Speech Synthesis of Code-Mixed Text

Sunayana Sitaram and Alan W Black
Carnegie Mellon University
Pittsburgh, USA
ssitaram, awb@cs.cmu.edu

Abstract

Most Text to Speech (TTS) systems today assume that the input text is in a single language and is written in the same language that the text needs to be synthesized in. However, in bilingual and multilingual communities, code mixing or code switching occurs in speech, in which speakers switch between languages in the same utterance. Due to the popularity of social media, we now see code-mixing even in text in these multilingual communities. TTS systems capable of synthesizing such text need to be able to handle text that is written in multiple languages and scripts. Code-mixed text poses many challenges to TTS systems, such as language identification, spelling normalization and pronunciation modeling. In this work, we describe a preliminary framework for synthesizing code-mixed text. We carry out experiments on synthesizing code-mixed Hindi and English text. We find that there is a significant user preference for TTS systems that can correctly identify and pronounce words in different languages.

Keywords: speech synthesis, code-mixing, multilingual systems

1. Introduction

Text-to-Speech (TTS) systems synthesize speech from text, and this text is usually assumed to be written in a standardized written form used for the target language. However, in many cases, a language is written using the script of another language, even if it has its own standardized written form. A common phenomenon that occurs in text in Social Media, Instant Messaging and email in bilingual and multilingual societies is code-switching, in which users type multiple languages in the same script, or sometimes retain the original script of the languages that are being mixed in the sentence.

Code switching is defined as switching between different languages in speech or text, in which the grammatical forms of the languages are preserved. Typically, one can identify one, or a few points in a sentence where code switching occurs. Code-mixing refers to the mixing of phrases, words and morphemes of one language into another language (Myers-Scotton, 1997). Lexical borrowing occurs when a word from one language is borrowed and used in another language while following the syntactic rules of the borrowing language.

Code switching can be thought of as an an inter-sentential phenomenon while code-mixing can be thought of as an intra-sentential phenomenon, although the techniques we describe in this paper are agnostic to the difference between the two. Code-mixing can also occur at the intra-word level, which we don't explicitly address in this work. From this point onwards, we choose to ignore the difference between code switching and code-mixing and use the term code-mixing as a general term to denote mixed languages in text.

Code-mixed text provides multiple challenges to TTS systems: first, the system needs to identify what languages are being mixed in the sentence, and then identify the language of each word or morpheme. Secondly, since languages being mixed may not have standardized spellings particularly if the text comes from social media, the system needs to normalize spellings. Lastly, the TTS system needs to ascertain the pronunciation of words in the sentence according to the pronunciation rules of the languages being mixed, while taking into account any influence of accents.

From the point of view of language resources, code-mixed languages can be considered to be low resource languages in general, because not many resources (particularly lexical) typically exist for such mixed languages. It is difficult to find labeled data and corpora for code-mixed languages, although there are some resources that have been developed in some language pairs, which we describe later. In many cases, one of the languages being mixed may be high resource, which can be exploited.

In this paper, we describe a preliminary framework for synthesizing code-mixed text. The organization of the paper is as follows. Section 2 relates this work to previous work in code switching and code-mixing. Section 3 describes our work on synthesizing code-mixed text written in different scripts. Section 4 describes our framework for synthesizing code-mixed text written in the same script. Section 5 concludes.

2. Relation to Prior Work

Code switching and code-mixing have been identified as challenges for language technologies ranging from Information Retrieval to Automatic Speech Recognition.

The Code Switching shared task at EMNLP 2014 (Solorio et al., 2014) consisted of data from 4 mixed languages (English-Spanish, Nepali-English, Arabic-Arabic dialect, Mandarin-English) and the task was to identify for each word which language it belonged to, or whether it was mixed, ambiguous or a named entity. Chittaranjan et al. (Chittaranjan et al., 2014) describe a CRF based approach for word level Language Identification for this task, in which they used various lexical and character-based features.

Recently, code-mixing in text has been studied for languages of the Indian subcontinent, which exhibit a lot of mixing with English and other Indian languages owing to

the large multilingual population and the number of languages.

Vyas et al. (Vyas et al., 2014) created a manually annotated corpus of code-mixed social media posts in Hindi-English and used it for POS tagging. They analyzed this corpus and found that 40% of the Hindi words in the corpus were written in Romanized script. They also found that 17% of the data exhibited code-mixing, code switching or both. They found that transliteration and normalization were the main challenges faced while processing such text.

Bali et al. (Bali et al., 2014) further analyzed this data to find that words fall into categories of code mixing, borrowing and ambiguous, with many borrowed words being written in English and many Hindi words being misidentified as English due to spelling. They suggest that a deeper analysis of morpho-syntactic rules and discourse as well as the socio-linguistic context is necessary to be able to process such text correctly.

Gupta et al. (Gupta et al., 2014) introduce the problem of mixed-script Information Retrieval, in which queries written in mixed, native or (often) Roman script need to be matched with documents in the native script. They present an approach for modeling words across scripts using Deep Learning so that they can be compared in a low dimensional abstract space.

A code switching corpus of interest is an Algerian Arabic-French corpus that contains 7000 comments from an Algerian news website (Cotterell et al., 2014). The unique feature of this corpus is that it contains Algerian Arabic text written in Romanized form ('Arabizi'), and Romanized Algerian Arabic tends to use Romanizations based on French orthography. This corpus has been manually annotated at a word-level with 'Arabic', 'French' and 'Other'.

Code Switching has also been studied in the context of speech, particularly for Automatic Speech Recognition (ASR) and building multilingual TTS systems.

Modipa et al. (Modipa et al., 2013) describe the implications of code-switching for ASR in Sepedi, a South African language and English, the dominant language of the region. They find that the presence of code switching makes it a very challenging task for traditional ASR trained on only Sepedi.

Vu et al. (Vu et al., 2012) present the first ASR system for code-switched Mandarin-English speech. They use the SEAME corpus (Lyu et al., 2015), which is a 64 hour conversational speech corpus of speakers from Singapore and Malaysia speaking Mandarin and English. They use Statistical Machine Translation based approaches to build code mixed Language Models, and integrate a Language ID system into the decoding process.

Ahmed et al. (Ahmed and Tan, 2012) describe an approach to ASR for code switched English-Malay speech, in which they run parallel ASRs in both languages and then join and re-score lattices to recognize speech.

Bilingual TTS systems have been proposed by (Liang et al., 2007) for English-Mandarin code switched TTS. They use speech databases in both languages from the same speaker and build a single TTS system that shares phonetic space. Microsoft Mulan (Chu et al., 2003) is another bilingual system for English-Mandarin that uses different frontends to process text in different languages and then uses a single voice to synthesize it. Both these systems synthesize speech using native scripts, that is, each language is written using its own script.

A TTS system that is capable of reading out social media text or informal messages produced by multilingual communities needs to be able to handle multiple languages. A Personal Digital Assistant also needs to be able to both recognize and produce natural code mixed speech while interacting with users who live in multilingual societies. To do this, the ASR, TTS and Machine Translation components of the system need to address challenges posed by code-mixing. With this motivation, our task for this work was to improve the pronunciation of a TTS system that has to read out code mixed text.

We divide the task of synthesizing such text into two main categories - synthesizing text which retains the script that the language is in, resulting in mixed scripts, and synthesizing text that (usually) uses the script of a single language, resulting in the use of a non-standard script for the other language.

3. Code-mixing with Mixed Scripts

In some text, people preserve the original scripts that the languages use while code mixing. This is seen in some language pairs being mixed, such as Chinese and Japanese with English. In cases where the scripts of the languages being mixed are different, we may be able to identify the language by looking at the script, and then use the appropriate TTS voice or Letter-to-Sound (LTS) rules to synthesize the text.

Ideally, we should get recordings from the same bilingual or multilingual speaker for all the languages that are being mixed and then switch between databases while synthesizing the different languages. However, getting such data and maintaining recording conditions across all the TTS databases may be difficult. Also, it may be difficult to anticipate which languages are being mixed in advance.

The Blizzard Challenge (Black and Tokuda, 2005) is an annual community-wide evaluation of TTS systems in which participants are given access to a common dataset to build synthetic voices from, which are then evaluated on a number of subjective metrics. From 2013-2015, the Blizzard Challenge included tasks on building systems for Indian languages. Since code mixing is very prevalent in many of the languages of the Indian subcontinent, the Blizzard Challenge added a multilingual task in 2014, in which English words were mixed with various Indian languages.

All the English words were written using the Latin alphabet, while the Indian languages were written in native script. The task was to synthesize test sentences containing such mixed sentences - however, the training synthesis databases contained no English in the recordings and corresponding prompts. There may have been some words written in the native script that were not native, like proper names, technical terms etc. in the data, but these were few in number.

Next, we describe our approach, the databases involved in these experiments and results from the Blizzard challenge.

3.1. Experiments and Evaluation

Since we only had data in the Indian languages to train from, we came up with a straightforward approach to deal with English words in Indian language sentences. When we detected a word in the Latin script, we used the US English text processing frontend to process it, which meant that all Non-Standard Words (abbreviations, numbers etc.) that were covered by the (much higher resource) US English frontend were also available to us. Then, we used a mapping between the US English phone set and the Indic phone set, which was common for all the lower resource Indian languages to convert the US English phonemes to Indic phonemes. This was a one-to-one mapping, which had its limitations, since some phonemes in English do not exist in the Indian languages and vice versa. Also, we could not incorporate contextual rules using this approach. However, the motivation behind using such a mapping was that it would implicitly capture the accent that a native Indian language speaker would use on English words by substituting phonemes from Indian languages.

We found that even with this simple approach, our system performed well in the Blizzard Challenge in 2014 and 2015. The languages that were included in the 2014 version of the challenge were Hindi, Gujarati, Assamese, Rajasthani, Tamil and Telugu and in 2015, Marathi, Bengali, Malayalam were included and Gujarati, Assamese and Rajasthani were excluded. The evaluation metrics used were similarity to speaker and naturalness, evaluated by paid listeners and volunteers.

Although in general we did not perform well on similarity to speaker due to our choice of synthesis technique (Statistical Parametric Synthesis as opposed to Unit Selection), our performance on naturalness was good. In particular, we had the best systems for Gujarati, Telugu, Tamil, Rajasthani and for the naturalness metric in 2014 and Hindi in 2015. The naturalness metric is not ideal for testing pronunciation quality of multilingual systems, since many other factors may also influence naturalness. However, it seemed like our approach was viable and was not influencing the quality of the system negatively.

4. Code-mixing with the Same Script

In many cases, when languages are mixed in text, the same script is used for all the languages that are being mixed. This creates the additional complexity of having to identify which language the words belong to. In addition, people may not follow standardized ways of writing the language that is using the "wrong" script. We extended the capabilities of our system to also be able to deal with same-script code mixing for some language pairs.

4.1. Data and Experiments

Our first task was to collect code mixed data, for which we crawled a Hindi recipe website in which the recipes were all in the native Hindi alphabet, Devanagari. The recipe descriptions were all in Devanagari, with a few words in English, such as titles, numbers etc. Interestingly, most of the comments submitted by users were in code-mixed English-Hindi, all written in Romanized script. We collected around 70k sentences from this website to create a code-mixed corpus. Two example sentences from the corpus are shown below with their translations.

Heavy Cream kya hai Ye kaha se milegi
Translation: What is heavy cream, where can it be found?

Dear nisha mujhe hamesha kaju barfi banane mein prob hoti h plzz mujhe kaju katli ki easy receipe bataiye
Translation: Dear Nisha I always have a problem making Kaju Barfi please give me an easy recipe for Kaju Katli.

In the first sentence, there is one point at which the code switching occurs, and the sentence is well written with correct spellings for the English words and reasonable transliterations of the Hindi words. In the second sentence, we see multiple locations in which English phrases and words are inserted ("Dear nisha", "prob" (problem), "plzz" (please), "easy receipe"). We also see that there are contractions and spelling errors. Some of the contractions ("h" for the word "hai") were common across the database and may be hard to normalize automatically in isolation even by humans.

Our goal for this part of the work was to be able to synthesize sentences such as those found in this corpus. We restricted our task to synthesizing Romanized text using our Hindi TTS system. Although this is may seem slightly artificial (synthesizing pure Romanized text using a Hindi voice), we can imagine finding the reverse case, where we have to use an English system to synthesize Romanized Hindi words that appear in Social Media posts, messages and emails. Also, as in the case of websites that have content in Devanagari but user submitted comments in Romanized Hindi and English, this setting may be more practical. An additional motivation for using the Hindi voice to synthesize these sentences was that these sentences were Hindi sentences (with Hindi being the matrix language) with English embeddings. We used the same Hindi TTS database from the Blizzard Challenge data for all our experiments.

First, we wanted to see how much of a difference we could make by using a Hindi pronunciation for the Hindi words. This involved both manually identifying the Hindi words in the Romanized sentence and replacing it with its correct (normalized) Hindi spelling, to be able to retrieve the correct LTS rules for it. This was, in a sense, "ground truth", or the best we could hope to get with our system.

We manually annotated and normalized 9 sentences by replacing the Hindi words with their Devanagari forms and synthesized them using our standard Hindi TTS system. We also synthesized the sentences using our current Indic frontend without any normalization, which meant that all the words went through the English frontend as described in the previous section. We asked 10 participants on Amazon Mechanical Turk to pick the system that was easier to understand, with a no difference option. Table 1 shows the results from this evaluation.

Table 1: *Subjective Evaluation of All-English Baseline vs Ground Truth*

Prefer Baseline	Prefer Ground Truth	No difference
18%	72%	10%

We expect that the Ground Truth system would be superior to the weak baseline in this case, but from the results we see that the preference for the Ground Truth was extremely high compared to the baseline. This is quite a large gap keeping in mind that the only difference between the two systems was the pronunciation of Hindi words in the sentences. This indicated that it would be interesting to see how close we could get to the Ground Truth by automating this process.

Our approach was the following: given Romanized text, first identify the English words in the sentence. Then, normalize all the other words to their standard spellings and try to recover the pronunciation of the normalized words. The next few sections describe these steps in more detail and the assumptions we made.

4.2. Language ID

First, we identified the English words in the English-Hindi mixed text. Our motivation to identify English words and not Romanized Hindi words first was that it is easier to find training data or resources to identify English words automatically and also that people writing in Romanized scripts were more likely to spell English words in a standardized way than Hindi words.

We took a naive approach to solving the Language Identification problem: if a word in the sentence was present in CMUdict (Weide, 1998), then we considered it to be an English word. Otherwise, we treated it as a misspelling of an English word or a Romanized Hindi word. Some words are ambiguous in isolation, and if they exist in the US English lexicon, they currently get marked as English.

The Language Identification step can be improved by making use of letter language models, taking into account context, using standard LID models on trained corpora etc., as described in the related work.

4.3. Spelling Normalization

After filtering out the English words present in the US English lexicon from the sentence, we normalized the spellings of the rest of the words. To do this, we used the Soundex algorithm (Russell and Odell, 1918). The Soundex algorithm encodes words such that spelling variants, which may have very similar pronunciation, are collapsed into the same code. Some characters such as those that represent vowels are ignored by Soundex.

Soundex only encodes consonants and ignores vowels. Each word is given a Soundex code, which consists of a letter followed by three digits. The letter is the first letter of the word and the digits encode the rest of the consonants in the word. Consonants that are similar share the same digit, for example, labial consonants (B, F, P, V) are encoded as the digit '1'. Soundex is used in many popular databases and is also used to index individuals and look up family names in the US census.

Taking the example of the words 'Smith' and 'Smythe', the Soundex algorithm works the following way. The first letter is retained in the code, and 'm' is mapped to 5 in both cases. The next letter, 'i' in 'Smith' and 'y' in 'Smythe' are both ignored. The letter 't' is mapped to '3'. The character 'e' is ignored in the case of 'Smythe'. In this way, both words have the same Soundex code and can be considered to be spelling variants according to this algorithm.

To normalize spellings in our data, we took the most frequent words of the code mixed recipe corpus as seeds and ran Soundex comparing each of these seeds to all the other words in the data. We formed clusters of spelling variants from these seeds by adding a word to the cluster if there was a Soundex match.

Then, we replaced the low frequency members of these clusters found in the sentences we wanted to synthesize with their seeds. In case a word belonged to more than one cluster we chose the seed to replace it with randomly.

In some cases (as in the case with the word "recipe") the seed words were English words that were found in the US English lexicon. In such cases, we treated these words as English words.

An extension of this could be to do a match phonetically, rather than by looking at vowels and consonants particularly for languages in which SoundEx implementations are not available.

4.4. Transliteration Model

After recovering the normalized spelling of the word in Romanized form, we then needed to recover its Hindi pronunciation. Instead of directly trying to recover the pronunciation from the Romanized form, we decided to reduce this problem into the problem described in the previous section, that is, synthesizing text that has mixed scripts, by transliterating Romanized Hindi into native Hindi script (Devanagari).

We explored transliteration standards and schemes that went from Hindi to Romanized, however, many of these standards used special marks such as diacritics to indicate vowel lengthening and most of them did not reflect how people actually type Romanized Indian languages on the web.

We trained a model on manually transliterated data from the FIRE dataset (Roy et al., 2013) that would give us a Devanagari word, given a Romanized word. The FIRE dataset is a useful resource due to the fact that it contains manually labeled words in Romanized Hindi with normalized spellings and transliteration in Devanagari. We used 1000 code mixed Hindi-English sentences and extracted the Romanized words marked as Hindi, and the Devanagari forms of these words. We found 1800 unique Romanized Hindi - Devanagari pairs in the dataset. The Romanized Hindi in the FIRE dataset was clean data with very few or no spelling variations for each word, which meant that our previous step of normalizing spellings was critical.

To build a model from Romanized Hindi to English, we followed the standard procedure to build Letter-to-Sound Rules in Festvox (Black et al., 1998). Usually, the input to that is a string of characters and the output is a string of phonemes, but in this case, both the input and output were strings of characters. We trained a Classification and Regression Tree for each grapheme in our Romanized Hindi word list which uses three previous and next grapheme contexts to predict Devanagari Hindi graphemes. Each Romanized Hindi grapheme aligned to none, one or two Devanagari graphemes.

4.5. Synthesis

Once the Language Identification, spelling normalization and transliteration stages were complete, we recovered the English words in Latin script and Hindi words in Devanagari script. This reduced the problem to the problem of synthesizing text in mixed scripts described in Section 3, and we used our standard system to synthesize this text.

4.6. Evaluation and Results

Once again, we carried out subjective evaluations on the Hindi TTS database with only the frontend being changed in each condition. In each case,10 listeners on Amazon Mechanical Turk listened to 9 pairs of sentences, which were the same 9 sentences that were used for the ground truth vs. baseline experiment.

Table 2 shows the preference for our technique, which we call Predicted, compared to the baseline where we treat all the words as English words, as described before. We can see that there was a significant preference for our method which does Language Identification, Spelling Normalization and Transliteration.

Table 2: *Subjective Evaluation of All-English Baseline vs Predicted*

Prefer Baseline	Prefer Predicted	No difference
16%	71%	13%

Next, we wanted to see how the Predicted system would compare against a system where all the Romanized words were treated as Hindi words. This is what would have happened if we had bypassed the Language ID step. Table 3 shows that there was a preference for the Predicted version, though it was not as significant as the preference over the All-English baseline. This could be due to the fact that there were more Hindi words than English words in the code mixed sentences and because the All-English baseline went through the English frontend and was then mapped to the Hindi phonemes.

Table 3: *Subjective Evaluation of All-Hindi vs Predicted*

Prefer All-Hindi	Prefer Predicted	No difference
30.5%	54%	15.5%

Finally, we wanted to see how close our Predicted system could get to the Ground Truth system. Table 4 shows that subjects had a preference for the Ground Truth system over the Predicted system, though the difference was not as high as the difference between the All-English baseline and the Ground Truth system.

Table 4: *Subjective Evaluation of Predicted vs Ground Truth*

Prefer Predicted	Prefer Ground Truth	No difference
28%	57%	15%

4.7. Future Work

We explored using the Algerian Arabic-English corpus (Cotterell et al., 2014) mentioned earlier for similar experiments, however, without hand labeled examples of Algerian Arabic-English, training data was not available to build a model for transliteration. One solution may be to use a transliteration resource such as the Google Transliteration API to go from Arabizi to Arabic, however, this may not be appropriate for the Algerian Arabic dialect. Furthermore, the closest TTS voice available to us in terms of language was in Modern Standard Arabic. Preliminary experiments showed that all these factors made it difficult to replicate the Hindi-English experiments for this data. However, given a reasonable amount of code mixed text and a mapping between the letters and the phone sets of the mixed languages, it should be possible to replicate the experiments done for Romanized Hindi.

The resources we used to make our Hindi TTS system capable of synthesizing Romanized Hindi and English words were a Language Identification System (in this case a US English lexicon), a moderate number of pairs of Romanized Hindi and Devanagari words, Soundex rules and a mapping between the US English and Hindi phoneme sets. While this mapping is a one-time cost, it requires some knowledge about the phonetics of both languages.

In addition, it may not always be possible to find good phonetic mappings between languages because some phonemes in one language may not exist in the other. In such cases, bilingual speakers may map phonemes to the closest possible phoneme, or borrow phonemes. **??** shows the mapping between English phones from the phone set used in our standard US English build and Hindi phonemes from the Indic phone set, along with the phonetic features assigned to each phoneme in both sets. In some cases, we did not map phonemes from English to very low frequency phonemes in our Hindi corpus, but chose to replace them with more frequent phonemes.

In all the experiments described above, we mapped phonemes from English to Hindi manually. However, we wanted to automate the process as much as possible.

The approach described by Nerbonne et al. (Nerbonne et al., 1996) uses Levenshtein distance to measure the distance between words in different dialects of Dutch, and uses this to group dialects together. A common set of words are compared across all dialects, with the distance being compared based on letters with different weights for insertions, substitutions and deletions. (Nerbonne and Heeringa, 1997) extend this work by using phonetic features and a variety of distance metrics.

Sriram et al. (Sriram et al., 2004) describe a technique for multilingual query processing, in which words in the queries are converted into a language independent 'common ground' representation, after which a weighted phonetic distance measure is used to match and rank queries. Phonetic features include vowel rounding, frontness, height, length, voicing, aspiration etc. which are the same features that are used in the standard Festival phonetic feature set. The authors also suggest weights that can be given to these phonetic features, since some of them may be more important than others while calculating phonetic

similarity.

Le et al. (Le et al., 2006) used a top-down bottom-up knowledge based approach for calculating phoneme similarity. They use models of similar phonemes cross lingually to create context dependent Acoustic Models in new languages. They used IPA rules to create a hierarchical graph of phoneme similarity, which splits phonemes into categories like Consonant-Vowel, Close-Back Vowels, Close-Front, Close-Back vowels etc. Acoustic Models were built with multiple languages and used for recognizing Vietnamese speech with a small amount of adaptation data.

Melnar et al. (Melnar and Liu, 2006) describe an approach to measuring phoneme distance cross lingually by representing phonemes as a feature matrix, with feature weights based on lexical frequency. In addition to traditionally used phonetic features, they also include corrolary features that represent allophonic realizations of the phones. This approach was shown to perform well on cross-lingual ASR tasks.

Pucher et al. (Pucher et al., 2007) describe phonetic similarity measures for ASR grammar optimization in which they compare minimum edit distance based measures, perceptually motivated measures and HMM-distance based measures and correlate them to word confusion in the ASR.

We implemented a simple weighted distance based metric to map English and Hindi phonemes in our system based on (Sriram et al., 2004). We conducted subjective tests comparing our manually mapped phonemes to the automatically mapped phonemes. However, we found that in subjective tests, listeners had a very significant preference for the manually mapped phonemes. This may be due to the fact that the manually assigned phonetic features may not be completely correct, and many phonetic features ended up getting the same weight in (Sriram et al., 2004).

Ideally, we should be able to automatically set these weights or learn this mapping from data. Recent work in creating vector representations of acoustics (Watts et al., 2015) could be a promising direction for creating such mappings automatically.

In this paper, we presented experiments on one language pair - Hindi and English. We are currently extending this work to cover more language pairs such as Spanish-English and German-English. We are also improving the components of our system, particularly the Language Identification and spelling normalization modules.

We did not explicitly address code-mixing that takes place at the morpheme level in our technique, and this would be an interesting future direction.

5. Conclusion

This paper presents preliminary work towards solving the problem of synthesizing code-mixed text. With the advent of smart phones and digital assistants in multilingual societies, we believe that this will be a very relevant problem to address in the future for speech processing. We provided a framework to synthesize code mixed text that came from Social Media in and presented experiments on Romanized Hindi-English using a corpus that we crawled from the web. Some of the ideas presented in this paper, such as spelling normalization can also be used while synthesizing single-language text from Social Media.

We conducted subjective listening tests to compare speech synthesized with our framework to baselines that treated all the words in the sentence as English words or Hindi words, and showed that there is a preference for speech synthesized with our technique. We also compared our technique to gold standard manually labeled and transcribed sentences with subjective tests and showed that there is still a gap between the two, which can be addressed with better Language Identification, spelling normalization and cross-lingual pronunciation rules. In all cases, we used minimal resources to extend the capability of our current system to make it capable of synthesizing code mixed text. In addition, better cross-lingual phonetic mapping techniques that make use of acoustics may eliminate the need to map phonemes manually.

6. Acknowledgments

The authors would like to thank Monojit Choudhury and Kalika Bali from Microsoft Research India for providing them with the FIRE dataset and for very useful discussions on code-mixing.

7. Bibliographical References

Ahmed, B. H. and Tan, T.-P. (2012). Automatic speech recognition of code switching speech using 1-best rescoring. In *Asian Language Processing (IALP), 2012 International Conference on*, pages 137–140. IEEE.

Bali, K., Sharma, J., Choudhury, M., and Vyas, Y. (2014). "i am borrowing ya mixing?" an analysis of English-Hindi code mixing in Facebook. *EMNLP 2014*, page 116.

Black, A. W. and Tokuda, K. (2005). The Blizzard Challenge 2005: Evaluating corpus-based speech synthesis on common datasets. In *Interspeech*.

Black, A. W., Lenzo, K., and Pagel, V. (1998). Issues in building general letter to sound rules.

Chittaranjan, G., Vyas, Y., and Choudhury, K. B. M. (2014). Word-level language identification using crf: code-switching shared task report of MSR India system. *EMNLP 2014*, page 73.

Chu, M., Peng, H., Zhao, Y., Niu, Z., and Chang, E. (2003). Microsoft Mulan-a bilingual TTS system. In *ICASSP*, volume 1, pages I–264. IEEE.

Cotterell, R., Renduchintala, A., Saphra, N., and Callison-Burch, C. (2014). An Algerian Arabic-French code-switched corpus. In *Workshop on Free/Open-Source Arabic Corpora and Corpora Processing Tools Workshop Programme*, page 34.

Gupta, P., Bali, K., Banchs, R. E., Choudhury, M., and Rosso, P. (2014). Query expansion for mixed-script information retrieval. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 677–686. ACM.

Le, V. B., Besacier, L., and Schultz, T. (2006). Acoustic-phonetic unit similarities for context dependent acoustic model portability. In *ICASSP*, volume 1, pages I–I. IEEE.

Liang, H., Qian, Y., and Soong, F. K. (2007). An HMM-based bilingual (Mandarin-English) TTS. *Proceedings of SSW6*.

Lyu, D.-C., Tan, T.-P., Chng, E.-S., and Li, H. (2015). Mandarin-English code-switching speech corpus in South-East Asia: SEAME. *Language Resources and Evaluation*, pages 1–20.

Melnar, L. and Liu, C. (2006). A combined phonetic-phonological approach to estimating cross-language phoneme similarity in an ASR environment. In *Proceedings of the Eighth Meeting of the ACL Special Interest Group on Computational Phonology and Morphology*, pages 1–10. Association for Computational Linguistics.

Modipa, T. I., Davel, M. H., and De Wet, F. (2013). Implications of Sepedi/English code switching for ASR systems.

Myers-Scotton, C. (1997). *Duelling languages: grammatical structure in codeswitching*. Oxford University Press.

Nerbonne, J. and Heeringa, W. (1997). Measuring dialect distance phonetically. In *Proceedings of the Third Meeting of the ACL Special Interest Group in Computational Phonology (SIGPHON-97)*.

Nerbonne, J., Heeringa, W., Van den Hout, E., Van der Kooi, P., Otten, S., Van de Vis, W., et al. (1996). Phonetic distance between Dutch dialects. In *CLIN VI: proceedings of the sixth CLIN meeting*, pages 185–202.

Pucher, M., Türk, A., Ajmera, J., and Fecher, N. (2007). Phonetic distance measures for speech recognition vocabulary and grammar optimization. In *3rd Congress of the Alps Adria Acoustics Association*, pages 2–5.

Roy, R. S., Choudhury, M., Majumder, P., and Agarwal, K. (2013). Overview and datasets of fire 2013 track on transliterated search. In *Fifth Forum for Information Retrieval Evaluation*.

Russell, R. and Odell, M. (1918). Soundex. *US Patent*, 1.

Solorio, T., Blair, E., Maharjan, S., Bethard, S., Diab, M., Gohneim, M., Hawwari, A., AlGhamdi, F., Hirschberg, J., Chang, A., et al. (2014). Overview for the first shared task on language identification in code-switched data. *EMNLP*, page 62.

Sriram, S., Talukdar, P., Badaskar, S., Bali, K., and Ramakrishnan, A. (2004). Phonetic distance based crosslingual search. In *Proceedings of the International Conference on Natural Language Processing*.

Vu, N. T., Lyu, D.-C., Weiner, J., Telaar, D., Schlippe, T., Blaicher, F., Chng, E.-S., Schultz, T., and Li, H. (2012). A first speech recognition system for Mandarin-English code-switch conversational speech. In *ICASSP*, pages 4889–4892. IEEE.

Vyas, Y., Gella, S., Sharma, J., Bali, K., and Choudhury, M. (2014). Pos tagging of English-Hindi code-mixed social media content. *Proceedings of the First Workshop on Codeswitching, EMNLP*.

Watts, O., Wu, Z., and King, S. (2015). Sentence-level control vectors for deep neural network speech synthesis. In *Interspeech*.

Weide, R. (1998). The CMU pronunciation dictionary, release 0.6.