L95 Assignment
Comparing the Label Attention Layer Parser and the Attach Juxtapose Parser

Gabriele Dominici
University of Cambridge
gd489@cam.ac.uk

1 Introduction

Nowadays, constituency and dependency parsers have achieved formidable results. Therefore, the difference between the top parsers of the metric used to evaluate them is minimal. The common metric used to evaluate a constituency parser is the Parseval score (Black et al., 1991), and the ones for dependency parsing are the Unlabeled Attachment Score (UAS) and Labeled Attachment Score (LAS). They capture the ability of the parser to produce similar results to the gold standard, but they do not state anything about the peculiarity of the parser. In this way, it seems that each of the state-of-the-art parsers is identical to the others because all of them have a similar evaluation score. For this reason, it is necessary to execute in-depth qualitative evaluations to understand the peculiarities in parsers' outputs. For instance, it can be helpful to understand which parser is more convenient to use in a specific task.

Specifically, for the scope of this project, I decided to analyse in-depth two state-of-the-art parsers (Label Attention Layer Parser (LAL) (Mrini et al., 2019) and Attach Juxtapose (AJ) (Yang and Deng, 2020)). Both of them use an encoder-decoder architecture. LAL uses a similar encoder as AJ but with a Label Attention Layer at the end. Apart from that, they significantly differ in several aspects, like the decoder part and the ability to produce dependency trees. For example, LAL can build a sentence's constituency and dependency tree simultaneously, while AJ generates only the first one. To compare them also on the dependency parsing task, I used the Stanford parser v3.5.1[1] to translate the AJ's constituency trees into dependency trees. I chose these two parsers because they achieve similar scores on constituency parsing in two completely different ways. It would be interesting to see how they perform on eleven specific sentences and how different their outputs are.

To address the project's aims, I automatically evaluate the parsers' constituency trees of the eleven sentences with the Parseval score and the parsers' dependency trees with UAS and LAS. Then, I manually inspected every tree to point out the strengths and weaknesses of every parser. I show some examples of the common error made by each of them.

The results show how these two parsers build similar constituency trees on short and simpler sentences, but there are some discrepancies in longer and complex sentences. Moreover, these differences increase in the dependency parsing task, where their outputs diverge even on some short sentences.

The code, the gold standard and the predictions of the parsers are accessible in a GitHub repository[2].

2 Background

The parsers that I chose are built starting from some modules or ideas. I described them here to make easier the understanding of the following sections.

2.1 HPSG parser

HPSG parser (Zhou and Zhao, 2019) creates a simplified Head-driven Phrase Structure Grammar (HPSG) (Pollard and Sag, 1994) tree, which combines both constituency and dependency relationships. It can be seen as a constituency tree which uses the HEAD attribute of HPSG to define dependency relationships. The head word of the phrase corresponds to the parent of the head word of its children in the dependency tree.

Using these trees allows a model to be trained jointly on constituency and dependency, which

[1] https://nlp.stanford.edu/software/lex-parser.shtml

[2] https://github.com/gabriele-dominici/L95_assignment

leads the model to have two advantages. Firstly, its performance increases owing to receiving additional information from the other task and secondly, a single model can solve both tasks.

2.2 CKY

CKY (Sakai, 1961) is a bottom-up parsing algorithm used for context-free grammar. It starts with single words, combines them into more complex constituencies according to the grammar, and iteratively, they are combined to create even more complex constituencies.

It is widely used thanks to its worst-case performance equals to $\mathcal{O}(n^3|G|)$, where n represents the length of the sentence and $|G|$ the size of the grammar.

2.3 Biaffine Attention Mechanism

The Biaffine Attention Mechanism parser (Dozat and Manning, 2016) is one of the state-of-the-art models in the dependency parsing task. It uses a Bi-LSTM with a deep biaffine attention layer that computes the probability of the dependency edge between every possible head and dependant in the sentence.

At the time of its release, it achieved results which were comparable to other state-of-the-art models, even if being more faster. Since then, some of the best models released have used the Biaffine Attention Mechanism to perform dependency parsing (Zhou and Zhao, 2019; Mrini et al., 2019).

2.4 Transition-based constituency parser

Transition-based parsers execute a series of actions to build the constituency tree one step at a time. A famous transition-based algorithm is the shift-reduce parser. It uses two buffers, S contains the subtrees created and B the remaining words to insert into the tree. It starts with S empty and B containing all sentence words. Then, it can perform two actions: *shift* remove the first element from B and put it into S, and *reduce* combine together two subtrees in S. The algorithm ends successfully when the B buffer is empty and S contains the complete parse tree.

There are different shift-reduce variants (Sagae and Lavie, 2005; Dyer et al., 2016) which modify the original approach. One of these is the In-Order Shift Reduce algorithm (Liu and Zhang, 2017), which only allows having a single subtree or terminal nodes in buffer S. It is the most similar variant to an incremental parser, according to (Yang and Deng, 2020).

3 Data used

To compare these two parsers, I have used the given eleven sentences, which vary from short and simple sentences to long and complex ones. The gold standard contains part of speech tag labels, constituency trees and RASP dependency trees (Briscoe et al., 2006).

However, both parsers used in this project produce Stanford dependency trees (de Marneffe and Manning, 2008). Therefore, I manually translated the RASP dependency relations to the Stanford ones. I tried to stick as much as possible to the relationships in RASP, but sometimes it was impossible. For instance, the *cc* and *conj* relationships in Stanford dependency have a completely different structure than the *conj* relation in RASP.

Moreover, the gold standard split hyphenated words (words that contain -), so I manually split the same words in the original sentences because both parsers considered them single words.

4 Label Attention Layer Parser

The Label Attention Layer Parser[3] (Mrini et al., 2019) (LAL) inserted in the HPSG parser, proposed by Zhou and Zhao (2019), a Label Attention Layer (Figure 1). It is a modified version of the classical self-attention, which allows the model to be more efficient and interpretable. It is more efficient because it uses a query vector instead of a query matrix inside the attention head, reducing the number of parameters. Furthermore, it is more interpretable owing to the learnt relation of every head with a specific syntactic category. The model has at least heads as the number of labels, and past experiments showed that some heads' contribution was mainly specific to a certain label.

LAL is an encoder-decoder model. The encoder is common to both tasks (XLNET (Yang et al., 2019)) and creates the word representations used at the next step to predict the constituency and dependency trees. It comprises a stack of self-attention layers, followed by some Label Attention Layers. Moreover, sentences are preprocessed according to Zhou and Zhao (2019) methods and HPSG trees are represented with the joint span representation (Zhou and Zhao, 2019).

[3] https://github.com/KhalilMrini/LAL-Parser

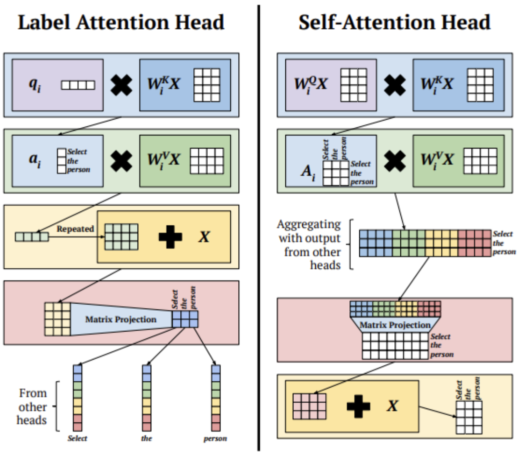

Figure 1: Image from Mrini et al., 2019, which shows the difference between the Label Attention Layer and Self-Attention Layer.

The model is jointly trained on the constituency and dependency tasks, so the loss of the model is the sum of the two specific losses described in Section 4.1 and Section 4.2, respectively. However, at training time, they used a CKY algorithm to predict the most probable HPSG tree starting from the constituency and dependency parsing outputs.

4.1 Constituency Parsing

To build the constituency parsing, the model calculates the score for each possible span in the sentence. First, the span representations are computed starting from the representations of words inside it. Then, a one-layer feed-forward network with layer normalisation is used to compute their score. Finally, a CKY algorithm is used to discover the best scoring tree with the highest sum of its span scores. The loss function related to this output is the hinge loss.

4.2 Dependency Parsing

The dependency tree is built using the Biaffine Attention Mechanism, which computes the probability of every word being the child of every other possible word in the sentence. This time, the model tries to minimise the negative log-likelihood of the correct dependency tree.

4.3 General considerations

LAL parser achieved state-of-the-art performance on constituency and dependency parsing tasks by exploiting the joint training. Moreover, using Label Attention Layer makes the model more interpretable. It allows us to inspect single heads and understand why the model mispredicts a label.

However, it does not predict part-of-speech tags but uses the Stanford tagger (Toutanova et al., 2003).

5 Attach-Juxtapose

Yang and Deng (2020) presented a novel transition-based system to tackle constituency parsing. It is called Attach-Juxtapose and incrementally builds a constituency tree inspired by psycholinguistic works (Marslen-Wilson, 1973; Sturt and Lombardo, 2005). It performs a series of actions (Figure 2), which compose a tree incrementally, starting from the first word and processing one subsequent word at a time. It has two possibilities: *attach* inserts a new node and optionally its parent in the tree, and *juxtapose* creates a new non-terminal node in the tree (internal node), which becomes the parent of the word that has to be added and of an already existing node in the tree. Therefore, at every step, there is always a single tree. Moreover, Yang and Deng (2020) demonstrates how this method is similar to shift-reduce algorithms, particularly the In-Order Shift Reduce, but with a smaller state space.

The Attach-Juxtappose parser (AJ)[4] uses this transition system and predicts the series of actions with an encoder-decoder model. Specifically, it uses the same encoder (XLNET) as the LAL parser and other parsers (Zhou and Zhao, 2019; Kitaev and Klein, 2018) to obtain word representations but then uses a GNN model to predict action to build the tree.

5.1 Constituency Parsing

The model starts with an empty tree, and at every step, it adds one word. Every time it should happen, the tree is passed to a Graph Convolutional Network (GCN) (Kipf and Welling, 2016), which spreads information through the tree. Initially, the tree leaves are initialised with the word representations given by the encoder, while non-terminal nodes are represented with a combination of the words inside that span. After several GCN layers, it uses the representation of the nodes that the next action can impact to predict it. At this step, the model has to decide where performs the action on the tree (target node) and, eventually, the labels of the related parent (parent node) and the internal

[4] https://github.com/princeton-vl/attach-juxtapose-parser

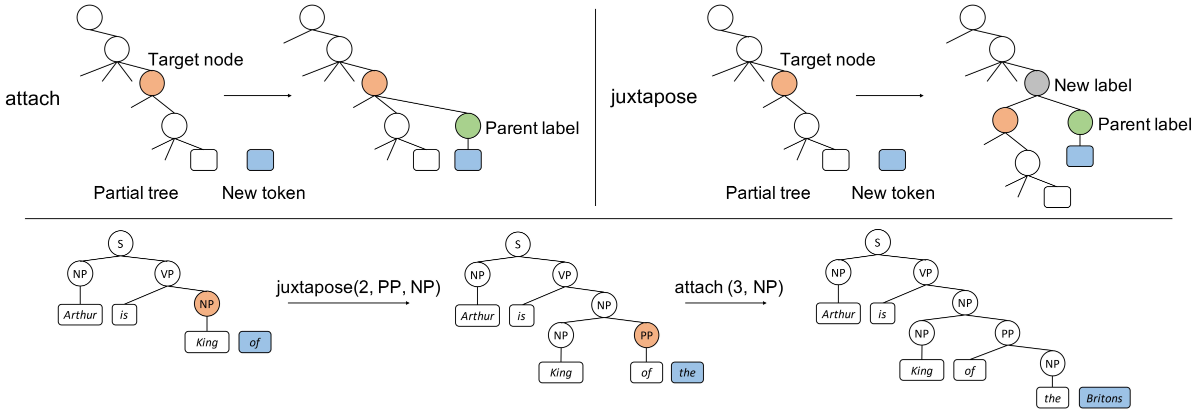

Figure 2: Image from Yang and Deng, 2020, which shows the possible action of the Attach-Juxtapose method.

node. The action can be inferred from the output of the model. Therefore, if the internal node is not predicted, the action to perform is *attach*, otherwise is *juxtapose*.

The model is trained by comparing the action chosen by the model with the oracle action, which is retrieved starting from the gold standard constituency tree. A constituency tree is always decomposable in a unique series of actions (oracle actions) that can be performed by the Attach-Juxatpose algorithm, according to (Yang and Deng, 2020). The loss function used by this parser is the sum of cross-entropy losses computed on each prediction element (target node, parent label and internal node label).

5.2 Dependency Parsing

AJ produces only constituency trees, so I used the Stanford parser v3.5.1[5] to transform them into dependency trees. It uses a transition-based system similar to the arc-standard system (Nivre, 2008), but it uses a neural network to choose the transaction among the possible ones.

The arc-standard system performs a series of actions creating a dependency tree, starting from a stack s, which contains the `ROOT` node, a buffer b which contains the words of the sentence and an empty set of relations A. At every step, it performs one of three actions: *left-arc* adds a left arc among two elements in s, the dependant is removed from s, and the relation is added to A; *right-arc* does the same but with a right arc; *shift* takes the first word from b and pushes it to s. It ends when b is empty, and s contains only the node `ROOT`.

The neural network is used to choose which action to perform, receiving the state of the stack and the buffer as input (Chen and Manning, 2014).

5.3 General Consideration

This parser achieved results comparable to other state-of-the-art model training only on the constituency parsing task, unlike most of the others. Another advantage of this parser is the possibility of producing valid trees from partial sentences because, at every step, there always be a single tree. If it used a unidirectional encoder, the architecture was fully incremental, allowing it to parse online sentences.

A little drawback that I found using it is the impossibility of parsing the `-` token. To handle this problem, I deleted the ones related to hyphenated words and reinserted them after constituency parsing. On the other hand, I substituted the ones that appear in pair with open and closing round brackets and changed back after constituency parsing together with their correct part-of-speech tag (`HYPN`).

Another crucial aspect to remember is that I employed two parsers and not only one to produce constituency and dependency trees.

6 Comparison

It is visible how these two parsers diverge consistently in their implementations while being extremely performant on constituency parsing.

They have completely different strategies for building trees. One uses attention mechanisms that exploit the information from the entire sentence (bi-directional), while the other uses GNNs to spread information among the already-seen words of the sentence (uni-directional). Moreover, the first one exploits extra information by creating a

[5] `https://github.com/dmcc/PyStanfordDependencies`

simplified HPSG tree to combine dependency and constituency trees, while the first focuses only on constituency relations. They also differ in the approach to building the best tree. The LAL parser uses the CKY algorithm to find the most probable tree, inspecting all the possibilities using the whole sentence. On the other hand, the AJ parser creates the tree incrementally, proposing a new transition method that uses only previous words.

The only shared details among these two parsers are the encoder-decoder backbone and the encoder itself. They both use XLNET (Yang et al., 2019) to compute the words' representations in the following steps. It means they start in the same way, using the same information, but they differ in how they use them. This is even more interesting because it is a comparison between two parsers and shows how the results can still differ using the same input representation.

However, I compared both parsers on the eleven sentences given on constituency and dependency parsing tasks. Firstly, I evaluated them automatically using some standard metrics. Then, I evaluated them qualitatively, analysing common errors.

Section A and Section B show, respectively, constituency and dependency trees mentioned in the following sections of both parsers and the gold standard ones.

6.1 Constituency Parsing

6.1.1 Quantitative evaluation

Table 1 shows the results of both parsers on every sentence and their mean. I used several metrics to capture different aspects of the proposed trees by the parsers. I evaluate them using classic Parseval scores[6] (Precision, Recall and F1-Score), which check how much different two trees are comparing nodes and their spans. Then, I used the Edit Distance score[7], which calculates how many actions are needed to transform one tree to another, the Accuray of the POS tagger, and the number of crossed brackets.

Results are similar because, overall, the analysed parsers propose similar trees. This is even more true for the first sentences, which are short and simpler than the last ones. This behaviour on simple examples could be caused by being trained on the same dataset and achieving extremely high performance.

Overall, the AJ parser outperforms the LAL parser on each metric used, but this widely depends on the poor performance of the LAL parser on the last sentence. On the others, the performances of the parsers are almost identical. Moreover, the POS tagger used by the AJ parsers always performed equally or better than the LAL's one, which implies a lower Edit Distance score. More tags labelled correctly means fewer renaming node actions.

Another noticeable thing which is valid for both examined parsers is that looking only at the number of crossed brackets, it does not seem that the trees we are comparing (predicted vs gold) are too different, or at least most of the common non-terminal nodes to both trees contain the same words. On the contrary, Parseval metrics and the Edit Distance suggest that there are many more differences in the trees.

It is hard to say anything more about the differences between these two parsers and their errors, using only these automatic evaluation metrics.

6.1.2 Qualitative evaluation

Comparing the prediction of the parsers with the gold standard manually can show many more details. On the other hand, it takes ages to inspect hundreds of them, so Kummerfeld et al. (2012) developed an algorithm which automatically categorises the errors made by a parser. It is very helpful to understand the behaviour of a parser on a dataset, and it shows the importance of evaluating parsers not only with metrics. However, in this project, it is not necessary for only eleven sentences, so I inspect them manually.

The first thing it is possible to notice is that the low Parseval scores (Recall in particular) on the first sentences are due to the different branching factors. Gold standard trees are almost always binary trees, while both parsers built non-binary trees. It causes to have fewer non-terminal nodes than the right tree, and it makes impossible to match gold standard labels. Despite that, the overall structure remains the same, leading to fewer crossed brackets. Figure 3 shows an example of this issue, which happens in all sentences, but is the only difference in sentences 2, 3, 4, 5 and 7. Parsers' predictions are the same for these sentences, and they are reasonable to me as much as the gold standard. In sentence 8, it happens the same, but this time, in the gold standard, the first non-terminal node has three children instead of 2 because of the binary tree. For this specific sentence, I prefer the solu-

[6] https://github.com/flyaway1217/PYEVALB

[7] https://github.com/JoaoFelipe/apted

		LAL parser						AJ parser					
No.	**# words**	**P**	**R**	**F1**	**#CB**	**ED**	**POS**	**P**	**R**	**F1**	**#CB**	**ED**	**POS**
1	10	0.83	0.62	0.71	1	6	0.80	0.83	0.62	0.71	1	5	0.90
2	10	0.83	0.56	0.67	0	6	0.90	0.83	0.56	0.67	0	5	1.00
3	11	0.75	0.60	0.67	0	9	0.64	0.75	0.60	0.67	0	6	0.91
4	10	0.83	0.45	0.59	0	8	0.90	0.83	0.45	0.59	0	8	0.90
5	15	0.62	0.67	0.64	0	10	0.93	0.62	0.67	0.64	0	10	0.93
6	16	0.75	0.63	0.68	1	10	0.94	0.75	0.63	0.68	1	10	0.94
7	21	0.78	0.54	0.64	0	21	0.71	0.78	0.54	0.64	0	19	0.81
8	27	0.83	0.59	0.69	0	19	0.89	0.87	0.59	0.70	0	19	0.89
9	53	0.62	0.44	0.52	4	52	0.72	0.59	0.42	0.49	5	42	0.92
10	18	0.70	0.39	0.50	2	14	0.94	0.78	0.39	0.52	2	14	0.94
11	40	0.37	0.28	0.31	10	49	0.82	0.57	0.44	0.50	4	34	0.93
Overall		0.67	0.49	0.56	1.64	18.5	0.81	**0.70**	**0.51**	**0.59**	**1.18**	**15.6**	**0.91**

Table 1: Automatic evaluation of the LAL parser and AJ parser on the eleven given sentences on constituency parsing. The last row shows the overall results. P, R and F1 are Precision, Recall and F1-Score, respectively, of Parseval. #CB means the number of Cross Bracketing, #ED represents the Edit Distance, and POS is the POS tag accuracy.

tion proposed by both parsers because they fully separate the first S from the second one. Figure 4 shows the gold tree and the tree produced by the AJ parser. Both parsers give a similar and reasonable prediction, but they differ slightly on the construction of the subtree for "old-fashioned means". The LAL parser separates "old-fashioned" to "means" under an ADJP tag, similarly to what is done in the gold standard, while the AJ parser puts all of them at the same level in an NP node.

Another difference between parsers' prediction and the gold standard is how they handle possession words in sentences 1 and 6. Figure 5 shows the subtrees for "my aunt's car" of sentence 6. Both parsers divide "my aunt's" from "car", which is reasonable to me because "my" is referred to "aunt" and not to the following NP, as written in the gold standard. It would be classified as an NP Internal Structure error by Kummerfeld et al. (2012). The same happened in sentence 1 with "my aunt's can opener". Again, both parsers act similarly, in contradiction to the gold standard.

In the first eight sentences, there are no significant differences between the two analysed parsers, and they seem to be more prone to the same errors, but as the complexity of the sentence grows, the dissimilarities increase too. For instance, in sentence nine, the LAL parser built a more similar subtree to the gold tree for "more or less unique functions" compared to AJ's prediction. In the gold and LAL trees, "more or less" modify "unique function", while the AJ parser unifies "more or less unique" under an ADJP node. AJ's solution is farther from the gold standard, but I prefer it because "more or less" refers to "unique" and together acts as a modifier of "functions". Figure 6 shows these differences.

In sentence ten, the LAL parser treats "87 - tag" as an ADJP node in "the original 87 - tag tagset" NP, which is correct, while the AJ parser made a Missing Node error (Kummerfeld et al., 2012) by putting them at the same level of the other words. Figure 7 shows this error. Differently, the gold standard labels "87 - tag" as an ADJP on its own, but it has another ADJP which contains "the original 87 - tag" and modifies the NP node of "tagset for the Brown Corpus". In my opinion, this is an error because the bigger ADJP node modifies only the word "tagset" and not the others.

The second half of the last sentence shows more differences. The AJ parser produced a more similar tree to the gold one than the LAL parser. It happened because the LAL parser separated "the different forms of the verbs" from "do (...), have and be", which are the examples of the verbs and are strictly related, while on the tree, they are considerably distant. It is translated in a high Edit Distance score. Moreover, the AJ parser separated words better also in the last part of the sentence, where there were examples inside brackets related to the word "do". On the contrary, the LAL parser put them at the same level as the words "do", "have" and "be".

To sum up, both parsers built trees similar to

		LAL parser		**AJ parser**	
No.	**# words**	**UAS**	**LAS**	**UAS**	**LAS**
1	10	0.67	0.67	1.00	0.89
2	10	1.00	1.00	1.00	1.00
3	11	0.80	0.80	1.00	1.00
4	10	0.89	0.89	0.89	0.89
5	15	0.93	0.86	0.93	0.86
6	16	0.86	0.80	0.86	0.80
7	21	0.75	0.75	0.75	0.75
8	27	1.00	0.96	0.79	0.79
9	53	0.83	0.63	0.71	0.54
10	18	1.00	0.87	0.94	0.87
11	40	0.50	0.50	0.61	0.58
Overall		**0.81**	**0.75**	0.80	**0.75**

Table 2: Automatic evaluation of the LAL parser and AJ parser on the eleven given sentences on dependency parsing. The last row shows the overall results.

the gold standard, which were penalised from the metrics due to different branching factors. However, their behaviour was similar in most of the sentences, producing most of the time the same kind of errors. This could suggest that the representations computed by the encode, which was sheared by both parsers, significantly impacted the results. On the other hand, when they received long and complex sentences, they produced different predictions with different kinds of errors.

6.2 Dependency Parsing

6.2.1 Quantitative evaluation

Table 2 shows the results of the two parsers on the eleven sentences. I evaluate them using the two classical metrics of the dependency parsing task: Unlabeled Attachment Score (UAS) and Labeled Attachment Score (LAS). They compare dependency relations without and with the label, respectively. For the project's scope, I avoided comparing relations with punctuation as dependent because, in the gold standard, they were missing.

Overall, these two parsers achieve similar scores in both metrics. The LAL parser scores slightly higher on UAS than the AJ parser, which achieves a marginally higher LAS score. However, looking at the performance on a single sentence, the AJ parser outperforms the other on short sentences, while, generally, LAL performs better than AJ on the longest sentences. A drop in the LAL performance in the last sentence might be caused by its poor performance on this sentence in constituency parsing. It is important to remember that the dependency parsing tree built by the LAL parser is strictly related to the constituency one.

Even this time, the results of these two parsers are close, making difficult to choose the best one by looking only at these results. It is even more arduous to say why one parser should best than the other.

6.2.2 Qualitative evaluation

The differences between the two parsers on this task are more than in the previous one. However, there is no clear pattern of recurrent errors, but there is a variety of them in the proposed solution for these eleven sentences. All the predictions are shown in Section B.

The first difference is visible in sentence 1. It has a challenging structure ("my aunt's can opener can open"), where the first "can" is a noun that modifies "opener", and the second is the auxiliary of "open". Therefore, LAL mispredicted the structure and connected both "can" to the verb "open" with the label `aux`. On the other hand, AJ perfectly predicts the whole sentence. Another error made by the LAL parser is in sentence 9. It connects "more or less" to the word "functions" instead of "unique". It is inherited by the constituency tree built by this parser, where it made the same error.

In sentence 9, also the AJ parser made a mistake. It mislabelled one of the dependant of the word "interjections". It predicted the `conj` relationship between "interjections" and "existential" instead of "there", which was modified by "existential". Furthermore, the AJ parser had problems with the hyphenated words because it did not connect the first part of the word to the second. For instance, in sentence 8 ("old - fashioned means"), it predicts an `amod` relation between "old" and "means" instead of predicting the same kind of relationship between "old" and "fashioned". The same happened in sentence 10 with the words "87 - tag tagset".

However, their behaviour is similar in some cases, and they make the same mistakes. A common error is to mispredict the `ref` dependency. It happens when a dependant of the head of an `NP` is the relative word introducing the relative clause modifying the NP. In sentences 5, 6 and 7, both parsers connect the relative word to the head of the relative clause with a `dobj` relationship instead of predicting a `ref` relation between the head of the `NP` and the relative word.

Moreover, both parsers are confused by words inside brackets. It happens where there is not only a list of examples but small sentences inside them. Therefore they were misled by the last brackets in sentence 9 and by brackets in sentence 11. However, they act in different ways in these scenarios.

In conclusion, each parser has a peculiar behaviour in some situations. For instance, the AJ parser handles wrongly hyphenated words, while the LAL parser strictly depends on the constituency tree it creates, being more prone to make the same mistakes in the two different tasks. On the contrary, many times, they blunder in the same way.

7 Discussion

None of the two parsers analysed outperforms the other clearly. Their performance is similar in both tasks if evaluated with common metrics, despite the differences in their architectures and approaches. However, looking at predicted trees, there are some discrepancies between them, but none of their error has a significant weight to prefer one instead of another. For instance, the LAL parser divides better a series of multiple adjectives before a noun in both tasks, while the AJ parser predicts better dependency trees for challenging situations where two equal words have a different meanings.

Another aspect which was not tested in this project is the parsing speed. However, according to Yang and Deng (2020), the AJ parser is slightly faster in producing constituency trees (33.9s to parse the whole PTB test set) than the LAL parser (40.8s).

To conclude, to understand these parsers better, more analyses can be done in further research. For example, more sentences should be inspected to understand the errors they are more prone to commit and categorise them using the approach proposed by Kummerfeld et al. (2012). Another interesting approach is to test their prediction in real downstream tasks, like Machine Translation or Information Extraction (Quirk and Corston-Oliver, 2006; Miyao et al., 2008).

8 Conclusion

For the scope of the project, I analysed two different parsers. The first one is the LAL parser, which uses a Label Attention Layer to make the model more efficient and interpretable. Using HPSG simplified trees, it produced both constituency and dependency relations. On the other hand, the Attach Juxtapose parser employed a Graph Neural Network to choose the action to perform in a new proposed transaction-based system: attach-juxtapose. It incrementally creates a constituency tree. To build the dependency tree, this time, I translated its predictions with the Stanford parser v3.5.1.

I compared them on both tasks with some standard metrics and manually each of the predictions given. Looking at them, no parser outperforms the other clearly or produces more important errors. However, each of them has its own behaviour in challenging situations, and their discrepancies increase with the sentence length.

It is difficult to choose the best parser because it depends on the real task where they should be employed. If the user needs both dependency and constituency tree, I would suggest using the LAL parser because of the less overhead and, overall, on both tasks, it produced more consistent predictions. On the other hand, the AJ parser is a slightly better solution for a constituency parsing tree because it is slightly faster and has also demonstrated to produce solutions with non-crucial errors. Moreover, its transition-based system can be employed in real-time applications.

References

E. Black, S. Abney, D. Flickenger, C. Gdaniec, R. Grishman, P. Harrison, D. Hindle, R. Ingria, F. Jelinek, J. Klavans, M. Liberman, M. Marcus, S. Roukos, B. Santorini, and T. Strzalkowski. 1991. A procedure for quantitatively comparing the syntactic coverage of English grammars. In *Speech and Natural Language: Proceedings of a Workshop Held at Pacific Grove, California, February 19-22, 1991*.

Ted Briscoe, John Carroll, and Rebecca Watson. 2006. The second release of the rasp system. page 77–80.

Danqi Chen and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750, Doha, Qatar. Association for Computational Linguistics.

Marie-Catherine de Marneffe and Christopher D. Manning. 2008. The Stanford typed dependencies representation. In *Coling 2008: Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation*, pages 1–8, Manchester, UK. Coling 2008 Organizing Committee.

Timothy Dozat and Christopher D. Manning. 2016. Deep biaffine attention for neural dependency parsing. *CoRR*, abs/1611.01734.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 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. Association for Computational Linguistics.

Thomas N. Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks.

Nikita Kitaev and Dan Klein. 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. Association for Computational Linguistics.

Jonathan K. Kummerfeld, David Hall, James R. Curran, and Dan Klein. 2012. Parser showdown at the Wall Street corral: An empirical investigation of error types in parser output. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1048–1059, Jeju Island, Korea. Association for Computational Linguistics.

Jiangming Liu and Yue Zhang. 2017. In-order transition-based constituent parsing. *Transactions of the Association for Computational Linguistics*, 5:413–424.

William Marslen-Wilson. 1973. Linguistic Structure and Speech Shadowing at Very Short Latencies. , 244(5417):522–523.

Yusuke Miyao, Rune Sætre, Kenji Sagae, Takuya Matsuzaki, and Jun'ichi Tsujii. 2008. Task-oriented evaluation of syntactic parsers and their representations. In *Proceedings of ACL-08: HLT*, pages 46–54, Columbus, Ohio. Association for Computational Linguistics.

Khalil Mrini, Franck Dernoncourt, Trung Bui, Walter Chang, and Ndapa Nakashole. 2019. Rethinking self-attention: An interpretable self-attentive encoder-decoder parser. *arXiv preprint arXiv:1911.03875*.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553.

Carl Pollard and Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. The University of Chicago Press, Chicago.

Chris Quirk and Simon Corston-Oliver. 2006. The impact of parse quality on syntactically-informed statistical machine translation. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 62–69, Sydney, Australia. Association for Computational Linguistics.

Kenji Sagae and Alon Lavie. 2005. A classifier-based parser with linear run-time complexity. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 125–132, Vancouver, British Columbia. Association for Computational Linguistics.

Itiroo Sakai. 1961. Syntax in universal translation.

Patrick Sturt and Vincenzo Lombardo. 2005. Processing coordinated structures: Incrementality and connectedness. *Cognitive Science*, 29(2):291–305.

Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 252–259.

Kaiyu Yang and Jia Deng. 2020. Strongly incremental constituency parsing with graph neural networks.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *CoRR*, abs/1906.08237.

Junru Zhou and Hai Zhao. 2019. Head-Driven Phrase Structure Grammar parsing on Penn Treebank. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2396–2408, Florence, Italy. Association for Computational Linguistics.

A Constituency parsing trees

This section shows the complete outputs of both parsers for the constituency parsing and the figures of the constituency trees mentioned in Section 6.1.2.

Sentence 1

GOLD (S (NP (PRP$ My) (NP (NN aunt) (NP (POS 's) (NP (NN can) (NN opener))))) (VP (MD can) (VP (VB open) (NP (DT a) (NN drum)))) (. .))

LAL (S (NP (NP (PRP$ My) (NN aunt) (POS 's)) (MD can) (VB opener)) (VP (MD can) (VP (VB open) (NP (DT a) (NN drum)))) (. .))

AJ (S (NP (NP (PRP$ My) (NN aunt) (POS 's)) (MD can) (NN opener)) (VP (MD can) (VP (VB open) (NP (DT a) (NN drum)))) (. .))

Sentence 2

GOLD (S (NP (DT The) (NP (ADJP (JJ old)) (NN car))) (VP (V (VBD broke) (RP down)) (PP (IN in) (NP (DT the) (NP (NN car) (NN park))))) (. .))

LAL (S (NP (DT The) (JJ old) (NN car)) (VP (VBD broke) (PRT (RB down)) (PP (IN in) (NP (DT the) (NN car) (NN park)))) (. .))

AJ (S (NP (DT The) (JJ old) (NN car)) (VP (VBD broke) (PRT (RP down)) (PP (IN in) (NP (DT the) (NN car) (NN park)))) (. .))

Sentence 3

GOLD (S (NP (ADVP (RB At) (ADVP (RBS least) (CD two))) (NNS men)) (VP (VP (V (VBD broke) (RP in))) (CONJP (CC and) (VP (VBD stole) (NP (PRP$ my) (NN TV))))) (. .))

LAL (S (NP (QP (IN At) (JJS least) (CD two)) (NNS men)) (VP (VP (VBD broke) (PRT (IN in))) (CC and) (VP (VB stole) (NP (PRP$ my) (NN TV)))) (. .))

AJ (S (NP (QP (RB At) (JJS least) (CD two)) (NNS men)) (VP (VP (VBD broke) (PRT (RP in))) (CC and) (VP (VBD stole) (NP (PRP$ my) (NN TV)))) (. .))

Sentence 4

GOLD (S (NP (NP (NNP Kim)) (CONJP (CC and) (NP (NNP Sandy)))) (VP (ADVP (RB both)) (VP (V (VBD broke) (RP up)) (PP (IN with) (NP (PRP$ their) (NNS partners))))) (. .))

LAL (S (NP (NNP Kim) (CC and) (NNP Sandy)) (DT both) (VP (VBD broke) (PRT (RP up)) (PP (IN with) (NP (PRP$ their) (NNS partners)))) (. .))

AJ (S (NP (NNP Kim) (CC and) (NNP Sandy)) (DT both) (VP (VBD broke) (PRT (RP up)) (PP (IN with) (NP (PRP$ their) (NNS partners)))) (. .))

Sentence 5

GOLD (S (NP (NP (DT The) (NN horse)) (CONJP (ADVP (RB as) (ADVP (RB well)) (ADVP (RB as))) (NP (NP (DT the) (NNS rabbits)) (S (WDT which) (S (PRP we) (VP (VBD wanted) (VP (TO to) (VB eat)))))))) (VP (VBZ has) (VP (VBN escaped))) (. .))

LAL (S (NP (NP (DT The) (NN horse)) (CONJP (RB as) (RB well) (IN as)) (NP (NP (DT the) (NNS rabbits)) (SBAR (WHNP (WDT which)) (S (NP (PRP we)) (VP (VBD wanted) (S (VP (TO to) (VP (VB eat))))))))) (VP (VBZ has) (VP (VBN escaped))) (. .))

AJ (S (NP (NP (DT The) (NN horse)) (CONJP (RB as) (RB well) (IN as)) (NP (NP (DT the) (NNS rabbits)) (SBAR (WHNP (WDT which)) (S (NP (PRP we)) (VP (VBD wanted) (S (VP (TO to) (VP (VB eat))))))))) (VP (VBZ has) (VP (VBN escaped))) (. .))

Sentence 6

GOLD (S (NP (PRP It)) (VP (VBD was) (NP (NP (PRP$ my) (NP (NN aunt) (NP (POS ’s) (NN car)))) (S (WDT which) (S (NP (PRP we)) (VP (VP (VP (VBD sold) (PP (IN at) (NP (NN auction)))) (ADVP (RB last) (NP (NN year)))) (PP (IN in) (NP (NNP February)))))))) (. .))

LAL (S (NP (PRP It)) (VP (VBD was) (NP (NP (NP (PRP$ my) (NN aunt) (POS ’s)) (NN car)) (SBAR (WHNP (WDT which)) (S (NP (PRP we)) (VP (VBD sold) (PP (IN at) (NP (NN auction))) (NP (JJ last) (NN year)) (PP (IN in) (NP (NNP February)))))))) (. .))

AJ (S (NP (PRP It)) (VP (VBD was) (NP (NP (NP (PRP$ my) (NN aunt) (POS ’s)) (NN car)) (SBAR (WHNP (WDT which)) (S (NP (PRP we)) (VP (VBD sold) (PP (IN at) (NP (NN auction))) (NP (JJ last) (NN year)) (PP (IN in) (NP (NNP February)))))))) (. .))

Sentence 7

GOLD (S (NP (NP (ADJP (JJ Natural)) (NNS disasters)) (PRN (PRN (: –) (NP (NP (NP (NNS storms)) (LST (, ,) (NP (NN flooding)))) (LST (, ,) (NP (NNS hurricanes))))) (: –))) (VP (VP (VP (VBP occur)) (ADVP (RB infrequently))) (CONJP (CC but) (VP (VBP cause) (NP (NN devastation) (S (WDT that) (VP (VP (VBZ strains) (NP (NNS resources))) (PP (IN to) (NP (NN breaking) (NN point))))))))) (. .))

LAL (S (NP (NP (JJ Natural) (NNS disasters)) (PRN (VBP –) (NP (NNS storms) (, ,) (NN flooding) (, ,) (NNS hurricanes)) (VBP –))) (VP (VP (VBP occur) (ADVP (RB infrequently))) (CC but) (VP (VB cause) (NP (NP (NN devastation)) (SBAR (WHNP (IN that)) (S (VP (VBZ strains) (NP (NNS resources)) (PP (TO to) (NP (VBG breaking) (NN point))))))))) (. .))

AJ (S (NP (NP (JJ Natural) (NNS disasters)) (PRN (HYPN –) (NP (NNS storms) (, ,) (NN flooding) (, ,) (NNS hurricanes)) (HYPN –))) (VP (VP (VBP occur) (ADVP (RB infrequently))) (CC but)

(VP (VB cause) (NP (NP (NN devastation)) (SBAR (WHNP (WDT that)) (S (VP (VBZ strains) (NP (NNS resources)) (PP (IN to) (NP (VBG breaking) (NN point))))))))) (. .))

Sentence 8

GOLD (S (NP (NP (NNS Letters)) (S (VP (VP (VBN delivered) (PP (IN on) (NP (NN time)))) (PP (IN by) (NP (ADJP (ADJP (ADJP (JJ old)) (HYPH -)) (JJ fashioned)) (NNS means)))))) (VP (VBP are) (ADJP (ADVP (RB increasingly)) (JJ rare))) (S (ADVP (, ,) (RB so)) (S (NP (PRP it)) (VP (VP (VBZ is) (ADVP (RB as) (ADVP (RB well)))) (S (IN that) (S (NP (DT that)) (VP (VBZ is) (ADVP (RB not) (NP (NP (DT the) (NP (ADJP (JJ only)) (NN option))) (ADJP (JJ available)))))))))) (. .))

LAL (S (S (NP (NP (NNS Letters)) (VP (VBN delivered) (PP (IN on) (NP (NN time))) (PP (IN by) (NP (ADJP (JJ old) (: -) (VBN fashioned)) (NNS means))))) (VP (VBP are) (ADJP (RB increasingly) (JJ rare)))) (, ,) (IN so) (S (NP (NP (PRP it))) (VP (VBZ is) (ADJP (RB as) (RB well)) (SBAR (IN that) (S (NP (DT that)) (VP (VBZ is) (RB not) (NP (NP (DT the) (JJ only) (NN option)) (ADJP (JJ available)))))))) (. .))

AJ (S (S (NP (NP (NNS Letters)) (VP (VBD delivered) (PP (IN on) (NP (NN time))) (PP (IN by) (NP (JJ old) (HYPN -) (JJ fashioned) (NNS means))))) (VP (VBP are) (ADJP (RB increasingly) (JJ rare)))) (, ,) (IN so) (S (NP (NP (PRP it))) (VP (VBZ is) (ADVP (RB as) (RB well)) (SBAR (IN that) (S (NP (DT that)) (VP (VBZ is) (RB not) (NP (NP (DT the) (JJ only) (NN option)) (ADJP (JJ available)))))))) (. .))

Sentence 9

GOLD (S (NP (NNP English)) (VP (ADVP (RB also)) (VP (VBZ has) (NP (NP (ADJP (JJ many)) (NP (NNS words) (PP (IN of) (NP (NP (ADJP (ADJP (JJR more)) (CONJP (CC or) (ADJP (JJR less)))) (NP (ADJP (JJ unique)) (NN function))) (, ,) (VP (VP (VBG including) (NP (NP (NNS interjections)) (PRN (-LRB- -LRB-) (LST (LST (UH oh) (, ,)) (UH ah)) (-RRB- -RRB-))) (, ,) (NP (NP (NNS negatives)) (PRN (-LRB- -LRB-) (LST (LST (RB no) (, ,)) (RB not)) (-RRB- -RRB-))) (, ,) (NP (NP (NN politeness) (NNS markers)) (PRN (-LRB- -LRB-) (LST (LST (UH please) (, ,)) (VP (VB thank) (NP (PRP you)))) (-RRB- -RRB-))) (, ,) (CONJP (CC and) (ADVP (ADVP (DT the) (ADVP (ADJP (JJ existential)) (“ ’) (ADVP (RB there)) (” ’))) (PRN (-LRB- -LRB-) (S (EX there) (VP (VBP are) (NP (NNS horses) (CONJP (CC but) (ADVP (RB not) (NP (NNS unicorns))))))) (-RRB- -RRB-))))) (PP (IN among) (NP (NNS others))))))))))) (. .))

LAL (S (NP (NNP English)) (ADVP (RB also)) (VP (VBZ has) (NP (NP (JJ many) (NNS words)) (PP (IN of) (NP (ADJP (JJR more) (CC or) (JJR less)) (JJ unique) (NN function))) (, ,) (PP (VBG including) (NP (NP (NP (NNS interjections)) (PRN (NNP -LRB-) (INTJ (INTJ (UH oh)) (, ,) (JJ ah)) (NN -RRB-))) (, ,) (NP (NP (NNS negatives)) (PRN (VBP -LRB-) (INTJ (DT no) (, ,) (RB not)) (NNP -RRB-))) (, ,) (NP (NP (NN politeness) (NNS markers)) (PRN (VBP -LRB-) (INTJ (INTJ (NN please)) (, ,) (NN thank) (PRP you)) (VBP -RRB-))) (, ,) (CC and) (NP (NP (NP (DT the) (JJ existential) (POS ’) (EX there) (” ’)) (PRN (NN -LRB-) (S (NP (EX there)) (VP (VBP are) (NP (NP (NNS horses)) (CC but) (RB not) (NP (JJ unicorns))))) (NNP -RRB-))) (PP (IN among) (NP (NNS others)))))))) (. .))

AJ (S (NP (NNP English)) (ADVP (RB also)) (VP (VBZ has) (NP (NP (JJ many) (NNS words)) (PP (IN of) (NP (ADJP (ADVP (JJR more) (CC or) (RBR less)) (JJ unique)) (NN function))) (, ,) (PP (VBG including) (NP (NP (NP (NNS interjections)) (PRN (-LRB- -LRB-) (INTJ (UH oh) (, ,) (UH ah)) (-RRB- -RRB-))) (, ,) (NP (NP (NNS negatives)) (PRN (-LRB- -LRB-) (INTJ (UH no) (, ,) (RB not)) (-RRB- -RRB-))) (, ,) (NP (NP (NN politeness) (NNS markers)) (PRN (-LRB- -LRB-) (NP (INTJ (UH please)) (, ,) (VBP thank) (PRP you)) (-RRB- -RRB-))) (, ,) (CC and) (NP (NP (NP (DT the) (JJ existential) (“ ’) (EX there) (” ’)) (PRN (-LRB- -LRB-) (S (NP (EX there)) (VP (VBP are) (NP (NP (NNS horses)) (CC but) (RB not) (NP (NNS unicorns))))) (-RRB- -RRB-))) (PP (IN among) (NP (NNS others)))))))) (. .))

Sentence 10

GOLD (S (NP (DT The) (NP (NNP Penn) (NP (NNP Treebank) (NP (NN tagset))))) (VP (VP (VBD was) (VBN culled)) (PP (IN from) (NP (DT the) (ADJP (JJ original) (NP (NP (CD 87) (HYPH -)) (NP (NN tag)))) (NP (NN tagset) (PP (IN for) (NP (DT the) (NP (NNP Brown) (NP (NNP Corpus))))))))) (. .))

LAL (S (NP (DT The) (NNP Penn) (NNP Treebank) (NN tagset)) (VP (VBD was) (VP (VBN culled) (PP (IN from) (NP (NP (DT the) (JJ original) (ADJP (CD 87) (: -) (NN tag)) (NN tagset)) (PP (IN for) (NP (DT the) (NNP Brown) (NNP Corpus))))))) (. .))

AJ (S (NP (DT The) (NNP Penn) (NNP Treebank) (NN tagset)) (VP (VBD was) (VP (VBN culled) (PP (IN from) (NP (NP (DT the) (JJ original) (CD 87) (HYPN -) (NN tag) (NN tagset)) (PP (IN for) (NP (DT the) (NNP Brown) (NNP Corpus))))))) (. .))

Sentence 11

GOLD (S (PP (IN For) (NP (NN example))) (S (NP (DT the) (ADJP (JJ original) (NP (NNP Brown)) (CONJP (CC and) (NP (NP (NNP C5)) (NNS tagsets))))) (VP (VBP include) (NP (DT a) (ADJP (JJ separate) (NP (NN tag) (PP (PP (IN for) (DT each)) (PP (IN of) (NP (DT the) (ADJP (JJ different) (NP (NNS forms) (PP (IN of) (NP (DT the) (NP (NNS verbs) (LST (LST (LST (VB do) (PRN (-LRB- -LRB-) (ADVP (ADVP (RB e.g.) (NP (NNP C5) (NP (NN tag) (NN VDD))) (PP (IN for) (VBN did))) (CONJP (CC and) (NP (NN VDG) (NN tag)) (PP (IN for) (VBG doing)))) (-RRB- -RRB-))) (, ,) (VB be)) (CONJP (CC and) (VB have)))))))))))))))) (. .))

LAL (S (PP (IN For) (NP (NN example))) (NP (DT the) (JJ original) (NNP Brown) (CC and) (NNP C5) (NNS tagsets)) (VP (VBP include) (NP (NP (DT a) (JJ separate) (NN tag)) (PP (IN for) (NP (NP (DT each)) (PP (IN of) (NP (NP (DT the) (JJ different) (NNS forms)) (PP (IN of) (NP (DT the) (NNS verbs))))))) (UCP (VBP do) (DT -LRB-) (NN e.g.) (NP (NP (NP (NP (NNP C5) (NN tag) (NNP VDD)) (PP (IN for) (ADJP (VBD did)))) (CC and) (NP (NP (NNP VDG) (NN tag)) (PP (IN for) (S (VP (VBG doing)))))) (NN -RRB-)) (, ,) (VB be) (CC and) (VB have)))) (. .))

AJ (S (PP (IN For) (NP (NN example))) (NP (DT the) (JJ original) (NNP Brown) (CC and) (NNP C5) (NNS tagsets)) (VP (VBP include) (NP (NP (DT a) (JJ separate) (NN tag)) (PP (IN for) (NP (NP (DT each)) (PP (IN of) (NP (NP (DT the) (JJ different) (NNS forms)) (PP (IN of) (NP (DT the) (NNS verbs) (NP (VP (VBP do)) (PRN (-LRB- -LRB-) (RB e.g.) (NP (NP (NP (NNP C5) (NN tag) (NN VDD)) (PP (IN for) (S (VP (VBD did))))) (CC and) (NP (NP (NNP VDG) (NN tag)) (PP (IN for) (NP (VBG doing))))) (-RRB- -RRB-))) (, ,) (VB be) (CC and) (VB have))))))))) (. .))

B Dependency parsing

This section shows the complete outputs of both parsers for the dependency parsing and the figures of the dependency trees mentioned in Section 6.2.2.

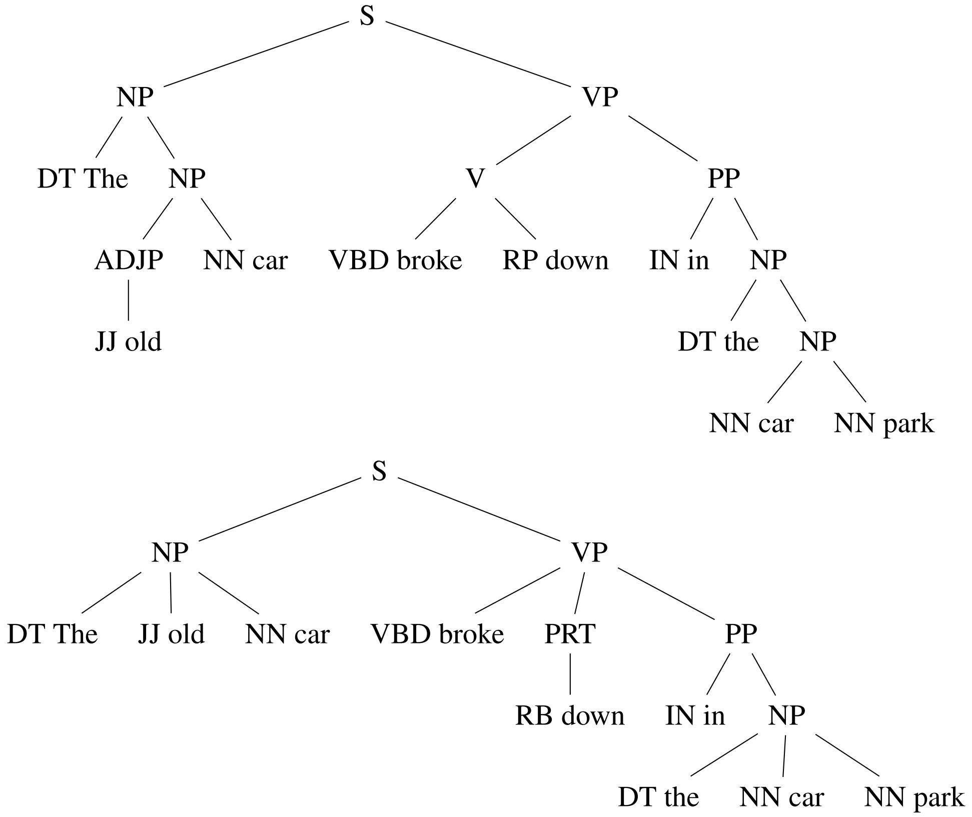

Figure 3: The top tree is the gold standard for sentence 2, while the bottom one is LAL's prediction. The AJ parsers built the same tree as LAL but with the RP tag for the word "down".

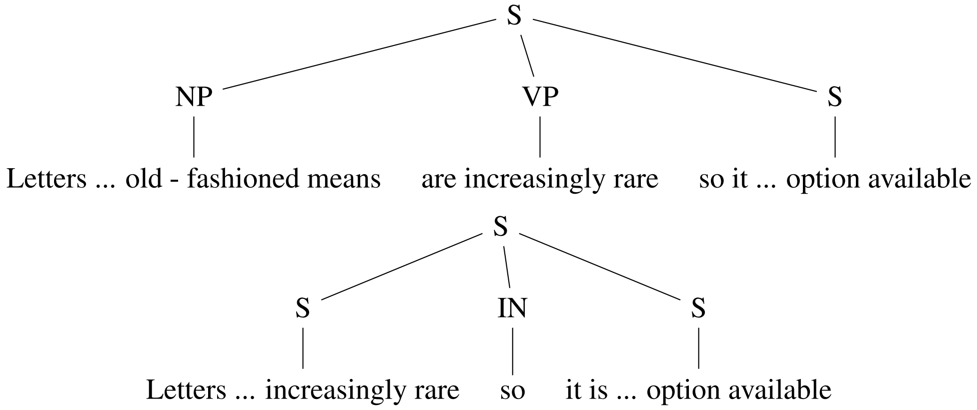
S
NP
VP
S
Letters ... old - fashioned means
are increasingly rare
so it ... option available
S
S
IN
S
Letters ... increasingly rare
so
it is ... option available

Figure 4: The top tree is the gold standard for sentence 8, while the bottom one is AJ's prediction. The LAL parsers built the same tree as AJ but with a different subtree of the words "old-fashioned means".

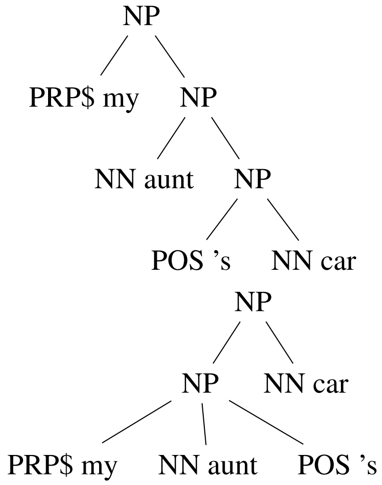

Figure 5: The top tree is the gold standard for a subtree of sentence 6, while the bottom one is LAL and AJ’s predictions.

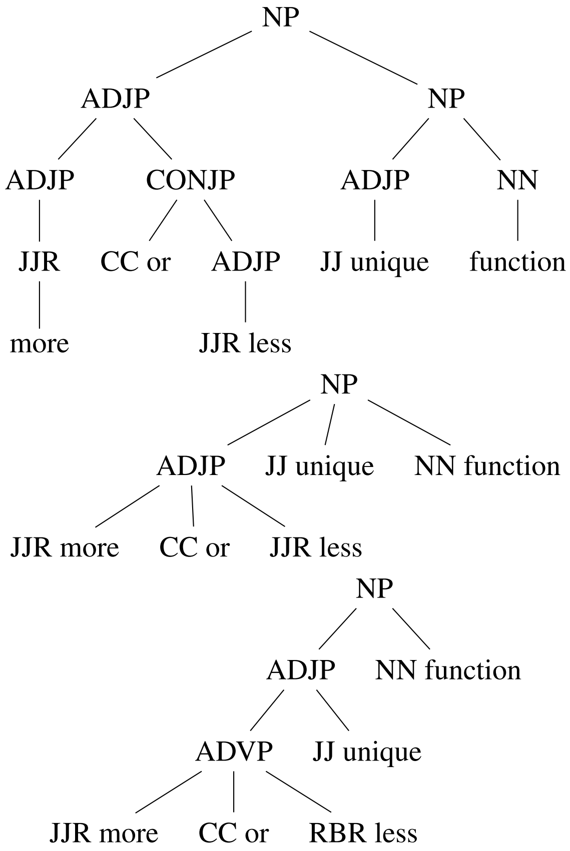

Figure 6: The top tree is the gold standard for a subtree of sentence 9, the middle one is LAL’s prediction, while the bottom one is AJ’s prediction.

	GOLD		LAL parser		AJ parser	
Word	**Head**	**Rel**	**Head**	**Rel**	**Head**	**Rel**
My	<aunt	poss	<aunt	poss	<aunt	poss
aunt	<opener	poss	<can	poss	<opener	poss
’s	>aunt	possessive	>aunt	possessive	>aunt	possessive
can	<opener	nn	<open	aux	<opener	dep
opener	<open	nsubj	>can	dep	<open	nsubj
can	<open	aux	<open	aux	<open	aux
open		ROOT		ROOT		ROOT
a	<drum	det	<drum	det	<drum	det
drum	>open	dobj	>open	dobj	>open	dobj

Table 3: Dependency parsing of sentence 1 of all parsers and gold standard.

	GOLD		LAL parser		AJ parser	
Word	**Head**	**Rel**	**Head**	**Rel**	**Head**	**Rel**
The	<car	det	<car	det	<car	det
old	<car	amod	<car	amod	<car	amod
car	<broke	nsubj	<broke	nsubj	<broke	nsubj
broke		ROOT		ROOT		ROOT
down	>broke	prt	>broke	prt	>broke	prt
in	>broke	prep	>broke	prep	>broke	prep
the	<park	det	<park	det	<park	det
car	<park	nn	<park	nn	<park	nn
park	>in	pobj	>in	pobj	>in	pobj

Table 4: Dependency parsing of sentence 2 of all parsers and gold standard.

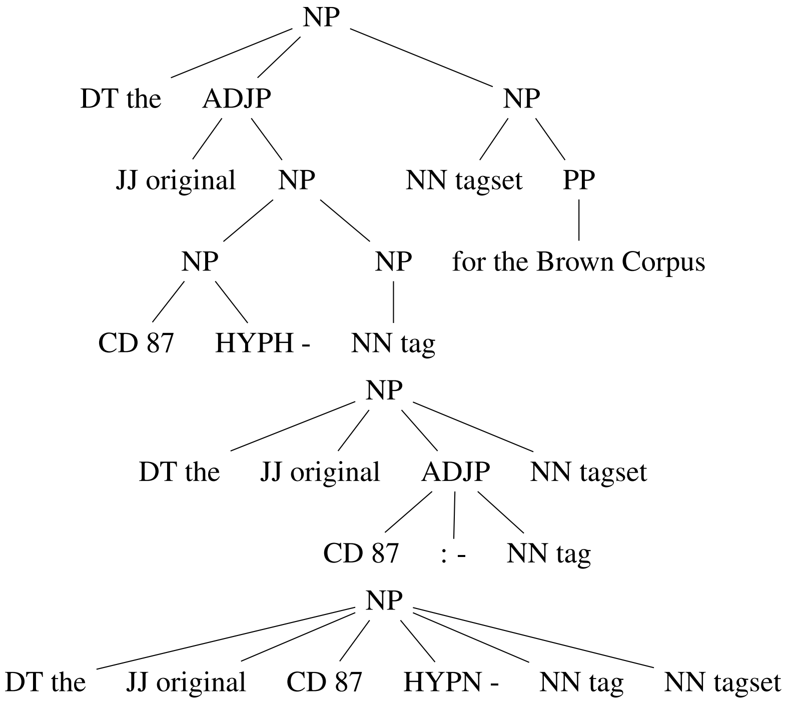

Figure 7: The top tree is the gold standard for a subtree of sentence 10, the middle one is LAL’s prediction, while the bottom one is AJ’s prediction.

	GOLD		LAL parser		AJ parser	
Word	Head	Rel	Head	Rel	Head	Rel
At	<least	advmod	<two	quantmod	<least	advmod
least	<two	quantmod	>At	mwe	<two	quantmod
two	<men	num	<men	num	<men	num
men	<broke	nsubj	<broke	nsubj	<broke	nsubj
broke		ROOT		ROOT		ROOT
in	>broke	prt	>broke	prt	>broke	prt
and	>broke	cc	>broke	cc	>broke	cc
stole	>broke	conj	>broke	conj	>broke	conj
my	<TV	poss	<TV	poss	<TV	poss
TV	>stole	dobj	>stole	dobj	>stole	dobj

Table 5: Dependency parsing of sentence 3 of all parsers and gold standard.

	GOLD		LAL parser		AJ parser	
Word	Head	Rel	Head	Rel	Head	Rel
Kim	<broke	nsubj	<broke	nsubj	<broke	nsubj
and	>Kim	cc	>Kim	cc	>Kim	cc
Sandy	>Kim	conj	>Kim	conj	>Kim	conj
both	>Kim	preconj	<broke	dep	<broke	dep
broke		ROOT		ROOT		ROOT
up	>broke	prt	>broke	prt	>broke	prt
with	>broke	prep	>broke	prep	>broke	prep
their	<partner+s	poss	<partners	poss	<partners	poss
partners	>with	pobj	>with	pobj	>with	pobj

Table 6: Dependency parsing of sentence 4 of all parsers and gold standard.

	GOLD		LAL parser		AJ parser	
Word	Head	Rel	Head	Rel	Head	Rel
The	<horse	det	<horse	det	<horse	det
horse	<escaped	nsubj	<escaped	nsubj	<escaped	nsubj
as	<well	mwe	<well	advmod	<well	advmod
well	>horse	cc	>horse	cc	>horse	cc
as	>well	mwe	>well	mwe	>well	mwe
the	<rabbit+s	det	<rabbits	det	<rabbits	det
rabbits	>horse	conj	>horse	conj	>horse	conj
which	>rabbit+s	ref	<wanted	dobj	<wanted	dobj
we	<want+ed	nsubj	<wanted	nsubj	<wanted	nsubj
wanted	>rabbit+s	rcmod	>rabbits	rcmod	>rabbits	rcmod
to	<eat	aux	<eat	aux	<eat	aux
eat	>want+ed	xcomp	>wanted	xcomp	>wanted	xcomp
has	<escaped	aux	<escaped	aux	<escaped	aux
escaped		ROOT		ROOT		ROOT

Table 7: Dependency parsing of sentence 5 of all parsers and gold standard.

	GOLD		LAL parser		AJ parser	
Word	**Head**	**Rel**	**Head**	**Rel**	**Head**	**Rel**
It	<car	nsubj	<car	nsubj	<car	nsubj
was	<car	cop	<car	cop	<car	cop
my	<aunt	poss	<aunt	poss	<aunt	poss
aunt	<car	poss	<car	poss	<car	poss
’s	>aunt	possessive	>aunt	possessive	>aunt	possessive
car		ROOT		ROOT		ROOT
which	>car	ref	<sold	dobj	<sold	dobj
we	<sold	nsubj	<sold	nsubj	<sold	nsubj
sold	>car	rcmod	>car	rcmod	>car	rcmod
at	>sold	prep	>sold	prep	>sold	prep
auction	>at	pobj	>at	pobj	>at	pobj
last	<year	det	<year	amod	<year	amod
year	>sold	tmod	>sold	tmod	>sold	tmod
in	>year	prep	>sold	prep	>sold	prep
February	>in	pobj	>in	pobj	>in	pobj

Table 8: Dependency parsing of sentence 6 of all parsers and gold standard.

	GOLD		LAL parser		AJ parser	
Word	**Head**	**Rel**	**Head**	**Rel**	**Head**	**Rel**
Natural	<disaster+s	amod	<disasters	amod	<disasters	amod
disasters	<occur	nsubj	<occur	nsubj	<occur	nsubj
storms	>disaster+s	appos	<hurricanes	nn	<hurricanes	nn
flooding	>storm+s	dep	<hurricanes	dep	<hurricanes	dep
hurricanes	>flood+ing	dep	>disasters	dep	>disasters	dep
occur		ROOT		ROOT		ROOT
infrequently	>occur	advmod	>occur	advmod	>occur	advmod
but	>occur	cc	>occur	cc	>occur	cc
cause	>occur	conj	>occur	conj	>occur	conj
devastation	>cause	dobj	>cause	dobj	>cause	dobj
that	<strain+s	nsubj	<strains	nsubj	<strains	nsubj
strains	>devastation	rcmod	>devastation	rcmod	>devastation	rcmod
resources	>strain+s	dobj	>strains	dobj	>strains	dobj
to	>strain+s	prep	>strains	prep	>strains	prep
breaking	<point	amod	<point	amod	<point	amod
point	>break+ing	pobj	>to	pobj	>to	pobj

Table 9: Dependency parsing of sentence 7 of all parsers and gold standard.

	GOLD		**LAL parser**		**AJ parser**	
Word	**Head**	**Rel**	**Head**	**Rel**	**Head**	**Rel**
Letters	<rare	nsubj	<rare	nsubj	<rare	nsubj
delivered	>Letter+s	vmod	>Letters	vmod	>Letters	vmod
on	>deliver+ed	prep	>delivered	prep	>delivered	prep
time	>on	pobj	>on	pobj	>on	pobj
by	>deliver+ed	prep	>delivered	prep	>delivered	prep
old	<fashion+ed	amod	<fashioned	amod	<means	amod
fashioned	<means	amod	<means	amod	<means	amod
means	>by	pobj	>by	pobj	>by	pobj
are	<rare	cop	<rare	cop	<rare	cop
increasingly	<rare	advmod	<rare	advmod	<rare	advmod
rare		ROOT		ROOT		ROOT
so	>rare	dep	>rare	dep	>rare	dep
it	<well	nsubj	<well	nsubj	<is	nsubj
is	<well	cop	<well	cop	>rare	parataxis
as	<well	advmod	<well	advmod	<well	advmod
well	>rare	advcl	>rare	ccomp	>is	advmod
that	<option	mark	<option	mark	<option	mark
that	<option	nsubj	<option	nsubj	<option	nsubj
is	<option	cop	<option	cop	<option	cop
not	<option	neg	<option	neg	<option	neg
the	<option	det	<option	det	<option	det
only	<option	amod	<option	amod	<option	amod
option	>well	ccomp	>well	ccomp	>is	ccomp
available	>option	amod	>option	amod	>option	amod

Table 10: Dependency parsing of sentence 8 of all parsers and gold standard.

	GOLD		LAL parser		AJ parser	
Word	**Head**	**Rel**	**Head**	**Rel**	**Head**	**Rel**
English	<has	nsubj	<has	nsubj	<has	nsubj
also	<has	advmod	<has	advmod	<has	advmod
has		ROOT		ROOT		ROOT
many	<word+s	amod	<words	amod	<words	amod
words	>has	dobj	>has	dobj	>has	dobj
of	>word+s	prep	>words	prep	>words	prep
more	<unique	amod	<function	amod	<unique	advmod
or	>more	cc	>more	cc	>more	cc
less	>more	conj	>more	conj	>more	conj
unique	<function	amod	<function	amod	<function	amod
function	>of	pobj	>of	pobj	>of	pobj
including	>word+s	vmod	>words	prep	>words	prep
interjections	>including	dobj	>including	pobj	>including	pobj
oh	>interjection+s	dep	>interjections	discourse	>interjections	discourse
ah	>interjection+s	dep	>oh	dep	>oh	dep
negatives	>interjection+s	conj	>interjections	conj	>interjections	conj
no	>negative+s	dep	>negatives	discourse	>negatives	discourse
not	>negative+s	dep	>no	dep	>no	dep
politeness	<marker+s	nn	<markers	nn	<markers	nn
markers	>interjection+s	conj	>interjections	conj	>interjections	conj
please	>marker+s	dep	<thank	discourse	<you	discourse
thank	>marker+s	dep	>markers	dep	<you	dep
you	>thank	dobj	>thank	dep	>markers	dep
and	>interjection+s	cc	>interjections	cc	>interjections	cc
the	<there	det	<there	det	<existential	det
existential	<there	amod	<there	amod	>interjections	conj
there	>interjection+s	conj	>interjections	conj	>existential	dep
there	<are	expl	<are	expl	<are	expl
are	>there	dep	>there	dep	>existential	dep
horses	>are	dobj	>are	nsubj	>are	nsubj
but	>horse+s	cc	>horses	cc	>horses	cc
not	<unicorn+s	neg	>horses	conj	<unicorns	neg
unicorns	>horse+s	conj	>horses	dep	>horses	conj
among	>interjection+s	prep	>there	prep	>existential	prep
others	>among	pobj	>among	pobj	>among	pobj

Table 11: Dependency parsing of sentence 9 of all parsers and gold standard.

	GOLD		**LAL parser**		**AJ parser**	
Word	**Head**	**Rel**	**Head**	**Rel**	**Head**	**Rel**
The	<tagset	det	<tagset	det	<tagset	det
Penn	<tagset	nn	<tagset	nn	<tagset	nn
Treebank	<tagset	nn	<tagset	nn	<tagset	nn
tagset	<cull+ed	nsubjpass	<culled	nsubjpass	<culled	nsubjpass
was	<cull+ed	auxpass	<culled	auxpass	<culled	auxpass
culled		ROOT		ROOT		ROOT
from	>cull+ed	prep	>culled	prep	>culled	prep
the	<tagset	det	<tagset	det	<tagset	det
original	<tagset	amod	<tagset	amod	<tagset	amod
87	<tag	number	<tag	number	<tagset	num
tag	<tagset	nn	<tagset	amod	<tagset	nn
tagset	>from	pobj	>from	pobj	>from	pobj
for	>tagset	prep	>tagset	prep	>tagset	prep
the	<Corpus	det	<Corpus	det	<Corpus	det
Brown	<Corpus	nn	<Corpus	nn	<Corpus	nn
Corpus	>for	pobj	>for	pobj	>for	pobj

Table 12: Dependency parsing of sentence 10 of all parsers and gold standard.

	GOLD		LAL parser		AJ parser	
Word	**Head**	**Rel**	**Head**	**Rel**	**Head**	**Rel**
For	<include	prep	<include	prep	<include	prep
example	>For	pobj	>For	pobj	>For	pobj
the	<Brown	det	<Brown	det	<Brown	det
original	<Brown	amod	<Brown	amod	<Brown	amod
Brown	<tagset+s	nn	<include	nsubj	<include	nsubj
and	>Brown	cc	>Brown	cc	>Brown	cc
C5	>Brown	conj	<tagsets	nn	<tagsets	nn
tagsets	<include	nsubj	>Brown	conj	>Brown	conj
include		ROOT		ROOT		ROOT
a	<tag	det	<tag	det	<tag	det
separate	<tag	amod	<tag	amod	<tag	amod
tag	>include	dobj	>include	dobj	>include	dobj
for	>tag	prep	>tag	prep	>tag	prep
each	>for	pobj	>for	pobj	>for	pobj
of	>each	prep	>each	prep	>each	prep
the	<form+s	det	<forms	det	<forms	det
different	<form+s	amod	<forms	amod	<forms	amod
forms	>of	pobj	>of	pobj	>of	pobj
of	>form+s	prep	>forms	prep	>forms	prep
the	<verb+s	det	<verbs	det	<verbs	det
verbs	>of	pobj	>of	pobj	>of	pobj
do	>verb+s	appos	<for	dep	<VDD	dep
e.g.	>do	dep	<for	discourse	<VDD	dep
C5	<tag	nn	<for	nn	<VDD	nn
tag	>e.g.	dep	<for	nn	<VDD	nn
VDD	>tag	nn	>tag	dep	>verbs	dep
for	>tag	prep	tag	prep	>VDD	prep
did	>for	pobj	tag	pobj	>for	pcomp
and	>tag	cc	>for	cc	>VDD	cc
VDG	<tag	nn	<for	nn	<tag	nn
tag	>tag	conj	>for	conj	>VDD	conj
for	>tag	prep	for	prep	>tag	prep
doing	>for	pobj	for	pcomp	>for	pobj
be	>do	conj	>for	conj	>verbs	conj
and	>do	cc	>for	cc	>verbs	cc
have	>do	conj	>for	conj	>verbs	conj

Table 13: Dependency parsing of sentence 11 of all parsers and gold standard.