Automatic Speech Recognition Report

(Kyriakos Kyriakou s2281922, Gregoris Georgiou s1950427)

1. Introduction

Humans use various methods to communicate with each other, such as speaking, making hand gestures, showing facial expressions, and more. However, speech is regarded as the most crucial form of communication (Benkerzaz et al., 2019). Automatic Speech Recognition (ASR) is an important technology that allows machines to recognize and interpret human speech. In recent years, Automatic Speech Recognition systems have become crucial in many applications around the world (Junqua & Haton, 2012; Li et al., 2015). Some of these include personal assistants, speech segmentation and labeling (transcription), and voice-controlled devices (Benkerzaz et al., 2019; Karpagavalli & Chandra, 2016).

In this paper, we present the results of an ASR study conducted on a small collection of recordings from ASR students on this year's (2023) course (Bell, 15 February 2023). The recordings consist of short utterances with words randomly selected from the famous tongue twister "Peter Piper picked a peck of pickled peppers. Where's the peck of pickled peppers Peter Piper picked?" (Bell, 15 February 2023).

The goal of this study is to evaluate the performance of the ASR system in several ways, including accuracy, speed and memory efficiency. In our experiments, we use the standard Word Error Rate (WER) metric to measure the accuracy of our output. Furthermore, we measure the time taken running the decoder of our system. Along that, we count the number of forward computations required to perform the decoding. Finally, we provide counts of the number of states and arcs in the Weighted Finite State Transducer (WFST) to compute a measure of the memory required for our decoder (Bell, 15 February 2023).

Our study is organized into four sections. In Section 2, we perform baseline experiments using a WFST designed to recognize any sequence of words from the vocabulary. We analyze and comment our experiments according to WER, and other measures mentioned above. In Section 3 we tune the WFST of our ASR system in order to improve our results. We experiment with varying techniques, such as changing the self-loop and final probabilities of the WFST. Moreover, we explore how transitions to each word based on unigram word probabilities can improve the system. Finally, we investigate the effect of adding a silence state between words in our WFST to handle pauses between words in the recording and thus create a more accurate and robust system. In Section 4, we implement Beam Search pruning into our Viterbi algorithm to avoid unnecessary computations along unlikely paths (Abdou & Scordilis, 2004). Finally, in Section 5, we investigate methods to further enhance our system. We experiment with improving the efficiency by employing a tree-structure lexicon and tree minimization. Lastly, we study how effective bigram model can be in building an ASR system.

2. Initial Systems - Task 1

The experiment setup in our study employs a 12-word (10 unique) lexicon from the tongue twister "Peter Piper picked a peck of pickled peppers. Where's the peck of pickled peppers Peter Piper picked?" (Bell, 15 February 2023). In our experiments, we used around 350 recorded utterances. Our WFST construction to represent the phones in the words is facilitated by the software OpenFST (Allauzen et al., 2007). In order to evaluate the system's performance and robustness, we have employed a data frame to store various details and metrics for the different transcriptions in each experiment. Specifically, we capture the following:

- **Transcription:** Refers to the original audio that is being transcribed.
- **Output:** Refers to the text generated by the ASR system after processing.
- **Substitutions:** Refers to the number of words in the output text that were substituted with incorrect words compared to the original transcription.
- **Deletions:** Refers to the number of words in the original transcription that has been deleted in the output.
- **Insertions:** Refers to the number of extra words added in the output text that were not present in the original transcription.
- **Errors Count:** Refers to the total number of error in the output, including substitutions, deletions, and insertions.
- **Words Count:** Refers to the total number of words in the actual transcription.
- **Word Error Rate (WER):** Refers to the percentage of word insertion, substitution, and deletion errors to the total number of words (Park et al., 2008). It is given by the formula:

$$\text{WER} = \frac{\text{S} + \text{D} + \text{I}}{\text{TotalWords}} \tag{1}$$

Model	WER(%)	#Errors[S D I]	Run-time(s)	#Forward Computations	#States,#Arcs
Baseline	76.0	1851 [698 71 1082]	821	35,277,696	127,252
Final Probability:					
1e-6	75.9	1847 [698 69 1080]	850	35,277,696	127,252
1e-4	76.0	1849 [699 69 1081]	855	35,277,696	127,252
1e-2	76.0	1850 [697 71 1082]	860	35,277,696	127,252
Self-loop probabilities:					
0.01	440.4	10719 [659 4 10056]	817	35,277,696	127,252
0.2	93.3	2270 [696 44 1530]	808	35,277,696	127,252
0.4	79.4	1933 [712 54 1167]	816	35,277,696	127,252
0.6	73.5	1788 [689 85 1014]	822	35,277,696	127,252
0.8	69.8	1699 [674 113 912]	832	35,277,696	127,252
0.99	63.4	1544 [680 216 648]	828	35,277,696	127,252
Change of grammar:					
Uni-gram	75.2	1830 [683 73 1074]	811	35,277,696	127,252
Bi-gram	69.0	1679 [547 84 1049]	859	33,409,728	127,341
Silence:	48.6	1184 [677 128 379]	866	37,459,218	133,268
Best Combination:	45,8	1114 [603 353 158]	920	37,459,218	133,268
Beam Search Pruning:					
1e-6	53.3	1298 [721 327 250]	335	14,080,501	127,252
1e-5	56.3	1370 [763 349 258]	297	12,449,188	127,252
1e-4	58.3	1419 [824 357 238]	262	10,902,159	127,252
1e-3	63.4	1533 [899 380 265]	219	8,973,358	127,252
1e-2	75.0	1826 [879 725 222]	155	6,189,628	127,252
Tree-structure Lexicon:					
Tree-structure Lexicon	76.0	1851 [698 71 1082]	621	25,936,584	97,192
With Minimization	76.0	1851 [698 71 1082]	561	24,409,518	81,170

Table 1. Results of all experiments.

- **Time for Decode:** Refers to the time taken by the ASR system to decode the input audio
- **Time for Backtrace:** Refers to the time taken by the ASR system to backtrack through the decoding process and generate the output text.
- **Total Time:** Refers to the total time taken by the system to process the input to generate the output text.
- **Forward Computations:** Refers to the total number of forward computations performed during the Viterbi decoding process, meaning every time the likelihood is computed along an arc in the WFST.
- **Memory:** Refers to the amount of memory used by the ASR system during decoding process. It's calculated by the number of states and arcs in the WFST.

2.1. Baseline System

We created our baseline system by constructing a WFST using a 12-word lexicon as shown on Figure 1. The start state of the baseline WFST has a uniform probability, $\frac{1}{12}$, of transitioning to each word. Also, each phone state, we set the self-loop probability to 0.5, thus the transitioning to the next phone state has probability of 0.5 (1 - self loop probability). Finally, for our baseline system, we decided to leave the default final state weights.

Table 1 displays the results obtained by our baseline system. Based on our baseline experiment, we observe a word error rate (WER) of 76%, coming from 1851 errors in 2434 words. Additionally, the time taken by the baseline system to decode the recordings was 821 seconds. By setting the self-loop and transition probabilities to 0.5 in our baseline system we get an acceptable level of performance. While this is not particularly high, it still suggests that there is a room for improvement in the system. We noticed that the obtained error rate is possible to be due of not giving different final weights to the final state for each word. Furthermore, we observed that in the output transcriptions, there are some silence words (mostly the word "the" and "of") appearing. This can be also noticed by the high number of insertion errors (1082, see table 1). An example of the mentioned is the following: original transcription: "pickled piper of peter" and output transcription: "the of pickled piper the of peter the". That is happening most probably due to the pauses at the start, in the middle, or at the end of the audio recordings, where the system incorrectly identifies as other words (silence words). In the following section, we experiment with different tuning techniques that could potentially improve the performance of the baseline system and mitigate the problem appearing from the pauses.

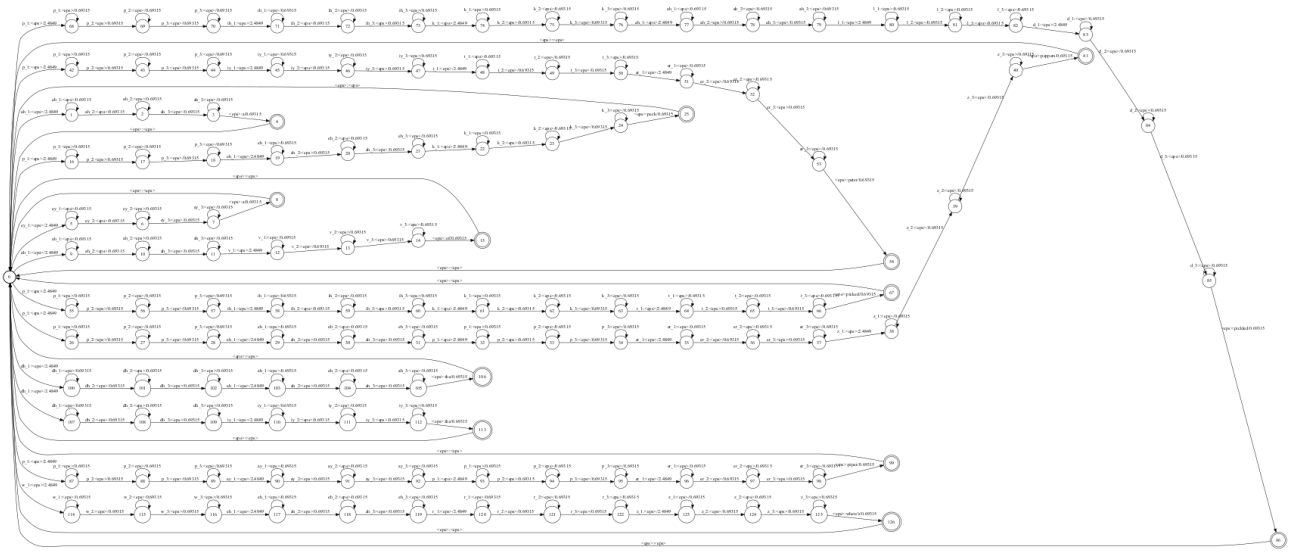

Figure 1. Baseline FST.

3. System Tuning - Task 2

In this section, we explore various tuning techniques that can be applied to improve the performance of our baseline ASR system. We begin by adjusting the parameters of our WFST by experimenting with varying self-loop probabilities, final probabilities, and using transitions to each word based on unigram word probabilities. Thereafter, we investigate how adding a silence state between words in our WFST to handle the pauses and noise in the recordings can effect the WER of our system. In the light of this, we apply the best profile from self-loop probability, and silence state, in a new system, in order to evaluate if the combination of those can lead to a more robust model.

3.1. Self-loop Probabilities

We explored the impact of varying self-loop probabilities on the performance of the system. We used self-loop probabilities of 0.01, 0.2, 0.4, 0.6, 0.8 and 0.99 (see Table 1) to evaluate our system. We try to investigate how self-loop parameters affect our system. By adjusting the self-loop probabilities, we can change the weight of repeated phones on the system in order to improve system's performance in handling repetitions or pauses in the utterances.

3.2. Discussion of Self-loop Probabilities Results

From Table 1 it can be observed that increasing the self-loop probability leads to a decrease in the WER, with the lowest WER achieved when using a self-loop probability of 0.99. The results show that using higher self-loop probabilities can help the system to handle repetitions and pauses in the utterances more effectively. This can be also observed by the number of insertions that decreases drastically as self-loop probabilities increase. When using a very small self-loop probability, the WER becomes not only worse that our baseline but incredibly bad with WER of 440.4%. However, it is worth noting that increasing the self-loop probability can also lead to a higher number of deletions as seen in $\#Erros[SDI]$ values for self-loop probabilities of 0.8 and 0.99. This suggest that as we assign higher probabilities, the most likely path may be less likely to include many phones from the recording (Pellegrini & Trancoso, 2011).

Our findings encourage the use of high self-loop probabilities as it is improving the performance of the ASR system. Nevertheless, assigning a very high self-loop probability might not always yield the best results, thus a careful tuning process involving multiple probabilities should be done.

3.3. Final Probabilities

A final probability is the probability of ending at a particular state in the FST. If this final probability is set in negative log-likelihood form, it is known as the final weight. All non final states have a final weight assigned equal to infinity, while all final states have a lower final weight. In figure 1, all final states are marked with a double circle like states 4,8,15 etc. During our experiments, we tried to change all final probabilities of all final states uniformly, which lead to no change in the accuracy of the decoder. Thus, we decided to change the final probabilities of selective final states, to reduce the chance that the Viterbi decoding ends at these particular states that has been assigned a higher final weight. All in all, it was decided to reduce the final probability, thus increase the final weights, only in the words that appear only once in the tongue twister, namely "a", "the" and "where's". To make the change have an impact to the accuracy, very low final probabilities were assigned to the selected words, more specifically $1e-6$, $1e-4$, $1e-2$.

3.4. Discussion of Final Probabilties Results

As one can clearly see from the results in table 1, altering the final probabilities of the selected words, made a very minor change in WER. To be precised, when compared

to the baseline performance, only final probability $1e-6$ decreased WER from 76.0% to 75.9%, which translates to 4 fewer errors, while $1e-4$ and $1e-2$ lead to 2 and 1 fewer errors respectively. Nevertheless, the almost meaningless impact to the accuracy of the decoder and the slight increase of the decoding run time from 820 to about 850 seconds that these changes in final probabilities introduced, forced us to not further experiment with final probability alterations.

3.5. Grammar-Unigram

As described in section 2.1, the probability of transitioning from state 0, the start state, to any word was uniform. By incorporating unigram probabilities to these transitions, we can create a more representative fst, and boost the performance of the decoder. Unigram probabilities for each word were calculated using all the recordings available. We iterated through all the recordings, counting the total occurrences of each word and then we divided by the total number of words in all the recordings. In this way, the unigram probability for each word was calculated. Furthermore, it has to be mentioned that we shouldn't leak information from training data, but since the utterances of words are randomly uniformly distributed, due to the fact that each student got his choice of words for each recording, this doesn't really affect our implementation.

3.6. Discussion of Unigram Grammar Results

The results of utilizing unigram grammar, are depicted in the "CHANGE OF GRAMMAR" section in table 1. When compared with the baseline model, incorporating unigram probabilities lead to decreasing the number of errors by 21, reflecting to a WER of 75.2 %, 0.8% less than the baseline. In light of this, it is verified that the introduction of unigram probabilities helps to improve the performance of the decoder.

3.7. Silence State

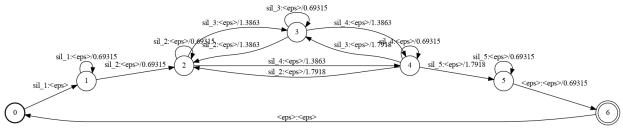

Figure 2. FST only for silence, which then was incorporated to Figure 1.

The problem of pauses before and after words in the recordings is clearly present with the use of the baseline model. For example, while the real transcript of the second recording is "where's peter", the baseline model predicts "the where's peter the", leading to high WER. It is obvious that this pauses are decoded by the Viterbi decoder as words like "the" or "of". With that said, to mitigate this problem an optional silence state was added in the WFST, which enables the Viterbi decoder to decode these pauses as silence, thus give back no output. Furthermore, the structure used to train the observation model for silence consists of five states arranged in a left-to-right topology, with the first and fifth states being the initial and final states, respectively and the second, third, and fourth states have an ergodic structure. This structure is depicted in figure 2. It was replicated and added to the baseline WFST portrayed in figure 2.1, introducing 6 new states and 16 new arcs, leading to a total of 133 states and 268 arcs. Moreover, the ergodic structure followed by the second, third and fourth state can help the model to generalize better and be able to classify variations of "silence" noises.

3.8. Discussion of Silence State Results

Results shown in table 1 suggest that the addition of the silence state had the biggest impact to the performance of the decoder. Even though the WFST now consists of more arcs and states, leading to more decoding run time and more forward computations, the improvement in the accuracy is exceptional. Compared to the baseline model, this model had about 700 less errors, leading to a WER of 48.6 %. The trade off between accuracy and decoding time is clearly valuable and the fact that the insertion errors dropped from 1082 to 379 clearly depict the extend of improvement. At last, the prediction given by the model with the silence state for the second recording is now "where's peter", which is exactly what the real transcript is.

3.9. Combined model

After exploring the effects of the silence state, self-loop probability, and final probability, we now introduce a model that incorporates two techniques previously discussed (silence and self-loop probability). We excluded the final probability technique, as it had almost no impact on improving performance. By combining these two tuning techniques, we aim to achieve an even better performance. We used the best performing profile for self-loop probability for our combined model (0.99). The results of this are discussed in the following section.

3.10. Discussion of Combined Model Results

The combined model that joins both a silence state and a high self-loop probability, achieved a WER of 45.8%, which is a significant improvement compared to the baseline (76%) and the silence model that achieved a WER of 48.6% (see Table 1). The use of silence state allows our system to handle the different pauses in the utterances, and the self-loop probability to handle repetitions. The combination of these techniques significantly improved our ASR system, as shown by the decreased WER and errors, therefore should be considered in other ASR systems.

4. Pruning - Task 3

In this section, we dive into the implementation of pruning in the Viterbi algorithm to avoid propagating computations along unlikely paths. Specifically, we have employed beam search pruning (beam width) (Abdou & Scordilis, 2004) and experimented with different pruning thresholds to de-

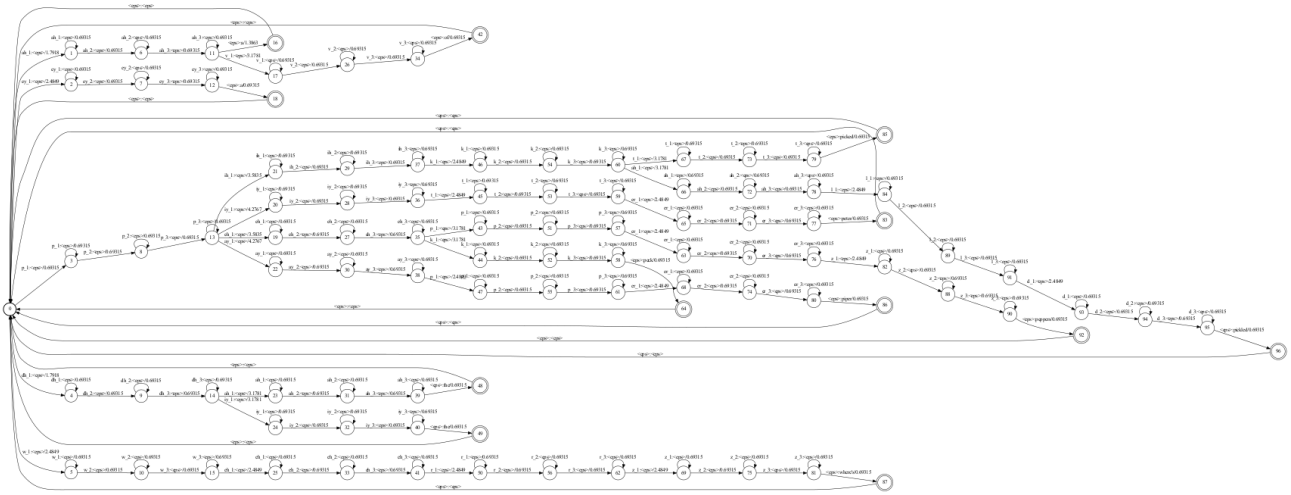

Figure 3. FST with tree-structure.

termine its effect on the accuracy and speed of our decoder. We implement pruning upon the best performing tuning profile achieved in Section 3. We use silence and self-loop probability of 0.99 as a base to improve with beam search pruning.

4.1. Beam Search Pruning

The basic idea of beam search pruning or beam width pruning (Ney & Mergel, 1992; Ortmanns et al., 1996) is to limit the number of active sequence of words that are selected during the decoding phase (Abdou & Scordilis, 2004). We select only the most probable sequence of words based on beam width. By pruning the unlikely sequences, we aim to increase the decoder's speed and reduce its computational cost (forward computations). However, there is a trade-off between the accuracy of the system (WER) and the beam width. The more the increase of the pruning (threshold), the more the accuracy drops as many sequences of words are discarded. In our experiments we evaluate several probability threshold values, namely 1e-6, 1e-5, 1e-4, 1e-3 and1e-2 (see table 1), and report our findings in the following section. Note that these thresholds are then translated using negative log-likelihood into beam widths to match the weight type of the WFST.

4.2. Discussion of Pruning Results

The results from Table 1 show the effect of applying beam search pruning with different pruning thresholds. As expected, increasing the pruning threshold (i.e. decreasing beam width) results in faster decoding speed with fewer forward computations, but at the cost of a higher WER. The WER only decreases slightly as the pruning threshold increases, indicating that the errors produced came from not including the correct sequence of words. Moreover the experiments show that pruning with lower threshold is more desirable as by using threshold value of 1e-6 decreases the forward computations to 14 million from 35 million, improves the run time speed by 2.4 times from the baseline system, while achieving a good WER of 53.3%. It is worth noting that even with relatively high pruning of 1e-2 threshold, the WER is still comparable with the baseline (75%). This suggests that beam search pruning can be an effective method for reducing the computational complexity of the ASR system, while still maintaining high accuracy. Finding the right balance between beam width and accuracy, can lead to an effective and robust system.

5. Advanced Topics - Task 4

The aim of this section is to explore more advanced techniques to improve the performance of our ASR system. Specifically, we investigate different methods for enhancing the system's efficiency by implementing a tree-structure lexicon and tree minimization. Additionally, we explore the effectiveness of improving the system's grammar by incorporating bigram.

5.1. Tree-Structure Lexicon

We explore the use of a tree-structure lexicon that could potentially improve the efficiency of the Viterbi decoder. To construct the tree-structure lexicon, we utilized the determinize() function from OpenFST library (Allauzen et al., 2007) based on Mohri et al. 2002, which creates an equivalent FST where each state is having only one transition per input label, thus eliminating multiple transitions (Mohri et al., 2002). The FST with tree-structure lexicon can be seen on Figure 3. In the Section 5.3 we discuss the findings.

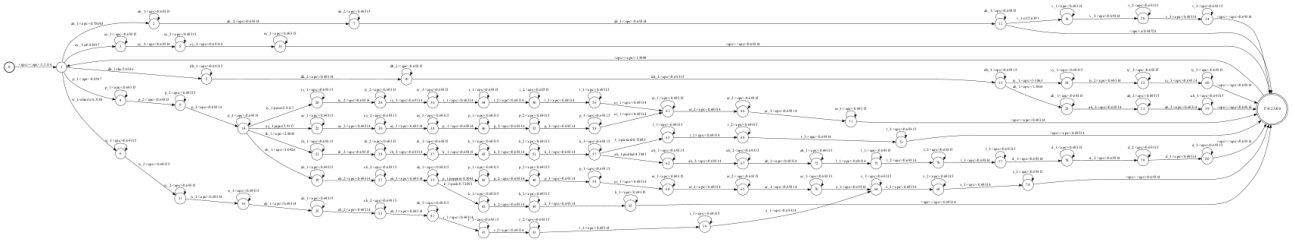

Figure 4. FST with tree-structure and minimization.

5.2. Tree Minimization

In addition to creating a tree-structure lexicon we experiment with minimizing our tree-structure FST. The minimize() function from OpenFST (Allauzen et al., 2007), is used to apply the minimization. Applying minimization makes the FST smaller by removing unnecessary redundant states and arcs while keeping its functionality the same. A more high-level definition is that it builds an equivalent transducer with the minimum number of states (see figure 4). We compare minimization's effectiveness compare to the the tree-structure lexicon and baseline in the following section.

5.3. Discussion of Tree-Structure Lexicon & Minimization Results

The results of our experiments with the tree-structure lexicon and minimization techniques show that there was a drastic reduction of run-time and number of forward computations compared to the baseline system (Table 1). More specifically, the run-time speed was reduced from 821s to 621s with tree-structure no minimization, and to 561 with minimization. Additionally, the forward computations were reduced by approximately 10 million for tree structure and 11 million with the minimized tree. Both of the methods showed to be more effective without any sacrifice on the word error count (WER). The efficiency improvement comes from the reduced number of states and arcs compared to the baseline as seen in Figures 3 and 4. The number of states have been reduced from 127 to 97 for tree-structure and to 81 with minimization. Furthermore, the number of arcs dropped from 252 to 192 and 170 when applied minimization. When comparing the results of the tree-structure with and without minimization, we can observe that the use of minimization did indeed improve more the efficiency without any drop in the accuracy. In light of this, the use of tree-structure lexicon and minimization showed effective changes in the efficiency of our decoder, something that makes it a worth-trying technique to explore.

5.4. Improving Grammar-Bigram

As discussed in section 3.5, grammar can pay a significant role for the decoder. In this section, bigram grammar is implemented in the WFST. The structure of the FST is exactly the same as the one used in section 3.5, but with some adjustments. The difference is that after the final state of each word, there is no arc returning to the initial state with probability 1. Instead, multiple arcs are drawn from each words final state to the first state of each word in the lexicon in order to take into account bigrams. Each bigram probability, similarly is calculated using all the recordings by counting the occurrences of each bigram in the dataset. In addition, the probabilities used to traverse from the start state to a word's path is the probability that this word is first in the bigram. Furthermore, the probabilities used to traverse an arc from the final state of the first word of the bigram to the first state of the second word, are calculated using conditional probabilities. For example, for the bigram "pickled peppers", the probability of the arc connecting the final state of the first word "pickled" to the first state of the second word "peppers" is calculated by:

$$\frac{prob(\text{bigram = "pickled peppers"})}{prob(\text{first word in bigram = "pickled"})}$$

5.5. Discussion of Bigram Grammar Results

As table 1 portrays, even though that the arc number in the WFST increased by 89, the number of forward computations decreased. In addition, the number of errors decreased by 172, translating to a drop of WER by 7.0% to 69.0%. By reviewing these results, we can confidently say that the improvement in grammar, boosted the accuracy of the decoder without having a great impact in the decoding speed.

6. Conclusions

Wrapping up this study, we learned how different techniques can lead to a more accurate and/or more efficient Viterbi Decoder. It was shown that higher self-loop probabilities assigned improve the accuracy of the decoder while any change on the final probabilities do not have any meaningful impact. Furthermore, special attention shall be given to introducing a silence state to mitigate the impact of random pauses in the recordings, since its improvement in the accuracy of the decoder is very meaningful and extremely important. Moreover, beam-search pruning technique can influence the decoding speed positively, without deteriorating the accuracy in large extend. Additionally, it was proven that improving the grammar used in the WFST using unigram or bigram probabilities have a positive impact to the accuracy of the decoder. At last, introducing a tree-structured lexicon and minimizing the WFST to remove redundant arcs and states can help improve the efficiency of the decoder without any impact to its accuracy.

References

Abdou, Sherif and Scordilis, Michael S. Beam search pruning in speech recognition using a posterior probability-based confidence measure. *Speech Communication*, 42(3):409–428, 2004. ISSN 0167-6393. doi: https://doi.org/10.1016/j.specom.2003.11.002. URL https://www.sciencedirect.com/science/article/pii/S0167639303001432.

Allauzen, Cyril, Riley, Michael, Schalkwyk, Johan, Skut, Wojciech, and Mohri, Mehryar. Openfst: A general and efficient weighted finite-state transducer library. In Holub, Jan and Žďárek, Jan (eds.), *Implementation and Application of Automata*, pp. 11–23, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-76336-9.

Bell, Peter. Automatic speech recognition 2022-23: Assignment, 15 February 2023.

Benkerzaz, Saliha, Elmir, Youssef, and Dennai, Abdeslam. A study on automatic speech recognition. *Journal of Information Technology Review*, 10(3):77–85, 2019.

Junqua, Jean-Claude and Haton, Jean-Paul. *Robustness in automatic speech recognition: fundamentals and applications*, volume 341. Springer Science & Business Media, 2012.

Karpagavalli, S and Chandra, Edy. A review on automatic speech recognition architecture and approaches. *International Journal of Signal Processing, Image Processing and Pattern Recognition*, 9(4):393–404, 2016.

Li, Jinyu, Deng, Li, Haeb-Umbach, Reinhold, and Gong, Yifan. Robust automatic speech recognition: a bridge to practical applications. 2015.

Mohri, Mehryar, Pereira, Fernando, and Riley, Michael. Weighted finite-state transducers in speech recognition. *Computer Speech Language*, 16(1):69–88, 2002. ISSN 0885-2308. doi: https://doi.org/10.1006/csla.2001.0184. URL https://www.sciencedirect.com/science/article/pii/S0885230801901846.

Ney, Hermann and Mergel, Dieter. Data driven search organization for continuous speech recognition. *IEEE Transactions on Signal Processing*, 40(2): 272 – 281, 1992. doi: 10.1109/78.124938. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-0026818496&doi=10.1109%2f78.124938&partnerID=40&md5=7603619cadf0362ae4edfe146e2ffdce. Cited by: 64.

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 vol.4, 1996. doi: 10.1109/ICSLP.1996.607215.

Park, Youngja, Patwardhan, Siddharth, Visweswariah, Karthik, and Gates, Stephen C. An empirical analysis of word error rate and keyword error rate. In *Proc. Interspeech 2008*, pp. 2070–2073, 2008. doi: 10.21437/Interspeech.2008-537.

Pellegrini, Thomas and Trancoso, Isabel. Error detection in broadcast news asr using markov chains. In *Human Language Technology. Challenges for Computer Science and Linguistics: 4th Language and Technology Conference, LTC 2009, Poznan, Poland, November 6-8, 2009, Revised Selected Papers 4*, pp. 59–69. Springer, 2011.