computations: The cumulative number of forward calculations during the Viterbi decoding. The number should have a strong positive correlation with run time and it is also an indication of *speed*.
- Number of states/arcs: The number of WFST states and arcs in the model. This is an indication of *memory* usage of each experiment, and they should grow proportionally as the lexicon size increases.
- Word Error Rate (WER): The sum of the minimum total number of substitution, deletion, insertion errors divided by the total number of words in the transcripts of the utterances. This is served as an indication of the recognition *accuracy*.

[1]The experiments are all executed on a basic DiCE desktop running Scientific Linux SL7.6.

In the baseline system, the start state has an uniform probability of 1/12 going into each word. All state[2] has an identical self-loop probability of 0.1 and a transition probability to the next state of 0.9 to serve as a baseline. In the end, each word WFST has a final state with an epsilon transition to the start state to be able to recognize word sequence. A illustration of this baseline WFST is shown in Figure 1.

Together with the provided monophone-neural observation model, the word sequence in an utterance is estimated by taking the word sequence with the maximum the final posterior probability $P(Word\_Sequence|Utterance)$, which is proportional to the Viterbi values of the last acoustic frame of the utterance at the final state (according to Bayes' theorem). As a result, we obtain a baseline model with its performance over 180 utterances shown in Table 1.

Without comparing with the other models performance, we already observe that the WER is enormous at 101.7%. This is likely a result of the rigid hand-assignment of transition probabilities and the absence of a "silence" model. We also cannot rule out the non-zero probability of an inadequate observation model in our system. At the very minimum, if there are four frames corresponding to each WFST state[3], the self-loop probability should be around 0.8 instead of 0.1 to obtain a better WER. The absence of a "silence" model in the current WFST can also partly explain the number of Insertion Errors (accounting for more than a half of the total errors), because there are possibly pauses at the beginning, between the word phrases and at the end of the recordings, that the current system is likely to recognize as other words.

It is worth noting that two errors are easily overlooked within the system that outputs phone symbols as we had experienced. Firstly, the system is easily made to recognize phones in a word repeatedly that should have outputted only once, e.g. "p p p p eh p er s" for "peppers". We attribute this error to a missing of an else condition to set `self.W[t][j] = []` when `arc.olabel==0` in the `forward_step()`. Because if `self.W[t][j]` has been set to some non-empty symbol by a previous incoming arc but the current arc i is a better path, `self.W[t][j]` should be set to

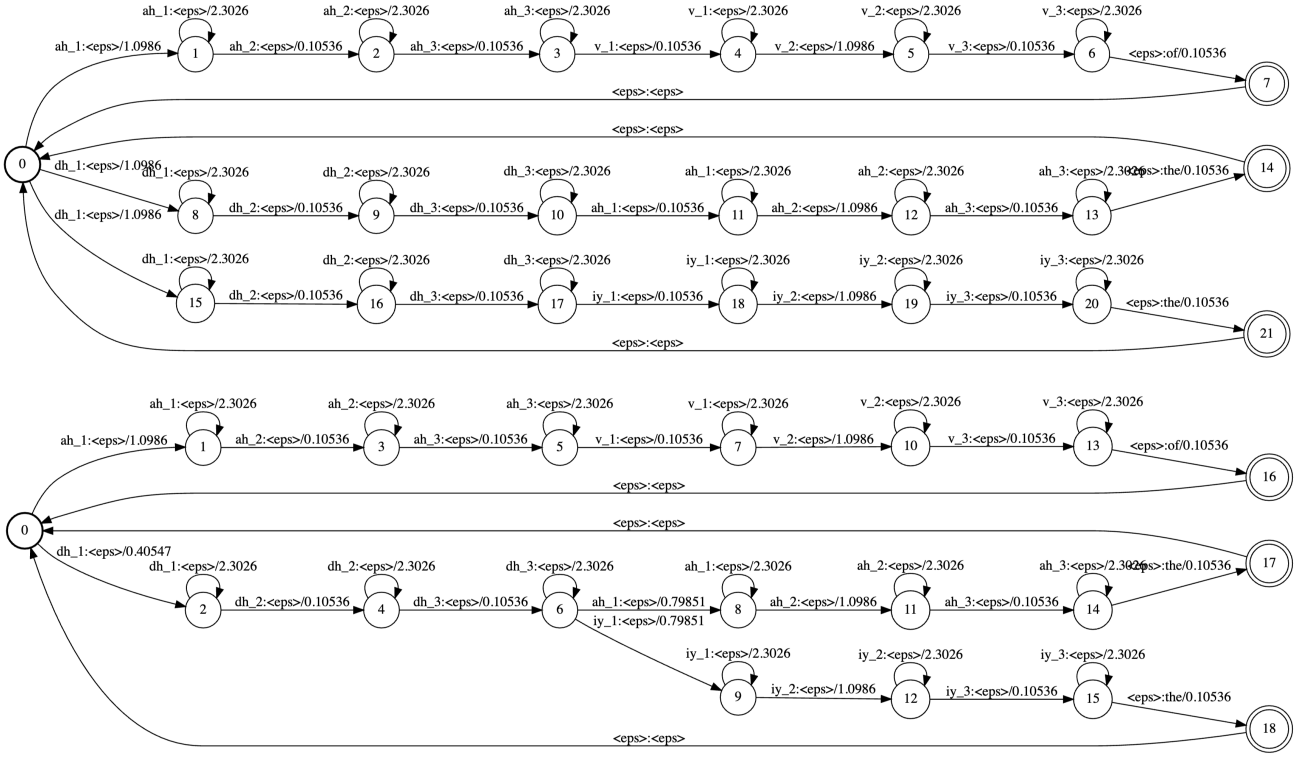
[2]Except the start state and final states.

[3]Estimated by the ratio of time steps (94803) and the number of the phones in the transcripts of all the utterances times 3 (the number states for each phone) which is 23727

Table 1. The performance of the initial "baseline" system, and the ones with various modifications, which are discussed in the following sections. The models under "-Improving the recognition accuracy" are discussed in Section. 2, and the models under "-Improving the computational efficiency" are discussed in Section. 3, e.g. the numbers in the first column under the header "self-loop probability" refers to models with such value of self-loop probability. "Run Time" is the time elapsed for decoding and back tracing as discussed previously. " Forward" indicates the number of forward probability calculations, " States/Arcs" indicates the number of WFST states and arcs in the model and "WER" shows the WER of the corresponding system along with its count decomposition in each of the three error categories [Substitution, Deletion, Insertion] (which become useful when analyzing their performance).

MODEL	RUN TIME	# FORWARD	# STATES/ARCS	WER [S D I]
-Baseline:				
	923.5	22753320	127/252	101.7% [728 205 1282]
-Improving the Recognition Accuracy:				
SELF-LOOP PROBABILITY:				
0.20	895.6	22754595	127/252	83.8% [696 284 844]
0.50	918.5	22753086	127/252	74.6% [668 368 589]
0.80	917.4	22752231	127/252	70.4% [620 427 486]
0.99	881.2	22753213	127/252	68.3% [602 539 345]
FINAL PROBABILITY:				
0.01	898.3	22871429	127/252	86.7% [739 273 876]
0.10	927.3	22817744	127/252	92.1% [733 244 1029]
0.45	930.3	22776547	127/252	96.5% [737 222 1141]
0.80	895.5	22759805	127/252	100.3% [726 214 1243]
USING BAUM-WELCH TRAINING:				
	944.0	22756424	127/252	66.4% [591 456 398]
ADDING SILENCE MODEL:				
	901.3	24153778	133/268	67.8% [657 351 467]
VARYING THE GRAMMAR:				
UNIGRAM	968.3	22757285	127/252	97.7% [713 227 1189]
BIGRAM	1395.4	34764516	127/384	95.8% [754 214 1117]
-Improving the Computational Efficiency:				
PRUNING BEAM WIDTH:				
0.10	362.4	5487107	127/252	132.5% [883 240 1762]
0.28	315.1	3937419	127/252	142.2% [895 499 1707]
0.46	294.4	3106100	127/252	151.8% [870 742 1693]
0.63	278.5	2676169	127/252	157.4% [858 806 1762]
0.81	270.0	2408269	127/252	152.5% [865 834 1622]
TREE-STRUCTURED LEXICON:				
	706.1	17096838	97/192	101.7% [728 205 1282]
LANGUAGE MODEL LOOK-AHEAD:				
0.10	269.7	2372157	98/193	121.2% [646 760 1233]
0.55	250.3	1534952	98/193	106.7% [188 1781 353]
0.99	246.5	1337040	98/193	102.5% [65 2030 137]

Figure 1. (On the top) The initial WFST in a three-word simplified version with start state marked in bold circle and final states in double circles. The word outputs are at the end of each word model. The transition probabilities as labelled on the arcs are in negative log values. (On the bottom) The tree-structured WFST of the same three-word lexicon as the figure above. Notice how nodes with the same input system, e.g. dh_1 on the top, are merged into a single state the state 2 on the bottom. The weights are also changed correspondingly.



empty again. Secondly, when there are multiple words in an utterance, it would remember nothing except the last word in the best path, e.g. outputting only "piper" in "where's peter piper". We found this is also caused by a missing else condition but in the traverse_epsilon_arcs (), that should set self.W[t][j] = self.W[t][i] when arc.olabel==0. Without having this, the output symbol would not be propagated and stored when there are two consecutive epsilon transitions.

2. Improving the recognition accuracy (Task 2 and a part of 4)

Five measures are experimented to improve the recognition accuracy as described in this section. This includes: varying the self-loop probabilities, varying the final probabilities of word WFST, utilizing the Baum-Welch training algorithm, adding the ergodic "silence" WFST and using different language model grammars.

2.1. Varying the self-loop probabilities

Hypothesis: As we estimated from Section 1, a better WER should be obtained when the self-loop probability for each state (non start, final states) is around 0.8, based on the total number of observation frames and the number of the phones in the transcription. Thus we perform experiments with varying self-loop probabilities of 0.20, 0.50, 0.80 and 0.99 (where 0.99 is to check if the system still improves with self-loop probability larger than 0.8), while keeping the other parameters of the system unchanged.

Result: The result is shown in Table 1 under the "self-loop probability" header. These statistics can be compared with the baseline system which has a self-loop probability of 0.1 and WER of 101.7%.

Conclusion: We can see the WER performance keeps improving as the self-loop probability increases, achieving the best WER when the parameter equals to 0.99 which is beyond our estimation (0.8). We attribute this to the large number of "silence" frames corresponding to the pauses in the utterances. These frames are recognized as new words because the system does not have a model for "silence",

which in the end contribute to the insertion errors in WER. So by increasing the self-loop probability, we indirectly impede the system's tendency of inserting new words.

To support our new hypothesis, we recorded the number of substitution, deletion and insertion errors when the self-loop probability is changed from 0.8 to 0.99, and found the numbers has changed from [620 427 486] to [602 539 345] respectively. The insertion error decreases by more than 25%, as we expected. While the deletion error increases by around the same percentage, as the system tends to recognize less words overall.

2.2. Varying the final probabilities

Hypothesis: The final probabilities add extra cost[4] to paths passing through the final states in the WFST. Based on the initial system, the system ought to have a lower WER when the final probability is smaller. Because of the same reason that a higher self-loop probability leads to better WER: it controls the system's tendency to over-generate words since there is no "silence" states in the WFST that can effectively model the pauses in audio.

Result: The result is shown in Table 1 under the "Final probability" header. The result of adjusting the final probability is not as effective as adjusting the self-loop probability. Even the best performing final probability parameter obtains a worse WER than any of the "self-loop probability" experiments.

Conclusion: Adding final probabilities does not have as promising a result as the previous experiment due to its limit ability in influencing any given path's cost. It was only able to add costs when the WFST produces words, compared to the self-loop probability experiment where it can add costs every time the same phone symbol is recognized at the last time step (the self-loop situation).

2.3. Using Baum-Welch training

Hypothesis: Baum-Welch training algorithm re-estimates the self-loop probabilities and transition probabilities for each arcs in the WFST, which should lead to a superior performance than the baseline system. The transition probability $\hat{a}_{ij}$ from state i to j is estimated by computing:

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T} \xi_{i,j}(t)}{\sum_{j=1}^{J} \sum_{t=1}^{T} \xi_{i,k}(t)} \tag{1}$$

where T is the total number of frames in a utterance, J is all the possible states in WFST. And

$$\begin{aligned} \xi_{i,j}(t) &= P(q_t = i, q_{t+1} = j | \mathbf{X}, \mathcal{M}) \\ &= \frac{\alpha_i(t) a_{ij} b_j(\mathbf{x}_{t+1}) \beta_j(t+1)}{\alpha_E} \end{aligned} \tag{2}$$

where $q_t = i$ refers to the state at time t is i, $\mathbf{X} = \mathbf{x}_1, ..., \mathbf{x}_T$ is the observation sequence, $\mathcal{M}$ is the WFST topology, $\alpha_i(t)$ is the forward probability of state i at frame t, $b_j(\mathbf{x}_{t+1})$ is the emitting probability of frame $t+1$ at state j, $\beta_j(t+1)$ is the backward probability of state j at frame $t+1$ and E the end state.

These two equations are used iteratively to update weights and the state occupation probabilities. We are aware that normally there should not be any overlap between the training and test data, but this was relaxed in our experiment since we only wanted a "proof-of-concept" that this algorithm can work with our setup and data.

Result: The training algorithm is repeated until the transition probabilities have converged. The resulting recognition performance is shown in Table 1 under the "Using Baum-Welch training" header. It achieve a WER of 66.4%, which is the best performance out of all the single techniques applied to improve the recognition accuracy.

Conclusion: Although the decoding run time is comparable to the baseline system, we should take into consideration the large amount of time spent on training the parameters. There is also the probability that the parameters will stop updating when we only get to a local optimum in the parameter space. But it is overall an effective unsupervised learning method for improving recognition accuracy.

2.4. Adding silence state in the WFST

Hypothesis: As speculated previously, with the baseline system, the pauses in audio files are modeled with words, which intuitively adds insertion errors to the result. The addition of an silence model should therefore alleviate this issue. A silence model as shown in Figure. 2 is composed into the baseline WFST as shown in Figure 1 to perform this experiment. The ergodic structure of the middle three states helps to model the variations of the pauses.

Result: The result is shown in Table 1 under the "adding silence model" header. The WER of 67.8% is a significant improvement compared to the baseline's 101.7% (reducing the number of insertion errors from 1282 to 467), while the other statistics are not much worse than the baseline.

Conclusion: It is as anticipated that the number of states and arcs increases by 6 and 16 respectively. Although this addition raises the number of total forward computations by around 2 million, the total run time is only raised by around 40 seconds (0.25 second per utterance), which overall is good improvement in performance. Therefore "silence"

[4] in negative log probability term

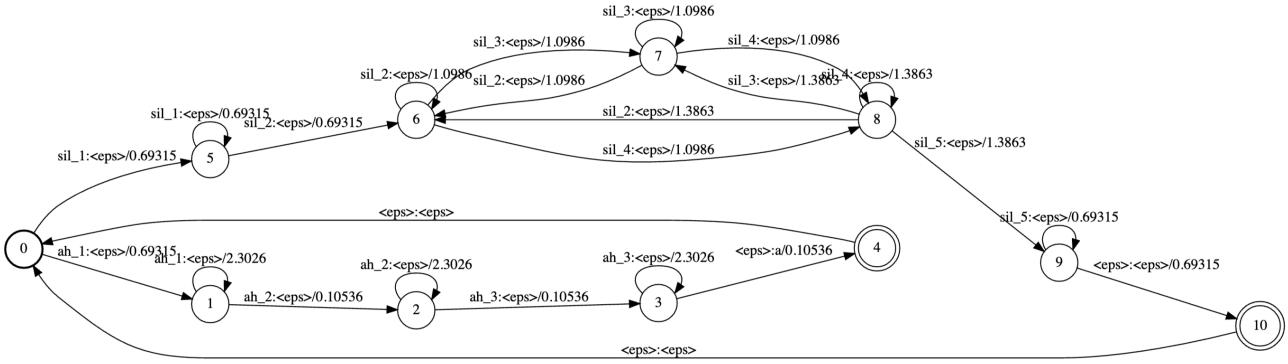


Figure 2. The 5 state left-to-right silence WFST from state 5,6,...,9 (with the middle 3 states having an ergodic structure) composed into the a simplified WFST, just as another word from the lexicon.

models should always be added to speech recognition systems whenever possible to lower the WER. Different typologies of the "silence" model are to be experimented for comparison.

2.5. Varying the grammar

Hypothesis: We attempt to use both the Unigram and Bigram language model in this experiment, with the models trained on the entire data set. We are aware that normally there should not be overlap between training and testing data, but this should not be a problem since we hypothesize that there won't be a significant improvement in WER even if we test on the training data. Due to the words in utterances are nearly all randomly sampled from an uniform distribution.

Result: The result is shown in Table 1 under the "Varying the grammar" header.

Conclusion: There are indeed not much improvements in WER with neither grammar. The Unigram model has nearly the same performance as the baseline model. The Bigram model has a slightly improvements over the baseline, likely because there are a few memorable two-words phrases e.g. "peter piper", "peck of" that would appear in the audio recordings more frequently and are thus modeled by the Bigram, while the rest of the words remain randomly ordered. The computational cost of the Unigram model is comparable to that of the baseline, while the Bigram model spends around 50% more run time, number of forward computations and arcs. The number of states is unchanged as we are able to add arcs between the last and the start state of word combinations to represent the Bigram conditions, as demonstrated in Figure 3.

Although these language models are not extremely useful in our toy experiment setup, in the case of recognizing normal speeches in the real world, they should be more helpful, especially with larger data set and more sophisticated N-gram models.

3. Improving the computational efficiency (Task 3 and another part of 4)

Three measures are experimented to improve the computational efficiency, or more specifically, the time cost and the memory space cost of the recognition process. This ranges from pruning the search space, modifying WFST to tree structure and language model look ahead.

3.1. Effect of pruning

Hypothesis In this experiment, pruning is implemented in the form of beam search as in (Lowerre, 1976). In the probability domain, paths with probability lower than the beam width θ times the probability of the best path at the current time step D is pruned. Thus the greater the beam width would results in more paths being pruned, inferring less computational time, forward path but also worse WER. In the experiment, 5 values of θ, evenly spaced between 0.1 and 0.99 excluding the right end point are used for computating the result.

Result: The result is shown in Table 1 under the "Pruning beam width" header.

Conclusion: From the result, even the system with the smallest pruning has a computation time less than a half, and the number of forward computations less than a quarter of the baseline system (as in Section 1). Of course this comes with a cost of 30.8% increase in WER, resulting in 132.5%, while the number of states and arcs are the same. With further pruning ($\theta = 0.81$), the number of forward computations is halved again, but results only in 25% of time saving (from 362.4s to 270.s) and a 20.0% increase in WER. Thus pruning widths of around 0.1 or less is probably more desirable than the larger ones, as they

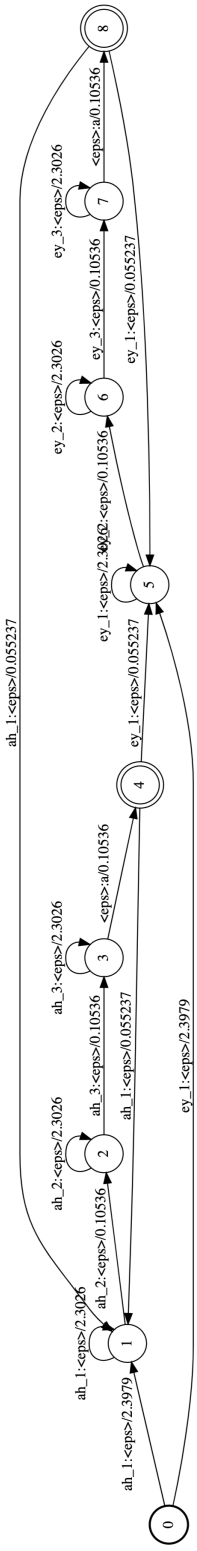


Figure 3. A simplified WFST with the Bigram language model. Notice that the different pronunciations of the same word are modeled with the same probability in our experiment. Comparing to the baseline WFST, the Bigram one has arcs going from the end of word WFSTs directly to the first state of words, e.g. to go from state 4 to state 5, the Bigram WFST has an direct arc whereas the baseline would only have arcs going from state 4 to state 0, then 0 to 5.

provide significant computation time saving without too much sacrifice in WER, as also discussed in (Huang et al., 2001).

3.2. Effect of tree-structured lexicon

Hypothesis: A tree structured lexicon as discussed in (Ortmanns & Ney, 2000) is experimented with the help of the determinize () function in OpenFst to generate equivalent WFSTs with no state having two transitions with the same input label (except for epsilon transitions as they are treated as regular symbols). This reduces the overall number of repeated states and arcs, thus should reduce the total run time and the number of forward computations.

Result: The result is shown in Table 1 under the "Tree-structured lexicon" header. A miniature version of the WFST used can be found on the bottom of Figure 1.

Conclusion: The tree-structured lexicon is quite effective, reducing 30 and 60 redundant states and arcs respectively. This results in a reduction of the computation time by 24% and the number of forward computations by 25%, while the WER has not been affected at all, since the tree-structured lexicon has a path of the equivalent cost for every path in the initial system. The technique offers substantial computational efficiency improvements without any cost in recognition accuracy. But we should also be aware of the disadvantage of the tree-structured lexicon over the linear lexicon that: we are **not** able to update the cost of recognizing different words at the very first arcs going into those word WFSTs (which can be used in e.g. pruning), but is possible when words do **not** share nodes and arcs (Kitaoka et al., 2005).

3.3. Effect of language model look-ahead

Hypothesis: The idea of language model look-ahead (Ortmanns et al., 1996) incorporates the tree-structured lexicon with beam search for an optimal improvements in the computational efficiency. Thus the enhancement should produce a better result than either the methods alone.

Result: The result is shown in Table 1 under the "Language model look-ahead" header.

Conclusion: This techniques yields exemplary result: with even the smallest pruning width (0.1) produces faster runtime, less computation cost and better WER than all of the ones with beam search alone. The one with language model look-ahead beam width of 0.99 produces faster runtime but, at the same time, better WER than the one with lower beam width. It even performs better in every category than the experiment with the tree-structured lexicon alone, except for 0.8% higher WER! Interestingly, the WER keeps improving as the beam-width become higher, which is exactly opposite to the results from Section 3.1. By decomposing the WER values, we observe that the number of substitution, dele-

tion and insertion errors have change dramatically, as the beam-width varies from 0.1 to 0.99, from [646,760,1233] to [65,2030,137] respectively. The number of substitution and insertion errors drop significantly, while the deletion errors soar. Furthermore, we notice that there are more than double amount of utterances that have no path reached to the end of the observations, which results in empty recognized words. It means a lot of the sentences that used to have recognition results (with a mixture of correctly recognized words and substitution, insertion errors) when the beam-width is low is now having empty results (which means zero substitution, insertion error). This explains the dynamics of the decomposition of the error counts. Therefore the improvement in WER is only an superficial effect of the underlying changes.

4. Discussion

In this report, we experimented with a range of techniques to improve the recognition accuracy and the computational efficiency with varying effects. For both our task at hand and real world applications, the best recognition accuracy should be obtained by combining the Baum-Welch training, the "silence" model with the Bigram grammar, and the tree-structured lexicon for improving the efficiency without degrading the WER. In any real world application with large vocabulary size, some forms of pruning would likely need to be employed for decoding at a reasonable speed. Although the language model look-ahead appears to have a better WER than the basic form of pruning, it should be used with scrutinize of the recognition output. The size of the pruning should be decided based on the particular application, with a constant trade off between recognition accuracy and speed.

Due to the Covid-19 pandemic, the extend to which combinations of the aforementioned techniques can contribute to the current task is untested, but we should be able to reduce the WER to sub-60% using the currently available methods.

References

Allauzen, C., Riley, M., Schalkwyk, J., Skut, W., and Mohri, M. Openfst: A general and efficient weighted finite-state transducer library. volume 4783, pp. 11–23, 07 2007. doi: 10.1007/978-3-540-76336-9_3.

Huang, X., Acero, A., Hon, H.-W., and Foreword By-Reddy, R. *Spoken language processing: A guide to theory, algorithm, and system development*. Prentice hall PTR, 2001.

Kitaoka, N., Takahashi, N., and Nakagawa, S. Large-vocabulary continuous speech recognition using linear lexicon search and 1-best approximation tree-structured lexicon search. *Systems and Computers in Japan*, 36(7): 31–39, 2005.

Lowerre, B. T. The harpy speech recognition system. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE, 1976.

Ortmanns, S. and Ney, H. The time-conditioned approach in dynamic programming search for lvcsr. *IEEE transactions on speech and audio processing*, 8(6):676–687, 2000.

Ortmanns, S., Ney, H., and Eiden, A. Language-model look-ahead for large vocabulary speech recognition. In *Proceeding of Fourth International Conference on Spoken Language Processing. ICSLP'96*, volume 4, pp. 2095–2098. IEEE, 1996.