ASR assignment report

s2282172 s2473164

Task 1

The baseline wfst we used consists of a start state, a sequence of subphone states for each word and a final state. The transition probability from the start state to each word is equal: $\frac{1}{num\ words}$. We use a left to right hmm typology to represents each word. The states are represented as subphones where each phone consists of three subphones, for example, phone ‘ey’ is represented as ‘ey_1’, ‘ey_2’, ‘ey_3’. All the non-start and non-final states have self-loop probability 0.1 and transition probability to next state 0.9. In the decoder, we use Viterbi algorithm without any pruning methods. The results are shown in Table 1.

Overall word error rate	144.62%
Total backtrace time	0.15s
Total decode time	976.26s
Total forward computations	32,513,904
Num states	116
Num arcs	230

Table 1. The results of the baseline wfst which uses equal transition probabilities to transit from start state to each word.

From the results, we can see the results are bad, both in speed and word error rate.

For the speed, we need around 2s for each utterance, and we find that most utterances are quite short, so it is obviously too slow to decode.

For the word error rate, we find that our baseline model can only recognize some isolated words compared to the transcriptions, also, the recognition results consist of lots of irrelevant words especially some short words such as ‘a’, ‘the’, ‘of’. For example, one of the transcriptions is ‘a peck of peter picked’, The recognition results of our models are ['the', 'the', 'the', 'of', 'pickled', 'of', 'peter', 'picked', 'the', 'picked', 'picked', 'the', 'picked']. From this example, we can see that our model only recognizes some isolated words and the results contain too many irrelevant short words such as ‘a’, ‘the’, ‘of’.

The reasons we think for the poor results:

1. We do not use any pruning methods, which makes our decoder inefficient by considering any possible states at each time step. Instead of this, we can use some pruning methods such as beam search by only considering the most possible states at each time step, we will experiment different pruning methods in next question.
2. We do not incorporate language model to the wfst, we use equal probability from the start state to transit to each word. However, if we check the unigram probability of all the transcriptions, we will find that some short words such as ‘a’, ‘the ‘ have much lower probability than other words. As a result, we will have much higher probability to transit to these than they would have. We will experiment different language models in the following questions.
3. We do not use any silence states between words. We know that there may have pause between words, but our wfst does not take it into consideration, as a result, the outputs of our decoder are always longer than the transcriptions. This reason, together with the last reason, cause the outputs contain too many irrelevant short words.
4. The self-loop probabilities and transition probabilities are set arbitrarily by hand, however, different self-loop probabilities and transition probabilities will have big effect to the results. These

probabilities should be trained using Baum Welch algorithm.

Task 2

Using unigram probabilities

In this experiment, we add unigram probabilities to our baseline model, instead of using equal probability to transit to each word from the start state, we use their unigram probabilities computed from all the transcriptions to transit to each word. The other structure and transition probabilities are the same as the baseline model. This helps to reduce the word error from 144.62% to 139.19%. The other metrics are almost the same as the baseline model.

Using unigram probability helps to reduce the word error rate, but only a small value. We think the reasons are:

1. In the transcriptions, the short words such as 'a', 'the' have much lower unigram probabilities than other words, however, the other words almost have the same unigram probabilities. So we suppose unigram probabilities can help to reduce the irrelevant short words, but the effect to other words are very limited. For example, for the example 'a peck of peter picked' shown in our baseline model, the output of the unigram model we used in this part is ['the', 'the', 'of', 'pickled', 'of', 'peter', 'picked', 'the', 'picked', 'picked', 'the', 'picked'], we can see that the irrelevant short words have been reduced, but the results for other words are not improved.
2. The unigram does not take the context of the words into consideration, which cause the improvement is limited, we can use bigram or trigram to further improve the performance. We will experiment bigram in the following questions.

Using different self-loop probabilities and transition probabilities

In this part of the experiments, we use different self-loop probabilities and transitions probabilities for the wfst with unigram grammar in the last part. The results are shown in Table 2.

Self-loop probabilities	Transition probabilities	Total word error rate
0.1	0.9	139.19%
0.2	0.8	102.34%
0.5	0.5	78.4%

Table 2. Results for wfst with unigram grammar with different self-loop probabilities and transition probabilities.

From the results, we have the following conclusions:

1. As we increase the self-loop probabilities and decreases the transition probabilities, the wfst can receive better performance. We think it is because when we increase the self-loop probabilities, during decoding, the model can have more opportunities to stay in the same state, thus it can help to reduce the output of the irrelevant short words as we see in the baseline model. For example, if we use self-loop probabilities 0.5 and transition probabilities 0.5, the output of the transcription 'a peck of peter picked' is ['the', 'of', 'pickled', 'of', 'peter', 'the', 'picked', 'the']. We can see that the output of the irrelevant short are significantly reduced compared to the Vaseline model and the model with unigram in the last question.
2. In our experiments, we set the self-loop probabilities by hand, although in this part, increasing the self-loop probabilities can help to improve the

performance. However, the self-loop probabilities and transition probabilities should be trained using Baum Welch algorithm.

transition probability to next state 0.9. We experiment with different p_s and p_f, the results are shown in Table 3.

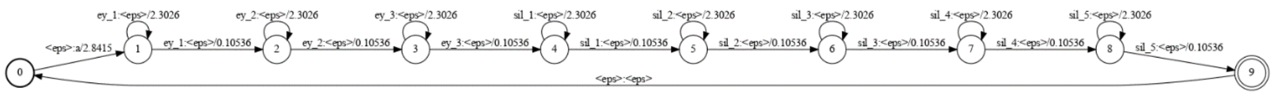

Figure 1. The example word_sil_wfst for 'a'.

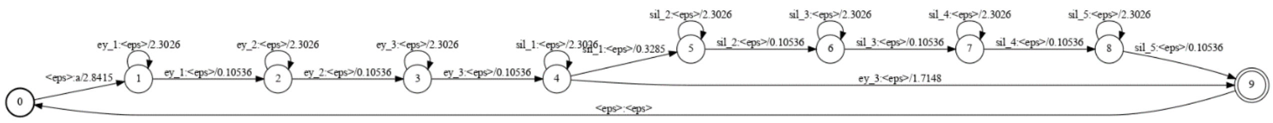

Figure 2. The example sequence_sil_wfst for 'a'.

Adding silence states between words

We use two kinds of wfst to add silence states between words, including word_sil_wfst and sequence_sil_wfst. In word_sil_wfst, during decoding, the decoder has to go through the silence states while in sequence_sil_wfst, when the decoder is in the final phone state of the word, it can go to the silence states or directly go to the word final state with given probabilities. We use different probabilities for the sequence_sil_wfst. In fact, the word_sil_wfst can be seen as special case of sequence_sil_wfst with probability $p_t \times 1$ going to the silence states, where p_t is the sum of transition probability of the final phone state of the word to other states. In all the experiments in this part, we add five silence states at the end of each word with a left to right topology, each silence state has self-loop probability 0.1 and the transition probability to next state 0.9. All the other non-start and non-final states in word_sil_wfst also have self-loop probability 0.1 and transition probability to next state 0.9. The final phone state in sequence_sil_wfst has self-loop probability 0.1, and transition probability $0.9 \times p_s$ to the silence states, $0.9 \times p_f$ to the word final states, the other non-start and non-final states have self-loop probability 0.1 and We also use unigram probability to transit to each word. The example word_sil_wfst for 'a' is shown in Figure 1. The example sequence_sil_wfst for 'a' is shown in Figure 2. The experiment results of the word_sil_wfst are shown in Table 3.

Overall word error rate	53.57%
Total backtrace time	0.14s
Total decode time	1387.72s
Total forward computations	47778204
Num states	166
Num arcs	330

Table 3. The results of the word_sil_wfst which has five silence states at the end of each word, during decoding, the decoder has to go through the silence states. We also use unigram probabilities to transit to each word.

Transition probability to silence states	Transition probability to the word final state	Overall word error rate
0.9×0.5	0.9×0.5	71.41%
0.9×0.8	0.9×0.2	69.72%

Table 4. The results of the sequence_sil_wfst which has five silence states at the end of each word. The results show different from the final phone state of the word to silence states and the word final state.

From the results, we have the following conclusions:

1. Adding silence states between words significantly decreases the word error rate, for bath word_sil_wfst and sequence_sil_wfst. Adding silence can especially reduce the outputs of the irrelevant short words. For example, the word_sil_wfst decoding output of the example 'a peck of peter picked' is ['a', 'of', 'picked', 'of', 'peter', 'peter'], compared to the baseline model and the wfst with unigram grammar, the irrelevant short words such as 'a', 'the' have been significantly reduced.
2. The higher the transition probability from the final phone state of the word to the silence state, the lower the word error rate is. The word_sil_wfst achieves the lowest word error rate, as it can be seen as a special case of sequence_sil_wfst with probability $p_t \times 1$ going to the silences states, where p_t is the sum of the transition probability of the final phone state of the word to other states. In theory, sequence_sil_wfst is more appropriate for the task as some words do not need silence states between them, such as 'speech recognition'. However, we suppose in the assignment, the lexicon contains words that do not have strong relations between each other and the utterances are almost arbitrary, so the wfst with high probability going from the final phone state to the silences can receive good performance.
3. However, because we only use unigram for both word_sil_wfst and sequence_sil_wfst, which does not take the context of the words into consideration, we suppose the improvement is mainly because of the reduction output of the irrelevant short words. We still have the problem of just recognizing isolated words as in the baseline model. We can see this from the transcription 'a peck of peter picked', and its output ['a', 'of', 'picked', 'of', 'peter', 'peter'].

Task 3

Beam search pruning

We use two different kinds of beam search for pruning. One is using a fixed beam size k (such as 10, 30, 50, 70) during decoding, at each time step, we sort the Viterbi probability of all the states and keep k states with the highest k probabilities and set the Viterbi probability of all the other states to 0. The other is using a beam width threshold θ (such as 0.01, 0.00001), we then multiply the Viterbi probability by the beam width and set all the states whose Viterbi probability lower than this value to 0. In the actual implementation, we use negative log probability. All the experiments in this part are base on the word_sil_wfst in the last task. The results of different beam size are shown in Table 5.

Beam size	Decode time percentage	Forward computation percentage	wer
word_sil_wfst does not use beam search	100%	100%	53.57%
10	13.96%	10.1%	88.7%
30	24.94%	20.46%	63.23%
50	36.59%	32.56%	55.79%
70	50.77%	44.79%	53.94%

Table 5. The results of different beam size.

The decode time percentage is computed as $\frac{decode\ time\ of\ beam\ search}{decode\ time\ does\ not\ use\ beam\ search}$, the forward computation is computed as $\frac{forward\ computations\ of\ beam\ search}{forward\ computations\ does\ not\ use\ beam\ search}$.

From the results, we have the following conclusions:

1. As we increase the beam size, the decoder have to consider more states at each time steps, thus both the decode time and the forward computations increase, however, the word error rate decreases as we increase the beam size.
2. Using a beam size 70, we can achieve almost the same word error rate as we consider all of the states at each time step, but we only need to do 44.79% forward computations, and the decode time also reduces to 50.77% of the model which does not use beam search. For our assignment task, we suppose a beam size 50 is enough, which only need 32.56% forward computations of the model does not use beam search, but have comparable word error rate as the model does not use beam search.

The results of different beam width are shown in Table 6.

Beam width	Decode time percentage	Forward computation percentage	wer
word_sil_wfst does not use beam search	100%	100%	53.57%
0.01	9.47%	5.53%	92.85%
0.0001	12.44%	7.8%	77.12%
1e-6	13.61%	9.06%	67.01%
1e-8	15.05%	10.59%	62.7%
1e-10	21.65%	17.1%	59.7%
1e-12	24.93%	20.33%	57.72%

Table 6. The results of different beam width.

From the results, we have the following conclusions:

1. The beam width need to be very small if we want to consider more states at each time step, which indicates most of the states have much lower Viterbi probability than the maximum probability at each time step. It also indicates it is necessary to use beam search during decoding to avoid consider the very unlike path.
2. As we decrease the beam width, the decoder has to consider more states at each time step, so the decode time and the forward computation percentage increase, and the word error decreases. If we use a beam width 1e-12, we can receive comparable word error rate as the model does not use beam search, but we only need 20.33% forward computations.

Comparison of both beam search methods

From the results, we can see that using both methods, we can receive similar word error rate as the model does not use beam search, but we need to do mech less forward computations. However, using beam width, rather than the fixed beam size, we can receive the same word error rate with fewer computations. For example, using beam width method, we can receive 62.7% word error rate with only 10.59% forward computations. But we need 20.46% forward computations to receive the same word error rate if we use a fixed beam size. Also, different wfst have different num of states, it will be difficult to choose a fixed beam size, using the beam width method, the beam size at each time step at be variable depending on the Viterbi probabilities, so using the beam width and multiply it by the Viterbi probabilities at each time step and then prune the unlike states should be a more general method.

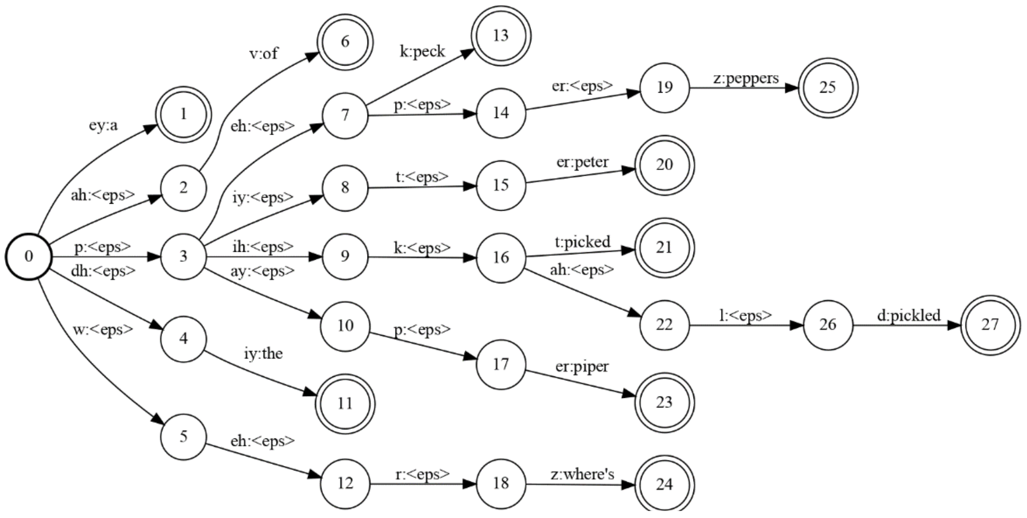

Figure 3. The tree structure lexicon. For clarity, the figure omits the arcs from the word final states to the start state.

Task 4

Using tree structure lexicon

In this experiment, we use tree structure lexicon rather than linear lexicon, the tree structure lexicon we used in the experiment is show in Figure 3. For clarity, the figure omits the arc from the word final states to the start state.
In our experiment, we use three suphone states for each phone, we also include silence states at the end of each word, which is similar to those in the word_sil_wfst in the beam search experiments.

The comparison of the num states and num arcs of the wfst which use tree structure lexicon and linear structure are shown in table.
From the results, we can see that using tree structure lexicon can significantly reduce the num states and num arcs, especially when we have a large vocabulary size.

	Num states	Num arcs
wfst uses tree structure lexicon	132	262
wfst uses linear structure lexicon	166	330

Table 7. Num states and num arcs of different transducer.

Using language model look ahead with tree structure lexicon.

In this experiment, we use language model look ahead with tree structure lexicon. At each time step during decoding, we compute language model look ahead probability for each state.

$$p_l = max_{w \in W(s)} p_w$$

p_w means unigram probabilities for each word, $W(s)$ is a set contains all the words that we can reach from current state.
We then multiply the language model look ahead probability with the Viterbi

probability and use it for pruning. In our actual implementation, we use negative log probability. We think using language model look ahead, we can prune the unlike path before we reach to the end of the words, so using the same threshold of the beam search, tree structure lexicon with language model look ahead will have better performance, especially when we use a small beam size or a large beam width (which means that the decoder needs to consider fewer states at each time step). We use a beam size 10 and beam width for the experiments, the results are shown in Table 8, Table 9.

	wfst with tree structure and language model look ahead	wfst with linear lexicon
Overall word error rate	115.37%	88.7%
Total backtrace time	0.12s	0.13s
Total decode time	133.65s	193.66s
Total forward computations	3379665	4826666

Table 8. Results for different transducers. A fixed beam size 10 is used.

	wfst with tree structure and language model look ahead	wfst with linear lexicon
Overall word error rate	103.78%	110.93%
Total backtrace time	0.12s	0.12s
Total decode time	64.88s	118.51s
Total forward computations	800841	2015340

Table 9. Results for different transducers. A beam width 0.1 is used.

From the results, we have the following conclusions:

1. Using tree structure lexicon can help to reduce the decode time and forward computations. We think it is because using tree structure lexicon, different words can share states, such as 'peter', 'picked'. The vocabulary size of the assignment is small, we suppose the advantage will be larger if we use a large vocabulary size.
2. In theory, using tree structure lexicon with language model look ahead, we can prune the unlike path before we reach to the end of the words, so using the same beam size or beam width, the wfst with tree structure lexicon and language model will receive better performance, especially we use a small beam size or a large beam width (which means the decoder needs to consider fewer states at each time step). And this is the case if we use beam width 0.1 for both of them. However, when we use a fixed beam size 10, the wfst with tree structure lexicon and language model look ahead has worse performance. We suppose it is because the utterances of the assignment are arbitrary, and many words have very similar unigram probabilities, which limit the performance of the wfst with tree structure lexicon and language model look ahead.

Using bigram probabilities

In this experiment, we use bigram probabilities which are computed from all the transcriptions to transit from one word to another. The wfst for each word is similar to word_sil_wfst we used in the beam search experiments, which have five silence states at the end of each word.

When we compute the bigram probabilities from all the transcriptions, we add a start symbol '#' at the beginning of each sentence to give the first word of the transcription bigram probability. We also add a end symbol '#' at the end of each sentence so that the probabilities of all the sentences of any length will sum to one, without the end symbol, the probabilities of the sentences which have a particular length, rather than all the sentences of any length, will sum to one.

The bigram_sil_wfst which we used in the experiments a start state, a sequence of subphone states for each phone and a final state. We also include a word start state and a word final state to each word so that we can use bigram probabilities to transit between words. The wfst has probability $p('w'|'\#')$ transits from the start state to each word, where 'w' is the corresponding word and '#' is the start symbol, and probability $p('\#'|'w')$ transits from each word to the final state, where '#' is the end symbol and 'w' is the corresponding word. All the other non-start, non-final, non-word-start, non-word-final states have self-loop probability 0.1, and transition probability to next state 0.9.

	unigram	bigram
Overall word error rate	53.57%	51.93%
Total backtrace time	0.14s	0.17s
Total decode time	1387.72s	1526.59s
Total forward computations	47778204	47778204
Num states	166	177
Num arcs	330	440

Table 10. The results of the wfst with bigram grammar.

From the results, we have the following conclusions:

1. The num states and num arcs of the model uses bigram are larger than unigram, which means that we need more space to store it. For our experiments, the num states is larger because we need extra word start for each word while in unigram, we only need a start state. The num arcs is larger because in bigram we need arcs from one word to all the other words. Because of the small lexicon in the assignment, the num states and num arcs of the bigram are just slightly than the unigram. However, as the size of the lexicon increases, the num states and the num arcs will grow rapidly, especially the num arcs. If we use a higher order language model such as trigram, then we have to make multiple state sequences for each word, as a result, the num of states and arcs will increase significantly. If the vocabulary is too large, then it may not be feasible to store the transducer.
2. The backtrace time and decode time are longer for the bigram model. We suppose it is because we have more states and arcs in the bigram model. Also, at each word final state, we have to consider the transitions to all the other words while in unigram we only need to transit to the start state.
3. The word error rate of the bigram is lower than the unigram. We think it is because using bigram, the model can take the context of words into consideration, thus be able to output some continuous words rather than the isolated words as in the baseline model and the unigram model. For example, the bigram model output of the transcription 'peter piper picked' is '['a',

'peter', 'piper', 'picked']'.

4. However, the improvement of the word error rate is not as large as we expect. We suppose it is because the utterances of the assignment are almost arbitrary, so many bigrams will have similar bigram probabilities, which leads to limited improvement. Also, we find there are some bigrams which are not appeared in all the transcriptions which results in some bigrams have zero probability. We can use some smoothing methods such as add-k smoothing and interpolation. We think another reason is that we directly use acoustic likelihood and language model probability without balancing them. We compute separate acoustic likelihood for each state and then multiply them to get the probability for the word, which can lead to underestimation for the probability for each phone. We can use a language model scaling factor and word insertion penalty to balance the acoustic likelihood and language model probability.