To be incremental or not: PreTra and RBGParser evaluated on garden path sentences

BACHELOR THESIS BSC ARTIFICIAL INTELLIGENCE
UTRECHT UNIVERSITY

Peter Maatman (5630177)
Supervisor: Meaghan Fowlie
Second reader: Anna Wegmann

November 6, 2020

Abstract

This study investigates the effect of incrementality on the accuracy of parsers on garden path sentences. We show that the incremental parser evaluated in this thesis achieves lower accuracies on garden path sentences than a non-incremental parser. We then explore one explanation for this worse accuracy of the incremental parser and we find two likely sources for error: either the scoring function, or the transition system is the culprit. We suggest further research to investigate which of of the two sources of error contribute the most.

Contents

1 Introduction

As Jurafsky and Martin say in their book Speech and Language processing, *"Ambiguity is perhaps the most serious problem faced by parsers"* [1]. There are a number of different types of ambiguity, in this thesis we will focus on *local ambiguity*. This type of ambiguity is very interesting because a reader will encounter ambiguity while reading the sentence, but by the time they reach the end of the sentence the ambiguity will be resolved. For example in sentence (1) the part *The old man* is parsed as a noun phrase. This makes the part *the boat* unexpected, but then we realize that *man* is actually the verb.

(1) The old man the boat.

It is reasonable to expect most modern parsers to be able to parse such ambiguous sentences correctly, because they are generally able to use features spanning the whole sentence. As opposed to people, who read sentences from left to right without having knowledge of what is to come.

However, recently interest in parsers that incrementally parse natural language have become more popular. The assumption that many current parsers rely on is that the whole sentence is known at the start of processing the input. But this is rarely the case in real world use cases of language processing such as smart speakers, Google Assistant, Siri[2]. These systems listen for user input over time and don't see the whole sentence the users utters at once. If these systems would only start processing once the whole sentence is known, sentences would be translated only after it was processed fully instead of on the fly [2].

It is important for the larger field of AI to be able to parse natural language. And because natural language is full of ambiguity, we need parsers that are able to handle that. As we have seen these ambiguous sentences likely present problems for modern parsers.

We aim to answer the question: how does incrementality impact the parsing accuracy on temporarily ambiguous sentences? To answer this question we examine two parsers, an incremental and a non-incremental parser. Next to this we try to determine what factors impact the accuracy of the incremental parser specifically. We find that, as expected, the non-incremental parser achieves higher accuracies on the temporarily ambiguous sentences than the incremental parser. We also identify two sources of error that might influence the accuracy of the incremental parser.

The remainder of this thesis is structured as follows: we will first introduce the background theory behind the sentences we use to evaluate the parsers, as well as the parsers themselves. We then describe two experiments we do to answer our research question.

2 Theoretical background

Several studies involving parsing have suggested that locally ambiguous sentences are problematic for parsers [1, 2, 3]. However, few have actually produced any work that shows this.

This section is further divided into four subsections. First we will shortly discuss what exactly garden path sentences are and why they are problematic for both human and computational sentence processing. We quickly go over the problem of dependency parsing and the annotation schema that is used. Then we will go over a non-incremental sampling based parser, RBGParser [4]. And lastly we will discuss an incremental, transition based parser, PreTra [2], which is based on RBGParser but uses a different decoding mechanism.

2.1 Garden path sentences

Garden path sentences are said to have three properties that make them hard to parse according to Jurafsky and Martin [1]:

1. The whole sentence is unambiguous, but an initial section of the sentence is ambiguous.
2. The ambiguous section has one prefered parse in the human sentence processing mechanism, while leaving the other possible parses as undesirable or less prefered.
3. The undesirable or less prefered parse of the initial section is the correct one.

These properties lead people "down the garden path": they initially choose the wrong parse for the sentence and are confused when they realize they chose the wrong one. These sentences can be so hard to parse, in fact, that people need to be shown the correct parse for them to be able to understand the sentence. Take Example (2); when you read this sentence it will seem as though you are reading a fairly normal, unambiguous sentence, until you reach the word "fell". There you realise that you have been misled and the sentence actually has a completely different structure. The correct structure has *raced* as part of a reduced relative clause, modifying *The horse*. And thus the sentence means "The horse, which was raced past the bar, fell" [1].

(2) The horse raced past the barn fell.

There are also less obvious garden path sentences, that can really only be noticed with precise reading time measurements or eye tracking[1]. In (3) *the solution* is often seen as the direct object of *forgot*, but instead it should be parsed as the subject of an embedded sentence. This misparse is hard to notice when reading the sentence, but can easily be shown with experiments where the participants take longer to read the word *was* than in control sentences[1].

(3) The student forgot the solution was in the back of the book.

2.2 Dependency parsing

In this thesis we are concerned with a dependency parsing task. Dependency parsing involves finding the dependency relation between words and phrases in a sentence. This dependency relation is a binary, asymmetric relation between words of a sentence. In computational linguistics these have been gaining popularity for a number of reasons, primarily because models trained on annotated corpora can easily be ported to a different domain or language [5]. A second reason is that these dependency relations generally don't describe a grammar. This makes them more robust in the face of unseen structures, as opposed to grammars that might not be able to encode a certain structure and thus be unable to parse that sentence. Figure 1 shows an example of the dependency relations between words in the sentence *"The old man the boat"* using the Universal Dependencies (UD) annotation schema [6]. Here the relations between the words are modeled as a dependency graph, where the relation between a word and its dependents is labeled through a directed arc. Generally speaking an artificial root node is inserted at the beginning of the sentence. These only serve to simplify the theoretical definitions and computational implementations, but can be left out in visual representations.

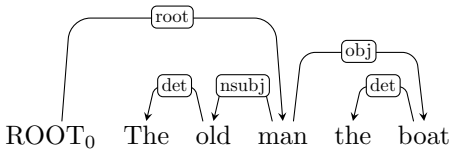


Figure 1: An example of a dependency parse using the UD annotation schema

2.3 Non-incremental parser: RBGParser

The sampling-based parser, RBGParser [4], focuses on using a highly expressive scoring functions that lead to simpler inference. With this approach it manages to outperform state-of-the-art parsers at the time it was published. Expressive scoring functions that go beyond first-order arc preferences lead to simpler inference. However, finding the structure that maximizes the score is known to be NP-hard[4]. This parser starts with a random initial candidate structure, and then stochastically climbs the scoring function to a higher scoring structure. A small example of how this works can be found in Figure 2. The distribution used to sample higher scoring structures is derived from the scoring function. The scoring function is defined to be $s(x, y) = \theta \cdot f(x, y)$, x denotes a sentence and $y \in \mathcal{Y}(x)$ is the corresponding dependency tree, θ are the learned weights that are applied to the feature vector $f(x, y)$ associated with the sentence and respective structure[4].

Sampling is done on a distribution that approximates independent samples

from Equation 1.

$$p(y|x, T, \theta) \propto \exp(s(x, y)/T) \tag{1}$$

The parameter T represents the temperature; this controls how concentrated the samples are around the maximum of $s(x, y)$. This distribution is used within a Metropolis-Hastings based sampling algorithm to determine if the moves in the sample space are allowed[4]. A second distribution determines the steps taken in the generation of the MCMC sequence. This second distribution follows the distribution described by Equation 2[4].

$$p(y'|x, y, T, \theta) \tag{2}$$

In the paper they explore two different proposal distributions, the first samples a new head for each word in the sentence and modifies one arc each time. The second uses a sampler for first-order scores, applying the Metropolis-Hastings algorithm[4].

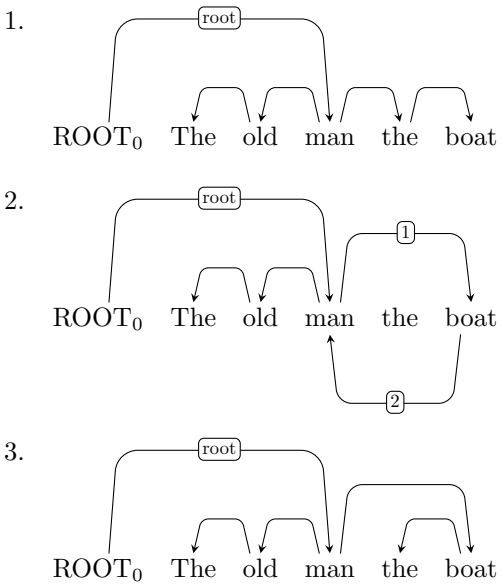


Figure 2: This example shows how RBGParser finds better scoring structures through random walk sampling. (1) shows the initial structure, and (2) and (3) modifications made on the arcs based on a random walk. The random walk generates new heads for K number of nodes. Each new head is sampled from a proposal distribution based on the learned weights θ, so these samples are guided in the right direction by how the distribution is set up. Only the heads for "man", "the", and "boat" are sampled while the sentence prefix remains fixed. In step (2) a loop forms from man $\rightarrow$ boat $\rightarrow$ man, such loops are erased, leaving only the edge from man to boat.

2.4 Incremental parser: PreTra

A different approach to parsing uses a "guide" to explore *transitions* that can be applied to a previous *configuration* to create a new configuration. A parser using such a system is a transition-based parser. A generic representation of the algorithm is described in Figure 3. These transition-based parsers differ in what configurations they use, and what kind of transitions between those configurations are allowed. The transition-based, incremental predictive parser *PreTra* proposed by Köhn [2] is based on this same principle.

The dependency structure used as *configuration* in PreTra has the property that it is always *connected*. This means that there is a path from every word to every other word in the structure, and one word is attached to the root node[2]. You can see in Figure 1 that the word *old* is connected to the word *boat*, if you ignore the directionality of the edges, via the path *old* $\rightarrow$ *man* $\rightarrow$ *boat*. This

```
def run_parser(parser, guide, sentence):
  configuration = parser.getInitialConfiguration(sentence)
  while configuration is not a terminal configuration do
    transition = guide.next_transition(configuration)
    configuration = configuration
      .makeTransition(transition)

  return configuration.getGraph()
```

Figure 3: The general algorithm describing a transition parser

property is not trivial to maintain on such a structure because the word needed to attach to the structure might not be part of the sentence prefix[2]. To solve this issue Köhn introduced prediction nodes in the tree that are stand-ins for words that are expected to be filled in a later step. A short example of how this works in practise can be found in Figure 4.

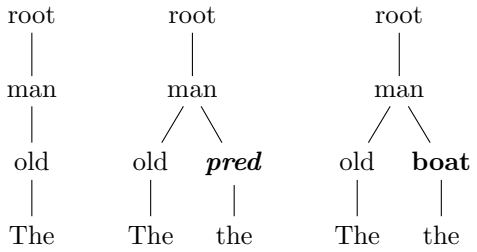


Figure 4: We assume we have parsed a sentences up to the point as shown in the left tree. The middle tree shows a structure after attaching the next word *the* to the tree. For this to be possible we have to add a *prediction* node in the tree that shows that we expect another word to be the head of *the*.

In PreTra the *configuration* is then defined as a tuple $\langle \mathcal{W}, \mathcal{P}, \pi, l \rangle$. $\mathcal{W}$ is the list of tokens, $\mathcal{P}$ is an unordered set of prediction nodes, $\pi : \mathcal{W} \cup \mathcal{P} \to \mathcal{W} \cup \mathcal{P} \cup \{0\}$ the set of directed edges, and $l : \mathcal{W} \cup \mathcal{P} \to L$ the labeling function for the edges [2].

The PreTra parser uses a beam search from the initial state. It generates new states repeatedly using a transition function and then obtains a beam from the generated states by scoring the new states with a scoring function and taking the top N structures. The initial state is defined as $\langle \emptyset, ([pred]), \{(1, 0)\} \rangle$, meaning a configuration starting with no tokens, that has one prediction node attached to the root[2].

The concrete transition function is composed of four functions that can generate new configurations, based on the current configuration and a new word, given some additional parameter. These four functions are defined as follows [2]:

attach(h) Attaches a new word w_{i+1} to an existing head $h \in \mathcal{W} \cup \mathcal{P}$
$\langle w_1...w_i, \mathcal{P}, \pi \rangle, w_{i+1}, h \rightarrow \langle w_1...w_i w_{i+1}, \mathcal{P}, \pi \cup \{(w_{i+1}, h)\}$

predictHead(h) Attaches a new prediction node to an already existing word or prediction node, and attaches the new word w_{i+1} to the new prediction node [2]. What happens in Figure 4 is exactly this operation.
$\langle w_1...w_i, \mathcal{P}, \pi \rangle, w_{i+1}, h \rightarrow \langle w_1...w_i w_{i+1}, \mathcal{P} \cup \{p\}, \pi \cup \{(w_{i+1}, p), (p, h)\}$

predictTwoHeads(h) Attaches a new word w_{i+1} to a new prediction node p_1. Then p_1 is attached to another new prediction node p_2, which is then finally attached to $h \in \mathcal{W} \cup \mathcal{P}$ [2]. An example of the result of this operation can be seen in the first parse of Example (4). There the word *man* has been attached with this operation.
$\langle w_1...w_i, \mathcal{P}, \pi \rangle, w_{i+1}, h \rightarrow \langle w_1...w_i w_{i+1}, \mathcal{P} \cup \{p_1, p_2\}, \pi \cup \{(w_{i+1}, p_1), (p_1, p_2), (p_2, h)\}$

replacePrediction(p) This last function is used to replace a prediction node with a concrete new word, where the the new word inherits all the dependency relations the prediction node had[2]. Note that this function takes a prediction node as parameter instead of a head node. Figure 4 also shows an example of this action, the prediction node in the middle tree is replaced with *boat* in the right tree, inheriting the dependency *the*.

$$\begin{aligned}\langle w_1...w_{i-1}, \mathcal{P}, \pi \rangle, w_i, p \rightarrow \langle w_1...w_{i-1} w_i, \mathcal{P} \setminus \{p\}, \\ \pi \cup \{(w_i, h) : (p, h) \in \pi\} \\ \cup \{(d, w_i) : (d, p) \in \pi\} \\ \setminus \{(p, h) : (p, h) \in \pi\} \\ \setminus \{(d, p) : (d, p) \in \pi\}\end{aligned}$$

Now to generate all the possible successors given a configuration and word, we first define the sets of possible successors that each of these functions can generate. And then take the union of those sets to get the final transition function[2].

$$\begin{aligned}\text{attach} &= \{attach^s_{w_i}(h) &&: h \in \mathcal{W} \cup \mathcal{P}\} \\ \text{predictHead} &= \{predictHead^s_{w_i}(h) &&: h \in \mathcal{W} \cup \mathcal{P}\} \\ \text{predictTwoHead} &= \{predictTwoHead^s_{w_i}(h) &&: h \in \mathcal{W} \cup \mathcal{P}\} \\ \text{replacePrediction} &= \{replacePrediction^s_{w_i}(h) &&: p \in \mathcal{P}\}\end{aligned}$$

We can now define the successor function as the union of the sets defined

above[2].

$$succ(s, w) = \text{attach}_w^s \cup \text{predictHead}_w^s \cup \text{predictTwoHeads}_w^s \cup \text{replacePrediction}_w^s$$

At each step in the parsing process this successor function is applied, and all the resulting dependency structures are scored with a scoring function f. From the scored dependency structures, the top N best scoring trees are kept for the next iteration in beam search fashion [2].

Köhn proposes two scoring functions in [2], one based on the same scoring system used in the restart-incremental parser incTP, introduced earlier in [2]. As well as a scoring function based on an LSTM, however this one did worse than the incTP based scoring function, so we will not be using that and instead use the incTP based scoring. The scoring system used for incTP has been used before in other parsers, including RBGParser [4]. PreTra has been built on top of RBGParser and reuses the implementation to extract the feature vector. A feature describes a specific combination of edges, the scorer uses a hash kernel to map the features to an index with a fixed upper bound [2]. A hash kernel is a function that hashes individual features and uses that hash to compute the index in a feature vector. This feature mapping is defined as $\phi(s)$, this maps a dependency structure to a fixed-length feature vector in which each feature can either be 0 if the feature is not present in the dependency structure or 1 if it is [2].

The final scoring function is then defined as $f_{TP}(s) = \phi(s) * \overrightarrow{w}$ [2]. In which $\overrightarrow{w}$ is a learned weights vector.

2.5 Errors due to beam search

The use of beam search has the potential to introduce errors in the parsing process. Köhn identifies three reasons that contribute to errors in PreTra [2]:

1. The scoring function rates a different structure in the beam higher than the correct structure.
2. The transition system can't produce the required structure.
3. The transition system could have produced the required structure, but in an earlier step a structure required to generate the correct structure fell of the beam.

Köhn does an experiment to investigate the third option as a source of errors in PreTra on a number of treebanks. In this experiment a flag is used to instruct PreTra to always keep the incremental structure with the least errors compared to the gold structure in the beam[2]. This makes it possible to investigate the coverage of the transition system and the quality of the scoring function. The experiment done by Köhn shows an expected high accuracy on complete sentences for a number of treebanks examined, however it falls short on the

Hamburg Dependency Treebank (HDT)[7]. The accuracy is also worse than his other incremental parser incTP, and because incTP and PreTra share a very similar scoring component this indicates that the errors were introduced by the transition system [2]. We will use a similar method on temporarily ambiguous sentences, but compare the results to RBGParser. Unfortunately RBGParser and PreTra only share a similar set of features and not a similar scoring system, this means that we will only be able to investigate the third source of errors.

3 Experimental setup

For this research we will be looking at the dependency parse results of garden path sentences from two parsers using a similar set of features. We use the non-incremental sampling-based RBGParser [4] and the incremental transition-based PreTra [2]. We compare their baseline performance from training to their performance on garden path sentences.

We also do a second experiment similar to Köhn [2] to account for one source of errors due to beam search in PreTra. For this experiment we only compare between two conditions on PreTra and do not compare the results to RBGParser. We use the same models used for the first experiment. A short description of the original experiment can be found in section 2.5.

3.1 Datasets

To train the two parsers in question we used the Universal Dependencies English Web Treebank (EWT) [8]. The baseline performance is evaluated on the test set of this treebank.

To evaluate the performance of these parsers on garden path sentences we build a small dataset of annotated garden path sentences. The sentences in this dataset were gathered from a number of websites and slides used in university courses that are available online, [1, 9, 10, 11, 12, 13, 14]. Appendix A shows a couple of examples from the final dataset, the whole dataset can be found on github[1]. We used UDPipe [15] to convert the plain sentences into CoNLL-X format. All dependencies were then manually annotated by both Meaghan Fowlie, and Peter Maatman.

3.2 Evaluation measures

Following Zhang et. al.[4] and Köhn [2] we will use the Unlabeled Attachment Score (UAS) as the evaluation metric of the parsers. The UAS reported here excludes punctuation on the garden path dataset following Martins et. al.[16] and Zhang et. al. [4].

3.3 Experimental details

Both parsers are trained using a reduced feature set that exclude the great-grandparent, parent-sibling-child, and global feature templates features to reduce training time. On PreTra this is known to lead to a 0.5 to 1 percentage points reduction in accuracy[2]. However, we did train PreTra with loss-augmented selection of the structure which increases accuracy by 0.5 percentage points [2]. We have done no further hyper parameter optimization on either parser. The

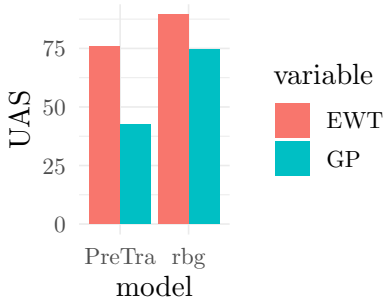
[1] `https://github.com/blackwolf12333/thesis_experiments/blob/master/data/sentences.conll.no_punct`

scripts and other code used to train and evaluate the parsers is available on github[2].

[2] https://github.com/blackwolf12333/thesis_experiments

4 Results

As a baseline test we have applied both parsers on the test set for EWT, here RBGParser achieves an unlabeled attachment score of 89.4%, and PreTra achieves an unlabeled attachment score of 75.8%. We compare these results to the scores achieved on the garden path dataset, which are 74.7% and 42.9% respectively. The accuracy difference of the same parser on the two different datasets shows that RBGParser is at an advantage compared to PreTra. The difference between the baseline, evaluated on EWT, and the garden path sentences for RBGParser is only 15%, while the difference for PreTra is 33%. This indicates that PreTra is at a disadvantage on garden path sentences compared to RBGParser.



Parser	EWT	GP
RBGParser	89.386	74.712
PreTra	75.800	42.406

Figure 5: Unlabeled attachment scores of PreTra and RBGParser, evaluated on EWT and our own garden path dataset

4.1 Beam search errors

When accounting for errors due to the use of beam search in PreTra we find that PreTra is very capable on the test set but errors are introduced because structures that are needed later are not kept in the beam. PreTra achieves an unlabeled attachment score of 95.79% in the "least errors" condition on the EWT test set, which is a 20% improvement over the baseline condition in this experiment. On the garden path set the "least errors" condition does not have such a drastic change in accuracy, going from 40.97% in the baseline condition, to 45.27% in the "least errors" condition, only a 4% increase.

As expected, the accuracy of PreTra when it can keep the structure with the least errors in the beam on EWT is very high. This result is very similar to the results achieved by Köhn [2]. However, the interesting part in this table is the relatively small improvement on the garden path sentences. Because the model used is the same this indicates that the transition system cannot generate the structures required for garden path sentences, or the scoring function rates different structures as better than the actual correct structure.

	EWT	GP
PreTra (least errors)	95.792	45.272
PreTra (baseline)	75.808	40.974

Table 1: This shows the results of PreTra compared to the results of running PreTra with the option to keep the structure with the least errors in the beam.

5 Discussion

Unfortunately EWT is a relatively small treebank, so training accuracy is not on par with the results presented in the papers that introduced RBGParser[4] and PreTra[2]. But since both parsers were trained on the same data it is still possible to compare their performance. The garden path dataset is also very small and thus might not represent all possible garden paths. This means that there is an unknown amount of wiggle room in the results presented on the garden path dataset. Future research should compile a larger dataset.

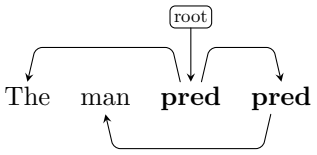
While we have shown that garden path sentences are hard to parse, it seems particularly hard for the incremental parser PreTra. One possible explanation for this bad performance has been explored here. But other experiments are required to conclusively say what the exact cause of the relatively bad performance is. To investigate which other factors play a part in the performance of PreTra we suggest an additional experiment.

A comparison can be made with Köhn's other incremental parser, incTP[2]. Because PreTra uses a scoring system based on the same scoring system in incTP this could rule out the scoring system as a factor in the performance of PreTra. A similar experiment is done by Köhn to show that the transition system in PreTra can't represent certain structures in the HDT[2].

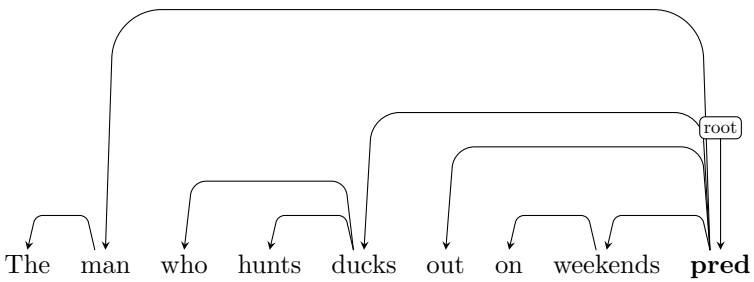
The parsing output generated by PreTra does not only include the final parse of the complete sentence, but also the best scoring structure for each incremental step. We inspected this output of the garden path sentences to get a clearer view of what the parser was doing while parsing these sentences. In this we observed that for a number of sentences the parser would score a tree with prediction nodes, that eventually would become an incomplete parse, higher. Meaning that a prediction node was left in the tree when the sentence was fully read. This means that the parser expected a different sentence structure early on and was not able to correct for this error with the given beam size. Example (4) should be read like *The man* is modified by the clause *who hunts*. The two parses below it are given by PreTra, the first parse (a) is the incremental parse after *The man* has been attached. This has resulted in two prediction nodes to indicate the expected verb and object. If we expect the parser to take the garden path, it should attach *hunts* to the root, replacing the prediction node. On the other hand, if the parser doesn't take the garden path we would expect it to attach *ducks* to the root. However, as can be seen in the complete parse (b), neither of those things happens. The prediction node that was attached to the root has never been replaced.

(4) The man who hunts ducks out on weekends

a.



b.



Because we can't view the other structures in the beam at that time it is hard to say exactly what structures are considered. But this does seem to support our finding that either the transition system does not cover the garden path sentences, or that the scoring function does not rank the correct structures high enough to remain in the beam.

6 Conclusion

The aim of this study was to investigate the impact of incrementality in parsers when evaluated on temporarily ambiguous sentences. We find that the incremental parser evaluated in this thesis is less accurate than the non-incremental parser when evaluated on garden path sentences. We have also found that the drop in accuracy is likely caused by either the transition system used, or the scoring function. Further research is necessary to conclude definitively which is the exact source of error. We suggest one method of investigating this problem. A second incremental parser with a similar scoring system to the one investigated in this thesis could be used to rule out the scoring method as a source of errors.

A major limitation of this study is the size of the datasets used. On the one hand the dataset used to train the parsers was relatively small, and on the other hand the dataset used to evaluate the parser on temporarily ambiguous sentences was very small. Future research should invest in building a larger dataset of temporarily ambiguous sentences. This will in turn also allow parsers to be trained on these sentences which might improve overall accuracy.

Within the broader AI community extra care needs to be taken when building incremental parsers to make sure that the parser also evaluates well on temporarily ambiguous sentences. As we have seen we can't assume that such a parser will be able to handle such sentences well, if at all.

References

[1] D. Jurafsky and J. H. Martin, *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition*. Upper Saddle River, N.J.: Pearson Prentice Hall, 2009, ISBN: 9780131873216 0131873210. [Online]. Available: http://www.amazon.com/Speech-Language-Processing-2nd-Edition/dp/0131873210/ref=pd_bxgy_b_img_y.

[2] A. Köhn, "Predictive dependency parsing," 2020.

[3] M. P. Marcus, "A theory of syntactic recognition for natural language.," Ph.D. dissertation, Massachusetts Institute of Technology, 1978.

[4] Y. Zhang, T. Lei, R. Barzilay, T. Jaakkola, and A. Globerson, "Steps to excellence: Simple inference with refined scoring of dependency trees," Association for Computational Linguistics, 2014.

[5] J. Nivre, "Algorithms for deterministic incremental dependency parsing," *Computational Linguistics*, vol. 34, no. 4, pp. 513–553, 2008. DOI: 10.1162/coli.07-056-R1-07-027. eprint: https://doi.org/10.1162/coli.07-056-R1-07-027. [Online]. Available: https://doi.org/10.1162/coli.07-056-R1-07-027.

[6] J. Nivre, M.-C. De Marneffe, F. Ginter, Y. Goldberg, J. Hajic, C. D. Manning, R. McDonald, S. Petrov, S. Pyysalo, N. Silveira, *et al.*, "Universal dependencies v1: A multilingual treebank collection," in *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, 2016, pp. 1659–1666.

[7] K. Foth, A. Köhn, N. Beuck, and W. Menzel, "Because size does matter: The hamburg dependency treebank," eng, in *Proceedings of the Language Resources and Evaluation Conference 2014 / European Language Resources Association (ELRA)*, Universität Hamburg, 2014.

[8] N. Silveira, T. Dozat, M.-C. de Marneffe, S. Bowman, M. Connor, J. Bauer, and C. D. Manning, "A gold standard dependency corpus for English," in *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*, 2014.

[9] C. Biggs. (2018), [Online]. Available: https://www.apartmenttherapy.com/garden-sentences-262915 (visited on 10/31/2020).

[10] H. Liu, "Dependency distance as a metric of language comprehension difficulty," *Journal of Cognitive Science*, vol. 9, pp. 159–191, Sep. 2008. DOI: 10.17791/jcs.2008.9.2.159. [Online]. Available: https://www.researchgate.net/figure/Dependency-structures-and-MDD-of-garden-path-sentence_fig3_273459859 (visited on 10/31/2020).

[11] VCM Lab. (2015), [Online]. Available: https://www.slideshare.net/vcmlab/cog5-lecppt-chapter09 (visited on 10/31/2020).

[12] I. Morris. (2014), [Online]. Available: https://slideplayer.com/slide/5838141/ (visited on 10/31/2020).

[13] Y. Tamura. (2015), [Online]. Available: https://www.slideshare.net/yutamura1/conceptual-plurality-in-japanese-efl-learners-online-sentence-processing-a-case-of-gardenpath-sentences-with-reciprocal-verbs (visited on 10/31/2020).

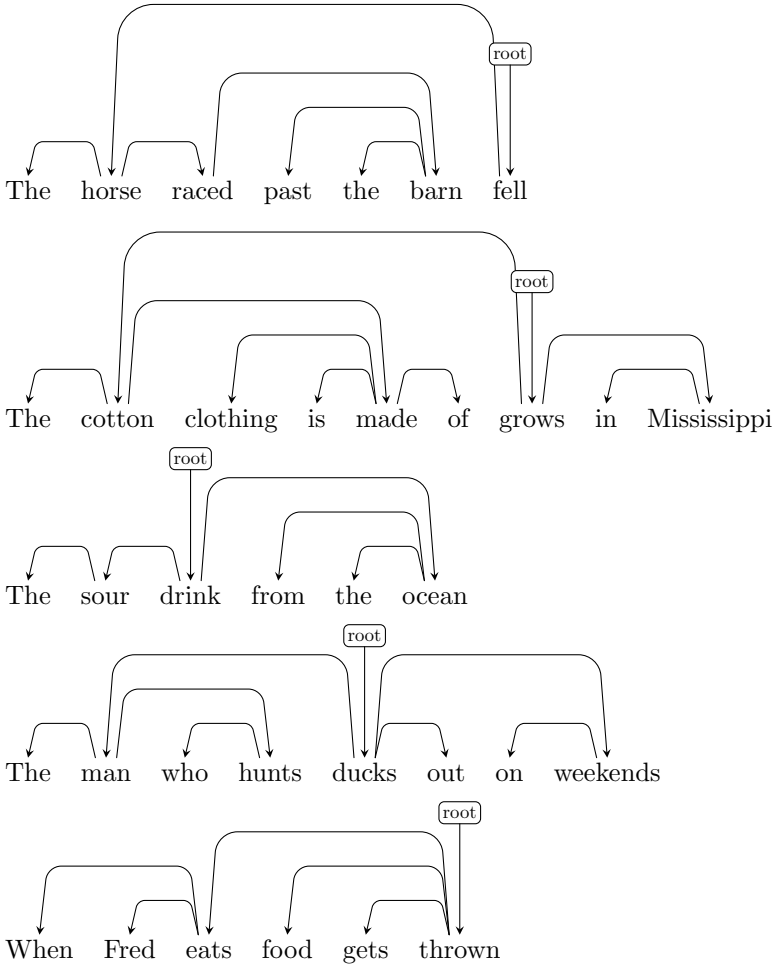
[14] Effectiviology. (2017), [Online]. Available: https://effectiviology.com/avoid-garden-path-sentences-in-your-writing/ (visited on 10/31/2020).

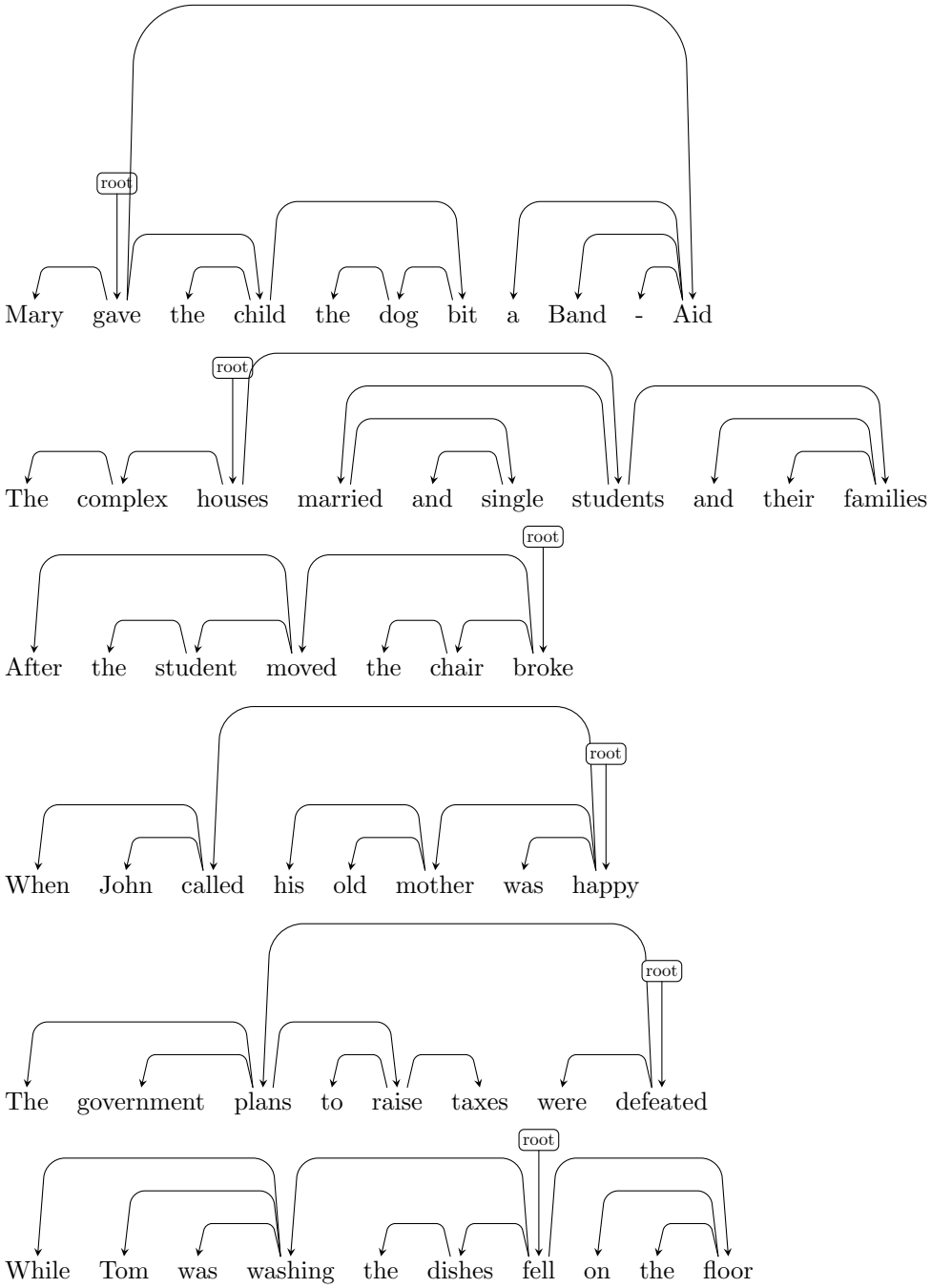
[15] M. Straka, "UDPipe 2.0 prototype at CoNLL 2018 UD shared task," in *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, Brussels, Belgium: Association for Computational Linguistics, Oct. 2018, pp. 197–207. DOI: 10.18653/v1/K18-2020. [Online]. Available: https://www.aclweb.org/anthology/K18-2020.

[16] A. F. Martins, M. B. Almeida, and N. A. Smith, "Turning on the turbo: Fast third-order non-projective turbo parsers," in *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2013, pp. 617–622.

A Garden path dataset

Below follows a sample of the dependency parses from the garden path dataset. The full dataset is available online at `https://github.com/blackwolf12333/thesis_experiments/blob/master/data/sentences.conll.no_punct`.




root
Mary gave the child the dog bit a Band - Aid
root
The complex houses married and single students and their families
root
After the student moved the chair broke
root
When John called his old mother was happy
root
The government plans to raise taxes were defeated
root
While Tom was washing the dishes fell on the floor