Enhancing ASR Systems: An Exploration of Weighted Finite State Transducers and Viterbi Decoding

s1945124

***Abstract*—Automatic Speech Recognition (ASR) systems are becoming part of daily life. From live captioning to automatic dictation, we have all used ASR in our lives. This report discusses a set of experiments on weighted finite state transducers (WFST) and Viterbi decoder, which are crucial parts of speech recognition systems. We probe WFSTs and Viterbi Decoder in a wide range of experiments and observe their effect on speech recognition by observing the change in accuracy and computational efficiency.**

***Index Terms*—Automatic Speech Recognition, WFST, Viterbi Decoder**

I. Task 1 – Initial systems

This section describes the data, and metrics used to perform all the experiments. A baseline system is described against which all the experiments will be compared.

A. Data

This report performs all the experiments on 318 utterances and their transcribed text. Recordings comprise of short utterances with words randomly selected from the tongue twister *"Peter Piper picked a peck of pickled peppers. Where's the peck of pickled peppers Peter Piper picked?"* (without punctuation). A 12-word lexicon is used. For the report, we have created a train/valid/test split of 70%/5%/25%. The train utterances were used for the purpose of learning language models (unigram & bigram). The validation set is used to find the best settings of various parameters in experiments. Finally, we used the test split to report the results. This is done to prevent any data leakage and report results on completely unseen data.

B. Metrics

The following metrics are used to evaluate the accuracy and efficiency of the ASR system:

- **Word Error Rate (WER):** The percentage of words that were incorrectly predicted. The lower the value, the better the performance of the ASR system with a WER of 0 being a perfect score. Word error rate can then be computed as:

$$\text{WER} = \frac{S + D + I}{N} \times 100$$

  Where:

  1) S is the number of substitutions
  2) D is the number of deletions
  3) I is the number of insertions
  4) N is the total number of words across all utterances (transcript)

- **Number of forward computation**: The cumulative sum of all *forward computation* steps taken during the Viterbi decoding, that is, every time we compute the likelihood along an arc in the WFST. It is closely related to the *decoding speed* which is reported as backtrace, decoding and total runtime in seconds.

- **Number of states and arcs:** The number of states and arcs in WFST model. It serves as a proxy for the memory size of the model. A larger lexicon should have more states and arcs, hence, the *memory* required is expected to increase proportionally to the number of states and arcs.

C. Baseline

In the baseline system, we use a 3-state WFST for each phone in the word. We create a linear phone WFST for all the words in lexicon. All the intermediate transitions output a ϵ symbol, except the final emitting arc from the last phone state of the word, which outputs the recognized *word*. The start state has a uniform probability of going to any word. All the states have a probability of 0.1 for staying in the same state and a probability of 0.9 to transition to next state. The final probabilities are set using a uniform distribution, indicating the probability of an utterance on any word. The final state is connected to start state via and ϵ-arc. Figure 1 illustrates the baseline WFST.

A pre-trained neural model is provided which provides the observation probability of a phone at any state in HMM, whereas the state transition probabilities are taken from WFST self-loop and transition values. We then perform Viterbi decoding which calculates the posterior probability of getting the most probable word sequence given an utterance; this is proportional to the value of the last acoustic frame. After running the Viterbi decoder over the test data, we obtain the results of the baseline model. Table II presents the evaluation accuracy and efficiency results of the baseline model.

The baseline performs poorly and as it can be seen from Table II that the WER achieved is 144.81% with 7 deletion, 740 insertion, and 216 substitution errors. We have a total of 665 words in the test data. Looking at an example utterance: *peter piper picked a peck of pickled peppers*, the ASR system predicts: *the picked of a picked picked peck of pickled of where's the*. It can be clearly

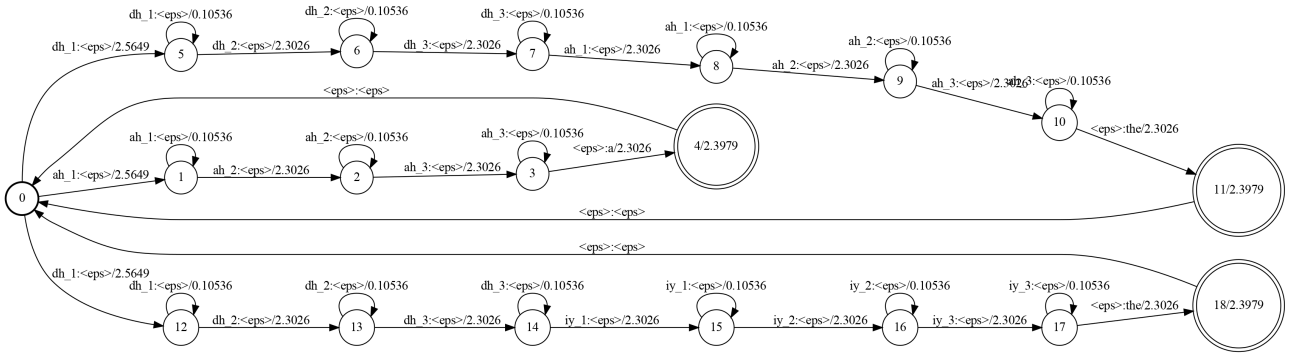

Fig. 1: Baseline WFST with uniform final probabilities

seen that the model is over-predicting, which can be attributed to the model predicting even when we have a silent acoustic frame. This suggests that adding a WFST for silence to the baseline model should result in a much lower WER.

II. Improving recognition (WER) of ASR system

The baseline system could be further improved by exploring the effect of various aspects of WFST. Four changes are further explored which are described in this section. We explored self-loop probabilities, and the final probabilities of the word WFST, adding WFST for silence and transitioning to the next word using unigram word probabilities.

A. Effects of the self-loop probabilities and system tuning

In this section, we observe the effect of self-loop probabilities and next-probability on WER of the system. We will also try to find a good setting of self-probability and next-probability after tuning the system with a grid search on self-loop probability and next-probability.

In subsection I-C, we observed that the model is over-predicting. One way to prevent the model from emitting more words is by increasing the possibility of staying in a single state. It could also be helpful in scenarios where a word in an utterance is spoken slowly (i.e. for a longer duration), or have a longer duration on the middle syllables such as "reoccurring", hence it will be able to recognize such words better. To explore this, we perform experiments by varying the self-loop probabilities and next probabilities in a range of [0.1-0.9] with 0.2 increments. At the end of this experiment we also tune the baseline system to find the best setting of self-loop probability and next probability. The result of `Tuned Baseline` is reported in Table II.

Hypothesis: We hypothesize that by keeping the next probability the same and increasing the self-probability, the WER will decrease.

Result: Table I shows the effect of WER when we increase the self-loop probabilities in the baseline model settings. The result from the best setting of the experiment is then evaluated on the test set and is reported under `Baseline + Self-loop` section in Table II.

Conclusion: The results presented in Table I show that the WER keeps improving as we increase the probability of self-loop. We believe that this improvement in WER is because higher self-loop probability is able to model the silence frames corresponding to pauses in the utterances. In Table II under `Baseline + Self-loop` section, we observe a decrease in insertion error by 55%, which further supports our hypothesis.

Self-loop Probability	Next Probability	WER.
0.10	0.9	118.97%
0.30	0.9	83.33%
0.50	0.9	78.16%
0.70	0.9	75.29%
0.90	0.9	72.99%
0.10	0.1	118.97%
0.30	0.1	70.69%
0.50	0.1	68.39%
0.70	0.1	67.82%
0.90	**0.1**	**67.24%**

TABLE I: Effect of self-loop probability on WER keeping next-probability constant in the baseline model.The WER is evaluated on Validation set. This not an exhaustive table, but other settings follow a similar trend.

B. Effects of the final probabilities

Hypothesis: The final probability of a state is the probability of ending in that state. We calculate the probability of a word ending as the final word of utterances in the training set. After observing the probability of all the words, we find that the distribution of any word ending up in the final position in given utterances is nearly uniform. Hence we hypothesize that if we set the final probability as calculated from the data or set them uniformly, it will not have any significant effect on WER on the test set.

Result: The results from the experiments are presented in Table II under section `Tuned Baseline + Learned Final Prob.` When compared with `Tuned Baseline` we observe a 1% increase in WER when the final probability is learnt from training data.

Conclusion: As hypothesised, we see that the difference in WER is not significant as the learnt distribution of final probabilities is close to uniform. Hence, we choose to use the uniform final probabilities to account for the randomness in data in all the consecutive experiments.

	Self-loop/Next Probability	Runtime in seconds (Backtrace/Decoding/Total)	Forward Computations	Number of States/Arcs	S/D/I Errors	WER
Baseline	0.1/0.9	0.046/319/432	10829163	127/252	216/7/740	144.81%
Improving Recognition Accuracy						
Baseline + Self-loop	0.9/0.9	0.047/316/428	10822893	127/252	219/16/327	84.51%
Tuned Baseline	0.9/0.1	0.056/321/434	10820865	127/252	213/37/208	68.87%
Tuned Baseline + Learned Final Prob.	0.9/0.1	0.038/318/428	10820247	127/252	210/29/224	69.62%
Tuned Baseline + Silence WFST	0.9/0.1	0.047/344/458	11481537	133/268	194/41/80	47.37%
Varying Grammar						
Unigram	0.9/0.1	0.055/323/437	10821298	127/252	208/30/226	69.77%
Bigram	0.9/0.1	0.049/469/583	16530192	127/384	191/41/229	69.32%
Unigram + Silence (UniG-Sil)	0.9/0.1	0.054/341/452	11481918	133/268	195/41/80	47.52%
Bigram + Silence (BiG-Sil)	0.9/0.1	0.141/2989/3159	112096287	991/2688	193/31/173	59.70%
Improving Computational Efficiency						
UniG-Sil + Tree Struct Lexicon	0.9/0.1	0.078/263/377	8801576	103/208	195/41/80	47.52%
UniG-Sil + Beam Search (0.01) (**A**)	0.9/0.1	0.032/78/190	1428773	133/268	316/51/138	75.94%
(**A**) + Tree Struct Lexicon	0.9/0.1	0.040/40/152	673571	104/209	321/61/113	74.44%
BiG-Sil + LM Pruning (0.01) (**C**)	0.9/0.1	0.084/2012/2159	74606412	673/1787	193/55/168	62.56%
(**C**) + Tree Struct Lexicon	0.9/0.1	0.084/1386/1518	51701210	462/1198	193/55/168	62.56%
(**C**) + Weight pushing + Beam Search	0.9/0.1	0.057/69/193	911357	462/1198	267/181/90	80.90%

TABLE II: Summary of results on test-set.

C. Adding a silence state in WFST

Hypothesis: The baseline model suffers from the issue of over-generation of words. As concluded in Section 1, this could be the result of predicting words when there is silence in the acoustic frame. Figure 2 illustrates the silence model added to WFST used for experiments. Here silence WFST has 5 states, here, states 2-4 has an ergodic structure to recognize the silence of various length. We hypothesize that adding a silence state in WFST will help model the silent acoustic frames in the audio, which will lead to lower word insertion errors and thereby effectively reducing the WER.

Result: The result in Table II under the section `Tuned Baseline + Silence WFST` shows a significant decrease of 31.21% when compared with `Tuned Baseline`, keeping all the other settings same. We also see a decrease of 60% in insertion errors, with no visible change in deletion and substitution errors. We also note that there is a 6% increase in computational steps with an increase in 6 states and 16 arcs.

Conclusion: The results supports our hypothesis and we observe a significant improvement in accuracy with a 6% loss in computational efficiency. Overall, this is a good trade-off. Further improvement can be done by experimenting with different topologies for silence WFST to model various duration of pauses.

D. Exploring unigram and bigram grammar on the system performance

The baseline model considers a uniform probability of transitioning to the next word, however, this information could better be modelled by a language model. In this experiment, we calculate the unigram and bigram grammar probabilities from the training data. This is done to avoid any information leakage from our evaluation datasets through the uniform grammar probabilities. Since the *sil* word is not observed in the lexicon, we used **Good Turing Smoothing** to redistribute the initial probabilities and assign a probability mass to the unobserved *sil* word.

To implement a WFST with unigram model, we add unigram probabilities on the transition arcs from the start word to all the linear phone WFSTs for all the words in the lexicon. Individual word WFSTs have a transition arc from their final states to the start state. Unigram WFST is similar to baseline WFST from Figure 1 with the difference of unigram probabilities on outgoing arcs from the start state instead of uniform probabilities. Silence WFST mentioned in subsection II-C is used to model pauses. Figure 2 illustrates a simple unigram WFST with silence.

The WFST for bigram has an arc going from start state to each word WFST with weight of $(start, word_i)$, where the pair denotes bigram probability of starting from $word_i$. The transition arc connecting $(word_i, word_{i+1})$ connects the end state of $word_i$ to start state of $word_{i+1}$, to model a transition between bigram pair. A bigram WFST of two words is shown in Figure 3. To model silence between bigrams we insert a silence WFST from subsection II-C between the bigram pair by adding a parallel arc from $word_i$ to *sil* and from *sil* to $word_{i+1}$. This ensures that after modelling any silence frames between bigrams we do not transition to any other word other than the target bigram pair. We assign the arc weights in a way that the total weight of arcs outgoing from $word_i$ to $word_{i+1}$ is equal to their bigram probability. This is illustrated in Figure 4, note that arcs from state 7 to 15 (word **a** to **sil**) and 7 to 8 (word **a** to **the**) sum up to bigram probability of *(a, the)*. We note that this method increases the number of states and arcs by an order of magnitude, and better alternatives might exist. In section III we will explore ways of improving the efficiency of our bigram model further.

Hypothesis: We find that the distribution of unigram probabilities obtained from the training set is very close

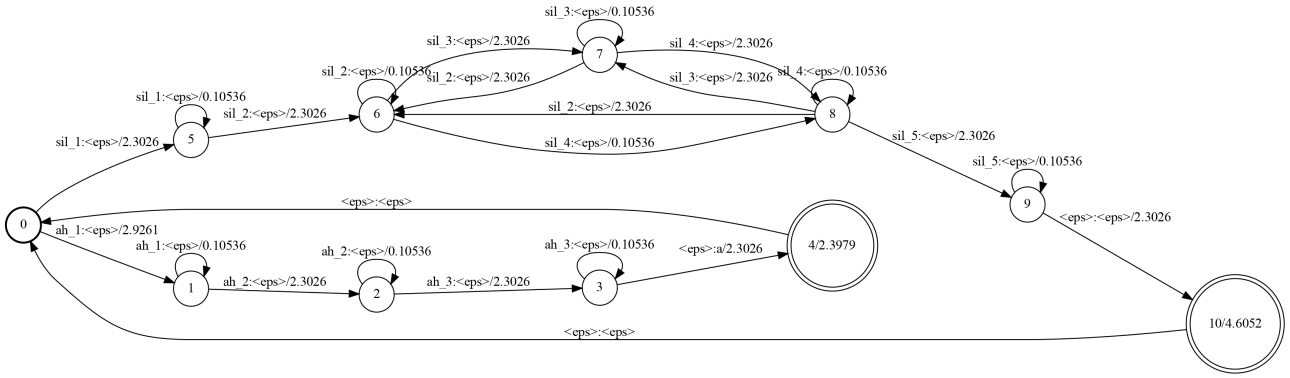

Fig. 2: Unigram WFST with silence WFST. State 6,7,8 have ergodic connection. Outgoing arcs from state 0 have different unigram probabilities.

to the uniform probabilities distribution of the next word, hence, we do not expect any significant change in WER in comparison to the baseline model. The reason for unigram grammar probabilities to be close to the uniform probability can be attributed to a random selection of words to create utterances in the given corpus.

Bigram grammar might not be always effective and is likely to perform poorly except for a few phrases such as "peter piper", "pickled pepper" etc. where a clear pattern is observed in the corpus. The training data also assigns very low probability to unlikely words such as "of pepper", words ending with "the/a/of" etc, however, it is observed that in the validation cases, such words indeed exists and bigram grammar always makes mistakes in such scenarios.

Also, due to the way silence is modelled between grammars, we are likely to see more deletion errors in cases of words with short durations such as "a", "of" etc. We believe that while speaking phrases like "a peck" , "peck of" there are usually no pauses, however, our WFST may lead to paths with silences in-between the bigram pair.

Result: The `Varying Grammar` section in Table II shows the results of Unigram grammar with and without silence WFST. We observe that the results are very similar to `Tuned Baseline` and `Tuned Baseline + Silence WFST`, which have uniform next-word transition probability. The results for bigram are reported under `Bigram` and `BiG-Sil` in Table II. For bigram grammar, without silence, we see only a very small improvement in substitution errors, which could be attributed to the prediction of frequently occurring two-word phrases correctly.

Conclusion: The randomness in the word sequences in utterances better fits a uniform model instead of a unigram/bigram model. Given that our experiment setup has random sentences, language modelling is unlikely to improve accuracy. However, in large vocabulary systems language with natural language, language grammar significantly improves the accuracy of ASR system.

III. Improving computational efficiency of ASR system

In this section, we try to analyze the efficiency of the Viterbi decoder and implement a form of pruning to avoid computations along unlikely paths. We will explore various methods to make the WFST efficient by decreasing the size of WFST and improving the decoding efficiency.

A. Standard beam pruning

In standard beam pruning we track the best state at any time t, and we prune the paths with a probability lower than beam width times the probability of the most likely path at time step t [Ortmanns et al., 1996a].

Hypothesis: A higher beam width will result in more paths getting pruned, which should result in fewer forward computations, hence, less overall computational time. However, it is a possibility that the best recognition sequence path has a probability lower than beam width at any time t, and may get pruned resulting in lower WER.

Result: Table III presents the results of running standard beam search on linear lexicon WFST with Unigram grammar. The final results on test data are reported under heading `UniG-Sil + Beam Search (0.01) (A)` in Table II.

Conclusion: The increasing percentage decrease in forward computations and increase in WER supports our hypothesis that more paths are pruned as beam width increases however, optimal paths may get pruned which results in lower accuracy.

B. Language model based pruning

In subsection II-D the WFST with bigram language model was an order magnitude more states and arcs. However, we may note that not all the bigrams have high probability. Such bigrams exists either because they were not observed in data and were added during language model smoothing. It is possible to reduce the overall size of WFST by adding a pruning threshold based on bigram probability i.e we will not add arcs if they they are below this threshold. Since we also add individual silence WFST for each bigram, this will also reduce the number of states resulting in a smaller WFST. This is similar to Language model pruning in [Ortmanns et al., 1996a].

Hypothesis: Pruning arcs with low bigram probability in Bigram based WFST should lead to slight increase in WER but considerable reduction in forward computations during Viterbi decoding. Also, we will get a smaller overall WFST.

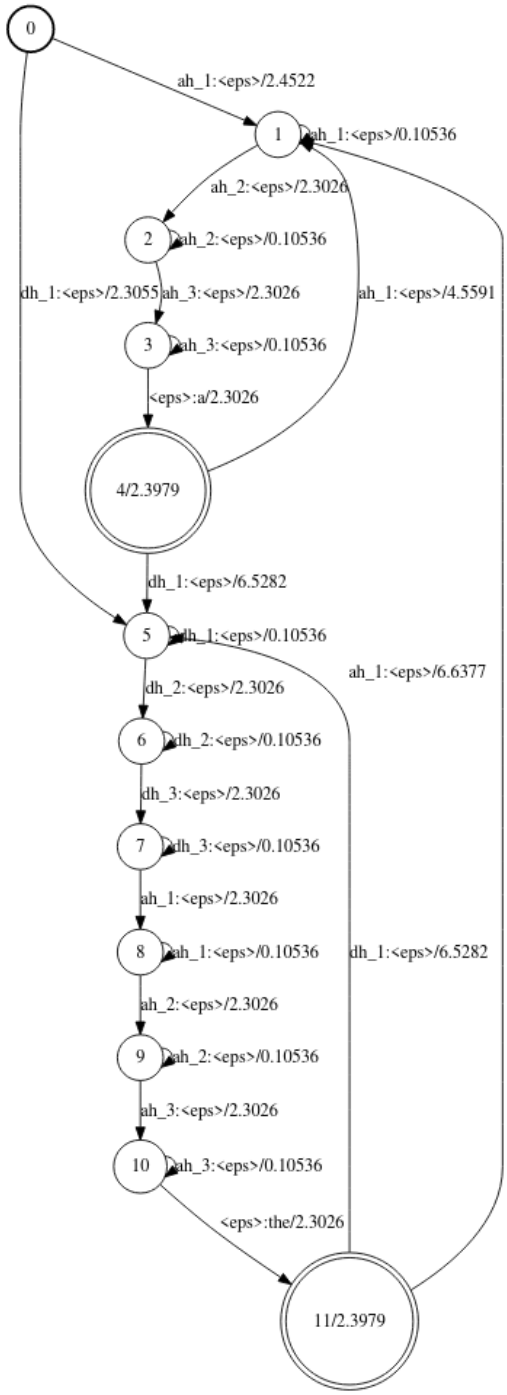

Fig. 3: Bigram WFST without silence

Result: Language model pruning results on test set are presented under heading `BiG-Sil + LM Pruning (0.01) (C)` in Table II,where 0.01 is the language model probability threshold and any arcs with lower probability will be pruned and won't exist in resulting WFST, this is illustrated in Figure 4, where there is no bigram from the word ***the***.

Conclusion: We see a considerable reduction of 318 states and 901 arcs from the Bigram WFST with silence. Also, a 33% increase in decoding speed is achieved with a loss of only 4.7% decrease in accuracy. This demonstrates pruning states and arcs with low probability could make

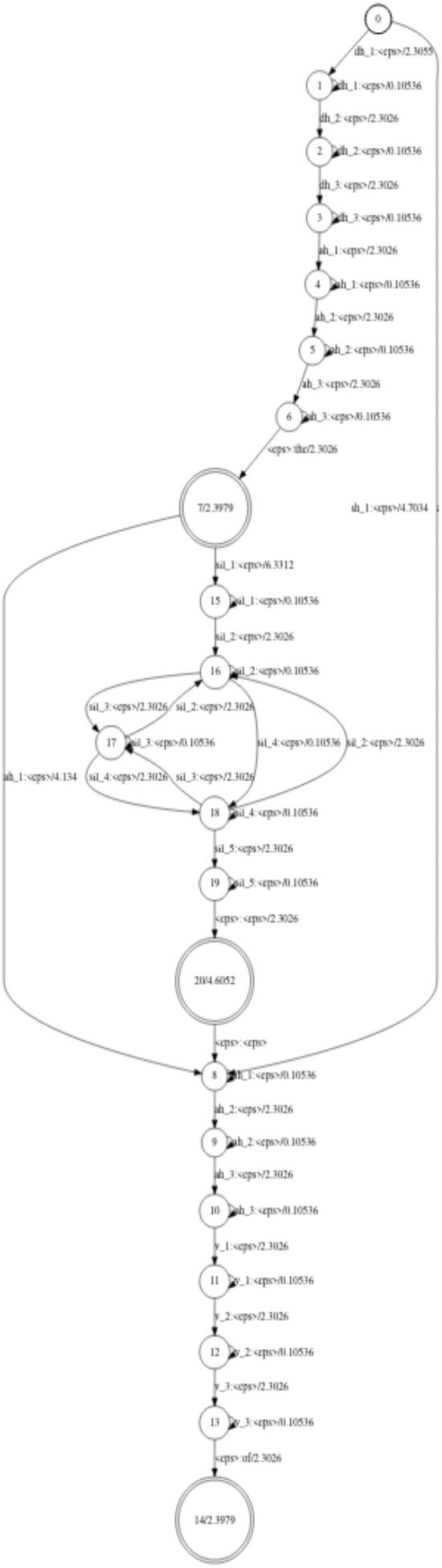

Fig. 4: Bigram WFST with pruned arc from **the** to **a**

the resulting WFST computationally efficient.

C. Tree structure lexicon

In linear lexicon we do not take advantage of shared prefix in our WFST. To take advantage of the shared prefix we convert our linear lexicon based WFST to tree structure lexicon. We implement this with the help of python-openFst's [Riley et al., 2009] $determinize()$ function. Determinization eliminates multiple paths to a sequence, this leads to a single path for each input sequence. This reduces the overall number of states and arcs, which results in lower computational memory and the number of forward steps taken during Viterbi decoding. Figure 5 shows the tree structure WFST equivalent of baseline WFST from Figure 1.

Hypothesis: Since determinization only eliminates multiple paths, we should not see any change in WER. However, since we have fewer states and arcs, we should get an increase in computational efficiency.

Result: Results are available under heading `UniG-Sil + Tree Struct Lexicon` in Table II. We can observe that the WER remained unchanged when compared with `Unigram + Silence (UniG-Sil)`, whereas we observe a decrease of 23% in forward computations. We also see similar results in `(C) + Tree Struct Lexicon`, where we converted a Bigram WFST to a tree structure lexicon.

Conclusion Based on the results we can safely conclude that tree structure lexicon is a very effective strategy to increase the computational efficiency of our ASR system without compromising on the recognition accuracy.

	% Decrease in forward steps		% Increase WER	
Beam Width	**Linear Lexicon**	**Tree Structure Lexicon**	**Linear Lexicon**	**Tree Structure Lexicon**
0.01	87.67	**91.98**	62.19	**45.11**
0.02	89.18	92.75	62.19	51.20
0.03	90.19	93.33	63.40	58.52
0.04	90.91	93.72	70.72	63.40
0.05	91.38	93.99	70.72	64.63
0.06	91.69	94.38	67.07	65.84
0.07	91.98	94.64	63.40	70.72
0.08	92.39	94.84	65.84	70.72
0.09	92.58	94.97	64.63	69.51

TABLE III: Comparison of Beam Search Pruning in Linear v/s Tree Structure Lexicon with Unigram grammar on Validation set

D. Beam search on tree structure lexicon WFST

Hypothesis: In subsection III-C we saw that tree structure lexicon is a very effective way of improving the computational efficiency of ASR. Here, we hypothesize that applying beam search pruning from subsection III-A on tree structure lexicon based WFST is more effective way of improving decoding speed than linear lexicon-based WFST.

Result: Table III shows the comparison of the percentage decrease in forward steps and percentage increase in WER for Unigram WFST. We see that we get a 4% more decrease in forward steps with less loss of accuracy when we apply beam search on tree structure lexicon. Also, from the test results under heading `(A) + Tree Struct Lexicon` in Table Table II we see that this method achieves slightly better WER with 52% further reduction in forward computations when compared with `UniG-Sil + Beam Search (0.01) (A)`.

Conclusion: Converting a linear lexicon-based WFST to a tree structure lexicon WFST is a better strategy when applying beam search pruning and results in significant improvement in computational efficiency.

E. Beam Search with weight pushing - a look-ahead strategy

In subsection III-A we saw that beam search is pruning the paths with probability lower than beam width probability of the current best path. We can accelerate the decoding speed further if we can know whether the path will result in a likely candidate at early time steps. We can do this by using $push()$ function in python-openfst [Riley et al., 2009] which pushes the weight towards the start state. This makes pruning in Viterbi decoding more efficient as the initial states will have most of the weight and we can prune the unlikely path in early time steps. Figure 5 illustrates weight pushed towards the start state in the tree structure lexicon WFST.

Hypothesis: Combining beam search for a tree structure lexicon-based WFST with pushed weights towards start state can significantly improve decoding performance. If the path to a prefix head has a lower probability than the beam width probability, we can skip all the paths through the prefix head. This will result in a faster decoding step. This is similar to the language model look-ahead method in [Ortmanns et al., 1996b] where we would like to incorporate language model probabilities as early as possible.

Result: The results of applying beam search on tree structure lexicon with pushed weights are shown in Table II shows the result on test set under heading `(C) + Weight pushing + Beam Search`. We observe a 100% reduction forward steps with a 30% decrease in WER in comparison to `(C) + Tree Struct Lexicon .`

Conclusion: Weight pushing in tree structure lexicon-based WFST provided a very effective way of improving decoding speed. However, we see a significant drop in recognition accuracy, which can be attributed to most of the paths getting pruned, which is evident from very high deletion errors of 181. The $push()$ method in python-openfst pushes most of the weight to the start state which may result in the path getting pruned at very early time steps. To improve this we need to push the weight to the common-prefix nodes to avoid very early pruning of candidate paths. The gain in decoding speed comes at a cost of poor accuracy and a better beam width needs to be explored on validation data before applying this strategy.

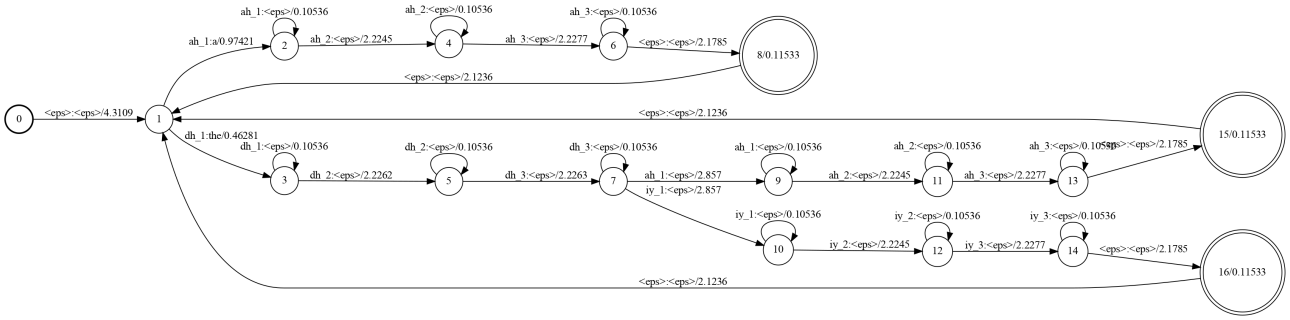

Fig. 5: Tree structure lexicon based Unigram WFST with weight pushed towards start state. WFST shows a common phone ***dh*** (state 7) is shared between the two pronunciations of word ***the***

IV. Discussion & Conclusion

In this report, we explored various methods to improve the computational efficiency and recognition accuracy of ASR systems. Overall, we find certain settings which are crucial to recognition accuracy and resulted in improvement of recognition accuracy. We find that modeling silence significantly improves recognition accuracy in all the settings and prevents over-prediction over silent frames. Self-loop probability also has a similar effect on over-prediction and modeling words with longer duration. We further explored unigram and bigram grammar to improve recognition accuracy and found that if the utterances consist of random word sequences, we do not get a significant improvement in accuracy and they might be more effective in cases where utterances consist of natural language.

We also explored various methods for improving computational accuracy. We found that pruning of WFST could be achieved by dropping arcs based on low bigram probabilities and converting the WFST to a tree structure with shared prefixes. Additionally, we found that the size of WFST is directly correlated with the decoding speed. We also found that beam search applied on tree structure WFST provides a better trade-off between decoding speed and loss in accuracy. Also, we should be cautious while using beam search over tree based WFST as we may get a superficial improvement in decoding speed while no word sequence is recognized at the end of Viterbi decoding.

In our toy experimentation setting, we set the weights of the self-loop and next transitions by running a grid search, which is not an efficient technique. In practice, we find all the arc weights by the Expectation-Maximization algorithm [Dempster et al., 1977] on training data, which iteratively finds the optimum arc weights. The ability to find arc weights specific to phone type is crucial to model acoustic properties such as silence, phone duration, lexical stress etc. A system failing to achieve this may perform poorly in real-world settings. A rule of thumb while optimizing ASR performance is to be aware of the context in which the speech recognition system will be applied.

References

[Dempster et al., 1977] Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38.

[Ortmanns et al., 1996a] Ortmanns, S., Ney, H., and Eiden, A. (1996a). Language-model look-ahead for large vocabulary speech recognition. In *Proceeding of Fourth International Conference on Spoken Language Processing. ICSLP '96*, volume 4, pages 2095–2098 vol.4.

[Ortmanns et al., 1996b] Ortmanns, S., Ney, H., Eiden, A., and Coenen-Lehrstuhl, N. (1996b). Look-ahead techniques for improved beam.

[Riley et al., 2009] Riley, M., Allauzen, C., and Jansche, M. (2009). OpenFst: An open-source, weighted finite-state transducer library and its applications to speech and language. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Tutorial Abstracts*, pages 9–10, Boulder, Colorado. Association for Computational Linguistics.