Proceedings of the LREC 2020 – 4th Workshop on Computational Approaches to Code Switching, pages 57–64
Language Resources and Evaluation Conference (LREC 2020), Marseille, 11–16 May 2020
European Language Resources Association (ELRA), licensed under CC-BY-NC

Code-Mixed Parse Trees and How to Find Them

Anirudh Srinivasan[1] **, Sandipan Dandapat**[2]**, Monojit Choudhury**[1]
[1]Microsoft Research, India [2]Microsoft R&D, India
{t-ansrin, sadandap, monojitc}@microsoft.com

Abstract

In this paper, we explore the methods of obtaining parse trees of code-mixed sentences and analyse the obtained trees. Existing work has shown that linguistic theories can be used to generate code-mixed sentences from a set of parallel sentences. We build upon this work, using one of these theories, the Equivalence-Constraint theory to obtain the parse trees of synthetically generated code-mixed sentences and evaluate them with a neural constituency parser. We highlight the lack of a dataset non-synthetic code-mixed constituency parse trees and how it makes our evaluation difficult. To complete our evaluation, we convert a code-mixed dependency parse tree set into "pseudo constituency trees" and find that a parser trained on synthetically generated trees is able to decently parse these as well.

Keywords: Parse Trees, Constituency Parsing, Code-mixing

1. Introduction

Code-mixing is a phenomenon observed in multilingual societies all throughout the world. Although it started off as mainly a spoken phenomenon, the need for computational methods for processing code-mixed text is ever growing as people are now code-mixing on social media and other online platforms more and more (Rijhwani et al., 2017).

Most work focusing on computational methods for code-mixing have been on tasks like LID (Solorio et al., 2014; Sequiera et al., 2015), NER (Aguilar et al., 2018) and POS tagging (Vyas et al., 2014). Although there are some works on code-mixed dependency parsing (Partanen et al., 2018; Bhat et al., 2018), there is no work that has focused on obtaining parse trees and the task of constituency parsing.

Having parse trees and a constituency parser for any language is extremely useful. It can be used for understanding the syntax of a sentence and checking whether a sentence is grammatically valid or not. Parse trees can also be used to build a probabilistic context free grammar (PCFG) that would help us understand the usage of different grammatical elements.

The work in this paper makes 3 contributions to the area of code-mixed parsing. Firstly, we propose a technique that modifies existing linguistic theory based code-mixed sentence generation processes to obtain parse trees of the sentences. The trees produced are also annotated in a manner that captures the parallels between the mixed languages that are used during the generation process. Secondly, we use these synthetic trees to train a neural constituency parser and evaluate the parser on synthetic and non-synthetic trees. This evaluation was not straightforward as there doesn't exist a set of non-synthetic code-mixed constituency trees. To address this issue, as the third contribution of the paper, we convert code-mixed dependency trees into what we call as "pseudo constituency trees" and show that the parser is able to parse these as well.

The rest of the paper is organized as follows. Section 2 talks about linguistic theories for code-mixing and how they can be used to obtain code-mixed parse trees. Section 3 talks about neural constituency parsers and the parsing technique chosen for our testing. Section 4 describes the evaluation method for the parser and results on synthetic data. Section 5 talks about further evaluating the parser using non-synthetic data and Section 6 concludes our discussion.

Author can be contacted at anirudhsriniv@gmail.com

2. Obtaining Code-Mixed Parse Trees

2.1. Background

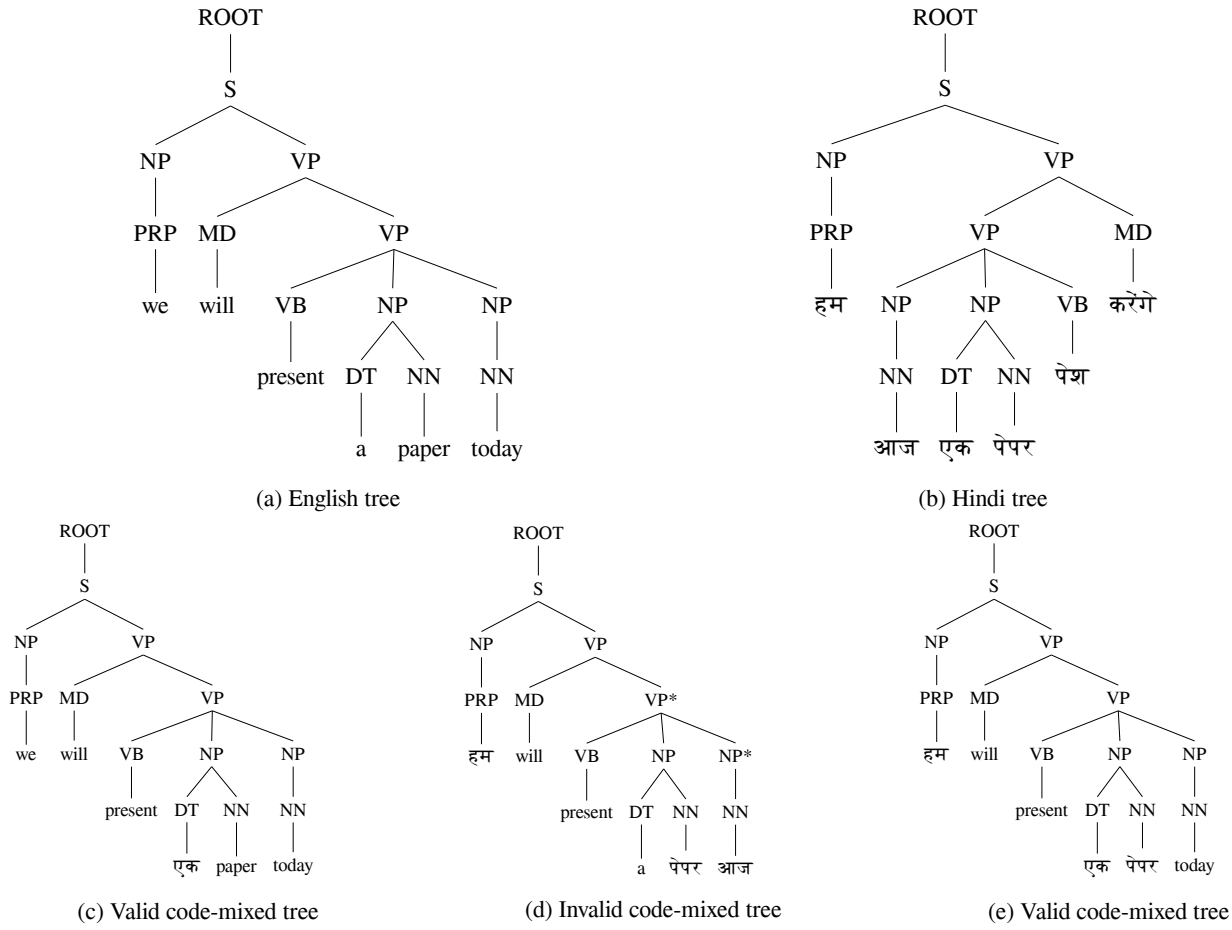
Researchers in linguistics have proposed multiple theories that aim to model code-mixing from a linguistics perspective. (Poplack, 1980; Sankoff, 1998; Joshi, 1982; Milroy, 1995; Di Sciullo et al., 1986; Belazi et al., 1994). On the whole, these theories draw parallels between the parse trees of a pair of parallel sentences in 2 languages and model code-mixing as the substitution of a subtree in one language with its equivalent in the other language, assuming that a set of conditions are satisfied. One of these theories is the Equivalence Constraint (EC) theory (Poplack, 1980; Sankoff, 1998). The work of Bhat et al. (2016) and Pratapa et al. (2018) make use of the EC theory to generate synthetic code-mixed sentences given a pair of parallel sentences. They show that using these generated sentences in language modeling showed an improvement in perplexity on a test set of non-synthetic sentences.

2.2. Generating Synthetic Code-Mixed Sentences

The method for generating code-mixed sentences in Bhat et al. (2016) first obtains equivalent parse trees of the parallel sentences in their respective languages. The method is briefly described here. The paper can be referred to for a detailed explanation. Assuming that $L1$ and $L2$ are the 2 languages,

1. Obtain a sentence in $L1$, its equivalent in $L2$ and a word level alignment between the two sentences
2. Obtain the parse tree of either of the sentences. If $L1$, is English, this can be done using a tool like the Stanford Parser (Klein and Manning, 2003)
3. Once the $L1$ tree is obtained, use the word level alignments to project the $L2$ words onto the $L1$ tree in a bottom-up manner. In this manner, the $L2$ tree is obtained

Figure 1a and 1b show monolingual trees for $L1$ and $L2$. Having these equivalent trees, EC theory allows the substitution of any number of words in the $L1$ tree with their equivalents in $L2$ tree as long as set of constraints are satisfied.



(a) English tree

(b) Hindi tree

(c) Valid code-mixed tree

(d) Invalid code-mixed tree

(e) Valid code-mixed tree

Figure 1: Monolingual and intermediate code-mixed trees. $L1$ is English and $L2$ is Hindi. Figure 1d is an invalid tree. The production being applied at the VP marked with * is the English production, which means that only the English productions can be applied for the terms on its RHS. However, the NP node on its RHS (marked with *) is deriving a Hindi word, which it is not allowed to, resulting in the tree being invalid. Had the English word been there (today, instead of आज), it would have been a valid sentence. Figures 1c and 1e are valid trees.

These constraints are detailed in the aforementioned paper. We will elaborate on one of these constraints in the next section. Since we are working at the parse tree level, we directly obtain the parse tree of the generated sentence every time we make substitutions. Figures 1c, 1d and 1e depict intermediate code-mixed trees obtained by making substitutions. The tree in Figure 1d is invalid and we explain why at the end of the next section.

2.3. EC Theory: Word Order

We focus on one of the EC theory constraints, as this will help us in better annotating the parse trees we obtain. For every production $u_0 \rightarrow u_1 u_2 ... u_n$ in the $L1$ tree, there must exist an equivalent production $v_0 \rightarrow v_1 v_2 ... v_n$ in the $L2$ tree such that

- u_0 and v_0 are the same non-terminal
- There exists a unique one-one mapping from the non-terminals in $u_1, u_2, ...u_n$ to the non-terminals in $v_1, v_2, ...v_n$

This simplifies down to 2 conditions: the LHS of the both the productions are the same and the RHS of both the productions have the same set of terms, possibly in a different order.

Given these definitions, we describe one of the constraints that are verified on the intermediate code-mixed tree (obtained after substitution of words from one language to the other). Starting in a bottom up manner, each non-terminal is assigned 2 language tags, one based on the word order of the production it's derived from and one based on the word order of the production being applied at it. If these tags match, this check continues up the tree. If the tags don't match at any point, the sentence is considered to be invalid.

This simplifies down to each production in the tree having a word order in its RHS that is either the $L1$ order or the $L2$ order, and that order determining which production ($L1$ or $L2$) can be applied for each term on the RHS. If the word order is same for both $L1$ and $L2$, either language's production can be applied for the terms on the RHS. Figure 1d shows an invalid code-mixed tree. The NP marked with * is assigned the tag of h from below as it is deriving a Hindi word and the tag e from above as the English word order is being applied at its parent node VP (marked with *), leading to a mismatch and making the sentence invalid.

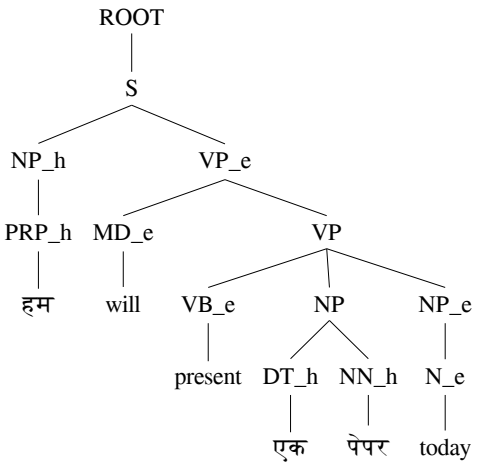


Figure 2: The final annotated code-mixed parse tree for tree in Figure 1e.

2.4. Annotating CM Parse Trees

We obtain the code-mixed trees directly each time we make substitutions in the monolingual trees. However, these simple trees do not capture the information provided by EC theory used in generating the tree. To address this, we annotate each non-terminal in the tree with a tag. This tag is determined by the language whose production (word order) was applied at that non-terminal. For leaf nodes, this would be the language of the word that takes it's place. For intermediate nodes, we added '_e' or '_h' depending on whether the $L1$ or $L2$ production was applied. This will ensure that the parser trained on these trees will learn the differences in word order for the productions in different languages. In cases where the word order is same for both languages, we do not add a tag. This is so that a parser will learn that the word order for that production would be same in both languages. Figure 2 shows the final tree generated by the process for the tree in Figure 1e.

3. Constituency Parsing

3.1. Background

Constituency parsing is the task of generating a valid parse tree given a sentence as input. One of the simplest methods for this task is the CKY algorithm (Younger, 1967). This algorithm takes in a set of CFG productions and builds up a tree for a sentence using a dynamic programming algorithm. There are variations of this algorithm that work with a Probabilistic CFG as well (Booth and Thompson, 1973).

Most of the early neural network parsers were simple encoder-decoder approaches where the sentence would be taken in by the encoder and the decoder would have to output the tree with no extra information being provided about a tree structure (Vinyals et al., 2015). These later evolved into methods where the decoder was constrained to output tokens that conformed to a valid tree structure (Ballesteros et al., 2015; Dyer et al., 2016). One negative aspect of these early neural methods is that they required extensive feature engineering to perform well (Thang et al., 2015).

3.2. Span Based Constituency Parsing

Span based parsing methods use a function to assign scores to spans in the sentence and use the CKY algorithm to build up the tree given these scores. Finkel et al. (2008) use Conditional Random Fields (CRFs) for the scoring purpose. More recently, there has been a line of work where neural networks have been used to score the spans, starting off with Durrett and Klein (2015) where a fixed-window based method is used. Stern et al. (2017) build upon this work by using RNNs instead of a fixed-window for the scoring and Kitaev and Klein (2018) use a transformer instead of the RNN. These methods achieve performance that is superior to the early neural network parsers without the complex feature engineering associated with most of them.

Dataset	Size	Height	RHS Length
En-Hi Train	421710	7.05 (1.96)	2.22 (2.34)
En-Hi Synth. Test	2740	7.10 (2.02)	2.21 (2.28)
En-Hi Real. Test	1381	5.40 (1.06)	3.38 (1.60)
En-Es Train	421710	8.16 (2.31)	1.75 (1.16)
En-Es Synth. Test	2542	8.13 (2.24)	1.73 (1.12)

Table 1: Statistics about Train and Test Datasets. Mean and Standard deviation (in brackets) reported for tree height and length of right hand size of productions.

3.3. Choice of Parser

We chose the span-based parser of Kitaev and Klein (2018)[1] for evaluating our trees. We chose this method as it requires only a set of trees as input for training and no information about the grammar of the language(s). This method achieves near state of the art performance on the Penn Treebank WSJ set.

This model runs the embedding of each token in the sentence through a transformer layer to produce contextual embeddings for each token, which are used to compute embeddings for each span in the sentence. This is then run through a scoring layer to produce scores for each span. These scores are used in a modified CKY-style parser to build up the most probable tree. For computing initial embedding of each token, we experiment with word embeddings over the combined vocabulary space of both languages and with multilingual BERT[2] (Devlin et al., 2019) (mBERT), which produces subword level embeddings. Lastly, the parsing model also learns to predict POS tags of tokens (using it while parsing), so we also report the POS tagging accuracy.

4. Evaluation

We created train, dev and test sets of synthetic trees from independent sets of parallel sentences, so there is no overlap in the trees between the 3 sets. Table 1 contains some statistics about the datasets. For both language pairs, we obtained parallel corpora by taking English sentences and running them through Bing Translator to obtain the parallel sentences. This allows us to perform this technique for languages that do not have large parallel corpora available for them. Since we are using an MT system, we end up with parallel sentences that are less likely to be semantic equivalents of each other and

[1] https://github.com/nikitakit/self-attentive-parser

[2] https://github.com/google-research/bert/blob/master/multilingual.md

	Hindi-English		Spanish-English	
Model	**Synth. Test**		**Synth. Test**	
	F1	POS	F1	POS
Word	38.22	96.07	33.20	90.23
mBERT	40.70	99.18	40.32	96.41

Table 2: Parsing F1 scores and POS tagging accuracies on Hindi-English and Spanish-English.

more likely to have one-one mapping between the words of both languages.

For Hindi-English, we used the sentences from the IITB Hi-En corpus (Kunchukuttan et al., 2018) which consists of sentences from multiple domains. For Spanish-English, we used the sentences from the corpus by Rijhwani et al. (2017) that mainly consists of sentences from social media (Twitter). We report parsing F1 scores for both these languages along with POS tagging accuracies. We call these test sets as "Synth. Test" as they contain synthetically generated trees. We report these results in Table 2.

4.1. Results

For Hindi-English we find that on our synthetic test set, we are able to get a F1 score of 38.22 using word embeddings. Using mBERT, we are able to get an increase of 2 points in the score. Both models are able to achieve high accuracies on POS tagging. For Spanish-English, similar to Hindi-English, using mBERT causes a boost in the F1 measure for parsing. For reference, we report results for English from (Kitaev and Klein, 2018) (which happens to be near state of the art), in which they obtain an F1 score of 92.67 using this same parsing technique.

4.2. Monolingual - Code-Mixed Performance Gap Analysis

Given that there is a big gap in the performance of our code-mixed parser and a state of the art (SOTA) parser on English, we perform a series of experiments with the aim of finding out the following: How much does each of the steps of our generation method contribute to the drop in the parser's performance? The processing done by our method can be divided into 3 stages:

1. Parsing the English sentence using the Stanford parser
2. Using alignments to project the English tree to obtain the $L2$ tree and an equivalent English tree
3. Substituting between the two trees to obtain a code-mixed tree

We already have the parsing results for trees produced after Step 3. We obtain trees produced after Step 1 and 2, train the parser on these trees and report the results on a test set. These trees are monolingual as code-mixing is done only in Step 3. We report results on the English trees that were used for Spanish-English code-mixing. These results are in Table 3.

We observe that there is a 20% drop in F1 comparing Step 1 to the SOTA English performance. This drop is due to

Parsing English Sentence

Model	F1
Word	67.98
mBERT	69.31

Obtaining equivalent English and $L2$ trees using alignements

Model	F1
Word	47.55
mBERT	50.93

Code-Mixing

Model	F1
Word	33.20
mBERT	40.32

Table 3: Parsing F1 scores and POS tagging accuracies after every step of our method. The first 2 tables are on monolingual (English) trees, while the third one is on code-mixed (Spanish-English) trees.

the errors introduced by using the Stanford parser to parse the English sentences. When we move from Step 1 to Step 2, we observe another 20% drop. This is the result of our method of projecting the English tree using alignments to obtain the $L2$ tree. The drop from Step 2 to Step 3 is around 10%. This is the actual complexity introduced to the parser by code-mixing, a much smaller difference than the 50% estimate from before.

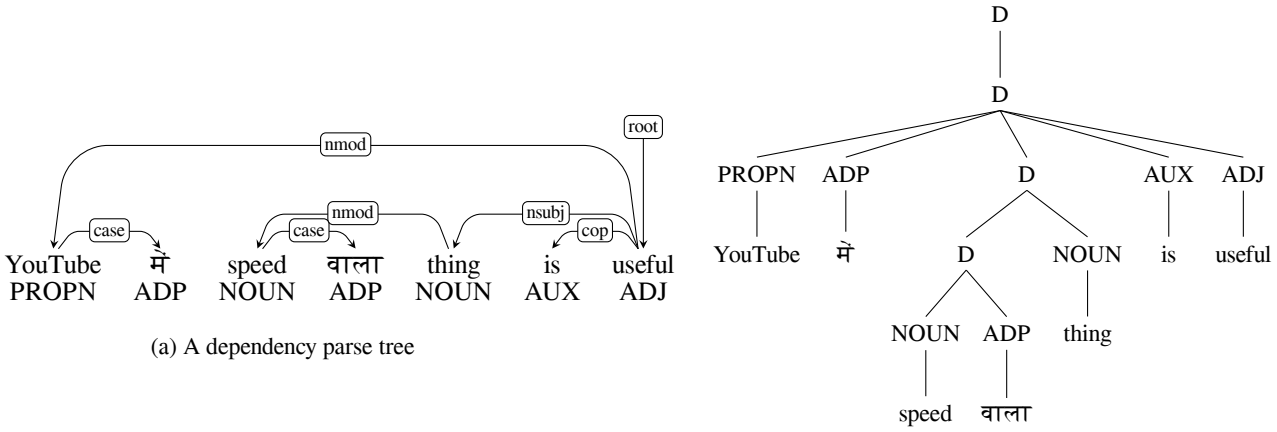
We can draw the following conclusions from this analysis. The complexity introduced by code-mixing brings in only a 10% drop in performance of the parser. The major reasons for the drop are the steps that obtain the parse trees for English and $L2$. Improvements to this technique can help in obtaining much better code-mixed parse trees.

5. Further Evaluation

5.1. Better Evaluation Methods: Using Non-Synthetic Trees

Evaluating the parser on synthetically generated trees alone is not sufficient. To get a thorough estimate of the usefulness of our synthetically generated trees, we have to take a parser trained on these trees and evaluate it on non-synthetic (real) trees. To accomplish, one would need a dataset of constituency parse trees of code-mixed sentences and to the best of our knowledge, such a dataset does not exist.

To address this and perform a more complete evaluation of our trees, we use of the work by Bhat et al. (2018) to come up with an evaluation technique. Their work proposes a Hindi-English code-mixed dependency parsing dataset. Dependency parse trees have a structure that is very different to constituency parse trees (see Figure 3). We convert these dependency trees to pseudo constituency trees and evaluate the parser with them.



(a) A dependency parse tree

(b) The converted constituency parse tree

Figure 3: A dependency parse tree and the constituency tree obtained by the conversion technique used.

5.2. Converting Dependency Trees to Constituency Trees

Although there has been a lot of recent work on converting dependency trees to constituency trees (Xia et al., 2008; Wang and Zong, 2010; Lee and Wang, 2016), most of these works require having the golden constituency tree for a dependency tree and train a machine learning based algorithm on such a golden tree set to learn the conversion. Since these resources are something not available for our case, we focus on the works of Collins et al. (1999) and Xia and Palmer (2001). These works propose an algorithm that will convert a dependency tree to a constituency tree in a deterministic fashion, outputting the structure alone of the constituency tree. This algorithm does not assign labels to intermediate nodes in the tree, a step that the aforementioned machine learning based algorithms try to achieve using labelled data.

5.3. Evaluation Methodology

We make use of Algorithm 2 from Xia and Palmer (2001) to convert the dependency trees in the train set of Bhat et al. (2018) (1381 tweets) to constituency trees. One aspect to note is that while this dataset is from Twitter, our synthetic trees are from a multiple-domain dataset. We assign the non-terminals without labels (all non-terminal except leaf level ones) the label 'D' and refer to the produced trees as pseudo constituency trees. Figure 3 shows a dependency parse tree and the pseudo constituency tree obtained by converting it. We use this as a test set on our trained parsers along with a metric we call "Unlabelled F1". This metric is needed as we don't have the labels for intermediate non-terminals, resulting in skewed scores being produced if we used the F1 score as per its original definition.

As per its original definition, the F1 score for parsing is calculated using precision and recall computed over successes and failures in the following manner: success is when a particular span in the sentence contains the same parent in the gold and the generated trees with the parent labels being the same, failure being otherwise. For our Unlabelled F1 measure, we relax the criterion of checking if the parent's labels match. We report this F1 score and POS accuracies in Table 4 under the Real Test column. We also calculate the Unlabelled F1 score for our Synth. Test set and report it and the POS accuracies(same value as in 2) for reference. Since we do not have English-Spanish dependency trees, we do not report any results on a Real Test set for that language pair.

Model	Real Test		Synth. Test	
	F1*	POS	F1*	POS
Word	30.32	40.25	46.93	96.07
mBERT	30.60	41.31	49.98	99.18

Table 4: *Unlabelled F1 and POS accuracies on the Hindi-English set of converted dependency trees.

5.4. Results & Error Analysis

We observe that the neural parser is able to perform decently on the Real Test set. There is a performance gap between the performance on this and on the synthetic test set. We also note that there is a huge gap in the POS tagging accuracy between the 2 sets. The domain mismatch between Real. Test and Synth. Test could contribute to the performance difference. We elaborate below on another possible reason.

Observing the distributions of tree height and production RHS length in Table 1, we observe a difference between the Train/Synth Test and Real Test sets. Given that these values (height, production RHS length) are integers and not continuous values, a difference of even 1 for their mean values is significant. We theorize that this is the side-effect of the algorithm used to convert dependency trees to constituency trees. In their work, Xia and Palmer (2001) refer to Algorithm 2, the algorithm that we've used, as the "Flattest Possible" algorithm, producing trees that are flatter (less in height) and wider (longer RHS of productions). This is clearly visible in our case, as the mean height is lesser and mean RHS length is more when comparing Real Test to both Train and Synth Test. Given this difference in the distribution between the train and test, the parser is not able to perform as well.

5.4.1. Error Analysis

We performed an analysis of where the parser makes errors and have listed some observations below. Appendix A con-

tains the gold tree from Real Test and the parser's predictions with more detailed explanations on how the parser is failing.

1. POS tag errors cause the parser to not capture smaller subtrees well

2. Longer sentences result in the parser not being able to capture any tree structure at all (i.e the root node derives most of the leaf nodes directly)

6. Conclusion

We present a technique to generate code-mixed parse trees given a pair of parallel sentences. We train a neural parser on these trees and report parsing F1 scores on a test set of generated trees. We also look into obtaining an evaluation set of non-synthetic trees and highlight the lack of such a resource in the community. Using an existing dependency parse resource, we evaluate our parser and observe that it is able to parse non-synthetic sentences as well, albeit not as well as it is able to perform on a set of synthetic sentences. The lack of code-mixed constituency parse set is something we've had to work around and the computational linguistics community would really benefit if such a resource exists.

A neural parser capable of performing on code-mixed sentences is a useful tool to have. Such a parser could be used to analyze code-mixed corpora and obtain statistics much more useful than values like Code-Mixing Index (CMI), Switching Point Fraction(SPF) etc.., statistics like what grammatical elements are likely to be switched more frequently and what are likely to be not. This information could further be used to sample from a set of large generated sentences to obtain more realistic sentences.

7. Bibliographical References

Aguilar, G., AlGhamdi, F., Soto, V., Diab, M., Hirschberg, J., and Solorio, T. (2018). Named entity recognition on code-switched data: Overview of the CALCS 2018 shared task. In *Proceedings of the Third Workshop on Computational Approaches to Linguistic Code-Switching*, pages 138–147, Melbourne, Australia, July. Association for Computational Linguistics.

Ballesteros, M., Dyer, C., and Smith, N. A. (2015). Improved transition-based parsing by modeling characters instead of words with lstms.

Belazi, H. M., Rubin, E. J., and Toribio, A. J. (1994). Code switching and x-bar theory: The functional head constraint. *Linguistic inquiry*, pages 221–237.

Bhat, G., Choudhury, M., and Bali, K. (2016). Grammatical constraints on intra-sentential code-switching: From theories to working models.

Bhat, I., Bhat, R. A., Shrivastava, M., and Sharma, D. (2018). Universal dependency parsing for hindi-english code-switching. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, volume 1, pages 987–998.

Booth, T. L. and Thompson, R. A. (1973). Applying probability measures to abstract languages. *IEEE transactions on Computers*, 100(5):442–450.

Collins, M., Hajic, J., Ramshaw, L., and Tillmann, C. (1999). A statistical parser for Czech. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 505–512, College Park, Maryland, USA, June. Association for Computational Linguistics.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June. Association for Computational Linguistics.

Di Sciullo, A.-M., Muysken, P., and Singh, R. (1986). Government and code-mixing. *Journal of linguistics*, 22(1):1–24.

Durrett, G. and Klein, D. (2015). Neural CRF parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 302–312, Beijing, China, July. Association for Computational Linguistics.

Dyer, C., Kuncoro, A., Ballesteros, M., and Smith, N. A. (2016). Recurrent neural network grammars. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 199–209, San Diego, California, June. Association for Computational Linguistics.

Finkel, J. R., Kleeman, A., and Manning, C. D. (2008). Efficient, feature-based, conditional random field parsing. In *Proceedings of ACL-08: HLT*, pages 959–967, Columbus, Ohio, June. Association for Computational Linguistics.

Joshi, A. K. (1982). Processing of sentences with intra-sentential code-switching. In *Coling 1982: Proceedings of the Ninth International Conference on Computational Linguistics*.

Kitaev, N. and Klein, D. (2018). Constituency parsing with a self-attentive encoder. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2676–2686, Melbourne, Australia, July. Association for Computational Linguistics.

Klein, D. and Manning, C. D. (2003). Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 423–430, Sapporo, Japan, July. Association for Computational Linguistics.

Kunchukuttan, A., Mehta, P., and Bhattacharyya, P. (2018). The IIT Bombay English-Hindi parallel corpus. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May. European Language Resources Association (ELRA).

Lee, Y.-S. and Wang, Z. (2016). Language independent dependency to constituent tree conversion. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 421–428, Osaka, Japan, December. The COLING 2016 Organizing Committee.

Milroy, J. (1995). *One speaker, two languages: Cross-disciplinary perspectives on code-switching*. Cambridge University Press.

Partanen, N., Lim, K., Rießler, M., and Poibeau, T. (2018). Dependency parsing of code-switching data with cross-lingual feature representations. In *Proceedings of the Fourth International Workshop on Computational Linguistics of Uralic Languages*, pages 1–17, Helsinki, Finland, January. Association for Computational Linguistics.

Poplack, S. (1980). Sometimes i'll start a sentence in spanish y termino en espanol: toward a typology of code-switching1. *Linguistics*, 18(7-8):581–618.

Pratapa, A., Bhat, G., Choudhury, M., Sitaram, S., Dandapat, S., and Bali, K. (2018). Language modeling for code-mixing: The role of linguistic theory based synthetic data. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1543–1553, Melbourne, Australia, July. Association for Computational Linguistics.

Rijhwani, S., Sequiera, R., Choudhury, M., Bali, K., and Maddila, C. S. (2017). Estimating code-switching on twitter with a novel generalized word-level language detection technique. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1971–1982, Vancouver, Canada, July. Association for Computational Linguistics.

Sankoff, D. (1998). The production of code-mixed discourse. In *Proceedings of the 17th international conference on Computational linguistics-Volume 1*, pages 8–21. Association for Computational Linguistics.

Sequiera, R., Choudhury, M., Gupta, P., Rosso, P., Kumar, S., Banerjee, S., Naskar, S. K., Bandyopadhyay, S., Chittaranjan, G., Das, A., et al. (2015). Overview of fire-2015 shared task on mixed script information retrieval.

Solorio, T., Blair, E., Maharjan, S., Bethard, S., Diab, M., Ghoneim, M., Hawwari, A., AlGhamdi, F., Hirschberg, J., Chang, A., and Fung, P. (2014). Overview for the first shared task on language identification in code-switched data. In *Proceedings of the First Workshop on Computational Approaches to Code Switching*, pages 62–72, Doha, Qatar, October. Association for Computational Linguistics.

Stern, M., Andreas, J., and Klein, D. (2017). A minimal span-based neural constituency parser. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 818–827, Vancouver, Canada, July. Association for Computational Linguistics.

Thang, L. Q., Noji, H., and Miyao, Y. (2015). Optimal shift-reduce constituent parsing with structured perceptron. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1534–1544, Beijing, China, July. Association for Computational Linguistics.

Vinyals, O., Kaiser, Ł., Koo, T., Petrov, S., Sutskever, I., and Hinton, G. (2015). Grammar as a foreign language. In *Advances in neural information processing systems*, pages 2773–2781.

Vyas, Y., Gella, S., Sharma, J., Bali, K., and Choudhury, M. (2014). POS tagging of English-Hindi code-mixed social media content. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 974–979, Doha, Qatar, October. Association for Computational Linguistics.

Wang, Z. and Zong, C. (2010). Phrase structure parsing with dependency structure. In *Coling 2010: Posters*, pages 1292–1300, Beijing, China, August. Coling 2010 Organizing Committee.

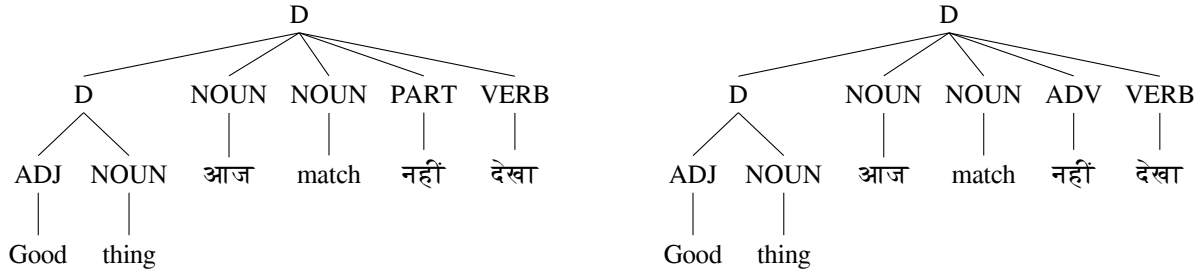
Xia, F. and Palmer, M. (2001). Converting dependency structures to phrase structures. In *Proceedings of the First International Conference on Human Language Technology Research*.

Xia, F., Rambow, O., Bhatt, R., Palmer, M., and Misra Sharma, D. (2008). Towards a multi-representational treebank. *LOT Occasional Series*, 12:159–170.

Younger, D. H. (1967). Recognition and parsing of context-free languages in time n3. *Information and control*, 10(2):189–208.

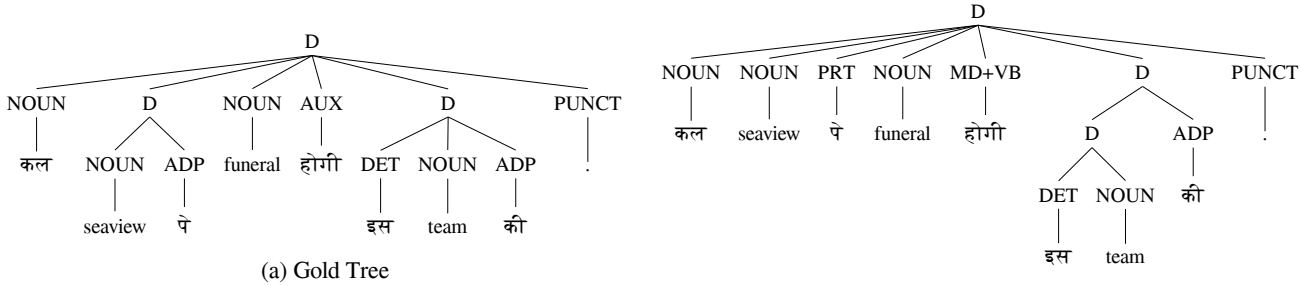
A. Error Analysis on Trees

Depicted below are 3 trees from Real Test and the parser's prediction for these trees. Gold trees are on the left and the parser's predictions are on the right. Errors made by the parser and possible reasons for the same have been mentioned along with the trees. The parsing F1 score for each tree is also reported, and the trees are ordered in descending order of this score.



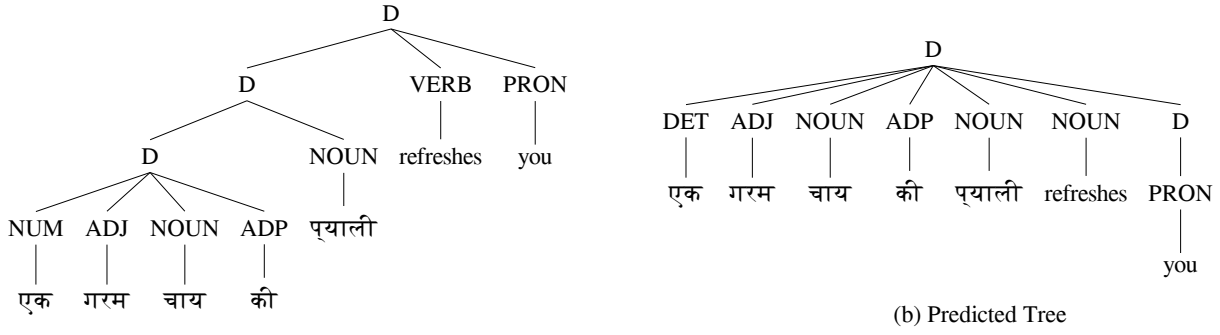
(a) Gold Tree (b) Predicted Tree

Figure 4: **F1: 100.0**: The parser predicts one of the POS tags incorrectly, but otherwise predicts the tree structure correctly.



(a) Gold Tree (b) Predicted Tree

Figure 5: **F1: 66.67**: The parser predicts the POS tag for पे incorrectly, leading to the subtree (D → NOUN ADP) not being captured properly.



(a) Gold Tree (b) Predicted Tree

Figure 6: **F1: 40.0**: There are a few POS tag errors in this case. As sentences get longer, the parser struggles to capture the tree structure of the original sentence and outputs a tree where the root node derives (almost) all the leaf nodes directly.