Diphone Synthesis Overview

***Abstract*—This report details the steps in building a unit selection synthetic voice based on diphones. This includes the development of a script, the recording of data, the creation of a database, and the process of synthesizing a target sentence. This process is used to build voices for 3 pointwise comparison evaluations which explore the effects of certain voice building steps. The first experiment investigates the importance of domain to designing the database. The second experiment explores the effect of database size and its relationship to domain. The third experiment takes a look at the pronunciation dictionary used at synthesis time.**

***Index Terms*—Unit Selection, Diphone Synthesis, Automatic Text Selection, Database Design, Speech Synthesis**

I. INTRODUCTION

This report seeks to explore a unit selection method for building a synthetic voice. Specifically, building a voice based on diphones which are algorithmically selected from a collection of recorded sentences. The process of building this voice involves many design choices which affect the end synthesis quality. Many of these choices have a cascading effect, influencing decisions made later in the synthesis pipeline. Ideally, we would make the best design decision at each step, but sometimes this choice is not obvious, so some experimentation is required. I perform three experiments to investigate how this method of synthesis might be improved. These experiments include using recordings from two different domains, adding more recordings to the database, and adjusting the choice of pronunciation dictionary.

II. DIPHONE UNIT SYNTHESIS

Diphone Unit synthesis is a type of unit selection. The base unit is a diphone and our database is a collection of diphones pulled from a recorded script. Diphones are considered a reasonable choice for unit selection in phoneme based languages like English [Taylor (2009), 491]. Further, they are preferable to phoneme units because they are better able to capture the co-articulation of two sounds. This is possible because diphones are sliced from an adjacent pair of phones and carry information about the transition between the two[1].

While some diphone based systems have a single diphone recorded for each context, unit selection takes this further by having many versions of each unit in the database. This helps account for context dependency, as a diphone can sound very different in different articulatory contexts (ex: interword vs. phrase final). However, the problem with attempting to represent every context is that the number of contexts is infinite. Luckily, many contexts will have similar acoustic realizations, so we only need to consider the contexts which have a discernible (by the human ear) effect on the final sound.

III. BUILDING THE DATABASE

A. Script Design

The first step in building the diphone database is designing the script which will be read by whoever is lending their voice to the synthesis system. Speakers cannot naturally utter diphones in isolation, so we must extract our phones from naturally read sentences [Taylor (2009), 487]. The script must cover a variety of diphone contexts, preferably as many as possible in as few sentences as possible. So, a script should be designed with diphone coverage in mind. The script can either be designed by hand or selected automatically from a corpus.

For this project I offer two similar methods of automatic script selection. The first was used to create the Arctic_A script and the second was used to create my Recipe Domain script. The goal was to create a domain script which was selected in a way similar to the Arctic_A script, which I did not design. I achieved this, in part, by using festvox tools which were pre-designed to aid in the script selection process [6]. The documentation page for these festvox tools notes that the CMU Arctic script was created using the same methods, but with an earlier version of the provided scripts [1]. Below, I outline the two methods in 5 steps:

Step 1: Selecting a Source Corpus

Arctic_A script: The Gutenberg Project corpora. An initial text corpus of 2.5 million words and 168 thousand utterances. [8]

Recipe Domain script: The FoodBase corpus, a resource of annotated recipes extracted from Allrecipes, a recipe centered social network. Comprising 56,700 utterances and 629,036 words, roughly 20 percent the size of the Gutenberg corpus used to create the Arctic Script. [9]

Step 2: Find "Nice Utterances"

Arctic_A script: Filter out sentences that are not between 5 and 15 words. The idea behind this step is that sentences which are either very short or very long are more difficult for a voice actor to read with desired consistency. Further, all the words of the utterance must be in the lexicon CMUDICT. This is so phone coverage is calculated using accurate phone labels for the words, rather than letter-to-sound rules. This process outputs 52,000 "nice utterances". [8]

Recipe Domain script: Run festvox script *find_nice* which will find "nice utterances" from the data set. A "nice utterance" is defined as one which is "of reasonable length, has only words in the frequency lexicon, no strange punctuation, capitals at the beginning, and punctuation at the end." Unlike with the Arctic set, I did not check words from the recipe domain

[1] Note the assumption that co-articulatory effects do not extend beyond the two phones

against the Unilex-gam lexicon (the lexicon used in this voicebuild) or the CMUDICT. This process outputs 21,000 nice utterances. [1][6]

Step 3: Search for the subset of utterances with the best phone coverage

Arctic_A script: 668 utterances were greedily selected from the larger corpus for best diphone coverage. [8]

Recipe Domain script: Run the festvox *dataset_select* script. This script greedily selects the utterances with the best diphone coverage and outputs them to a file. 494 utterances were selected for best diphone coverage.[2]

Step 5: Manual Adjustments

Arctic_A script: The output sentences were hand pruned based on a visual examination, leaving a final set of 593 utterances and 5258 words. [8]

Recipe Domain script: I manually added some further utterances for better domain specific coverage. My goal was to cover numbers and types of measurements often used in recipes. The final set is 553 utterances and 5737 words. This is purposefully similar in size to the Arctic script to allow for better comparisons during experimentation.

B. Recording the Script

Once we have a script, the next step is to make the recordings. We face two main design choices, the recording conditions and the choice of the speaker. Our goal is to record natural speech by having the speaker read the carefully selected sentences. In this case I am the only speaker available. I am not a professional, but I did my best to deliver a functional performance by remaining consistent across all recordings. This includes maintaining the same voice level during a recording session, and limiting the number of sessions to two. However, it should be noted that I was ill during the first recording session, which may negatively affect the voice quality of some recordings. In an ideal scenario, the speaker would be more consistent so as not to influence the listener during evaluation.

C. Labeling the Text Utterances from the Script

In order to segment our diphones for our database we need to label the recordings read out by our speaker. In theory, we could successfully hand label the audio, but this is labor intensive and could cause disparities between our database labels and the labels of our automatically labeled target sequence. It is preferable to automatically label the speech using the front end of our TTS system. This way, we can expect the units of our database to have the same labels as the units predicted for our target sentence during synthesis.

We put the scripts text through the TTS front end and get a phonetic sequence for each sentence. In this voice build we use HTK to get a master label file containing the phonetic transcription of all the utterances.

It should be noted that the phonetic labels for our speech are dialectally dependent. In other words, they will vary depending on which pronunciation dictionary is selected. Thus, it is important to choose a pronunciation dictionary which matches the accent of the recorded speaker, in my case this is the Unilex-gam dictionary.

In this system we have a predefined phone set and a lexicon of terms with their associated pronunciations. For terms which are not in the lexicon there is a system of letter-to-sound rules. These rules are not always accurate, but we can add missing words to the dictionary to improve the system. This may be necessary in a domain with a lot of unusual terminology such as in a medical field.

For my Recipe Domain script, there were 85 terms not in the lexicon, accounting for 188 words out of 5730 words or roughly 3 percent of the text. The greater the percentage of words in the recording script which are not in the lexicon, the more important adding terms to the dictionary becomes in creating a good system.[3]

D. Aligning the Labels with the Waveform

Once we have the labels we need to align these labels to the waveforms. We can do this using the forced alignment method provided by the HTK speech recognition toolkit. This forced alignment process involves training HMMs to recognize the words in the script recordings. These HMMs are then used to perform "recognition" on the same data (the script recordings parameterized as MFCCs) used to train them. As a byproduct of the recognition process, phone and word boundaries are found for the sentences [Taylor (2009), 479].

There are a few design choices involved in the alignment process. First, we can, if we choose, train the HMMs on different data and then attempt to use them to align our database. However, there is no real reason to do this as we are not looking to build a generalizable speech recognizer, only to align our data. So, it is simpler and best to train and align on the same data. Using different data, or a subset of our data is more likely to lead to false alignments.

Even when training and aligning on the same data, we might end up with some misaligned labels in which the start or end timestamp of a phone label is incorrect. Though the alignment is an automatic process, we are able to hand fix alignment errors. However, this is a tedious process and fixing a given error is unlikely to create a worthwhile improvement on the quality of the voice unless the error is somehow prolific and egregious.

IV. Synthesizing the Utterance

A. Selecting among Candidate Utterances

At synthesis time, we have a target sequence. In this case, we provide a raw text sentence to Festival. Our synthesizer must select the units from the database which best match the target sequence. For each base diphone in the target sequence there are many candidate matches (slices of

[2] This diphone coverage algorithm may not be working perfectly because I did not check my corpus against the lexicon, so in electing for diphone coverage the algorithm may be assuming incorrect diphones for some out of vocab words (which are expanded with letter to sound rules).

[3] By chance, only two of these missing words appeared in my evaluation sentences ("cilantro" and "hoisin"). The letter-to-sound rules for these terms turned out to be fairly accurate, so adding words to the dictionary was not necessary to achieve good intelligibility.

waveform) in the database. We need a method of comparing the linguistic qualities of the target diphones to the linguistic qualities of the candidates and finding the two which best match.

One major design choice of a synthesis system is deciding how to define "best" in terms of candidate selection. Hunt and Black define best as the diphone candidate with the "lowest cost. [7]" The cost of a diphone candidate is calculated using an algorithm which combines two components, a target cost and a join cost. Target cost is a measure of the distance between the target diphone and a candidate diphone. Join cost is a measure of how well two candidate diphones are able to join together. Minimizing the join cost gives us smoother sounding concatenation points.

B. Calculating Target Cost

Festival calculates the target cost using linguistic information about the target diphone and the candidate diphones. A set of feature categories are defined and then the individual values for each feature are calculated for each diphone. We can then use these features to measure the distance between two diphones. We are essentially asking 'how similar are these diphones based on this set of features.'

Our features can be anything that the front end of our TTS system is able to extract from our target sequence, such as phonetic context, stress and word position. This information is stored in utterance structures.

Some features are more important than others in our cost calculation. In order to account for this, weights are applied to the different features. Our target cost measure takes a target and candidate unit, calculates the distance between the units for each feature independently, then weights these features and combines them to form a total target cost for those two diphones. The more mismatched the context of a target and candidate diphone, the higher the target cost value. An exactly matching candidate would have a target cost of zero.

It should be noted that this method makes some naive assumptions for the sake of simplicity. Features are treated as independent of one another, even though we should expect certain features to interact during acoustic realization. Further, this feature based method makes implicit predictions about the acoustic output. The system assumes that, by selecting the best units based on these features it is outputting the best sounding speech. A more advanced system might incorporate acoustic features directly. For example, we could use a method of partial synthesis which would synthesize an acoustic representation from the target text and then compare these to the waveforms of the diphone candidates in the database. [Taylor (2009), 504]

C. Calculating Join Cost

Selecting candidate units based on the target unit is only half the battle. We must also consider how well two consecutive candidate diphones will sound once concatenated together during synthesis.

One of the benefits of unit selection is allowing for increased variability in diphone contexts. However, this means there is more variability at the diphone edges, so we cannot assume that two units will join together well. Hunt and Black approach this problem by comparing the last frame in the left diphone with the first frame in the right diphone [7]. Festival effectively uses this method by comparing features for these two frames.

In choosing features for the join function, we are lucky enough to be able to compare waveforms, since all of the candidates in our database have associated waveforms. So, we can more easily use acoustic features like MFCCs. In this voicebuild, the join cost is calculated by measuring the mismatch between MFCC vectors, F0, and power. We can investigate the relative effect of each of these features on voice quality by adjusting the weighting of each feature in the cost calculation.

Setting the weight of the power equal to zero notably affects the naturalness of the synthetic voice by making the speech vary erratically in volume. Yet, the speech still remains relatively intelligible.

Removing F0 as a feature (by setting it to zero) can create a machine-like quality to the voice, as human speakers tend to have smooth transitions between F0 values.[4] Some signal processing could be used to smooth these F0 transitions, but too much modification to the original signal comes with its own problems (see section V), so its preferable to have F0 as a factor in the join cost.

Setting the MFCC based feature weight to zero is the feature most likely to affect the intelligibility of the voice as it can lead to the algorithm joining formants which are very different from one another. This results in audible skipping sounds at the join points. The effect of these bad joins is most important to sounds which rely on formants for their quality, such as vowels [10].

The aforementioned features are default weighted to a value of 1.0 int his system and used to calculate join costs between pairs of diphones in the database. The goal is to minimize the cost. Any sounds in our database which were contiguously recorded will automatically have a join cost of zero. Further, for certain features the join costs will be considered zero if they are beneath some threshold of human perception as we cannot hear the difference between perfect matches and some near perfect matches.

D. Selecting the Best Sequence

The target cost functions and join cost functions help us choose from among our candidates for each target diphone. We could simply choose the best candidate for each diphone independently, but that may not necessarily give us the best complete sequence. The best candidate sequence is the one with the lowest total cost. We can compute the lowest total cost by combining the target and join costs for every diphone in the target sequence. However, this is a massive amount of computation. Luckily, we are able to use dynamic programming to make finding the best sequence computationally tractable.

The Viterbi algorithm allows us to search through a lattice of candidates for our target sequence and find the best sequence. This algorithm adjusts computation so it grows linearly with the length of the target sequence and the number of candidate targets. However, even with dynamic programming, the number of joins between sets of candidates

[4] Unvoiced sounds are less affected as they have no F0

is still an exponential relationship. Thus, computing the best sequence can still be a slow task.

There are some measures we can take to reduce the number of join costs that need to be calculated. First, we can implement some sort of pruning. For example, during beam search we prune candidates based on the width of the beam. This assumes that some candidates are so bad we should not bother calculating them.

We can also make best use of the zero join cost trick by improving our database to hopefully lead to more instances of zero cost joins. However, increasing the size of our database may do more harm than good by increasing the overall number of candidates. It would likely need to be done strategically. We could also go the other direction and make the database smaller so we have fewer candidates to compute joins between. However, with fewer candidates we are sacrificing quality.

A final option is to prune candidates based on target cost before computing join costs. We can use target costs to rank the candidates and remove all those which do not meet some minimum threshold. This can be helpful so long as the threshold is set properly and only very bad candidates are removed.

For the voice builds used in the experiments for this report, pruning parameters are left to the default as set in Festival's Multisyn engine.

V. Waveform Concatenation and Signal Processing

At synthesis time, once a best sequence of diphones has been selected, these waveforms are pulled from the database and concatenated in order. Unit selection was in part devised to allow for less signal processing at synthesis time than is required by standard diphone synthesis [4]. Too much signal processing can lead to degradation of the original signal, making it sound less natural. Further, in order to perform signal processing we must predict what processing to do from the target text sequence, which is a difficult task. By having multiple versions of each diphone, unit synthesis aims to make signal processing less necessary.

In best case scenario unit selection, no modification is needed and the synthesis is purely based on concatenation of the units. However, our database is always finite, so there are likely to be points where the waveforms are joined; some signal processing might be necessary in order to smooth out the transition. This smoothing is a pitch synchronous process, which is why the waveforms in our database have their pitch periods marked during processing.

VI. Experiment 1

A. *Introduction*

The construction of the database used for unit selection is a complicated design task. As designers, we have access to many kinds of source text through the internet. We need to consider what texts to use for script design. It makes sense to ask what kind of sentences we expect to synthesize and then design a script from text of the same domain. This experiment aims to investigate how the choice of domain for the recording script affects the final synthesized sentences. Presumably, a voice will be evaluated as better when the evaluation sentences match the recording domain.

B. *Methods*

In order to evaluate the quality of a speech synthesis system, it is wise to use human listeners as the end goal is to achieve intelligibility and naturalness, two metrics which are better measured by a listener than by an objective metric. [Taylor (2009), 535]

For this evaluation I gathered 5 naive listeners and had them perform a simple pointwise comparison task between sentences. Each sentence was synthesized twice, one using a voice build with the Arctic_A database, and one using a voice build with my Recipe Domain database. Upon an initial listen, I determined that the synthesized sentences were largely intelligible. So, listeners were asked to select which voice they think sounds "most natural."

I chose to use 20 evaluation sentences from the Recipe Domain. Though more evaluation sentences may provide more information, asking a listener to perform too many tasks runs the risk of them becoming bored or sloppy. My goal was to design a test which would take no longer than 30 minutes for a single listener.

These 20 evaluation sentences were taken from the Recipe Domain corpus which was used to build the Recipe Domain script. First I removed the sentences from the corpus used to make the Recipe Domain script. I then reran the same script selection algorithm from section III to get a subset of sentences which had been greedily selected to maximize phonetic diversity. From this subset of 472 sentences I selected the top 15 sentences. The goal is to maximize the phonetic diversity in the evaluation sentences to get a better sense of the voice build's capabilities. I then added 5 handmade sentences in order to round out the evaluation set with very frequent Recipe Domain terms, like numbers and measurements.

C. *Results*

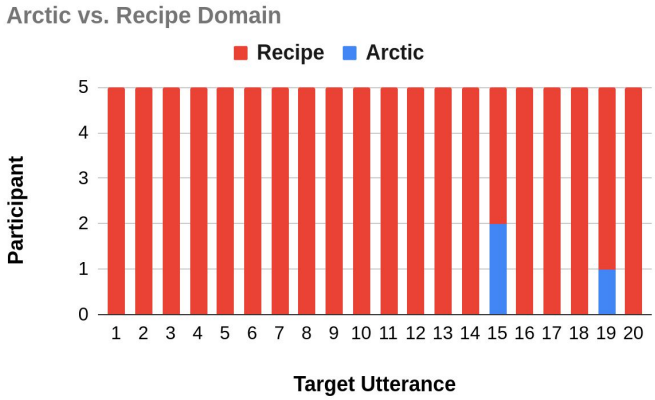

Figure 1: Results of a pointwise comparison between the Arctic voice build and Recipe Domain voice build when evaluated on 20 sentences by 5 listeners

D. *Discussion*

The Recipe Domain specific voice is clearly more natural when synthesizing Recipe Domain evaluation sentences based on this set of results. I suspect this is in part

due to a greater occurrence of zero-cost joins from contiguous recordings in the Recipe Domain database. Though both voices were mostly intelligible, the Recipe Domain voice had noticeably fewer instances of inter-word joins. This is unsurprising as many of the words in the evaluation sentences also occurred in the Recipe Domain script and so were made up of contiguously recorded diphones. These words were less likely to occur in the Arctic script so synthesized words are more likely to be made up of imperfectly joined diphone slices.

Given more time, it would be a good idea to extend this experiment by using 20 evaluation sentences selected from the Arctic Domain. If a similar pattern emerges (where these new sentences are better synthesized by the Arctic voice build,) then we would have further evidence in support of building a database from the appropriate domain.

VII. Experiment 2

A. *Introduction*

As a follow up to the previous experiment, I decided to combine the Recipe Domain database with the Arctic database and build a new combination voice. The goal is to see whether more data leads to an improvement over the Recipe Domain voice, despite the added data being from a different domain. It is often assumed that using a larger set of recordings will lead to better results during synthesis as there will be better diphone coverage. However, this may not necessarily be the case if the data is not all chosen with the domain in mind.

B. *Methods*

For the experiment the evaluation sentences are the same as those used in experiment 1.

The evaluation is performed using the same listeners and the same pointwise comparison method, but they were also given a third option, to say the synthesized sentences sounded the same[5]. This time one voice was built using the Recipe Domain database of 553 utterances. The second voice is built using a combination of the Recipe Domain and Arctic databases, resulting in a database of 1146 utterances.

C. *Results*

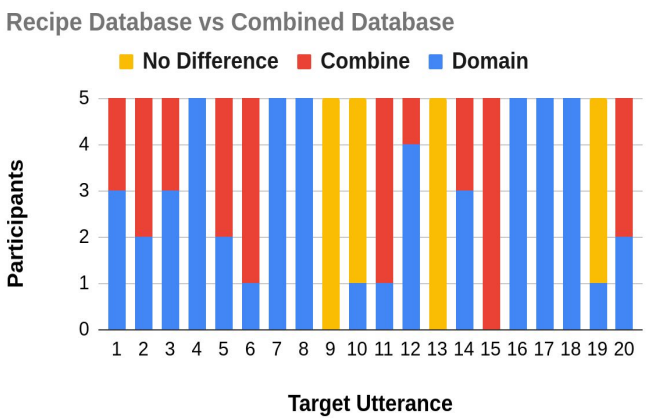

Figure 2: Results of a pointwise comparison between the Recipe Domain voice build and a combination voice build when evaluated on 20 sentences by 5 listeners

D. *Discussion*

As expected, the voice built from the combination database results in fewer missing diphones during synthesis of the evaluations sentences, though only two were gained. This is the result of having more candidates available thanks to the larger database. Despite this fact, listeners generally preferred the sentences synthesized with the Recipe Domain voice build over those built with the combination voice build. This indicates that it may be better to have less data so long as it fits the domain than to have more data which is less tailored to the task.

It is useful to consider how the lexical diversity of a given domain might affect the best way to design the database. A domain with less lexical diversity might benefit more from having a database well curated to that domain than from having a larger database. In the case of this study, we can look at the difference in lexical diversity between the Arctic script and the Recipe Domain script[6]. A simple type/token calculation indicates that the Arctic script is more lexically diverse than the Recipe domain script[7]. This intuitively makes sense, as recipes tend to have a repetitive form and a narrow topical focus, whereas the Guttenberg corpus is more broad. Perhaps these qualities in the Recipe Domain make is more beneficial to have high quality synthesis of the more common phrases which comes from a well curated domain specific script, rather than the greater coverage that comes from a larger, but less curated script.

This could have implications for script design. Perhaps a preliminary analysis of the domain can best indicate whether to maximize for domain specific lexical coverage or for overall diphone coverage. For a domain where lexical diversity is more limited, maybe the system would benefit from maximizing its database of utterances to contain most of these words (which would result in a lot of zero join costs for words) and then augmenting with sentences which represent interword diphones (sentences which have the best diphone

[5] After giving the synthesized sentences an initial listen, prior to designing the test, I noticed a few of them sounded identical (ore more likely near identical) between the two voice builds. This is unsurprising as the combination voice would have all the data present in the Recipe Domain voice

[6] It would have been better to look at the whole corpus if possible, rather than extrapolating from the scripts

[7] Domain Type/Token Value: 0.223
Arctic Type/Token Value:: 0.340

coverage for boundaries between words). For example, say we wanted to build a voice to read a popular series of children's books with a very limited vocabulary, we should consider lexical coverage in the script.

However this might not work so well for a very lexically diverse domain, where we have too many word types to reasonably get most of them in our database. In this case we might be better off focusing on diphone coverage so as best to synthesize unseen words.

VIII. Experiment 3

A. *Introduction*

As a final experiment I investigated the use of a pronunciation dictionary. The goal was to see if I could create an intelligible voice of a specific accent using recordings and labels from a different accent. If this worked well, it would be a highly convenient way to avoid needing to record accent specific data for a given voice build. However, this assumes that the voice build would still have the diphones necessary for synthesis and that they could be concatenated naturally.

B. *Methods*

For the experiment the evaluation sentences are the same as those used in experiment 1.

The evaluation is performed using the same listeners and the same pointwise comparison method as experiment 1. The only difference is the voice builds being compared. Both voices were built using the Recipe Domain database and labeled using the unilex-gam dictionary (the dictionary which matches the speaker). At synthesis time, one voice uses the Unilex-rpx dictionary, while the other uses the Unilex-gam dictionary.

C. *Results*

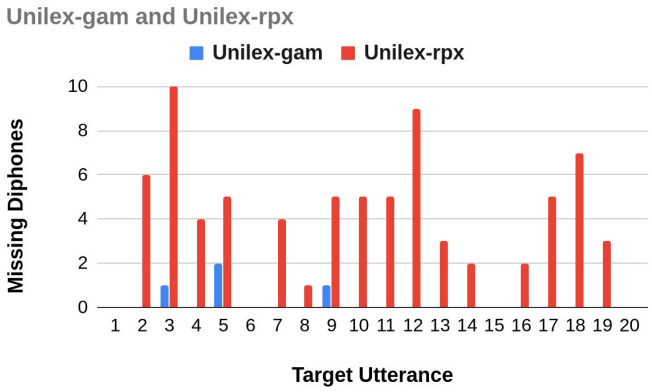

Figure 3: Comparison between two voice builds of missing diphones at synthesis time for 20 evaluation sentences. One voice uses the Unilex-gam dictionary and the other uses the Unilex-prx dictionary

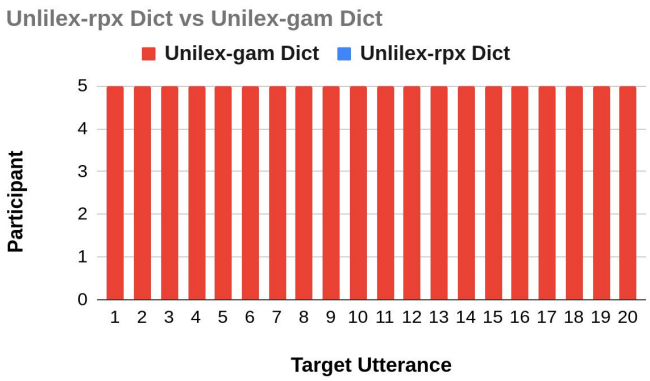

Figure 4: Results of a pointwise comparison between a voice build using the Unilex-gam dictionary and a voice build using the Unilex-prx dictionary. Evaluated on 20 sentences by 5 listeners

D. *Discussion*

Choosing the Unilex-rpx dictionary at synthesis time, results in very poor synthesis. Many more diphones are missing during synthesis time (see figure 3) and the speech becomes largely unintelligible for many of the utterances due to inter-word silences. So much so, that all 5 listeners preferred the utterances synthesized using Unliex-gam for all 20 evaluation sentences.

This mismatch in dictionaries likely leads to far fewer instances of zero-cost joins between contiguous sounds. This again highlights how much effect these contiguous words have on the overall quality of the output. Instead this voice sounds much more sliced up, with noticeable joins in the middle of many words.

These results demonstrate the necessity of remaining consistent with pronunciations throughout the build, all the way back to the original recordings. Ideally, we would use a speaker with dictionary perfect pronunciation. Further, in order to build a voice which could synthesize sentences with different accents we would need to take a different approach in designing the pronunciation dictionary. Perhaps using something like the keyword lexicon as described by Fitt and Isard could make a more generalizable voice build [5].

IX. Conclusion

This investigation of diphone unit synthesis has emphasized some key choices in the development of a voice build, specifically the importance of domain, its relationship to amount of data, and the structure of the pronunciation dictionary. It has become clear that a good voice cannot be built without a good database. Thus, script selection is vital and must be tailored as best as possible to the task at hand. Unfortunately, this means that these unit selection voices, while functional within their domain are not very generalizable. It is difficult to build a decent voice from data not specifically recorded for that task and building these voices is no small effort.

X. References

[1] A. W. Black, K. Lenzo, Building voices in the Festival speech synthesis system, 2000, 89-90, http://festvox.org/bsv.

[2] Black, A. W., & Campbell, N. (1995). Optimising selection of units from speech databases for concatenative synthesis.

[3] Black, Alan & Taylor, Paul. (1997). The Festival Speech Synthesis System: System Documentation.

[4] Clark, R. A., Richmond, K., & King, S. (2007). Multisyn: Open-domain unit selection for the Festival speech synthesis system. Speech Communication, 49(4), 317-330.

[5] Fitt, S., & Isard, S. (1999). Synthesis of regional English using a keyword lexicon.

[6] https://github.com/festvox/festvox

[7] Hunt, A. J., & Black, A. W. (1996, May). Unit selection in a concatenative speech synthesis system using a large speech database. In 1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings (Vol. 1, pp. 373-376). IEEE.

[8] Kominek, J., Black, A. W., & Ver, V. (2003). CMU ARCTIC databases for speech synthesis.

[9] Popovski, Gorjan & Seljak, Barbara & Eftimov, Tome. (2019). FoodBase corpus: a new resource of annotated food entities. Database The Journal of Biological Databases and Curation. 2019. 10.1093/database/baz121.

[10] Syrdal, Ann. (2001). Phonetic effects on listener detection of vowel concatenation. 979-982.

[11] Taylor, P. (2009). Text-to-speech synthesis. Cambridge university press.

XI. Appendix

A. Evaluation Sentences

1. In a bowl, mix the avocados, black beans, corn, onion, salsa, cilantro, and lemon juice.

2. In a medium bowl, place the sour cream, Neufchatel cheese, white sugar and raspberry extract.

3. Set the processor on medium and slowly add salsa until the texture is lumpy.

4. Mix well, add additional hoisin sauce to thicken mixture to your desired consistency if needed.

5. Layer the ingredients in the order given in a one quart wide mouth canning jar.

6. Broil until cheese is melted.

7. Season with balsamic vinegar and garlic salt.

8. When all the cheese has melted, stir in salt and nutmeg.

9. Heat the oil in a large, heavy skillet over medium heat.

10. Cover with crabmeat, squeeze juice of one lemon over the crabmeat.

11. Drop by rounded tablespoons onto greased baking sheets.

12. With a spoon, draw the chocolate up the sides of the cups until evenly coated.

13. Stir ice cream to soften.

14. When melted, dip egg shapes in chocolate.

15. It tastes best if chilled overnight.

16. For best results, allow bologna to pickle 90 days.

17. Place on ungreased baking sheet; bake 18-20 minutes or until golden brown.

18. Cover bowl with plastic wrap; let dough rest for 30 to 60 minutes.

19. Evenly divide dough into 16 pieces.

20. Heat oil in deep fryer to 375 degrees Fahrenheit.