Gamesourcing Mismatched Transcription

Yu Yao Chang

December 3, 2015

ABSTRACT

Transcribed speech is an essential resource to develop speech technologies for different languages of the world. However, native speakers of most languages of the world may not be readily available online to acquire transcribed speech. The goal of this research is to explore the possibility of acquiring transcriptions for speech data from non-native speakers of a language, referred to as mismatched transcriptions. The two main problems tackled in this work are: 1) How do we motivate non-native speakers to provide transcriptions? 2) How do we refine the mismatched transcriptions? Firstly, we design a novel game that facilitates the collection of mismatched transcriptions from non-native speakers. In this game, players are prompted to listen to sound clips in a foreign language and asked to transcribe the sounds they hear to the best of their abilities using English text. The misperceptions by the non-native speakers are modeled as a finite memory process and implemented using finite state machines. The mismatched transcriptions are further refined using a series of finite-state operations.

The main contributions of this thesis are as follows: 1) Creation of a streamlined game for crowdsourcing transcriptions for speech data from non-native speakers. 2) Algorithms that process the resulting mismatched transcriptions and provide the closest sounding English words. 3) Experiments describing various modifications to the above-mentioned algorithms and results showing their effect on the accuracy of the English words that are produced as output.

CONTENTS

1 INTRODUCTION

Globalization has brought the world closer and has consequently encouraged the introduction of foreign languages in new lands. For popularly spoken languages, it is easier to find a native speaker compared to a language that is not commonly used. The goal of this research is to build labeled data for languages for which resources of native speakers are difficult to find.

Labeled data refers to the attachment of transcription to some audio data containing speech in a certain language. There could be multiple transcriptions for the same phrase or word in a language. Differences in perceived sounds by the transcriber cause variations in the resulting transcription. These differences are much smaller for native transcribers compared to non-native transcribers.

The resulting labeled data would be useful for building speech technologies for the language in question. Transcriptions are very important in building technologies such as speech recognition and even translators, and they are hard to acquire for languages whose native speakers are difficult to reach.

The approach used to collect the labeled data uses non-native speakers to generate the transcription, then attempts to recover the original transcription by using finite state models of cross-lingual perception. The phenomenon of having subjects transcribe audio in a foreign language is dubbed "Mismatched Transcriptions" (Jyothi and Hasegawa-Johnson 2015). In order to motivate transcribers and create a streamlined process for collecting labeled data, a game called "Soramimi Words" is created as a medium. The game interface allows for uniform interface and process to be presented to all subjects.

The process envisioned is as follows: a phrase from a foreign language would be played to a subject, and the subject would transcribe the phrase in English letters, to the best of their abilities. The subjects are to be told beforehand that their transcriptions would not have to make sense–what is important is that the subject captures the sounds that they heard, in any syllables or phonemes they feel is appropriate given their language backgrounds. The only requirement for the subject is that letters from the English alphabet would have to be used for the transcribing process.

A phrase in Hindi, for example, will sound different to a native speaker of Hindi and another who does not know the language at all.

One possible reason is that the subject would try to transcribe a sound clip from Hindi using existing knowledge of phonemes perhaps from a more familiar language.

This approach will not yield a perfect transcription; the probability of a transcription matching with the language's more recognized transcription is very low. One reason would be that the process relies on English letters to spell out sounds in a different language; and it is not necessarily true that a phoneme set for a language is a proper subset of the set of English phonemes. There is also the possibility that the subject is not giving a best effort attempt at transcribing; given the amount of data needed to build the labels and consequently the implied number of transcribers needed, the possibility of collecting sub-optimal transcriptions is very high.

An additional step is then needed to refine the subject's transcriptions. One possible solution would be to hire a human proctor that would "check" the proximity of the subject's transcription against what the proctor hears. However, this introduces an element of subjectivity into the process and could create more uncertainty in the output. The proctor may agree more with some transcriptions than others. It is also difficult to even determine if a transcription is genuinely poor or simply a drastically different interpretation of the perceived sound relative to the proctor's. Providing a form of "answer key" (strictly speaking, none exists as there is no "perfect" transcription) or guide for the proctor also deviates from the purpose of gathering interpretations from a wide audience.

An alternative approach would be to have subjects compare transcriptions and eliminate the need for a centralized human checker. The idea of a multi-subject pool works as follows: subjects would listen to an audio file in a foreign language and write down their best attempt transcription in English; then the subjects would be prompted to tweak their transcriptions and attempt to match each others' transcription inputs. Once the two transcriptions match, the refining process is complete. This approach faces bigger challenges in the earlier stages of launch due to the higher requirement for the number of subjects involved in refining the outputs.

The approach taken in this research automates the refining step. After the subject types in the transcription, the subject would then be exposed to the same audio files previously heard, but with the order shuffled and audio concatenated with one other audio file. The subject would then be asked to listen one more time to the newly generated audio files and type in the closest sounding English word. A computer would then take the subject's transcriptions for the respective audio files and generate its own set of English words to be compared to. This way, the transcription can be refined by measuring the proximity between the subject's word output and the computer's word output.

The algorithm is a finite state model that takes in transcriptions as inputs and outputs English words and is explained in more detail in Chapter 4. It should be noted that in order for the subject's transcription to be evaluated fairly, the computer's output itself would have to be reasonable. That is, against perfect transcriptions, the computer should output truly the closest sounding English word. This is evaluated by both quantitative measure of edit-distance between phonemes in the transcription and in the output word and subjective measure of human evaluation judging the correlation between the output word and audio. Similarly, with an ideal finite state model that outputs the closest sounding English word, the evaluation of the player should be fairly accurate. That is, the subject should be able to correctly identify which English word (as the word is generated through the subject's own transcription, therefore, it is truly the sound the subject has perceived) sounds the most similar to the audio—if the subject was attempting the transcription in good faith. The main challenge in creating the finite state model is that there is no universally acknowledged mapping from English letters to a set of phonemes. Frequently in English, the same sequence of letters produces several sets of phonemes and vice-versa.

Lastly, in addition to a streamlined procedure to collect labeled data, there is a need for participants. One of the important factors in recruiting participants is creating motivation. Many studies are carried out by awarding participants with small monetary compensation. It is ideal to minimize the amount of monetary incentive the research awards and maximize the number of subjects who provide good-effort transcriptions. For that, an alternative form of compensation is needed to reward participants. One main motivating factor that exists for humans is pleasure. Humans would naturally tend to do something pleasurable or enjoyable such as playing video games. The concept of embedding important studies in games is introduced by Von Ahn (2006), who illustrates how much time people spend in playing video games and how certain studies were conducted by creating games to conduct experiments. This research will take a similar approach and turn the above described process into a game, which will be described in more detail in Chapter 3.

The main contributions of this paper are follows:

- Establish a process as described above for crowd-sourcing labeled data.
- Create an algorithm to output the closest sounding English words based on an input transcription.
- Create a game that encompasses the crowd-sourcing process and makes use of the algorithm developed.
- Create several methods of mapping phonemes to letters.

- Conduct experiments with the algorithm to see which phoneme-to-letter mappings would yield better outputs.

Chapter 2 will introduce prerequisite knowledge to understanding the discussion in the paper.

2

PREREQUISITE KNOWLEDGE

This paper assumes certain basic working knowledge of concepts in algorithmic computing. This chapter will clarify and explain these concepts.

2.1 FINITE STATE TRANSDUCERS

Finite state transducers (FSTs) are variations of finite state machines (FSMs). FSTs are widely used in the field of speech recognition and are used in this paper to identify English words based on transcription of foreign sounds (Mohri et al. 2002). FSMs are abstract machines that have states with transition arcs between the states and are often useful in modeling dynamic systems. All FSMs have a start state and an end state. When an end state is reached, no more transitions can be made. The transition arcs are directed and labeled with input to the state that prompted the transition between states.

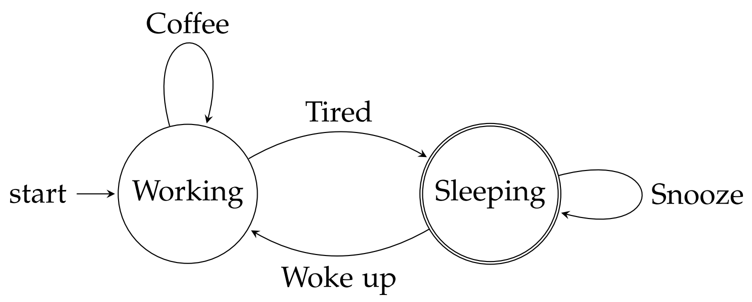


Figure 2.1: Simple example of a finite state machine modeling life of a college student.

While Finite State Acceptors (FSA) only takes in inputs to prompt state changes, FST have outputs on their transition arcs and weights for each transition arc. That is, now for a given input, the state machine will produce an output. The weight of each transition introduces the concept of the shortest path, which is the path with the lowest total cost from the start state to some end state. There could be multiple end states in a FST. There is also the concept of n-shortest paths, which would then list the n-paths with the lowest path costs in the FST. This concept of shortest path would be useful later on to minimize the edit distance from the input to the output. For example, suppose the input to a FST is a sequence of letters, and the desired output is the closest spelled English word (like the feature auto-correct). In this case, the FST can be designed such that if some end state is reached in the FST, an English word would be output. Transitions from one state to another would read in the input letters in order, one at a time, and the final transition output would be an English word. However, since the original input might not be a valid English word, it might be necessary to modify the original letter sequence to match with a valid English word. In that case, the FST could include modification arcs–arcs that take in nothing but output an English letter; these arcs would be considered insertions and would have a cost associated with them. With the design of such an FST, the shortest path from the start state to some end state would yield the smallest edit distance. A similar concept is utilized in this work's finite state model, except using phonemes instead of letters.

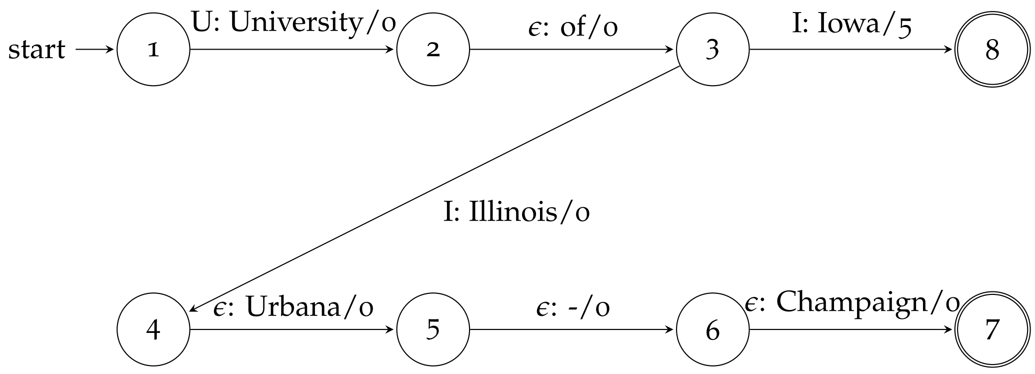


Figure 2.2: Example of an FST. With input letters "UI", the output of the shortest path will spell out "University of Illinois at Urbana-Champaign"

Compositions of FSTs involve combining two FSTs such that the input of the first FST yields a corresponding output in the second FST. Assume there are two FSTs: FST A that takes in an input string x and outputs a string y with weight z and FST B that takes in an input string u and outputs a string v with weight w.

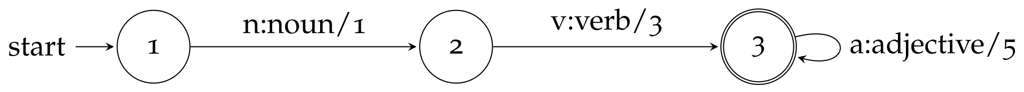


Figure 2.3: Diagram of FST A

The composition of FST A and FST B will result in a new FST, say C, that takes an input string y and outputs a string v with weight z+w

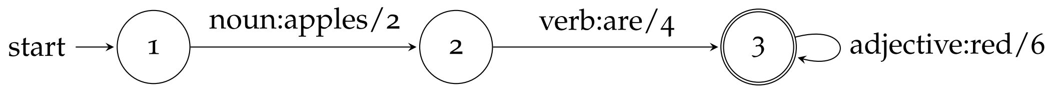


Figure 2.4: Diagram of FST B

(using tropical semiring). If there is no corresponding input/output pairs between FST A and FST B, then no new arc would be produced.

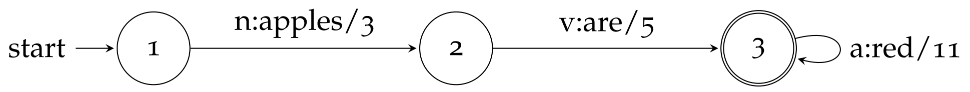


Figure 2.5: Diagram of composition of FST A and FST B

The FST creation and simulation in this research is done with the library OpenFST (Mohri et al. 2000). Usage of FST and design details of the finite state model will be discussed in Chapter 4.

2.2 EDIT DISTANCE

Edit distance refers to the minimum number of edits needed to change one string to another. Edits include: insertion, deletion, and substitution. For example, the word "Handy" has an edit distance of 1 to the word "Candy"—the edit needed was to substitute the letter "H" with "C".

Edit distance can be calculated using FSTs. An FST representing a word can be constructed by having its inputs and outputs on each transition arc be letters in the word (in proper order). Then edits can be handled by adding on arcs with weights of 1. Insertion would include an arc from one state to itself, with an input of epsilon and output of the letter to be inserted, deletion. Deletion would include an arc from one state to the next, with input the original letter in the word sequence and output of epsilon. Substitution would include an arc from one state to the next, with input the original letter in the word sequence, and output the letter to replace the original. The edit distance for a given word would then simply be the total path cost from the start state to the end state (if all arc weights for edits are set to 1).

In figure 2.6, the word "Handy" can be changed to "Candy" by taking the arc from state 1 to 2 with output "C". The edit distance between "Handy" and "Candy" would then be 1, since that is the total path cost from the start state to the end state. Similarly, "Handy" can be changed to the word "Hand" with edit distance 1 by taking the arc of output ϵ (deletion) from state 5 to 6.

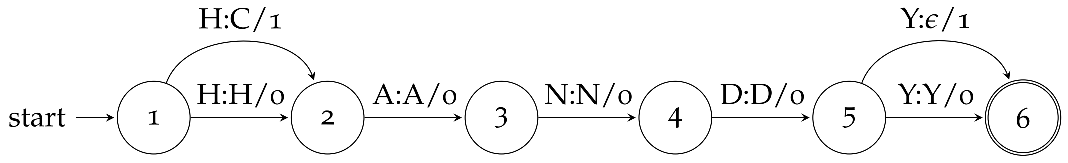


Figure 2.6: Diagram of computing distance from the word "Handy" to various other words.

The next chapter will discuss the design of the process of collecting labeled data, and further clarifying the need for a finite state model in refining user input.

3

SORAMIMI WORDS

The process of collecting labeled data was formulated as a game in order to attract more transcribers. The name of the game is ″Soramimi Words″. ″Soramimi″ is a Japanese word that generally means ″mishearing″. Soramimi Words roughly means ″mishearing words″ which is the intended situation for transcribers: to mishear words spoken in a foreign language and transcribe those words into perhaps nonsense English letters.

A form of a game is ideal for creating an incentivized uniform process for collecting desired labeled data. A uniform process would minimize inconsistencies from one process to another. A more enjoyable game would theoretically attract more transcribers and, in turn, increase the amount of data gathered.

To be more exact, with the rise of more gaming platforms (including mobile devices), the amount of time humans spend in playing video games has drastically increased. Games could be used as a medium that naturally prompts people to perform certain tasks. In fact, there have been successful attempts at making games that use the general public to collect data for studies (Von Ahn 2006). One of the more famous examples was called the ″ESP Game″, which involved having two users look at a picture, and type words describing the picture—much like the mismatched transcription process, but using pictures instead of audio files as label targets. The purpose of the game is for the two users to agree on a label for the same exact image. On the user end, the game is amusing and fun because it is interesting for humans to at least attempt to understand how other humans think. Humans are inherently social animals (in varying degrees), and social behaviors such as comparing different viewpoints proved to be interesting enough to gather enough users to play the ESP Game. The game ended up collecting more than 10 million labels in the first few months of launch. The resulting data was very useful in optimizing search engine results. By understanding how or what users think when trying to describe a picture, search engines could greatly improve search result yields given certain keywords. Similarly the process of mismatched transcription also look to games as a solution as a motivator for crowd sourcing label data.

The main purpose of Soramimi Words is to collect transcriptions of certain phrases in various languages from a random selection of subjects. However, a simple interface of playing sound clips and typing in transcription does not provide much incentive. Games are good incentives to play only when they are entertaining. The amount of amusement that a foreign sound provides is very limited; perhaps the duration of said amusement may last longer for a younger subject; however, younger subjects may not be the ideal ones to provide accurately labeled data.

Social interactions could be a good incentive for users to play games. However, the format of a game described for multi-players could be complicated, making it difficult to convince a user to play. For example, it is difficult in the early stages of the game to synchronize two random strangers as they go through the transcription process.

Soramimi Words is broken up into two stages: the first stage is where the player would be prompted to listen to a sound clip (which would be a short clip of words in the language the game is currently collecting labeled data for), and the second stage is where the player is prompted to listen to the shuffled clips again and attempt to come up with what they think are the closest sounding English words. Before the second stage, the player is told that their inputs will be evaluated against the computer's selections of best matching English words. The computer's output is generated from transcriptions in the first stage. Each input in the first stage is fed into an algorithm that produces an English word that has the least edit distance from the phonemes presented in the transcription. The player's transcription is used instead of a more standard transcription as the computer's goal is to create the closest sounding English word to the current player and verify the player is giving a good faith attempt transcription should the two English words be similar. The player's transcription is the best representation of what the player heard. The player's inputs in the second stage would then be evaluated against those computed outputs. In addition to providing an extra step of refinement, the presence of the computer was meant to provide a competitor to the player, thus prompting the player to give their best efforts.

Soramimi Words is available to play at `soramimiwords.web.engr.illinois.edu`. The user interface utilizes a minimalistic approach in order to ease the player into knowing how the game operates. The game was demonstrated to the general public at the Beckman OpenHouse at the University of Illinois at Urbana-Champaign in Spring 2015. No user data or inputs were recorded at the event. The exhibition was purely to observe user reaction to the games. The audience at the OpenHouse were mostly young children and community families which is not the ideal audience to collect labeled data. While most of the audience found the game to be generally entertaining, one ob-

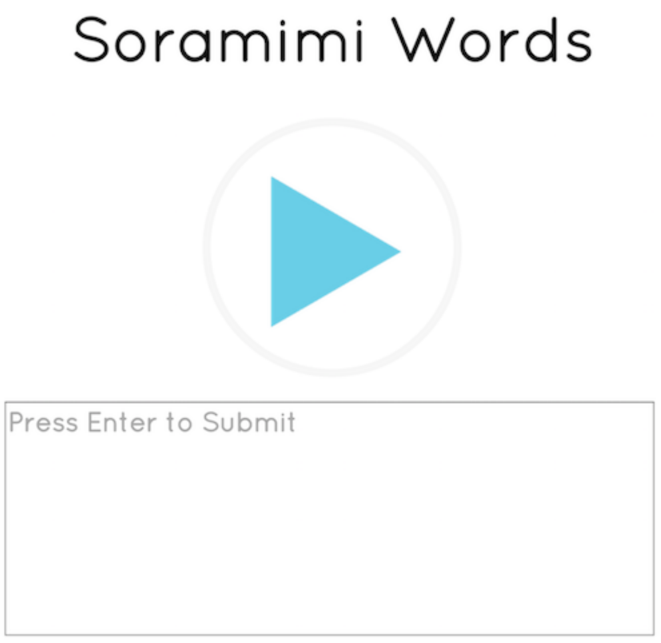


Figure 3.1: Screen shot of Soramimi's first stage gameplay

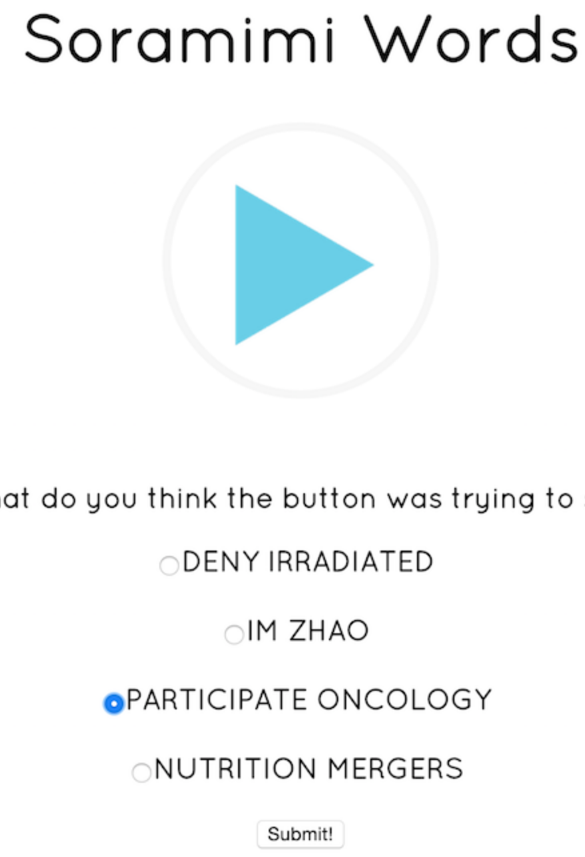
vious flaw was that the game was too long. It was difficult to keep the attention of the audience. The initial version of the game had a tutorial narrative that included a story of how a machine is challenging players to transcribe foreign words. General reactions hinted that the narration was too long. Later revisions of the game had the tutorial summarized in a few GIF images. Compared to words, moving pictures are more capable of capturing audience attention.

The flow of the game is straightforward: the player clicks on the blue triangle on top of the page to hear a sound clip, and types in appropriate transcriptions into the box. The grey circle around the blue triangle is a gauge that fills up as the player submits the entries to indicate progress. The blue triangle was chosen because it is the standard industry form of a play button, thus creating a mapping from the button's shape to the intended usage of the button. Similarly, there are no other buttons throughout game play; and the only other interactive component for the player is the text box; this minimizes the ”gulf of evaluation”: the difference between the intended purpose of an object versus the user's perceived purpose (Norman 2013). In future iterations of the game, it could be plausible to add a help button that flashes after a period of user inactivity in order to remind the user of the original task.

The second stage of the game is intended to refine the user's inputs. The user is given instructions to click on the play button, listen to the clips, and write down what they think is the closest sounding

English word. The sound clips in this stage are obtained by concatenating each clip with the succeeding sound clip from the order of clips in the first stage. This was done to introduce some variety to prompt the user to further think about the sound they have heard. This stage could also be used to weed out purposefully poorly attempted transcriptions.

Elaborating on how the audio files are chosen and presented to the players in the game: In the first stage of the game, 6 random sound clips containing a short phrase (3 to 4 syllables) in a foreign language are chosen from a pool of about 100 audio files. For the OpenHouse, clips in Hindi were played. The clips are then labeled numerically 1 through 6. Then in the second stage, the clips are shuffled in order (a possible permutation would be 2,6,1,5,3,4). The clips are then concatenated one after the other ([2, 6], [1, 5], [3, 4]) and played again to the player. Since each audio transcription generates one English word, each multiple choice option will contain two English words– and the player will listen to the sound clip again and identify the correct corresponding English words.


Soramimi Words

What do you think the button was trying to say?

DENY IRRADIATED

IM ZHAO

[✓ PARTICIPATE ONCOLOGY]

NUTRITION MERGERS

Figure 3.2: Screen shot of Soramimi's second stage gameplay

Originally, the process was designed so that the subject would type in what they think is the best sounding English word. However, in an effort to further shorten the refinement process (and the game in general), the second stage of the game was simplified to asking the subject to choose one set of English words from a list which would match the closest to the audio files. One option in the multiple choice would be the output English words created from the player's first stage transcription; the rest of the choices would be selected at random from a dictionary. To increase the difficulty of the game, it is also conceivable to develop a method to find a secondary closest sounding English word sequence instead of choosing English words at random. This deviates from the previous process where the computer's output word was never revealed to the subject. This change made the process easier to explain and decreased the amount of creative thinking the subject would have to do. Since the second stage of the game is simply for verifying the validity of transcriptions, multiple choice suffices.

Consider a perfect algorithm that does output the closest sounding English word given a transcription. The user's input in the second stage should then match closely to the computer's choice of closest sounding English word. If the user's choice of closest sounding English word is vastly different from the computer's choice of English word, then there are two possibilities: Either the user was not seriously attempting transcription in the first stage of the game, or the algorithm was faulty and produced incorrect English words as output. The former situation is expected to occur every now and then as players come and go. The latter prompts the researcher to perfect the algorithm such that there are no false positives in detecting faulty transcriptions. In the next chapter, the process of creating the finite state model will be discussed in more depth.

4

THE ALGORITHM

The algorithm involved in this research essentially recreates a phenomenon known as mishearing. Mishearing is the process of taking a set of phonemes, and then with some insertion, deletion, and or editing of phonemes, creating a different set of phonemes–producing a completely different word. Following the description of the game in the previous chapter, the input of the algorithm would be a string of nonsense syllables–transcribed from a sound clip in a foreign language from a random subject. The output of the algorithm would be, the closest sounding English word to the input.

This algorithm is a finite state model involved in the refining stage of the game. As mentioned before, the method of detecting a poor transcription from the player hinges on how much the player's choice of English word matches with the computer's choice of English word. For this error detection to work, the algorithm cannot be a noisy medium that outputs a suboptimal English word.

The algorithm generally works as follows: First, an FST is created that takes each letter in the input string as both input and output on the transition arcs.

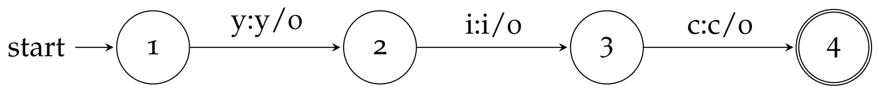


Figure 4.1: Diagram of initial input FST A

Second, an FST is created that maps each letter to potential phoneme mappings.

Figure 4.2 only shows three potential mappings between letters and phonemes. The actual FST consists of more transitions that cover more letter-to-phoneme mappings, not included above for clarity in illustrating this particular example. In fact, coming up with a good letter-to-phoneme mapping is very important in finding good outputs in the algorithm. This topic of configuring the letter-to-phoneme mapping will be discussed in much greater detail later in the next chapter.

The third step is to compose the two FSTs mentioned above.

The above FST gives the letter "y" two possible phonemes to map to. However, the above phoneme mapping might not be sufficient

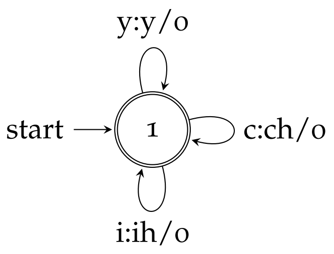


Figure 4.2: Diagram of FST B, showing three potential mappings from letter to phoneme

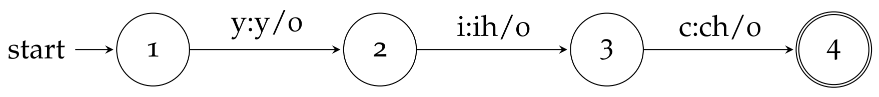


Figure 4.3: Diagram of FST A crossed with FST B as introduced above

to form an English word. In other words, the three phonemes "y ih ch" may not correspond to any English word. In order to get a match with any English word, the FST would have to allow for insertion and deletion of phonemes. In a more general case, the number of insertions includes all the letter-to-phoneme mappings that exist. However, due to space constraints and readability concerns, the following FST has only a few extra insertions and deletions allowed in the FST. This new FST will be called FST C.

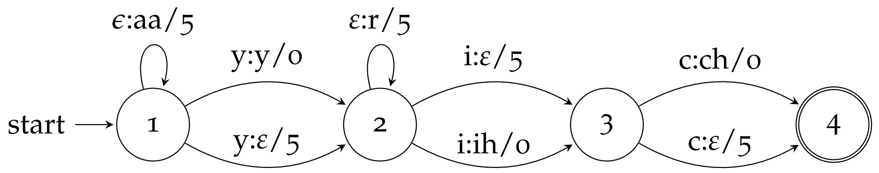


Figure 4.4: Diagram of FST C: product of FST A crossed with FST B, with allowed insertion and deletion of phonemes

Simply by human observation, in the FST above, there are two possible paths from the start state to the end state that would have an output of English words. The first has the phoneme sequence of "ih ch" (itch) with a total path cost of 5, where the first letter y's sound is deleted; and the arc with input letter y and output ϵ (nothing) is taken from state 1 to 2. The second has the phoneme sequence of "aa r ch" (arch) with a total path cost of 20. State 1 will take the self-loop and insert the phoneme "aa" with cost 5, take the arc that deletes the letter y with another cost 5, take the self-loop at state 2 that inserts the phoneme "r", take the transition that deletes the letter i from state 2 to 3, then take the transition from 3 to 4 with input letter c and output phoneme ″ch″.

The next step in the model is to create another FST to identify possible English words in FST C shown above. In essence, this FST automates the thinking process of mapping phonemes to English words presented above. This FST will have one start state and one end state. From the start state, transitions will take in phonemes and output English words in the very last transition. The end state will only be reached (an English word would only be produced as output) if the correct phoneme sequence is taken. The data for the FST is taken from CMUDict, which provides phoneme-to-word mappings. Again, for clarity, the FST below is greatly simplified and only contains two words from the dictionary. In the actual algorithm model, every word contained in CMUDict is included in the FST of phonemes to English words.

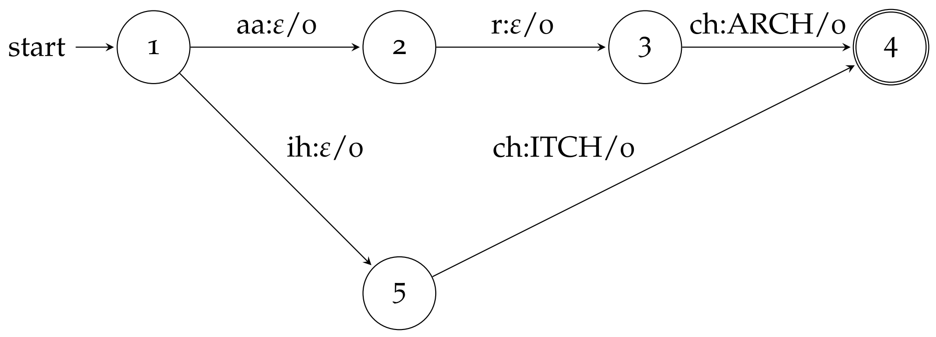


Figure 4.5: Diagram of FST D: with phonemes as inputs, and the last transition to an ending state with an arc output of a corresponding English word created from the path

Lastly, FST C will be composed with FST D to create FST E. The optimal English word will then be found by finding the shortest path from the start state to the end state.

Notice that, as discussed before on inspection of FST C, the path costs of creating the outputs ″arch″ and ″itch″ are 15 and 5 respectively. The English word with the least path cost is the most desirable one. Path cost is linearly related with edit distance. The more modification (insertion/deletion/replacement of phonemes) needed for an input, the greater the edit distance, and thus the greater the difference between the original input and the output in terms of phonemes. The result can be evaluated subjectively by having human subjects evaluate the output English words given an audio. On the other hand, the result can be evaluated objectively by calculating the edit distances between the input phonemes and the phonemes of the output English word.

The algorithm is implemented in C++ using the OpenFST library. The insertion and deletion costs for the FSTs are both set to 5. There

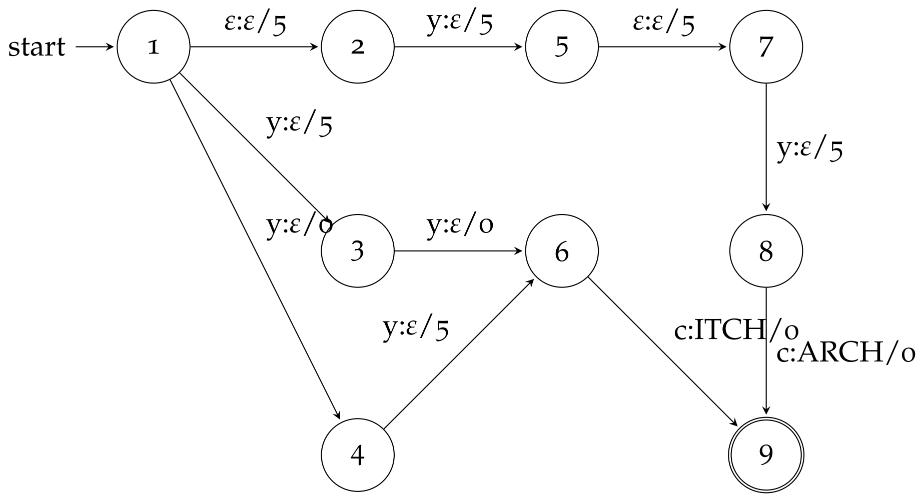


Figure 4.6: Diagram of FST E: composition of FST C and FST D, used to find the English word with least edit distance of phonemes from the original input

could be room for more research on the optimal insertion and deletion costs. The immediate effect of having a lower deletion cost would be shorter output words, and similarly, longer words with higher insertion cost. In this research, it does not matter what the edit costs are, as long as they are greater than 0.

The algorithm in its present state is not particularly fast. With larger inputs, the FSTs grow much larger and composition of FSTs and finding shortest paths takes longer to finish executing. Execution speed is a concern as this algorithm is used in a game. Nobody likes to play games with long loading times. One remedy attempted was to cut down the number of words contained in the dictionary. CMU-Dictionary is used for phoneme to word mapping, and the dictionary itself contains over 100,000 words and possible pronunciations. The dictionary also contains obscure words and sometimes words that are not English (and not popularly used in English). Therefore, for the sake of time and quality of output, the dictionary was reduced in size.

In order to find which words would be best to prune out, Google n-grams is used (Michel et al. 2011). Google n-gram contains information on frequency of words that appeared in books over the past 200 years. If the frequency of the word is over a threshold, then the word is added to a list. Then, each word in CMUDict is checked to see if it exists in the list. If the word was not in the frequent word list, it is pruned out of the CMUDict. The threshold for pruning can be lowered or raised depending on how many frequent words are included in the list; the initial threshold was set to word frequency of 100, and the dictionary size was pruned down to around 15,000, which was plenty for the purposes of the game.

5

EXPERIMENTS: LETTER TO PHONEME MAPPING

A big factor in determining the accuracy of the algorithm depends on the mapping between letters and phonemes. When an input string comes in, there are multiple ways the letters in the string can be mapped to phonemes. In fact, it is difficult to perfectly extract what the subject meant by certain combinations of letters, as there are deviations between hearing perception of the audio and spelling. There are letters in English that are silent, or have different pronunciations depending on the arrangements of letters.

A very bad mapping produces very bad outputs. For example, a mapping could simply map every letter in the input string to the phoneme ″aa″. The result would simply produce the English word with the largest amount of the sound ″aa″. The average edit distance for a bad mapping would also be higher than a good mapping, as a good mapping would provide more information on the possibilities of what letters could sound like, and the computer would not have to perform as many changes to the phoneme sequence to find a matching sequence that produces an English word.

The first mapping provided to the algorithm was done manually based on the knowledge of the English language by the author. The mapping took 35 phonemes in English and mapped each one to an English letter. This mapping is very minimal and is missing mappings from certain letters to phonemes; for example, the letter x was never mapped. In addition, there are multiple letters to multiple phonemes mappings that are unaccounted for in this first iteration.

The next iteration relies on a naive implementation of expectation maximization (EM, see Dempster et al. 1977). First we take the words that contains the same number of letters as to phonemes, and create a one-to-one mapping from those letters to phonemes. Then, the result is taken as a base list to compute more mappings from the rest of the words repeatedly. For each word in the dictionary, the list of phonemes is iterated through: if there exists a mapping between the letter and phoneme (from the previous iteration of finding mappings), then the phoneme is marked as mapped. At the end of this first iteration, there might exist phonemes and letters that are unmapped. This approach then maps each unmapped letter and phoneme to adjacent 2 or 3 phonemes and letters respectively.

Table 1: First iteration of letter-to-phoneme to letter mapping

Letter	Phoneme
a	aa
a	ae
a	ah
a	ao
a	aw
a	ay
b	b
c	ch
d	d
d	dh
e	eh
e	er
e	ey
f	f
g	g
h	hh
i	ih
i	iy
j	jh
k	k
l	l
m	m
n	n
n	ng
o	ow
o	oy
p	p
r	r
s	s
s	sh
t	t
t	th
u	uh
u	uw
v	v
w	w
y	y
z	z
z	zh

In edge cases where there are two letters or two phonemes adjacent to each other that are not mapped, then the case of multiple-to-one mapping is also considered. The mapping is then stored in a dictionary, along with the frequency of the mapping. At the end of the algorithm, mappings with frequency less than a set threshold (in this case the threshold was set to be the average number of frequencies) are considered bad and thrown away. This process is then repeated with the new list of mappings to generate the final list of mappings.

With the second iteration, the assignments of letters and phonemes are fairly generous. If a letter or phoneme is unmapped, it is mapped to two or three of its adjacent phonemes or letters; while this method picks up numerous different mappings, it also picks up numerous erroneous mappings, such as mapping a consonant letter to a vowel phoneme. The original intention was for the frequency pruning to get rid of such anomalies. However, even after increasing the pruning threshold, quite a few good mappings with fewer frequencies were pruned out, while only a few bad mappings were pruned out and the rest remained. It is very probable that certain letter and phoneme combinations are frequent enough to skew the pruning process. Therefore, after the results were generated, the file was in-

Table 2: Second iteration of letter to phoneme to letter mapping using naive EM

Letter	Phoneme
a	ao
a	aw
a	ay
bb	b
d	dh
dh	dh
e	ey
f	ff
kk	k
ll	l
o	oy
tt	t
t	th
u	uh
y	y
z	zh
si	zh
p	p
c	k
k	k
ch	k
fa	f
f	f
v	v
w	w
s	sh
sh	sh
r	aa-r
t	s-t
l	l
el	eh
ar	eh
a	eh
e	eh
ai	ey
r	er
or	er
ur	er
e	er
gr	g
g	g
gg	g
u	y-uw
b	b
rt	r
r	r
m	m
n	eh-n
i	ih-ng
g	jh
j	jh
s	s
si	s
so	s
c	ch
a	ae
n	n
nc	n
nd	n
y	iy
a	iy
e	iy
i	iy
e	ih
i	ih
u	y
a	r-ey
e	r-eh
e	er-z
d	d
ti	t
t	t
n	ih-n
o	ow
h	hh
u	uw
z	z
a	aa
a	ah
e	ah
o	ao
g	ng
n	ng
e	s-eh
i	r-ih
e	r-iy

spected manually, and obvious incorrect mappings such as the letter c mapping to phoneme "ah" were removed from the result.

The first set of mappings was completely manual, the second set of mappings was semi-automatic due to the last step of manually removing incorrect results. The second set of mappings is good at finding many mappings, with the downside of a high false-positive rate. Ideally, it would be preferable to have a completely automated process in generating mappings; as humans may provide good/bad mappings not detected due to incomplete understanding of the English language or simply human error.

The next step in generating sets of mappings is to completely automate the process of creating mappings. To this end, an open source library called Carmel is used. Carmel is a toolkit that does EM training given training data and a FST model (Graehl and Knight 2009). The FST model given would be all possible mappings from phoneme to letter, and the training data would be words and their corresponding phoneme sequences. Running EM training of Carmel on the data would then output the probability of each mapping of letter to phoneme. The FST model provided to Carmel only supports a maximum of two letter mappings, as any larger mappings may take too long. Unlike the previous naive EM implementation, Carmel will not output erroneous mappings; but the mappings Carmel provides will not be complete or broad. In other words, the outputs will never contain erroneous mappings between letters and phonemes, but would not be complete as the original FST model provided only supports 2 letter mappings.

After Carmel assigns probability to each transition arc (phoneme to letter mapping), a threshold is introduced such that mappings with low probabilities are not considered valid. The number of mappings generated by Carmel is much greater than the previous two iterations and will be included at the end of this chapter in tables.

After the generation of the three sets of phoneme-to-letter mappings, experiments were then run to test how well each set did in terms of improving the output English word.

As mentioned before, given an accurate transcription of sound clips, the finite state machine model will ideally output a very close sounding English word such that the effect of mishearing will be achieved. Given a list of 50 Hindi clip transcriptions, the transcriptions were inputs to the finite state models, and the output English words were collected for each set of mappings. It should be noted that the list of Hindi words was not used in previous iterations of working with the algorithm; and the finite state model was not changed or tweaked based on the output English words. The author then listened to the sound clips corresponding to each transcription, and rated the output English words on a three-point scale: Bad, Okay, Good.

Table 3: 3 Point rating results for the 3 sets of mapping

Rating	Manual Mapping	Naive-EM	Carmel
Bad	26	15	18
Okay	43	38	32
Good	24	40	43

The rating criterion is as follows: If the output English word has no correlation with the audio file, that is, it is impossible to hear even traces of the word's syllable in the audio file, then that entry gets a Bad rating. If the output English word has some correlation with the audio, and one can make out a few syllables of the English word present in the audio, then the entry gets an Okay rating. Lastly, if the output English word has high correlation with the audio, such that one can really mishear the audio to be the English word, then the entry gets a Good rating. Table 3 shows that Carmel had the largest number of Good ratings and the least number of Bad ratings while manual mapping contains the largest number of "Bad" and "Okay" ratings across the three. There is an increase of "Good" ratings going from manual mapping, naive-EM, to Carmel.

A further step is taken to refine the output of the finite state model—that is, calculating the average edit distance between the original set of phonemes to find the closest sounding English word. This approach quantifies the evaluation method for the mappings. A lower average edit distance would indicate the mapping was more complete, covering more phonemes to letters and consequently lowering the number of edits to the original input.

The edit distance is calculated as follows: each Hindi audio file had a transcription and a phoneme sequence, both of which were provided by a native Hindi speaker. Each algorithm is then given the list of Hindi transcription, and the edit distance between the English word and provided phoneme sequence for each audio file.

Table 4: Average edit distance for the 3 sets of mapping for a set of 50 Hindi transcriptions

Manual Mapping	Naive-EM	Carmel
3.90	3.73	3.75

As expected, from least automation to most automation, the number of edits needed to generate an English word decreased. In conjunction with the results of subjective evaluations of proximity of English word outputs, the intended result of having the best mapping stemming from a completely autonomous process is reached.

Table 5: Part 1 of 2: Phoneme to letter mappings generated by Carmel

Letter	Phoneme	Letter	Phoneme	Letter	Phoneme
a	aa	t	t	de	d
o	aa	o	uh	dg	jh
a	ae	u	uh	ei	ay
a	ah	o	uw	ey	ay
e	ah	u	uw	ed	d
i	ah	w	uw	ea	eh
o	ah	v	v	et	eh
u	ah	w	v	er	er
a	ao	u	w	ei	ey
o	ao	w	w	ey	ey
o	aw	e	y	ea	iy
i	ay	i	y	ee	iy
y	ay	j	y	ey	iy
b	b	u	y	eu	oy
c	ch	y	y	ed	t
t	ch	s	z	eu	uw
d	d	z	z	ew	uw
a	eh	g	zh	es	z
e	eh	j	zh	fe	f
a	ey	s	zh	ff	f
e	ey	z	zh	gh	aw
f	f	at	ae	gg	g
w	f	al	ao	gh	g
g	g	au	ao	gu	g
h	hh	aw	ao	g	g-ah
e	ih	ao	aw	ge	jh
i	ih	au	aw	gi	jh
y	ih	ai	ay	ge	zh
e	iy	ai	eh	he	dh
i	iy	ar	er	he	hh
y	iy	ai	ey	ht	t
d	jh	ay	ey	hw	w
g	jh	au	ow	ig	ay
j	jh	as	zh	ir	er
c	k	bb	b	ie	iy
k	k	be	b	i	w-ih
q	k	b	b-ah	je	zh
l	l	cc	ch	ke	k
m	m	ch	ch	le	l
n	n	ci	ch	ll	l
n	ng	cz	ch	me	m
o	ow	ch	k	mm	m
p	p	ck	k	m	m-ah
r	r	ce	s	ne	n
c	s	ch	sh	nn	n
s	s	ci	sh	ng	ng
z	s	dd	d	ou	ah

Table 6: Part 2 of 2: Phoneme to letter mappings generated by Carmel

Letter	Phoneme	Letter	Phoneme
or	ao	s	z-ah
ou	ao	si	zh
ou	aw	su	zh
ow	aw	s	zh-ah
or	er	ti	ch
oa	ow	tu	ch
oe	ow	th	dh
ow	ow	ti	sh
oi	oy	te	t
oj	oy	tt	t
oy	oy	th	th
oo	uh	ur	er
ou	uh	ue	uh
oo	uw	uh	uh
ou	uw	ul	uh
ph	f	ue	uw
pe	p	ui	uw
pp	p	u	y-ah
re	er	u	y-uh
re	r	u	y-uw
rr	r	ve	v
se	s	wh	hh
ss	s	we	w
sc	sh	wh	w
sh	sh	x	g-z
si	sh	x	k-s
ss	sh	xi	zh
sz	sh	ze	z
se	z	zh	zh

6 CONCLUSION

Labeled data for languages where resources of native speakers are difficult to reach could be instead generated via methods of crowd sourcing. By establishing a streamlined process embedded in a game, the creation of Soramimi Words aims to collect labeled data via crowd sourcing. In an attempt to refine the transcriptions obtained from the game, a finite state model is utilized. Given the player's transcription input, the model outputs the closest sounding English words. If the player sincerely attempts to provide a reasonable mismatched transcription, when exposed to the same audio clip and given a list of choices of English words, the one generated by the finite state model should stand out as familiar, sounding closest to the audio clip.

Improvements on the finite state model in the game Soramimi Words is explored by researching different mappings between phonemes and letters in English. The techniques for generating the mappings went from completely manual, semi-automatic and partially manual, to completely automatic. The initial mapping was generated by the author where 39 phonemes from CMUDict were taken and assigned English letters (Weide 2007). This approached relied heavily on the assigner's knowledge of the English language. The second approach is a naive implementation of expectation maximization, where a set of one-to-one phoneme-to-letter mappings was generated examining words with the same number of letters and phonemes; newer mappings were then generated by based on the previous set of assignments, and unmapped phonemes were assigned adjacent letters in the words. The new mappings were then stored along with the frequency of the mappings, and mappings with frequency lower than the average number of total mappings were discarded. The process was then repeated with the new set of mappings generated. The iterative process generated a large number of mappings, including, inevitably, erroneous mappings (e.g. mapping a phoneme that belongs to a vowel letter to a consonant). Those erroneous mappings were removed manually.

The last set of mappings was generated automatically via a tool called Carmel, an open sourced library that trains FSTs via EM. While this tool generates only completely valid mappings (unlike the previous approach, where the assignment of unmapped phonemes to letters is fairly generous) the range of assignments is quite limited as the resulting FST would take up too much memory if an FST allowing more than two letter assignments were created.

Experiments were performed on the three sets of English phoneme-to-letter mappings. In both a subjective evaluation, where words generated by each mapping were evaluated by the author for how closely they resembled a certain Hindi sound clip, and objective evaluation, where the average edit distance needed from Hindi audio transcription to the closest sounding words was generated, Carmel generated mapping was the most effective.

With the creation of the game Soramimi Words and the finite state model in the game, this work aims to collect labeled data that will in turn be useful in building speech technologies that will advance understanding of the languages of the world.

REFERENCES

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.

J. Graehl and K. Knight. Carmel finite-state toolkit. Technical report, 2009. URL `http://www.isi.edu/licensed-sw/carmel/`.

P. Jyothi and M. Hasegawa-Johnson. Acquiring speech transcriptions using mismatched crowdsourcing. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

J.-B. Michel, Y. K. Shen, A. P. Aiden, A. Veres, M. K. Gray, J. P. Pickett, D. Hoiberg, D. Clancy, P. Norvig, J. Orwant, et al. Quantitative analysis of culture using millions of digitized books. *science*, 331 (6014):176–182, 2011.

M. Mohri, F. Pereira, and M. Riley. The design principles of a weighted finite-state transducer library. *Theoretical Computer Science*, 231(1):17–32, 2000.

M. Mohri, F. Pereira, and M. Riley. Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, 16(1):69–88, 2002.

D. A. Norman. *The design of everyday things: Revised and expanded edition*. Basic books, 2013.

L. Von Ahn. Games with a purpose. *Computer*, 39(6):92–94, 2006.

R. L. Weide. The CMU pronouncing dictionary. Technical report, 2007. URL `http://www.speech.cs.cmu.edu/cgi-bin/cmudict#about`.