Universidad del País Vasco Euskal Herriko Unibertsitatea

FACULTY OF MATHEMATICS AND PHYSICS Charles University

Co-funded by the Erasmus+ Programme of the European Union

Annotation Inconsistencies in Universal Dependencies: Identification, and Correction

Author: Akshay Aggarwal
Advisors: Daniel Zeman, Charles University, Prague

Koldo Gojenola, UPV-EHU, Spain

Euopean Master's Programme in Language and Communication Technologies
EM-LCT

Final Thesis

September 2019

Departments: Computational Architectures and Technologies, Computational Science and Artificial Intelligence

---

Abstract

This research attempts at identification and correction of inconsistencies in different treebanks. The inconsistencies might be related to linguistic constructions, failure of the guidelines of annotation, failure to understand the guidelines on annotator's part, or random errors caused by annotators, among others. The work also proposes a metric to test the similarity of different treebanks in the same language, when the annotation guidelines remain the same. We offer solutions to some previously identified inconsistencies in the scope of Universal Dependencies Project in a language neutral manner, the solutions being reliable enough to not need a human annotator in the pipeline.

Acknowledgements

I would like to express my gratitude to my supervisors, Daniel Zeman from Charles University for his dedicated guidance as well as his recommendation of the thesis topic; and Koldo Gojenola from UPV/EHU for his valuable remarks and the much needed push to allow me to keep things structured.

I am very thankful to Prof. Markéta Lopatková and Prof. Vladislav Kuboň from Charles University, Prague and also Dr. Bobbye Pernice from Universität des Saarlandes for their immense help in handling all the difficulties encountered during the two years of studying in LCT programme.

I would also like to express my most sincere thanks to the numerous people on reddit who have constantly helped me with their pointers, and indulged me in discussions to have me see things in a language-independent manner, while also helping me with annotation tasks as and when required.

A very sincere vote of thanks is also extended to Charles University in Prague for the university's provision of computing and storage resources. A very heartfelt thanks is also due, to the brilliant researchers and teachers therein. The thesis would not have been possible without their brilliant shaping of the foundations of the subject within me. As such, I would also like to thank every teacher, and professor I have ever studied from.

Special thanks are also due to the following sources for the provision of computing resources.

Computational resources were provided by the CESNET LM2015042 and the CERIT Scientific Cloud LM2015085, provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures"

Computational resources were supplied by the Ministry of Education, Youth and Sports of the Czech Republic under the Projects CESNET (Project No. LM2015042) and CERIT-Scientific Cloud (Project No. LM2015085) provided within the program Projects of Large Research, Development and Innovations Infrastructures.

Finally, I would like to thank all the people who supported me during the two years, my family and friends. Special thanks are due to my friends in Prague for helping me keep my sanity in place.

Contents

1. Introduction

According to Wikipedia definition of the word[1], a treebank is a parsed text corpus, which annotates syntactic or semantic structure. Built usually (but not always) on top of a POS-annotated corpora, a treebank might seek to include phrase structure (Example- PennTreebank [Marcus et al., 1994]), dependency structure (Example- Prague Dependency Treebank [Böhmová et al., 2003]) or semantic information (Example- FrameNet [Baker et al., 1998]).

A treebank can be constructed manually, by linguists spending a considerable time developing the treebank; or semi-automatically, wherein the data is automatically annotated, and then checked for consistency. Regardless of the method used for creating a treebank, it is an essential element in the field of computational linguistics. A treebank can be used to study linguistic structures, find out features associated with a language, or to understand the constructional peculiarities within a language, among others.

In this work, our main focus is on syntactic treebanks and especially dependency treebanks, rather than semantic ones. Therefore, the term 'treebank' shall be used to refer to a syntactic (dependency) treebank henceforth, unless specified otherwise.

1.1 Inter-conversion of Treebanks

There exist a multitude of treebanks for different languages as they can be seen on Wikipedia[2], for example. As noted by Kakkonen [2006], there exist a variety of formats and annotation schemes even for the treebanks for the same language. A well known example to this is the case of distinctive POS tagging schemes for PennTreebank[3] and for British National Corpus[4], both of which are meant for annotation of English language. Kakkonen, in his work also notices that there exist tools which are meant to work for a particular tagset/annotation scheme. Notice that given enough similarities in annotation schemes, a conversion process can be drafted from one treebank to another, to make use of the resources available for the latter.

This conversion process of a treebank from one annotation scheme to another can be either 1:1 (one-to-one mapping) or n:m (many-to-many mapping). As in Machine Translation, the approach can also be pivot-based, i.e. conversion to an intermediate set, and then from the intermediate set to the desired set. For example, Interset [Zeman, 2008] uses the pivot-based approach, implemented as a Perl library. However, the mapping can still be deterministically applied, as in the case of POS tagsets for example.

It is important to note that not all the conversions are deterministic. If we consider an example of a dependency treebank where the dependency structure is changed from

[1]`https://en.wikipedia.org/wiki/Treebank`
[2]`https://en.wikipedia.org/wiki/Treebank#Syntactic_treebanks`
[3]`https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html`
[4]`http://www.natcorp.ox.ac.uk/docs/c5spec.html`

function-word head to content-word head structure, the entire dependency structures need to be modified, thus introducing problems. Such problems can be characterized by loss of information, loss of language-specific patterns, and induced inconsistencies in the data, among others.

Knowing the downside of fully-automatic conversion techniques, one can argue that we could do the task of treebank conversion manually, rather than automatically. This is not an ideal proposition because of multiple reasons as listed:

1. The treebanks differ in size, ranging from thousands of tokens to billions of tokens. An example would be WikiText-2 dataset [Stephen Merity et al., 2016], which contains around 2 million words, extracted from Wikipedia articles. The manual annotation on data as large as this requires time, money and significant human effort.

2. In case of multiple annotators for a given data, the different annotators may not always agree on the annotation principles for the same amount of data. This is especially the case when the guidelines are not specific enough, or in cases where the local grammatical annotation differs from the guidelines.

3. In case of low-resource languages, it might be difficult to find a technically-sound annotator for the language, thus rendering the process to be painfully slow, and in some cases inaccurate.

To combat this problem, an approach of semi-automatic conversion is preferred over manual or fully-automatic conversion. The semi-automated conversion procedure relies on converting the data from one annotation style to another automatically, followed by a human annotator verifying the results, and correcting them, if needed. A trade-off between the fully-automatic and manual conversion techniques, this approach is considerably faster than the manual approach, and allows the conversion process to be controlled for quality-check as compared to a fully-automatic approach. However, it is worth noting that there can be significant iterations (or revisions) of the treebanks needed before the converted data is again available at par with or better than the data quality in original scheme. Since the research breakthroughs and improvements don't wait for the data to be perfect, the task of checking for consistency, and/or quality of the treebanks has gained momentum in recent years as a research problem.

1.2 Universal Dependencies (UD) Project

As elaborated in the previous section, there are multiple and (possibly) conflicting annotation styles, even for the treebanks for the same language. Like any other measurement criteria where the standardized unit (in form of SI unit, or ISO standards) was needed to be defined, the different annotation styles required a similar form of standardization.

A rather more detailed history of UD Project can be accessed on UD homepage[5]. This is a shorter version thereof. Although there already existed annotation schemes that were

[5]`https://universaldependencies.org/introduction.html#history`

used as de facto standards, with the example of The Stanford dependencies [Marneffe et al., 2013], Google universal tagset [Petrov et al., 2012], HamleDT [Daniel Zeman et al., 2014], among others. However, there was still the problem of which annotation style to go for. McDonald et al. [2013] in their Universal Dependency Treebank (UDT) Project tried to provide with a universal annotation language, covering 6 languages in 2013, and expanding to 11 languages the following year.

With the modifications resulting in development of HamleDT 2.0 [Zeman et al., 2014], and Universal Stanford Dependencies (USD) [de Marneffe et al., 2014], the Universal Dependencies (UD) Project was thus born in 2014 as a means of unifying all the novel features of different annotation formats as a universal annotation scheme consistent among different languages.

The version 1.0 of UD (also referred to as UDv1.0) [Nivre et al., 2015] was launched in January 2015, and covered 10 treebanks in 10 different languages. With the iterative methodology, the project evolved to contain 146 treebanks in 83 languages in UDv2.4 [Nivre et al., 2019]. It is worth noting that not all the treebanks in UD are manually annotated. Rather, most treebanks are semi-automatically converted from the original source to the UD format according to a set of guidelines[6].

1.3 Motivation for the Problem

Since the introduction of UD, it has fast become a standard reference to compare scores relating to parser performance (Che et al. [2018], Alonso et al. [2017]), study of language-specific features [Alzetta et al., 2018], and for Shared Tasks on UD [Zeman et al., 2018]. Given how different UD treebanks are being considered as benchmarks for comparison of different scores, it only makes sense to be considered them as Gold Standard (GS) data.

We discussed earlier how many of the UD treebanks are generated through a semi-automatic process, and thus are liable to contain a significant amount of errors. Such errors are detrimental in a GS, because of multiple reasons including, but not limited to:

1. In the case of parser evaluation, the parser learns errors from the data as well, replicating them when used on test data. While this affects parser evaluation scores, it also means that the parser does not learn the features of the language correctly, thus causing increasingly more errors on the real world data (data not in the treebank).

2. Since semi-automatic conversion is also likely to introduce more errors, this can result in inflating/deflating already known errors/features. These patterns can become a nuisance on the treebank-level or might disappear altogether. Consider the case of a language-feature F which is a rare phenomenon in language L, with the relative occurrence of $x_0\%$ in the original data. Due to conversion process, it is possible that the relative occurrence might change to $x_1\%$, where $x_1 \neq x_0$.

[6]`https://universaldependencies.org/guidelines.html`

- In case of the inflation of error ($x_1 > x_0$), the data which otherwise did not exhibit F suddenly starts displaying the pattern, thus affecting the quality of the data.
- In case of the deflation of pattern ($x_1 < x_0$), the data might not exhibit F at all, increasing its rarity. Considering the case of parser evaluation as above, the parser might decide to overlook this feature in entirety, thus losing out on essential data.

3. With respect to identification of language-specific features, it is very possible that a lot of features might start getting wrongly associated with a language (the case of inflation as above) or they might be deemed a rare status (the case of deflation as above). Such instances, while seemingly harmless for high/medium resource languages, can pose serious problems with respect to low-resource languages, impacting the way the given language is studied.

It is worth noting that the problems as mentioned above are but a subset of multiple problems associated with an erroneous GS, and how they affect UD and the research around it. As such, these problems need to be minimized as much as possible, eliminating them too, in an ideal case. However, doing the task manually is again a difficult task and the automatic methods are not always 100 % reliable and/or effective. This is in part also because of the different properties of languages, different language families, et al.

1.4 Formal Problem Statement

Having learned about the UD project, problems concerning semi-automatic conversions, and possible effects of these problems within the scope of UD, we can now formulate our problem statement for the scope of the thesis as follows:

Given the different treebanks in UD, the thesis aims to identify inconsistencies in treebanks, and provide a corresponding automated correction tool for them. The inconsistencies might be related to linguistic annotation, improper adherence to guidelines, lack of guidelines related to an observed phenomenon, annotator caused error, among others. The proposed methods should ideally not require a human annotator for verification, and should be as language-neutral as possible. However, language-specific methods can also be investigated and reported.

1.5 Data Source

This work was started in January 2019, when UDv2.3 was the latest release. As such, a majority of the experiments contained herein were first developed for UDv2.3. However, with the release of UDv2.4, most experiments were carried over to UDv2.4 data. It is worth noting that there are some experiments which were never done for UDv2.3 and were started from scratch on UDv2.4.

Nonetheless, there are some experiments that work well for UDv2.3 and UDv2.4 throughout, and there are some that work better for only one of the releases, mainly owing to the difference in count of the corresponding error pattern(s). To facilitate the understanding of experiments better, each experiment shall contain a note specifying the dataset (which also mentions the release version of UD) on which the experiment was conducted.

1.6 Organizational Layout of the Document

Now that we have formalized the problem statement, this subsection deals with organization structure of the rest of the document. We first continue the preface of the document by very quickly noting a few conventions that are used throughout this document. In Chapter 2, we take a look at the different categorisation of errors, and then the typology of different problems identified in UD treebanks. We continue the document with Chapter 3, containing the background on the research pertaining to the problems as discussed in Chapter 2. In the subsequent chapters (Chapters 4 - 5), we layout the individual problems, and elaborate on the method/approach undertaken to solve the problem(s). Given that a lot of different problems were identified, and tried to be solved (some without success), Chapter 6 focuses on the problems that could not be solved, while offering viewpoint on what can be done in future, and/or why the used approach failed in practice. In Chapter 7, we discuss on some of the open problems as identified by the other authors, which were not undertaken in the current work. We officially conclude the document with a chapter on Conclusions.

Attached to the document are also a series of Appendices. The appendices contain the data meant to help the reader understand some of the terms used through out the document, with an example being a list of ISO language codes used throughout.

1.7 A Brief Overview of Conventions Used

This section is an overview on some of the important conventions used throughout the length of the document.

1. The following conventions hold with respect to the UD terminology. A short introduction to different terms associated with UD can be accessed in Appendix A.1.
   - Unless otherwise mentioned, 'POS' refers to 'UPOS'.
   - Syntactic Relations in UD can be referred to by either of 'relation(s)', 'deprel', or 'dependency relation'. Unless otherwise mentioned, the instances refer to the 'udeprel' part of the relation.
2. The POS tags, as well as dependency relations are formatted in the same formatting style, with one essential difference. Both the categories are marked orthographically using a separate tag in LaTeX, with the tag being referred to as `\verb`. We refer to

this formatting style as ‘tag’ category.
Table 1.1 shows the difference in formatting as below:

Text Format	Command	Output
Regular	Lorem ipsum	Lorem ipsum
Italic	`\textit{Lorem ipsum}`	*Lorem ipsum*
Bold	`\textbf{Lorem ipsum}`	**Lorem ipsum**
Tag	`\verb|Lorem ipsum|`	`Lorem ipsum`
Tag2	`\texttt{Lorem ipsum}`	`Lorem ipsum`

Table 1.1: Illustration of Formatting styles

- The POS tags are always capitalized (written in upper-case), while the deprels are always non-capitalized (written in lower-case).
- In the event of the Figure and Table Captions, or in (sub-)headings, the formatting style associated with ‘Tag’ category cannot be used. The formatting style of ‘Tag2’ category is used instead.

3. When referring to specific files that are part of the standard release of the accompanying module, the filename is referred to with ‘Tag’ category formatting.

4. The use of ‘Tag’ category is also reserved for nomenclature of problems. Thus, a problem identified as `ProblemX` will act as the unique identifier for the problem across the length of the document.

5. The languages are referred to by their language-identification codes whenever possible.

   - A complete list of languages in UDv2.4, with their identification codes can be seen in Appendix A.2.
   - The language codes are also formatted using ‘tag’ category as defined above. Given the nature of the dependency relations, it should be easily possible to disambiguate the language code from the former.
   - In case of an unclear distinction, the language name corresponding to the language-code shall follow in parentheses.

6. The name of the different treebanks are written in the format of `LanguageCode`-treebank_name. The truecasing in the name of the treebank is optional.
   For example, the SynTagRus treebank for `ru` can be referred to by either of `ru`-syntagrus or `ru`-SynTagRus.

7. The tokens taken from a language other than `en` follow a pattern when mentioned inline:

- For the tokens with Latin based orthography, the token is marked in bold, followed by a literal translation in parenthesis. Consider the following example from `nl`, written inline in text with `en`.

  Example 1. Lorem ipsum text **hier** (here).

- For the tokens with non-Latin based orthography, the token is again marked in bold, followed by the transliteration of the token in italics, and the literal translation of the token in regular case, separated by a semi-colon. The transliteration, and the translation are mentioned in the parenthesis following the token. Consider the following example from `ru`, written inline in text with `en`.

  Example 2. **да** (*da*; yes), this is Lorem ipsum text here.

- For the case where LTR (Left-To-Right) languages are mentioned inline with RTL (Right-To-Left) languages, the transliteration and translation are written for the tokens in the order of utterance. Consider the following example, assuming **A, B and C** is written in RTL as **C and B ,A**. The example would be written inline with `en` in the following way:

  Example 3. This is the Lorem Ipsum for RTL language- **C and B ,A** (*A, B and C*; A, B and C)

2. Problems Identified in UD Treebanks

Ever since the UD project was introduced in 2014, and since the revision of guidelines in UDv2, there have been multiple publications that highlight the problems in UD treebanks. Some of the problems highlighted in these publications have been found to be global in nature (i.e. they occur in almost all treebanks, regardless of the language), while the others are related to a specific group of languages. Before we start discussing the problems, we should specify the general kind of errors.

Agrawal et al. [2013] define different kinds of errors that can be found in a treebank. The first kind are the random errors, characterised by the inconsistencies introduced by the annotators owing to the distractions while undertaking the annotation procedure. The systematic and recurrent errors are introduced not in isolated scenarios as random errors, but can be found across the treebank in a consistent manner. These errors are usually related to the guidelines of the treebank, in either of two ways. The guidelines could be misunderstood by the annotator(s), and/or the guidelines might themselves be unclear (or not appropriate to handle some cases), leaving the annotator(s) in a jeopardy. Alzetta et al. [2017] extend the definition of systemic and recurrent errors to also include the cases of conversion errors, caused by improper mapping of original annotation scheme to a new scheme. Throughout the length of this document, we focus on the errors of the second kind (systemic and recurrent errors), and try to correct them.

It is worth pointing out why the experiments listed in the section were chosen to work on, and not others. As we will see, apart from the first problem listed in next few sections, all of the error typologies were pointed out from a common source [Alzetta et al., 2017]. The authors of the paper note that the mined patterns were found to be common across different sections of the `it` treebank, and across different languages as well. Owing to the success of the algorithm in determining the typology of inconsistencies, it makes sense to use it on different datasets to flag the inconsistencies within them as well.

2.1 Intra-Language Inter-Treebank Harmony

UDv2.4 contains 146 treebanks in 83 languages. As such, there are multiple languages with more than one treebank, with some containing up to 5 treebanks. A list of all such languages, with the associated treebanks can be seen in Appendix A.3. Regardless of the differences in genre or the teams involved for building the treebank, the different treebanks for a language should be close to each other if the annotation scheme remains the same. However, this is not often the case, owing to different sources the original treebanks originated from. The problem of determining the degree to which the different treebanks differ from each other is not yet entirely solved. We propose a new metric for the purpose of determining the degree to which two treebanks differ in this section. This is not an error pattern, but a metric proposal problem.

Alonso and Zeman [2016] note that if the two treebanks from the same language are as similar as possible, the differences in parsing accuracy (training parser on one of the treebanks, and using it to parse the other language) would be due to differences in dataset size, and domain change; but not due to differences in dependency convention.

Rosa and Zabokrtsky [2015] show that KL-Divergence score of POS trigrams can be effectively used for source selection for POS Tagging. In their approach, they are able to select effectively not just a singular source, but also for source-weighting in multi-source transfer. Computing the KL-Divergence on POS trigrams, they call the measure as KL_{cpos^3}, defined as follows:

Definition 1.

$$KL_{cpos^3}(tgt, src) = \sum_{\forall cpos^3 \in tgt} f_{tgt}(cpos^3) \cdot \log \frac{f_{tgt}(cpos^3)}{f_{src}(cpos^3)} \tag{2.1}$$

where $cpos^3$ is a coarse POS tag trigram, and

$$f(cpos_{i-1}, cpos_i, cpos_{i+1}) = \frac{count(cpos_{i-1}, cpos_i, cpos_{i+1})}{\sum_{\forall cpos_{a,b,c}} count(cpos_a, cpos_b, cpos_c)} \tag{2.2}$$

with $f_{src} cpos^3 = 1$ for each unseen trigram.

Intuitively, treebanks of the same language should be a better fit for single-source transfer than a treebank from another language, and so KL_{cpos^3} for a single-source transfer can be considered as a good benchmark for deciding this. However, KL_{cpos^3} is a variant of KL-Divergence, and thus is non-symmetric. Therefore, we should rely on a metric that essentially calculates the divergence of the treebanks from each other, in both directions.

Combining the above two observations, we can arrive at a definition of treebank harmony.

Definition 2. Given two treebanks A and B, we say the treebanks are in harmony with (or, are harmonious to) each other, iff

1. The difference in labelled attachment scores (LAS) when trained on one treebank and tested on another, denoted by θ_{LAS}, is less than or equal to a given threshold θ_1. Mathematically, it can be represented as:

$$\theta_{LAS} = |LAS_{x,x} - LAS_{y,x}| \leq \theta_1 \qquad \forall [x, y \in \{A, B\}] \tag{2.3}$$

   where $LAS_{P,Q}$ indicates LAS when trained on P and tested on Q.

2. The difference in KL_{cpos^3} scores of the treebanks calculated in both directions, denoted by θ_{POS}, is less than or equal to a given threshold θ_2.
   Mathematically, it can be represented as:

$$\theta_{POS} = |KL_{cpos^3}(x, y) - KL_{cpos^3}(y, x)| \leq \theta_2 \qquad x, y \in \{A, B\} \tag{2.4}$$

   where $KL_{cpos^3}(P, Q)$ indicates KL_{cpos^3} score of Q as an estimator for P.

Mathematically, we denote two harmonious treebanks A, B as $A \doteq B$ over (θ_1, θ_2). The relation of harmony is reflexive, symmetric, but not transitive.

As mentioned earlier, there exist up to 5 different treebanks (not including PUD) for a given language. More often than not, the treebanks cover different domains, and are of different sizes. As such, it becomes an important criteria to determine the appropriate values for θ_1 and θ_2. If the values are too large, we run the risk of saying the treebanks are harmonious even when they might not be. Also, if the values are too small, we could be overlooking at the effect of domain change and dataset size, to mistake the two treebanks as being non-harmonious to each other.

We return to this problem in Chapter 4, when we look at it in detail with instances to determine as a metric on which to base the success of experiments in the research.

2.2 Problems Caused by Change of Guidelines in UDv2

A summarized version of changed guidelines from UDv1.x to UDv2 can be accessed online[1]. Most of the changes in guidelines could be processed in an automatic manner, for example the renaming of particular POS tags or dependency relations. There were still changes that could not be applied deterministically, and those form the majority of the problem conversions as we shall see in this section.

It is important to note that the changes had to be applied to 64 treebanks in 47 languages as they moved from UDv1.4 to UDv2.0, and so the analysis might be limited to these 64 treebanks only in this case. However, it is worth scouting for these patterns in the newer treebanks, given how some (if not all) of them might be a cause of concern therein.

The task of POS tagging, and dependency parsing of the input sentences were done with the help of UDPipe [Straka and Straková, 2016], which contains the trained models for UDv1.2, UDv2.0, UDv2.3 and UDv2.4. The tree structures shown henceforth are generated as per Parsito format [Straka et al., 2015].

2.2.1 conj_head

In the changed guidelines, there were two changes with respect to conjunction tags `CCONJ` and `SCONJ`; and the dependency relations, `cc` and `conj`. The changes are listed as follows:

1. The POS tag `CONJ` in UDv1 was changed to `CCONJ` in UDv2, to make it more parallel to `SCONJ`.

2. The coordinated structures are attached to the immediately succeeding conjunct in UDv2, as opposed to UDv1 where they were attached to the first conjunct.

Let us look at an example to understand this better. Consider the following example, taken from UDv2.4 `en`-LinES treebank. The dependency graph for this input, as per UDv1.2 and UDv2.0 looks as shown in Figure 2.1 and 2.2 respectively.

[1]https://universaldependencies.org/v2/summary.html

Example 4. This mode conforms closely to the ANSI-92 Level 1 specification, but is not ANSI-92 Level 1 compliant.

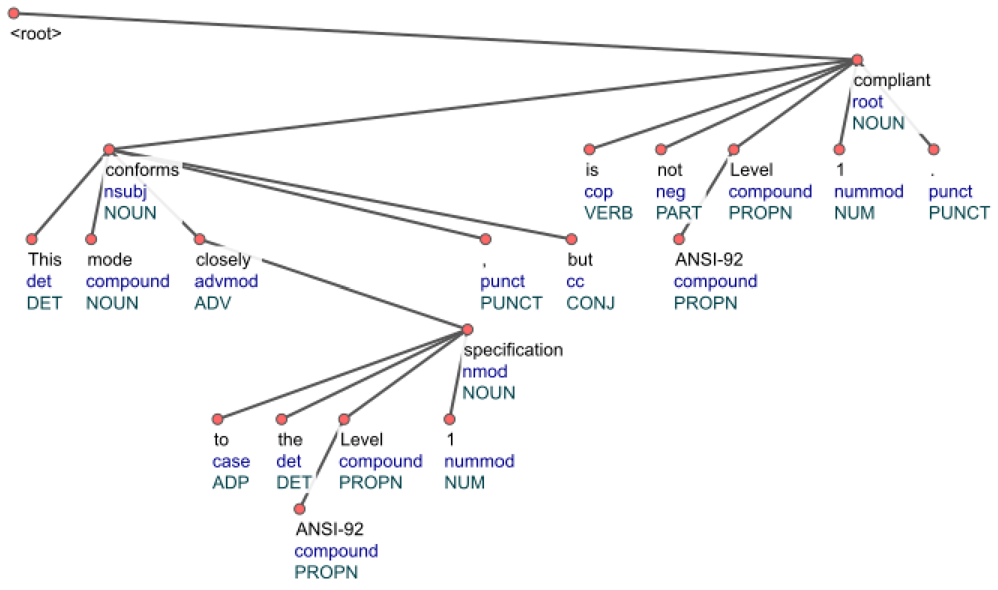


Figure 2.1: Tree Structure, as per UDv1.2 for Example 4

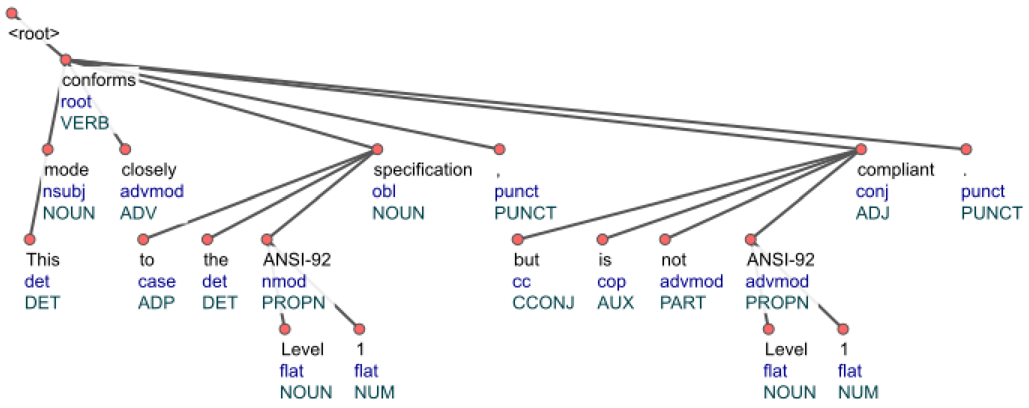


Figure 2.2: Tree Structure, as per UDv2.0 for Example 4

Due to the data size disparity in UDv2.0 and UDv1.2, the POS tags are aligned better in the former over the latter. Let us take a look at the tree structures with respect to *but* token. We can notice the difference in two aspects:

1. POS tag of the token is changed from `CONJ` to `CCONJ`.
2. Given the two conjuncts viz. *conforms*, and *compliant*, the token is linked to the former in UDv1.2, whereas the same token is linked to the latter in UDv2.0.

Of the two changes in guidelines, the first one (renaming of tag) can be applied deterministically, and automatically throughout the treebank(s). The second change, however, can be classified as head identification error. The pattern in question was also identified by Alzetta et al. [2017] in their paper, where they note that it contributes to 24.65 % of total discovered error instances. This implies that this is indeed a major error category, and needs to be handled well. We do take a look at this error type in our experiments, in Chapter 5, and discuss about it in detail therein.

2.3 Open Problems

2.3.1 Problems with Unfinished Experiments

This segment deals with problems that have been identified in the context of UD, but the experiments on the attributed problem are not yet complete. Nonetheless, we define such problems in short here.

Non-projective Structures

Let us understand non-projectivity through the following example from LinES treebank in `en` data, and the tree structure as shown in Figure 2.3.

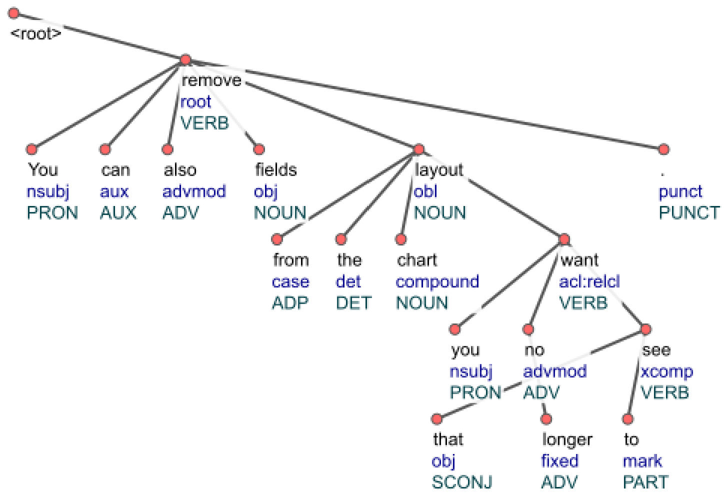


Figure 2.3: Sample Non-projective Tree

In the graph, notice the edge going from *see* to *that*. We can see that the edge crosses over another edge in order to link the two tokens. Informally, presence of such crossing edges in a tree makes it non-projective in nature.

To define the concept of non-projective structures in a formal manner, we need to define a few notations. We use the same notations as used by Mambrini and Passarotti [2013].

If a node j depends on a node i, we call node j as a child node of i (also, i is parent node of j), represented as $i \rightarrow j$. We use $i < j$ to denote the node i preceeding node j in the tree T. A node v lying in between the nodes i and j in the tree can be represented as $v \in (i, j)$. Also, we use the notation $v \in Subtree_i$ if node v is part of the subtree rooted at node i. From Havelka [2007], we can define the condition of projectivity of a tree as follows:

Definition 3. A given tree T is projective in nature iff

$$i \rightarrow j \ \& \ v \in (i, j) \implies v \in Subtree_i \qquad \forall i, j, v \in T \tag{2.5}$$

If a given tree does not satisfy the above condition, it is said to be non-projective in nature. Furthermore, in case of non-projectivity, node v is said to be in gap, represented as $v \in Gap_{i \leftrightarrow} j$. The double headed arrow signifies the nodes being considered irrelevant of their order of occurrence in the tree.

In Mambrini and Passarotti [2013], the authors analyze the occurrences of the non-projective structures in `grc`, where they also study the effect of genre on the counts of non-projective structures (edges, and trees) in the language. Nonetheless, there has been no study conducted on the state of non-projective structures within the scope of all treebanks in UD to the best of our knowledge.

While non-projectivity is a characteristic of some languages, and especially more so of certain genres (poetry, for example); the increasing count of non-projective trees has been shown to affect dependency parsing in a negative way. Owing to semi-automatic conversion scheme, a lot of non-projectivities might also be introduced artificially. Thus, it becomes important to not only identify such cases of false non-projectivities (i.e. the cases which should have been marked as projective, but were annotated as non-projective), but also to remove them as it affects the treebank quality in general.

2.3.2 Problems Outside Scope of Current Research

Auxiliary as Head

Auxiliaries was another category that underwent significant changes in guidelines when moving from UDv1.4 to UDv2. Even though there was an extensive discussion about auxiliaries, and the changes that could be made; there were still problems. Some of these problems were anticipated, while there were others which could not be anticipated at the time. The following are the list of changes for auxiliaries from UDv1.4 to UDv2:

1. The definition of `AUX` was extended to include copula verbs, and non-verbal TAME (Time, Aspect, Mood, Evidentiality. Might/might not include Voice and Polarity) particles.

2. The definition of `AUX` was also extended to include `cop` (copula) verbs as they perform grammaticalized function in nominal clauses.

3. The `aux` relation was also expanded to include non-verbal TAME particles, as in the case of `AUX`.

4. The relation `auxpass` was removed from UDv2.0, making it as a subcategory of the larger `aux` relation, in the form of `aux:pass`. Essentially speaking, `auxpass` was demoted to a sub-category of `aux` relation.

In the discussion of this problem, we refer to the case when an auxiliary (`AUX` or `aux`) is treated as the head of a dependency relation. Although allowed in certain cases, the auxiliary should not be marked as the dependency head in general sense. Consider the following example in Figure 2.4, taken directly from Alzetta et al. [2017].

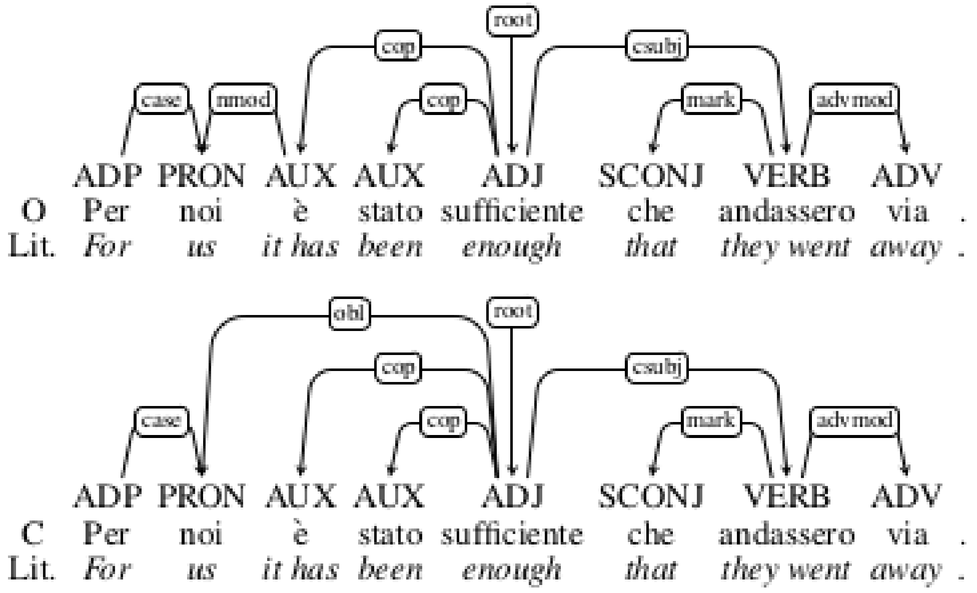

root
cop
csubj
case
nmod
cop
mark
advmod
ADP PRON AUX AUX ADJ SCONJ VERB ADV
O Per noi è stato sufficiente che andassero via .
Lit. For us it has been enough that they went away .
obl
root
cop
csubj
case
cop
mark
advmod
ADP PRON AUX AUX ADJ SCONJ VERB ADV
C Per noi è stato sufficiente che andassero via .
Lit. For us it has been enough that they went away .

Figure 2.4: Example taken from Alzetta et al. [2017]

In the figure, **O** refers to the original (incorrect) instance, whereas **C** refers to the corrected instance. Notice how the incorrect instance has *è* (`AUX`) serving as a dependency head. Alzetta et al. [2017] notice that this particular error, classified as a head identification error, contributes to around 13 % of the total discovered erroneous instances.

2.3.3 Problems with Failed Results

`AUX` and `VERB` Distinctions

There is a significant overlap between `VERB` and `AUX`. The distinction between `AUX` and `VERB` is not always explicit, and is very liable to be affected by the agreement between the definitions of the terms in UD, and according to the traditional language-grammar. This is noted in part in the guidelines for UDv2 as well, where the following point is noted, with reference to the definition[2] of `AUX`:

```
[AUX] is often a verb (which may have non-auxiliary uses as
well) but many languages have nonverbal TAME markers and
these should also be tagged AUX.
```

One of the proposed change in guidelines was to get rid of `AUX` altogether[3]. However, as per findings of de Lhoneux and Nivre [2016], a parser is not able to learn the distinction between the two categories, when they are merged together. The authors observe a decrease in parsing scores when the two categories are not explicitly separated. This was the principal motivation behind keeping the two separate, but there are still overlaps.

In UDv2.4, it was proposed to limit the `AUX` by a list. The list would essentially identify all auxiliaries by a common definition, and thus would be able to create a better distinction between the two conflicting categories of `AUX` and `VERB`. This could be realized just in part though, principally because of the conflicts between traditional grammar-based definitions of the two categories, and the definitions as per UD.

With respect to this particular error type, the experiment to separate the classes of `AUX` and `VERB` was unsuccessful with respect to UDv2.4, when not using the aforementioned list. We do discuss the failed experiment in Chapter 6 nonetheless. It is important to note that there still exist problems with respect to the differentiation between the two categories, as can be seen in the list of open issues on the subject[4].

[2]https://universaldependencies.org/u/pos/all.html#aux-auxiliary
[3]https://github.com/UniversalDependencies/docs/issues/275
[4]https://github.com/universaldependencies/docs/issues?utf8=%E2%9C%93&q=is%3Aopen+aux

3. Previous Research

In this chapter, we discuss some of the solutions that have been proposed/used by the different researchers. The solutions discussed here are limited in the scope of the problems identified in the last chapter. It is important to note that there have been numerous papers studying the different treebanks in UD, and the set of problems encountered while changing the annotation from the guidelines for UDv1 to UDv2. While such research is helpful in pointing out cases where the annotating teams had difficulties during the conversion procedure, we do not discuss those references here.

3.1 Error Mining Methods

Error mining in treebanks can be done in multiple ways. There is a possibility of using hand-written rules, and scouting for the patterns in the relevant treebank, which works for finding inconsistency typologies that are known beforehand. The other approach is to combine the statistical approach, with the manually defined rules [Ambati et al., 2011]. This method is what is often called as heuristics based search, since it identifies a lot of patterns, which can then be used to look for errors in the data (in some cases, this can be done automatically). The last approach is automatic scouring for error patterns within the scope of the treebank, also known as automatic error mining methods.

Boyd et al. [2008] first introduced the idea of error mining methods in dependency treebanks using variation nuclei, expanding on the idea of using n-grams based variation nuclei for discontinuous annotations from Dickinson and Meurers [2005]. It is important to point that the original idea was for syntactic annotation, while the method proposed in this literature was specifically tuned for dependency treebanks. This is often referred to as the first automatic error mining method in dependency treebanks.

The original method for discontinuous structural annotations involved looking at n-grams within a sentence that might have been separated by tokens. This was in turn, an extension of the previous research on n-gram variation nuclei for continuous annotation [Dickinson and Meurers, 2003a,b]. By extending the method to discontinuous annotations, Dickinson and Meurers were able to look at more patterns in TIGER corpus. Moreover, this meant that instead of looking at plain POS tags and identifying the variations therein, the words could now be looked at in order to generalize the context.

The method proposed by Boyd et al. was looked at in context of UD Treebanks by de Marneffe et al. [2017] for three languages (`en`, `fi`, `fr`). The authors further extended the method to use word lemmas instead of simply using word forms, and also evaluate on the automatically annotated treebanks to identify more inconsistencies. The first extension of using lemmas works well for languages that are not too morphologically-rich (`en`, `fr`), but fails otherwise. The second extension is done at the cost of a drop in precision, but without a significant gain in recall.

The method proposed by Boyd et al. has an inherent problem instance of data sparseness. De Kok et al. [2009] implemented an algorithm based on n-grams and suspicion sharing across the n-grams by extending the methods of Sagot and de La Clergerie [2006] and Van Noord [2004]. Their approach however, relies on classifying each sentence within the results of a parsed corpus as a parsable or unparsable sentences. This again is not very optimal for large treebanks.

In their work, Dell'Orletta et al. [2013] devised an unsupervised algorithm called LISCA (*LInguiStically-driven Selection of Correct Arcs*), which attempts to find the errors in dependency parsing by building a statistical model on the data from a gold standard. The algorithm learns the probability of the dependency arcs, given the gold standard, capturing the linguistic data and thus ranking the dependency arcs on the test data by their probability of occurrence.

In Alzetta et al. [2017], the authors identify errors in newspaper section of Italian UD Treebank by using the same algorithm. They narrow the search space for the errors by binning the arcs according to the scores into 10 bins of equal size and an extra bin to include the extra cases, and manually inspecting the bins for errors, while concentrating on the last two (and the extra) bins containing the arcs with lowest scores. Analysing the data, 36% of the arcs in the low ranking bins consisted of random errors, while the remaining ones were found to be systemic and recurrent errors (even in treebanks of different languages).

While the algorithm mentioned above successfully points out the erroneous arcs in the different datasets, it is sensitive to the genre of the data. The authors note that the data should ideally belong to the same register/genre for the algorithm to function at its best. While this is problematic because in some treebanks it is not possible to separate the data from different genres, there is also a possibility of unavailability of enough data in a particular genre (i.e. a single genre contributing in a very small manner to the size of the treebank).

Added to these difficulties is the difficulty of training the algorithm. The algorithm essentially needs to be trained on a gold standard data, from which it builds a statistical model that is used to generate the probability scores of a dependency arc. In case of languages with no high-quality parsers available or for low-resource languages, this poses a cold-start problem where we do not have the data to train the algorithm, and so the algorithm cannot be used at all.

We tried solving this problem of cold-start by using the method of k-fold cross validation (with varying values of k). We selected one fold as test data, and trained the algorithm to build a statistical model on the remaining data. The experiment is under manual evaluation and owing to the large data size, would not be completed in time to include it in the current research document.

3.2 Treebank Harmonization

Owing to different annotation schemes for the different treebanks of a language, there is no standard evaluation metric to compare the relative distance of treebanks' annotation to each other to the best of our knowledge.

Alonso and Zeman [2016] compared the treebanks for `es` in UDv1.3. They assess the similarity of the different treebanks using dependency parsing. A high-efficiency parser was trained on one of the treebanks, and then trained on the another. The idea was to notice the drop in LAS scores, and if the difference in scores was more than what was intuitive, the treebanks were marked as not similar enough. The same technique of evaluating the different treebanks for `ru` against each other was also used in Droganova et al. [2018]. In their work spanning the different `ru` treebanks in UD, Droganova et al. also point out problems with individual treebanks. This can again be used to compare and scout for patterns that are present across the different treebanks for the language.

Nivre and Fang [2017] proposed an evaluation metric called CLAS (Content-based LAS) score that disregard the punctuation and other functional nodes, evaluating LAS based on content words only. The idea here was to give equal treatment to the languages with weak morphology and languages with strong morphology. For example, a single inconsistency in `fi` will affect the parsing score more than a single inconsistency in `en` owing to the differences in the extent of morphology used by the languages. The metric was evaluated as a secondary measure in CoNLL 2017 Shared Task [Zeman et al., 2017]. The primary metric for the Shared Task was macro-averaged score for the different languages. It was reported that the there is no significant performance difference in parser performances when the evaluation metric was changed from macro-averaged LAS score to CLAS score.

4. Experiment 1: Intra-Language Inter-Treebank Harmony

We previously defined the term harmony between two treebanks in Definition 2 (Section 2.1). We also mentioned that we need to optimize the two parameters (θ_1, θ_2) for any languages to not over-reject or over-accept the effects of dataset difference, and domain change.

4.1 Dataset

Before we start tuning the hyper-parameters, we need to define our dataset. This experiment was conducted entirely on UDv2.4 data, without exceptions. To minimize the effect of dataset size disparity, we remove from consideration all the treebanks which are missing train data, as listed in Appendix A.5. From these treebanks, we remove only the ones which do not contain any train data whatsoever.

Furthermore, there are treebanks which have data in the format where it needs to be fetched from another corpus and is not readily available for usage. We discard such treebanks as well from the consideration. Thus, the effective dataset for this entire experiment can be seen in Table 4.1 as follows:

Language	Count	Treebank Names
cs	4	CAC, CLTT, FicTree, PDT
en	4	EWT, GUM, LinES, ParTUT
es	2	AnCora, GSD
et	2	EDT, EWT
fi	2	FTB, TDT
fr	3	GSD, ParTUT, Sequoia
gl	2	CTG, TreeGal
grc	2	Perseus, PROIEL
it	4	ISDT, ParTUT, PoSTWITA, VIT
ko	2	GSD, Kaist
la	3	ITTB, Perseus, PROIEL
lt	2	ALKSNIS, HSE
nl	2	Alpino, LassySmall
no	3	Bokmaal, Nynorsk, NynorskLIA
pl	2	LFG, PDB
pt	2	Bosque, GSD
ro	2	Nonstandard, RRT
ru	3	GSD, SynTagRus, Taiga
sl	2	SSJ, SST

Continued on next page

Language	**Count**	**Treebank Names**
sv	2	LinES, Talbanken

Table 4.1: Dataset for the Experiment on Harmony Between Treebanks, UDv2.4

4.2 Tuning Parameter θ_1

In accordance with Alonso and Zeman [2016], we report the LAS scores in form of confusion matrices for each of the languages. The horizontal rows mark the treebank the parser was trained on, while the vertical column marks the treebank on which the parser was tested. The results can be as seen in Table 4.2.

es	AnCora	GSD
AnCora	86.36	68.07
GSD	68.81	85.30

et	EDT	EWT
EDT	85.19	75.60
EWT	72.76	92.35

fi	FTB	TDT
FTB	92.30	48.65
TDT	52.17	87.78

gl	CTG	TreeGal
CTG	82.02	25.27
TreeGal	56.67	90.97

grc	Perseus	PROIEL
Perseus	70.54	43.23
PROIEL	32.02	75.84

sv	LinES	Talbanken
LinES	91.37	74.50
Talbanken	76.05	91.99

fr	GSD	ParTUT	Sequoia
GSD	91.07	78.99	78.17
ParTUT	78.38	96.12	77.32
Sequoia	78.59	78.23	93.85

ko	GSD	Kaist
GSD	63.95	28.07
Kaist	32.76	72.27

lt	ALKSNIS	HSE
ALKSNIS	90.02	51.66
HSE	47.26	88.98

nl	Alpino	LassySmall
Alpino	88.10	79.06
LassySmall	76.56	92.34

pl	LFG	PDB
LFG	98.72	67.10
PDB	84.11	88.13

pt	Bosque	GSD
Bosque	87.91	63.63
GSD	36.67	88.67

ro[a]	Nonstandard	RRT
Nonstandard	00.00	00.00
RRT	00.00	84.49

sl	SSJ	SST
SSJ	94.43	56.01
SST	71.82	88.07

[a]Annotated data not in correct CONLL-U format, UDPipe parser not trainable or testable on the data

`la`	ITTB	Perseus	PROIEL
ITTB	86.29	38.34	39.38
Perseus	29.18	77.34	38.53
PROIEL	35.83	34.12	75.82

`ru`	GSD	SynTagRus	Taiga
GSD	90.56	70.62	61.85
SynTagRus	73.55	86.59	67.22
Taiga	70.02	69.75	84.62

`no`	Bokmaal	Nynorsk	NynorskLIA
Bokmaal	91.47	82.27	61.20
Nynorsk	85.26	91.12	62.01
NynorskLIA	69.79	68.66	89.03

`cs`	CAC	CLTT	FicTree	PDT
CAC	85.25	72.57	80.78	78.55
CLTT	68.92	93.83	64.40	63.64
FicTree	74.20	69.51	92.30	74.25
PDT	81.61	76.21	83.15	84.85

`en`	EWT	GUM	LinES	ParTUT
EWT	88.29	77.06	75.56	70.30
GUM	76.36	91.48	75.50	71.39
LinES	70.41	69.81	90.83	67.05
ParTUT	66.70	68.25	69.79	94.09

`it`	ISDT	ParTUT	PoSTWITA	VIT
ISDT	90.10	88.48	64.79	79.22
ParTUT	84.72	94.18	62.40	76.94
PoSTWITA	81.71	81.71	84.83	75.33
VIT	82.57	82.25	63.26	86.26

Table 4.2: LAS Scores (in %) for Different Treebanks per language, UDv2.4

A couple of inferences can be made quite clearly from the data, wherein we see how the performance of the parser model is affected with respect to treebanks. Let us take a look at the parser performances for `pt`, and `gl` for example. As can be seen clearly, the parser performs differently on different treebanks for each of the languages. Even with significant considerations for the disparities in size, genre, etc. the two treebanks can be considered really divergent from each other.

Similarly, if we look at `ko` parser scores, we can see that the parsers perform really badly even when training and testing data remains the same. This is testimony to the fact that the treebanks do not follow consistent annotation scheme within itself. As a result of the inconsistency within the treebank itself, the parser trained on such data automatically performs worse for the other treebank.

We have almost discounted for the size of the data by selecting the treebanks with some training data available, but not entirely. Consider the case of `cs` treebanks for example. The token counts of the train data in different treebanks for `cs` is as shown in Table 4.3. Owing to smaller training data, the parser trained on `cs`-CLTT data performs the poorest among the parsers in the language. At the same time, due to larger training data, the parser performs the best when trained on `cs`-PDT data. Therefore, we need to still regulate for train data size disparities.

Treebank	Sentence Counts
CAC	23 478
CLTT	860
FicTree	10 160
PDT	68 495

Table 4.3: train Size of `cs` Treebanks

If we take a look at the sub-table containing LAS scores for parsers in `fi`, the two parsers perform unsatisfactorily on the other treebank. However, looking at the data in Table 4.4, we can safely discount the effect of size disparity as the sole reason for the low performance. Looking at the GitHub repositories for the two treebanks[1][2], and the base reference for TDT Treebank [Haverinen et al., 2014], we can understand the genre distribution in the two treebanks, as listed in Table 4.5.

Treebank	Sentence Counts
FTB	14 981
TDT	12 217

Table 4.4: train Size of `fi` Treebanks

Treebank	Genres
FTB	grammar-examples
TDT	news, wiki, blog, legal, fiction, grammar-examples

Table 4.5: Genres in `fi` Treebanks

Notice that while the TDT Treebank contains multiple genres, FTB is composed of a singular genre. In sub-section 4.3.2, we study the effect of genre classification with respect to parser performances. It is worth noting that while the genre classification can easily be done in case of some treebanks (as in case of `fi` above), it might not always be possible to do so.

[1]`https://github.com/UniversalDependencies/UD_Finnish-FTB`
[2]`https://github.com/UniversalDependencies/UD_Finnish-TDT`

4.2.1 Optimization for Size Disparity

To properly account for size disparities, we need to keep in mind two aspects, as listed:

1. The data is homogeneous in nature. By homogeneous, we mean that the data should be crawled from same or very similar sources. Also, the term also implies that the data should be from a singular genre so that genre based fluctuations do not affect the scores.

2. We should have sufficient data available, so as to be able to compare the different sized splits, and their relative performances.

`hi`-hdtb treebank is one of the treebanks that satisfies both the above conditions. The treebank's split of train, test and dev data is as shown in Table 4.6. As an additional data reference, we use news section of `hi`-pud. The size of the splits of the PUD treebanks in general, is as shown in Table 4.7.

split	Sentence Counts
dev	1 659
test	1 684
train	13 304

Table 4.6: Size of `hi`-hdtb treebank

Category	Sentences
News	500
Wiki	500
Total	1 000

Table 4.7: Sentences in PUD treebanks

We process the train data of `hi`-hdtb in a workflow, defined as follows:

1. Split the data into sizes in the range of {5, 10, ..., 95} % of the original data size.

2. On the split data, train a UDPipe Parser for the split, with the dev data given as heldout, and report the scores on `hi`-hdtb test data, as well as news section of `hi`-pud.

3. Report the LAS Scores in a graphical format.

The graph in Figure 4.1 shows two curves that demonstrate the performance of the parser as the training data increases in size. In case of `hi`-hdtb train data, the splits until around 12.6% are less than the size of the test data. That is marked clearly in the graph as the lowest performance score by the parser. As the split size increases, and thus the number of training instances, the parser performs better and better, the performance increasing monotonically.

In case of `hi`-pud news section, even at 5%, the size of training data exceeds the size of test data. However, the parser performance is significantly lower than on `hi`-hdtb test data. Also, unlike the previous curve, the performance is largely varying, with multiple local minima.

Even though the data belongs to the same genre in the two testing sets, there is considerable difference between the two. In the case of `hi`-hdtb data, the dataset is crawled from

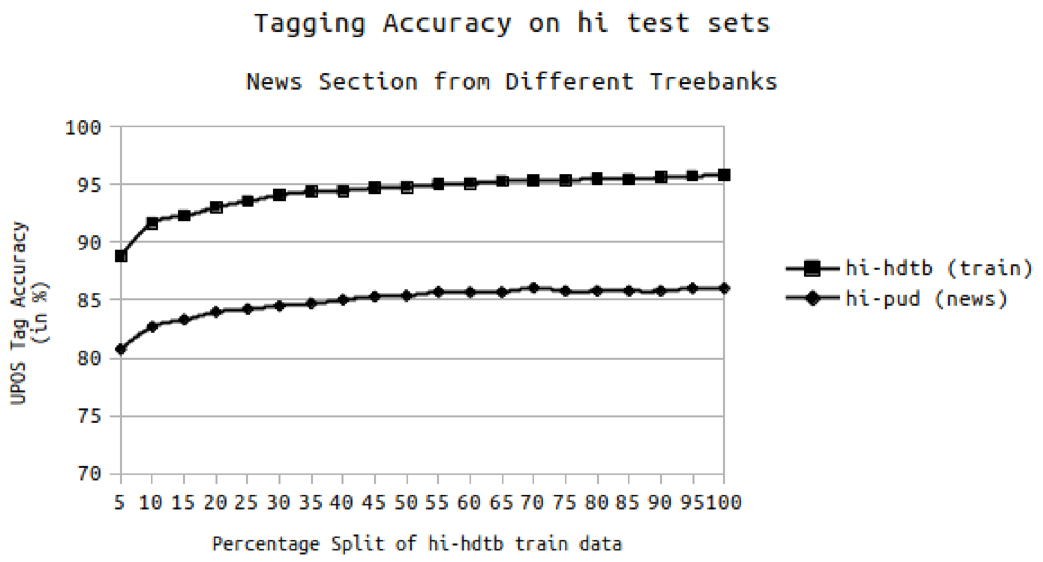


Figure 4.1: LAS for size disparity

newspaper dailies, and thus contains news articles, as they were printed. In the case of `hi`-pud news section though, the data was originally in either of English, German, French, Italian or Spanish, and they were translated to Hindi by using English as pivot language. Thus, there are instances where the nuance of the original language might be lost, since the news reporting style in the two languages is not very often similar.

Nonetheless, we can make some general statements about the performance by looking at the given data. We notice that based on the available data for training, the performance of the parser can vary as much as 10% when on the same data (i.e. data belonging to same documents as well).

Since the parser performance starts saturating at different points, we can describe the saturating points in the form of size factors, and how the LAS score can be affected when considering the effect of size alone. We give a formula for the upper bound of the difference in metrics based on size first, and then illustrate it with an example.

Definition 4. Given two treebanks, A and B, with α as the size ratio of the treebanks, we can bound θ_{LAS} by an upper bound given by $\theta_{1,size}$ as below.

$$size(A) > size(B) \implies \alpha = \frac{size(A)}{size(B)}$$

$$\theta_{1,size} = \begin{cases} \frac{min(10,\zeta)}{32} + \frac{\gamma}{16} + \frac{\epsilon}{8} + \frac{\delta}{4} + \frac{\beta}{2} & \text{if } \theta_{LAS} > 1 \\ 1 & otherwise \end{cases} \tag{4.1}$$

where $\zeta = \lfloor \alpha - 80 \rfloor$, $\gamma = \lfloor \alpha - \zeta - 60 \rfloor \leq 20$, $\epsilon = \lfloor \alpha - \zeta - \gamma - 40 \rfloor \leq 20$, $\delta = \lfloor \alpha - \zeta - \gamma - \epsilon - 20 \rfloor \leq 20$, $\beta = \lfloor \alpha - \zeta - \gamma - \epsilon - \delta \rfloor \leq 20$, $\beta, \delta, \epsilon, \gamma, \zeta \geq 0$

Example 5. We are given two treebanks, containing the same genre of data, differing in the size. Let us assume the two given parameters as:

$$\alpha = 92.5$$
$$\theta_{LAS} > 1$$

We can estimate the upper bound on θ_{LAS} as follows:

$$\begin{gathered}
\zeta = \lfloor \alpha - 80 \rfloor = \lfloor 92.5 - 80 \rfloor = \lfloor 12.5 \rfloor = 12 \\
\gamma = \lfloor \alpha - \zeta - 60 \rfloor = \lfloor 92.5 - 12 - 60 \rfloor = \lfloor 20.5 \rfloor = 20 \\
\epsilon = \lfloor \alpha - \zeta - \gamma - 40 \rfloor = \lfloor 92.5 - 12 - 20 - 40 \rfloor = \lfloor 20.5 \rfloor = 20 \\
\delta = \lfloor \alpha - \zeta - \gamma - \epsilon - 20 \rfloor = \lfloor 92.5 - 12 - 20 - 20 - 20 \rfloor = \lfloor 20.5 \rfloor = 20 \\
\beta = \lfloor \alpha - \zeta - \gamma - \epsilon - \delta \rfloor = \lfloor 92.5 - 12 - 20 - 20 - 20 \rfloor = \lfloor 20.5 \rfloor = 20 \\
\theta_{1,size} = \frac{10}{32} + \frac{20}{16} + \frac{20}{8} + \frac{20}{4} + \frac{20}{2} = 0.3125 + 1.25 + 2.5 + 5 + 10 = 19.0625 \\
\theta_{LAS} \leq \theta_{1,size} = 19.0625
\end{gathered}$$

We placed an upper cap in the formula, with respect to the parameter ζ. This is important, since we do not want the parameter to become large enough to dominate the other calculated parameters. Consider the following example:

Example 6. Consider two treebanks with same genre, and with difference in the size metric. Let us the assume the two given parameters are:

$$\alpha = 120$$
$$\theta_{LAS} > 1$$

In this case, we have the calculated parameters

$$\zeta = 40; \quad \gamma = \epsilon = \delta = \beta = 20$$

If we calculate the parameters with respect to ζ and γ without an upper cap, we get

$$\frac{\zeta}{32} = 1.25$$
$$\frac{\gamma}{16} = 1.25 = \frac{\zeta}{32}$$

At this point, the final parameter ζ starts having an equal influence as that of γ.

In case of a high enough value of α parameter, the parameter shall influence the results as heavily as other parameters, if not more. We do not want that to happen, and thus we restrict the calculated value by using the limit of 10.

4.2.2 Optimization for Genre Distribution

Optimization for Genre distribution is considerably more difficult than that for size. Very often, the genre distribution in the treebanks is not proportional. Also, we need to account for the fact that it might be easy in some cases to identify the proportions of the genre distribution easily, as we showed in Table 4.5. However, it might not always be the case, especially in the case if the treebank has been converted multiple times, i.e. the original annotation may be lost/unavailable.

For the optimization procedure, we use `fi` treebanks, the counts and genre for which were given in Tables 4.4 and 4.5 respectively. The distribution counts of different genres in `fi`-tdt is given in Table 4.8.

Category	Sentences Count
grammar-examples	2 002
legal	1 082
news	1 120
wiki	2 269
Others	5 744

Table 4.8: Distribution Counts of various genres in `fi`-tdt treebank

Since we are accounting for the genre differences, it only makes sense to keep the size of the dataset constant. We can compare the impact of addition of genres to the data in two ways. The first is by adding genres in fixed proportions so that the proportion of each genre in the treebank is equal, while the size of treebank remains constant. The second approach is to add an equal number of instances of each genre. We would use the first approach in our case, because a parser trained on the second approach has a possibility of learning more general features of the language, and performing better, rather than focusing on the proportion of the genres involved therein.

Definition 5. In a treebank, consider n different genres, with count of genre i in the treebank given by G_i. The total size of the treebank T can be given by:

$$|T| = \sum_{i=1}^{n} G_i$$

To study the effect of genre variance, we need to add genres to the data, such that the total size of the treebank remains the same. We say we arrive from a given treebank T to a modified treebank $T^{'}$ with the number of genres given by n and $n+1$ respectively in the following way:

1. The size of the treebank doesn't change, i.e.

$$|T^{'}| = |T|$$

2. The added genre in $T^{'}$ has equal number of instances to other genres. We refer to the number of instances for a genre in $T^{'}$ by $G^{'}$.

$$\begin{aligned}
\forall i \in \{1, ..., n\}[G^{'}_{i+1} = G^{'}_{i}] \\
\forall j \in \{1, ..., n-1\}[G_{j+1} = G_{j}] \\
\implies \sum_{i=1}^{n+1} G^{'}_{i} = \sum_{j=1}^{n} G_{j} \\
\implies \forall i \in \{1, ..., n\}[(n+1) \cdot G^{'}_{i} = n \cdot G_{i}]
\end{aligned}$$

With the proposed addition methodology, we will be able to study how the addition of genres in proportional increments (proportional to the size of the treebanks) affects the comparison metric. We modify a bit the methodology for the study of the metric in this case. We first generate the data for the experiment in the following manner:

1. Let the ordered set of different genres in `fi`-tdt treebank be represented as $X = \{$wiki, grammar-examples, legal, news$\}$, and individual genre be represented by x_i such that $\forall i, x_i \in X$.
2. Split the original genres in `fi`-tdt treebank.
3. For each split (one split contains data from one genre), we downsample it to the sentence counts as in Table 4.9. We refer to the downsampled split containing genre x_i by T_{x_i}. We further split all T_{x_i} into 1:1 ratio to achieve T_{x_i}-test and T_{x_i}-train splits.

Genre	Original Size	Downsampled Size
wiki	2269	1800
grammar-examples	2002	900
news	1120	600
legal	1082	450

Table 4.9: Downsampling Genre-wise Data for `fi`-tdt Treebank

4. For $i \in \{1, 2, 3, 4\}$, we start combining different T_{x_j}-train and T_{x_j}-test splits in the following manner:
   (a) train$\{i\}$ contains instances from T_{x_j}-train splits such that $j \in \{1, ..., i\}$.
   (b) test$\{i\}$ contains instances from T_{x_j}-test splits such that $j \in \{1, ..., i\}$.
   (c) $\mid$ train$\{i\}$ $\mid$ = $\mid$ test$\{i\}$ $\mid$ = $\mid$ `fi`_ftb-train $\mid$ = $\mid$ `fi`_ftb-test $\mid$ = 1800
   $\forall i \in \{1, 2, 3, 4\}$

Essentially, i in train$\{i\}$ or test$\{i\}$ refers to the number of genres contained. We can summarize the distribution of instances per genre in either of test$\{i\}$ or train$\{i\}$ splits as in Table 4.10.

T	G_i	$\|T\|$
train1	900	900
train2	450	900
train3	300	900
train4	225	900
test1	900	900
test2	450	900
test3	300	900
test4	225	900

Table 4.10: Data Split of `fi`-tdt for studying effect of Genre Distribution

For studying the variability of LAS scores, we train the data on each train split, and test the trained parser on all the test splits. Note that we do not need to ascertain for the size difference of the treebanks, since we made sure that remains as the case. We define the workflow for the experiment as follows:

1. For all train splits, do:
   (a) Train a UDPipe Parser on the split, with the default arguments being passed to the parser.
   (b) Get LAS scores of the trained parser on all the test splits.
2. Report the computed LAS scores.

Instead of reporting confusion matrices for the experiment, it makes more sense to represent the data graphically in a combined format for all the trained parsers. The resulting graph can be seen in Figure 4.2. We define ΔG as a metric first, and then point out some observations from the graph before trying to deduce an upper bound on θ_{LAS} scores.

Definition 6. We define ΔG as the difference in number of genres in training and testing data. Mathematically, it can be expressed as
G_{train} = The number of genres present in training data
G_{test} = The number of genres present in testing data

$$\Delta G = G_{train} - G_{test} \tag{4.2}$$

We use $|\Delta G|$ to refer to the absolute difference in genre counts.

The following observations can be made from Figure 4.2:

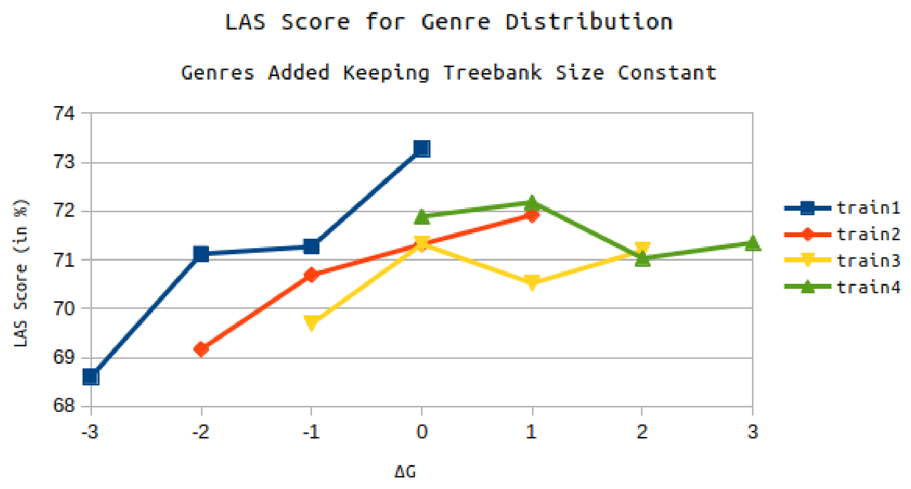


Figure 4.2: LAS Scores for Genre Optimization

1. The lesser the G_{train} value, the more drastic is the effect of addition/removal of genre.
2. The effect of removing a genre is more detrimental to the parser performance than when a genre is added.
3. As more genres are added ($\Delta G > 0$), the parser performance is not significantly affected.
4. As more genres are removed ($\Delta G < 0$), the parser performance is affected significantly.

Definition 7. Observing the points noted above, and the graphical data, we can safely bound θ_{LAS} by a metric $\theta_{1,genre}$. The metric is defined as follows:

$$\theta_{LAS} \leq \theta_{1,genre} = \begin{cases} 2 & \text{for } \Delta G \leq 0 \\ \left\lfloor \frac{\Delta G}{2} \right\rfloor + 2 & \text{for } \Delta G > 0 \end{cases} \tag{4.3}$$

Since the addition of genres was not seen to have affected the LAS scores by more than an absolute value of 2%, we keep it that way for $\Delta G \leq 0$. The added factor in case of a negative ΔG value is to account for the fall of parsing quality, taken with an upper limit.

Like we mentioned earlier, the effects of genre addition/removal are more detrimental when G_{train} value is lower. This can be attributed to the fact that in case of decreased count of genres, the parser is not able to learn the genre-independent features of the language. In case of multiple genres, the parser relies less on genre-specific data, and learns genre-independent data better and thus the addition/removal of genres doesn't impact the LAS scores to a detrimental degree.

4.2.3 Other Factors

We took into account the two major factors that cause a dip in parser performance. However, there are a few other factors that also affect the parser performance. We discuss some of those factors in this subsection, without performing any experiments on them. Thus, we do not account for these features in our definition of harmony of treebanks, as we argue that these features should be corrected in treebanks (wherever necessary, and possible), rather than being accounted for.

We do not account anywhere for genre-specific vocabulary in any of the cases, since Alonso and Zeman [2016] and Droganova et al. [2018] prove in their experiments that LAS scores are not affected by genre/topic-specific vocabulary.

Very Long Sentences

We define very long sentences as the sentences with more than 25 tokens. As the sentence length increases, the distance of the nodes from the root of the sentence also increases. Collins [1996] showed how the distance between a token and the sentence root affects parser performance for syntactic trees. We can safely extrapolate on that information to extend it to the case of dependency parsing as well.

As the number of very long sentences increase in the treebank, the parser performance on the individual sentences decreases, and therefore the total score on the entire treebank as well. While it might not always be possible to get rid of such very long sentences from the treebank, care should be taken to keep the count of such sentences as minimal as possible, or a separate parser could be trained on such sentences separately.

Non-Projective Structures

We discuss non-projective structures later in the document in Chapter 7. However, it is worth mentioning at this point that the presence of non-projective structures has been known to affect the parsing quality. The greater the number of non-projective structures, the greater the difference in the parser performance. Minimizing the number of non-projective structures in the treebanks is a definite way to reduce the degree by which the treebanks differ.

Differences in Annotation Strategies

In different treebanks from the same language, there might not be agreements in the annotation scheme. As Droganova et al. [2018] note, the difference in annotation of `бы` (*by*; would) in `ru`-SynTagRus and `ru`-Taiga as an auxillary in the different treebanks causes the parser trained on `ru`-SynTagRus to be able to predict only 5% of functional relations in `ru`-Taiga. Other annotation inconsistencies, like those of tokenisation of multi-word entities (MWEs), parataxis, `SYM` vs `PUNCT`, etc. also cause a dip in parser performance. Such inconsistencies are often a result of individual team's decision on annotation strategies differing from each other.

A correction on this aspect requires a more coherent dialogue between the different teams responsible for development of different treebanks, and should be encouraged. While it might not always be possible to catch these differences by an automatic tool also because we need to identify the areas where the annotation might be inconsistent between treebanks; the concept of variation nuclei [Boyd et al., 2008] can be used to some extent.

Other Incorrigible Factors

Apart from the above mentioned factors, the treebanks on the same language can also be influenced by the regional differences in the language. Consider the case of `en`, and with reference to the difference in the spelling distinctions between American English and British English. In case of a lexicalized parsing scenario, the differences in spelling can make differences on how the parser reacts to different tokens. Another notable example is with respect to Spanish spoken in Spain, and the variation(s) of it with respect to Spanish as spoken in Latin America.

Also there can be differences in the source of the treebank data. Consider the example of `ru`-Taiga treebank. The treebank was meant to capture the nuances of the online communication in `ru`, and thus incorporates many features of the online expression of the language, including but not limited to non-standard orthography of tokens, difference in casing from the formal standards, topic hashtags. Such associations are almost always parsed differently, and the parser is almost guaranteed to perform sub-optimally when trained/tested on such data, and tested/trained on a treebank not exhibiting such features.

4.2.4 Brief Discussion on θ_1 metric

While θ_1 metric compares the two treebanks to identify the problems in relation to each other, computing LAS scores of a treebank on itself can be considered as a reliable measure as well. Consider the case of `ko`-GSD in Table 4.2. A LAS of around 63% indicates either of two things:

1. The parser is not able to learn the features of the language, and we should try a different parser.
2. The annotation scheme within the treebank is not uniform.

We can check for the first case by checking with the multiple architectures of parsers for example. However, in the second case, all the parsers would yield lower scores than normal for the treebank. This was also one of the intents in reporting the scores in Table 4.2 in confusion matrices, and not reporting on simply the difference of LAS scores.

The architecture of a parser used for calculating θ_1 metric also would essentially change the variation of the scores, based on genres or on size. For example, the size-based disparity scores are in line with Velldal et al. [2017] where they report the similar patterns with respect to size variations. It might be an interesting study to extend the study based on this to different architectures and study the variation of scores in cases of genre-distribution as well as size-disparity.

4.3 Tuning Parameter θ_2

We briefly discussed on the metric KL_{cpos^3} in Section 2.1. In this section, we shall try to optimize θ_2 value, given θ_{POS} values for different treebanks as given in Table 4.11, with the scores listed as percentage of 1 (example- if the absolute value is 0.005, it will be reported as 0.5 since 0.5% of 1 = 0.005).

es	AnCora	GSD
AnCora	-	0.683
GSD	0.683	-

et	EDT	EWT
EDT	-	1.082
EWT	1.082	-

fi	FTB	TDT
FTB	-	0.511
TDT	0.511	-

gl	CTG	TreeGal
CTG	-	1.902
TreeGal	1.902	-

grc	Perseus	PROIEL
Perseus	-	5.423
PROIEL	5.423	-

ko	GSD	Kaist
GSD	-	4.722
Kaist	4.722	-

lt	ALKSNIS	HSE
ALKSNIS	-	0.795
HSE	0.795	-

la	ITTB	Perseus	PROIEL
ITTB	-	2.979	4.7
Perseus	2.979	-	10.874
PROIEL	4.7	10.874	-

nl	Alpino	LassySmall
Alpino	-	0.818
LassySmall	0.818	-

pl	LFG	PDB
LFG	-	1.38
PDB	1.38	-

pt	Bosque	GSD
Bosque	-	0.066
GSD	0.066	-

ro	Nonstandard	RRT
Nonstandard	-	0.746
RRT	0.746	-

sl	SSJ	SST
SSJ	-	2.607
SST	2.607	-

sv	LinES	Talbanken
LinES	-	0.413
Talbanken	0.413	-

fr	GSD	ParTUT	Sequoia
GSD	-	1.159	0.587
ParTUT	1.159	-	1.19
Sequoia	0.587	1.19	-

`ru`	GSD	SynTagRus	Taiga
GSD	-	1.837	2.976
SynTagRus	1.837	-	1.009
Taiga	2.976	1.009	-

`no`	Bokmaal	Nynorsk	NynorskLIA
Bokmaal	-	0.129	1.199
Nynorsk	0.129	-	1.099
NynorskLIA	1.199	1.099	-

`cs`	CAC	CLTT	FicTree	PDT
CAC	-	4.553	0.844	0.851
CLTT	4.553	-	6.024	5.358
FicTree	0.844	6.024	-	0.493
PDT	0.851	5.358	0.493	-

`en`	EWT	GUM	LinES	ParTUT
EWT	-	0.581	1.483	1.947
GUM	0.581	-	0.817	1.466
LinES	1.483	0.817	-	0.394
ParTUT	1.947	1.466	0.394	-

`it`	ISDT	ParTUT	PoSTWITA	VIT
ISDT	-	0.398	2.29	0.001
ParTUT	0.398	-	4.732	0.429
PoSTWITA	2.29	4.732	-	2.287
VIT	0.001	0.429	2.287	-

Table 4.11: θ_{POS} Scores for Different Treebanks per language, UDv2.4

4.3.1 Optimization for Size Disparity

To properly account for size disparities, we need to keep in mind two aspects, as listed in Section 4.2.1. We use the same dataset as we did while accounting for the size disparities for θ_1 parameter. However, we modify the workflow for the experiments as follows:

1. Split the data into sizes in the range of {5, 10, ..., 95} % of the original data size.
2. On the split data,
   (a) Compute θ_{POS} of the split data with the original data.
   (b) Compute θ_{POS} of the split data with news section of `hi`-pud data.

(c) Train a UDPipe Model on each split with the dev data given as heldout, and report the UPOS Tag Accuracy Score (in %) on the two sets of testing data (`hi`-hdtb and news section of `hi`-pud), in a graphical format.

3. Report θ_{POS} scores for the different splits in a graphical format.

The results for the Accuracy of the Tagger Model(s) can be as seen in Figure 4.3. As can be seen, the Tag Accuracy Score increases as the split size increases, except for some minor irregularities, which can be considered as within the scope of error. The above inference holds true in the case of both sets of testing data. It is also interesting to note that the rate of increase in Tag Accuracy is almost similar in both cases, thus making the two graphs almost parallel to each other.

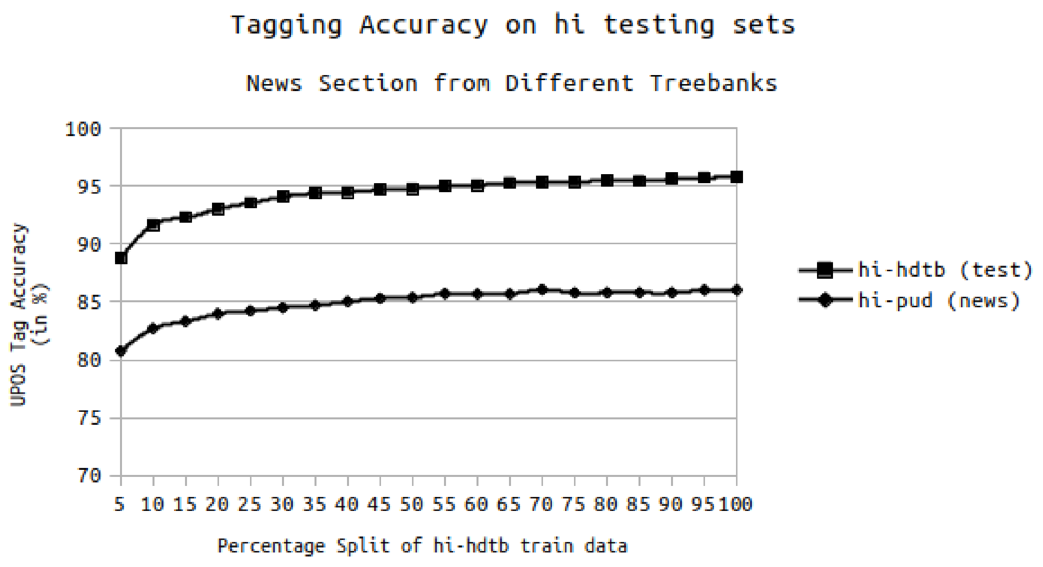


Figure 4.3: Tagging Accuracy Scores for size disparity

The results of difference in KL_{cpos^3} scores, also referred to as θ_{POS} scores, can be seen in Figure 4.4, where the constant line in the graph denotes θ_{POS} score for the two sets of test data. This figure requires a more thorough description than the last one, but we can summarize a few points based on a first glance:

1. In case of `hi`-pud news data, as the size of split increases, the θ_{POS} scores also starts increasing. Alternatively phrasing, even when the calculations are done on relative frequencies (cf. Definition 1), the θ_{POS} score starts diverging as the frequency counts increase.

2. The lowest score in case of θ_{POS} for `hi`-pud news section is almost the same as the score for when the data is compared against `hi`-hdtb test set.

3. In case of hi-hdtb test data, initially we observe a dip in the score, followed by a increase, and then the decreasing value again. Although expected to be monotonously increasing/decreasing, the behaviour marks the presence of some trigrams that make the scores diverge, before the sizable presence of the trigrams in question start causing the scores to converge.

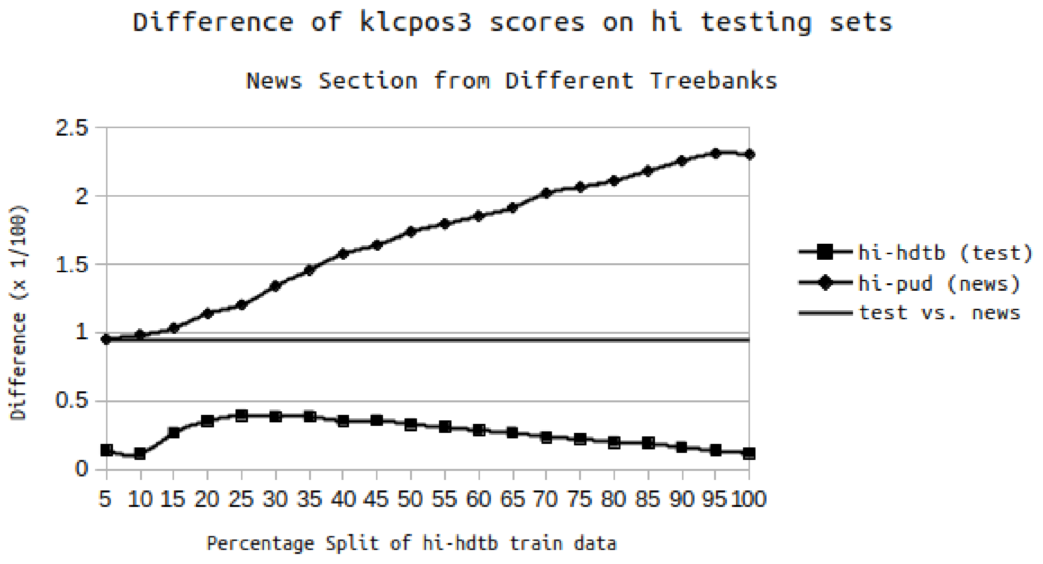


Figure 4.4: θ_{POS} scores for size disparity

It is interesting to note that in Figure 4.3, the difference in tagging performance on the two test sets of data was almost always constant. This should have been reflected when calculating θ_{POS} scores, but the graphs for the scores are diverging in nature, as can be seen in Figure 4.4.

With respect to the given data, the following points can be concluded for θ_{POS} metric, for the size-based disparity:

1. If the data comes from same sources (and is in same genre), we expect the metric to be less than 0.01, even if the size difference is 20x.

2. If the data comes from different sources (more likely case, owing to how different treebanks come from different sources), the metric score diverges until a certain point, until saturating (cf. Figure 4.4 for hi-pud data).

In our experiment, a size difference of 26x (news data of hi-pud, and train set of hi-hdtb, in number of sentences) caused the metric to be still less than 2.6 (10% of 26). In general, it should be safe to define the condition for θ_2 based on size, as follows:

Definition 8. Given two treebanks A, B such that the size difference between the two is α, and the parameter θ_{POS}, expressed as percentage of 1. The optimal value of the parameter θ_{POS} can be upper bounded by value of $\theta_{2,size}$ as follows.

The value $\theta_{2,size}$ can be limited between 0, and the larger value of 10% of α, or 1, subjected to a maximum value of 7.
Mathematically,

$$size(A) > size(B) \implies \alpha = \frac{size(A)}{size(B)} \implies \theta_{POS} \leq \theta_{2,size} = min(max(1, \frac{\alpha}{10}), 7) \quad (4.4)$$

The definition above bounds the value of θ_2 parameter, for size disparity. Consider the following example for two treebanks, which contain the instances from same genre:

Example 7. We are given two treebanks, containing the same genre of data, differing in the size. Let us assume the two given parameters as:

$$\alpha = 25$$
$$\theta_{POS} \text{ (absolute)} = 0.026 \implies \theta_{POS} = 2.6\% \text{ of } 1$$

We can compute $\theta_{2,size}$ as follows:

$$\theta_{2,size} = min(max(1, \frac{25}{10}), 7) = min(max(1, 2.5), 7) = min(2.5, 7) = 2.5 < \theta_{POS}$$

Thus, the given treebanks in this instance would not be harmonious with respect to parameter $\theta_{2,size}$.

There is a specific reason for the choice of limiting the $\theta_{2,size}$ value to 7. In most cases, the treebanks are split in train, dev and test in the proportion of 8:1:1. Considering a treebank that does not contain a dev set, the split would be in ratio of 9:1 for train, and test data. In the formula for KL_{cpos^3}, the choice for the base of the log is not important, affecting only the absolute values. However, if we use base 2, it makes it easier for the difference to be calculated in terms of number of bits. Noticing that $\log_2 90 \approx 6.49$, we include the case of unseen trigram pairs on either side, before taking a ceiling of this value, to mark the upper limit at 7.

4.3.2 Optimization for Genre Distribution

For optimization of the genre distribution with respect to POS scores, we used the same dataset as in Section 4.2.2. We computed KL_{cpos^3} score between all the train splits, all the test splits, as well as between every possible pair of train and test split. The results for the computed scores can be seen in form of confusion matrices in Table 4.12.

	test1	**test2**	**test3**	**test4**
test1	-	0.644	0.145	0.306
test2	0.644	-	0.496	0.258
test3	0.145	0.496	-	0.159
test4	0.306	0.258	0.159	-

	train1	**train2**	**train3**	**train4**
train1	-	0.75	0.272	0.421
train2	0.75	-	0.434	0.298
train3	0.272	0.434	-	0.158
train4	0.421	0.298	0.158	-

	train1	**train2**	**train3**	**train4**
test1	0.384	0.381	0.079	0.067
test2	1.036	0.242	0.745	0.612
test3	0.552	0.183	0.312	0.148
test4	0.672	0.037	0.433	0.288

Table 4.12: KL_{cpos^3} Scores for Genre Optimization

We can notice from the table that except for the score for train1 and test2 in the last part of the table, all the values are at most 1. This was a peculiar case, and when the experiment was retried with other different splits, the average scores were found to be less than 1 in this case as well.

We can very well say that addition/removal of genre does not affect POS distribution of the data. As such, we don't account for genre disparity in case of θ_{POS} scores.

4.4 Combining Optimized Values; Further Discussion

In our experiments, we optimized the two parameters θ_1 and θ_2 based on size and genre variations, by considering only one at a time. This is not always possible since the premise of the genre-distribution based study is the isolation of individual genres. Also, while some genres are extremely different (like wiki data and internet slang), some others are not very much so (like wikipedia and nonfiction). Realistically, we would need to factor genre-distribution and size-disparity together. In the sense, there can be different number of genres, and with each genre having a different size in the treebanks being compared. In such a case, the metric would need to be compared for size per genre, on basis of genre-distribution and finally, the overall size of the treebank.

While the metric reported in this experiment is far from perfect, it gives something to start comparing the data in different treebanks. We discussed the metric for calculating POS distributions, as well as LAS scores. It is not possible to identify and localize the source of errors using the metric, but it narrows the search space on where to look for. This, of course, only holds true if the genre distribution can be identified, and size composition of different genres identified. It would be an interesting study for future to learn if the effects of genre addition/removal are uniform across all genres, or are certain genre pairs more accommodating to each other (in the sense of less variance in LAS scores between them).

5. Experiment 2: conj_head

As discussed in Section 2.2.1, the problem identified as conj_head refers to the head identification error. This error is characterized by the coordinating conjunction being linked to the previous conjunct, rather than by the next conjunct. The latter of the two is as per UDv2 guidelines. We shall treat this problem in this section, with a glance through some of the observations on the problem in Section 5.1, allowing us to define our effective dataset in Section 5.2. We elaborate on our proposed solution to the problem, and the explanation of the algorithm used in the experiment in Section 5.3 and 5.4 respectively. We finally evaluate our experiment in Section 5.5.

5.1 Observations Pertaining to the Problem Statement

The problem of identifying the coordinating conjunction, and separating it from subordinating conjunction is a problem in itself, with the boundaries between the two sometimes not being explicit. Nonetheless, in our treatment of the problem, we shall identify coordinating conjunctions with their UPOS tag (`CCONJ`), and limiting ourselves to a particular deprel, `cc`. Notice that this distinction is necessary, and needs to be marked explicitly owing to the discussion of problem related to multiple deprels being associated to the UPOS tag (cf. Section 7.2). We take a look at the different issues associated with the problem that makes it a difficult one to solve in following subsections.

5.1.1 Direction of Dependency

One of the intuitive methods of approaching the problem at hand is to isolate the problematic token in a tree, and then check if the dependency edge to this token is in the correct direction. However, the identification of the correct direction can be non-trivial if worked in a language-independent manner. Consider the case of `sa`, and how it differs from `en`, as in Example 8. In `en`, the coordinating conjunction occurs in between the different conjuncts. In the given example for `sa`, the conjunction is linked with the last conjunct in a form that is typical of the language.

Example 8.
Text (`sa`): तस्य त्रयः पुत्राः परमदुर्मेधसः वसुशक्तिः उग्रशक्तिः अनन्तशक्तिश्च इति बभूवुः ।
Translit: *tasya trayaḥ putrāḥ paramadurmedhasaḥ vasuśaktiḥ ugraśaktiḥ* ***anantaśaktiśca*** *iti babhūvuḥ .*
Lit.: His three sons extremely-stupid Vasushakti Ugrashakti **Anantashakti-and** known-by-these-names there-were .
Translated: There were his three extremely stupid sons, called Vasushakti, Ugrashakti, and Anantashakti.

Note how in the data, the coordinating conjunction च (*ca*; and) is attached to the last conjunct, unlike in English where the conjunction (*and*) exists as a token on its own. It is also worth pointing out that the token referred to above is not the only coordinating conjunction in the language, with other conjunctions may/may not be attached to the last conjunct.

We have not yet talked about the case of RTL languages. Consider the case of Hebrew, for example. Written in right-to-left manner, the token for coordinating conjunction is added on to the next word as a prefix. However, this conjunction token is not the only prefix used in the language, and a singular word can have multiple prefixes. It would still have been possible to isolate the prefix if the associated character were reserved only for prefixes of such nature. However, the character in question can also occur as the first character in a word, without implying conjunction. The same process is elaborated in Example 9. The associated character is ו (*w*). The rules for listing the transliteration, as well as the translation are the same as defined for inline usage of a word from RTL language.

Example 9.
Text (`he`): התפוח והכלב; ורד; הייתי בטיול ובגדול נהנתי
Translit: *hṯp̄wḥ* ***w**hḵlḇ;* ***w**rḏd;* *hṯ ḇṭwl* ***wḇḡḏwl*** *nhnṯ*;
Lit.: the-apple and-the-dog; Rose; I-was on-a-trip **and-in-large** I-enjoyed;
Translated: The apple and the dog; Rose; I was on a trip and in general I liked it;

Although the problem may seem complex, it is not so. Effectively, we can consider the cases of `sa` and `he` as similar, differing only in aspect of prefix, or as suffix. Once we are able to identify the relevant affix, the problem can be simplified to that of direction problem. Notice that the identification of the relevant affix is a tokenisation problem, which is outside the scope of this research. In our treatment of the languages, we assume (and are given in CONLL-U representation) the syntactic tokens split into the smaller syntactic words.

5.1.2 Asyndetic Coordination

Asyndetic coordination refers to the case where the coordinating conjunction is omitted. A typical example of this is listed below, where comma (or some other punctuation) delimits the different tokens, and acts as conjunction marker. While this may be frequent in some languages, the alternative approach of using a conjunction between every conjunct is also possible.

Example 10.
Asyndetic: A, B, and C.
Notice the lack of a conjunction between A and B.
Non-Asyndetic: A and B and C.
The conjunction (and) is present in between every conjunct.

In either of the case, we need to restrict our focus on the present coordinating conjunction, and make sure the conjunction is linked to the next conjunct. The problem here remains the same, making sure the conjunction is attached in the right direction.

5.1.3 Nested Conjunctions

It can be argued that nested conjunctions can not be handled the same way as the other conjunctions in the scope of the problem. We use the examples as given in UD guidelines on the problem[1], without adding conjunctions in between the tokens.

Example 11.

1. A, B, C
2. (A, B), C
3. A, (B, C)

Without using the enhanced dependencies, the trees for the first two examples cannot be distinguished in the trees. Only the last example can be distinguished from the first two. However, if there are conjunctions present, all the three cases are distinguishable from each other.

It is important to note that we work with the hypothesis that the conjunction is located always close to the conjuncts. This is intuitive, but in the event of this being not the case, the conjunction will introduce a non-projectivity into the sentence. We do not want to introduce non-projectivities in the sentence where it was not already, although it might happen that we end up removing some of the non-projectivities in the process.

5.1.4 Conjunction Sandwich

We have so far discussed only the cases where the direction of dependency is wrong. Such cases are easily detectable. However, there is one more case which is significantly more difficult to determine. Consider the example of a subtree in Figure 5.1. Notice that token 'B' is the conjunction, while 'A', and 'C' are conjuncts. The conjunction 'B' should be correctly linked to 'C'. The node 'D' refers to the shared parent of nodes 'A', 'B', 'C', or the root of the sentence, as the case might be.

In the above figure, we can observe that the direction of the association of the conjunction is correct. However, that does not mean that it is linked to the correct head. This problem can be present in the default annotation, or might be introduced after the tree has been corrected for the misdirected dependency. We refer to such cases as a Conjunction Sandwich, since the conjunction is sandwich-ed in between the conjuncts, with no way of knowing the conjuncts. In the figure, we explicitly mention that the surrounding tokens (or subtree heads) are conjuncts. However, it is fair to assume that neither of the two would be labelled by the deprel, and that makes such cases even harder to detect. In our experiments, we tried to detect such cases, without any success. We therefore do not deal with such cases in this research.

[1]`https://universaldependencies.org/u/dep/all.html#nested-coordination`

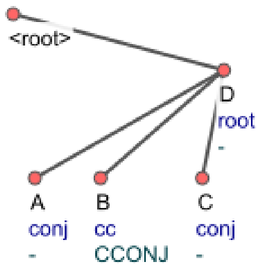


Figure 5.1: Possible Case of Conjunction Sandwich

5.2 Dataset Definition

The experiment was initially started on UDv2.3, but owing to the release of UDv2.4 in May 2019, the experiment was transported entirely to UDv2.4. It is worth noting that there were far more cases of this problem being identified in UDv2.3, rather than in UDv2.4. Nonetheless, there exist significant cases of the problem in UDv2.4 as well.

We limit our treatment of the problem to `af`, and `ar`. The treebank for working with `sa` is too small, and so we discard it from the dataset, owing to the smaller size of the treebank. The other languages are not considered since the number of erroneous instances with respect to attachment in wrong direction is too small. Note that we don't conduct any experiments on agglutinative languages, owing to the complexity of the resulting agglutinated token. Again, as in case of `sa` and `he`, it should be possible however to do so once the agglutinates have been identified, and isolated.

With respect to treebanks, `ar` has 2 different treebanks, not including `ar`-PUD. We focus our attention on `ar`-PADT treebank here because the other treebank for the language is delexicalised, and requires a license in order to obtain the textual data.

5.3 Experimental Setup

We start by identifying the cases where the direction of dependency is inverted. Once such cases have been identified, we start by flipping the direction of attachment, towards the most likely conjunct, as defined in previous section.

While `ar` is a RTL language, it is written in CONLL-U format in LTR format. As such, given that `af` is already an LTR language, the algorithm for identification, and correction of the direction of attached dependency is the same for both languages. Table 5.1 lists the counts of instances identified as attached in the wrong direction, in comparison to the total number of instances.

We limit our treatment of the problem to the case where we do not need to change the level in the tree (where the node is incorrectly attached) by more than 1. In essence, the wrong attachment of the conjunction can be corrected by finding a conjunct that is in the same level as the wrong conjunct, or is within the subtree of this wrong conjunct. In very rare cases, it might be necessary to attach the conjunction to the parent of the wrong

Language	Misdirected	Total	Percentage
af	1 829	1 832	99.836
ar	1 411	13 855	10.184

Table 5.1: Counts of Coordinating Conjunctions attached in wrong direction

conjunct it is attached to. If none of these is valid, we hypothesize that the annotation for the sentence was faulty, and thus it requires manual inspection for it to be corrected.

With respect to the above statement about the change of level being restricted to a maximal value of 1, there are 3 trees possible, as shown in Figures 5.2, and 5.3. The tokens follow the same conventions as discussed in previous subsection on Conjunction Sandwich.

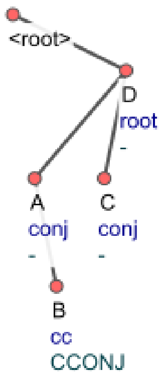
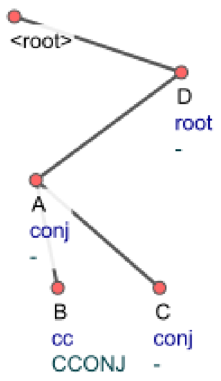


Figure 5.2: Possible Wrong Attachments of a Coordinating Conjunction: Both conjuncts at same level

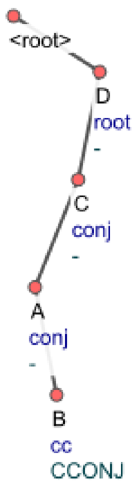
Notice that while the 3 cases are separate, there is no deterministic way of knowing what case an instance might refer to. As such, we handle all the 3 cases in decreasing order of priority, i.e. we try to handle the case as in Figure 5.2 first, failing which we try to solve it with respect to the case as in Figure 5.3a, and eventually as in Figure 5.3b. If a particular instance is still not corrected after the consideration of the last case, we leave it untouched. We describe in detail the algorithm for the task in the next subsection.

5.4 Algorithm

We start with defining some wrapper functions in Algorithm 1 and 2. While the first one checks for the coordinating conjunctions that are attached in wrong direction, the second one tries to change the parent of the given node x to a new parent z. In case the new association would be non-projective, the function rolls back to the previous parent. If the projectivity is preserved, the function returns a **true** value, which allows us to terminate the function whenever the function call is made inside another function. The function also checks against making the node as attached directly to the technical root of the tree, thereby making sure there is just one root node at any instance.



(a) Wrong conjunct at higher level



(b) Correct conjunct at higher level

Figure 5.3: Possible Wrong Attachments of a Coordinating Conjunction

Algorithm 1 misdirectedDependency()

```
Input: Node $x$
1: if $x.upos$ == “CCONJ” and $x.udeprel$ == “cc” and $x.parent.id < x.id$ then
2:     return true
3: end if
4: return false
```

Having defined our wrapper functions, we start by trying to attach the conjunction at the same level to other siblings in Algorithm 3. We first check to see if there is a single remaining sibling that does not have a POS tag of `X`, `PUNCT`, or `SYM` since we want to avoid the linking of the conjunction to these POS tags. We do these checks in lines 4-15, and in case of a single sibling being available, attach the node therein, and return a **true** value.

In case the condition is not met, we try to find the nearest sibling that has the deprel as `conj`, and try attaching the conjunction there. Again, this condition might fail. As a last resort, we try to find indirect dependents of the conjunct the conjunction is attached to, and link the conjunction there. We limit the set of indirect dependents by restricting the deprels to `obl`, `xcomp`, `nmod`, and `nsubj`.

Notice how we decide on whether or not the algorithm terminates by continuously checking the condition of projectivity, and returning a value from the function only if the condition of projectivity with respect to the new parent is maintained. It is also important to note that we always limit our search for a suitable candidate to cases where the candidate occurs later than the conjunction we are trying to rehang.

Algorithm 2 setParent()

```
Input: Node $x$, Original Parent $y$, New Parent Candidate $z$
1: if $x$ will be attached non-projectively to $z$ or $z$ is $ROOT$ then
2:     $x.parent \leftarrow y$
3:     return false
4: else
5:     $x.parent \leftarrow z$
6:     return true
7: end if
```

Algorithm 3 attachToSibling()

```
Input: $node$ such that $misdirectedDependency(node)$ == true
1: {Try to attach to a sibling node}
2: $count \leftarrow 0$
3: $origParent \leftarrow node.parent$
4: for all $siblings$ of $node$ do
5:     if $siblings.upos$ not in [“$X$”, “$PUNCT$”, “$SYM$”] then
6:         $TargetSibling \leftarrow siblings$
7:         $count \leftarrow count + 1$
8:     end if
9: end for
10: if $count == 1$ then
11:     {Just one sibling, attach to this sibling}
12:     if $setParent(node, origParent, TargetSibling)$ then
13:         return true
14:     end if
15: end if
16: {More than one siblings, narrow search by deprels}
17: for $sibling$ of $node$ do
18:     if $sibling.udeprel ==$ “$conj$” and $sibling.id > node.id$ then
19:         if $setParent(node, origParent, sibling)$ then
20:             return true
21:         end if
22:     end if
23: end for
24: for $sibling$ of $node$ do
25:     if $sibling.udeprel$ in [“$obl$”, “$xcomp$”, “$nmod$”, “$nsubj$”] and $node.id < sibling.id$
        then
26:         if $setParent(node, origParent, sibling)$ then
27:             return true
28:         end if
29:     end if
30: end for
31: return false
```

If there is no suitable candidate in the same level as the current level of the conjunction, we try to ascend one level and try to attach the node to the next aunt (parent's sibling) in Algorithm 4. We do not set any checks with respect to deprels, but still keep a check on the condition of projectivity and the node order.

Algorithm 4 attachToAunt()

```
Input: $node$ such that $misdirectedDependency(node)$ == true
 1: {Try to attach to the first relevant aunt node}
 2: $origParent \leftarrow node.parent$
 3: $aunts = []$
 4: for $sibling$ of $origParent$ do
 5:     if $sibling.id > node.id$ then
 6:         $aunts$.append($sibling$)
 7:     end if
 8: end for
 9: if $aunts$ is not empty then
10:     $setParent(node, origParent, aunts[0])$
11: end if
12: return false
```

In the event that a suitable candidate is not found, a **false** value is returned. This implies that our search for a suitable candidate has failed even after trying to ascend one level. As last resort, we try to attach the conjunction to the grandparent, while preserving projectivity in Algorithm 5.

Algorithm 5 attachToGrandparent()

```
Input: $node$ such that $misdirectedDependency(node)$ == true
 1: {Try to attach to the grandparent node}
 2: $origParent \leftarrow node.parent$
 3: $grandparent \leftarrow origParent.parent$
 4: if $setParent(node, origParent, grandparent)$ then
 5:     return true
 6: end if
 7: return false
```

Having established all the possible cases, we can wrap them all in a nice function that takes care of all the cases, in priority order. The final algorithm is as given below:

Algorithm 6 fix_conj_head()

```
Input: $node$ such that $misdirectedDependency(node)$ == true
1: if $attachToSibling(node)$ then
2:     return
3: else if $attachToAunt(node)$ then
4:     return
5: else if $attachToGrandparent(node)$ then
6:     return
7: else
8:     Do Nothing
9: end if
```

5.5 Evaluation and Results

We implement the algorithm in form of a Udapi-python block[2] [Popel et al., 2017]. The runtime of the block for the data is as mentioned in Table 5.2, as run on Ubuntu 18.04 (64-bit) on a 4-core Intel i5-6300 HQ processor.

Language	RunTime (in ms)
af	68
ar	162

Table 5.2: Runtime for Algorithm with Udapy Python Block

After applying the algorithm on the data, there were 17 (0.92 % of identified misdirected instances), and 359 (25.44 % of identified misdirected instances) cases in af and ar data respectively, which could not be handled. With respect to the unhandled cases in both the datasets, the algorithm is designed to work in a way that it does not over-generate. As such, all the cases where the algorithm would have over-generated were not handled by the algorithm.

We hypothesized earlier that if the rehanging of the node requires a change in more than one level (of the level of wrong conjunct), it is likely to be an annotation error that needs manual correction. We found that to be true for more than 50% of the cases in either treebank with respect to all the unhandled cases.

For the evaluation, we randomly sample 100 instances identified with wrong dependencies before the correction from each treebank, and report the score on the counts of wrong dependencies before, and after the correction algorithm. The scores can be seen in Table 5.3.

With the defined algorithm in previous subsection, we were able to correct around a significant amount of flagged error cases, just by identifying the direction of dependency.

[2]Code available at `https://github.com/Akshayanti/conj_head.git`

Language	Corrected Instances
`af`	95
`ar`	97

Table 5.3: Results of Experiment on `conj_head`, evaluated with 100 random samples

Even with a consideration of 5% error, the algorithm is able to fix the dependencies effectively.

The algorithm was further tested on `grc`-PROIEL and `grc`-Perseus data, and the UDPipe parsers trained on the corrected data. The LAS scores when the parsers were tested on itself and on the other treebank, are as shown in Table below.

`grc`	PROIEL	Perseus
PROIEL	75.84	32.02
Perseus	43.23	70.54

Table 5.4: Before correction

`grc`	PROIEL	Perseus
PROIEL	77.86	32.14
Perseus	43.55	70.70

Table 5.5: After correction

Although there is not a significant change in LAS scores before and after the correction, the general increase in the scores can be attributed to the increased uniformity of the directions of association.

6. Negative Experiment: AUX vs. VERB

We discussed in brief the problem of separating instances labelled AUX and VERB in Section 2.3.3 earlier. We shall treat this problem in this chapter, with a glance through some of the observations on the problem in Section 6.1, followed by the definition of the working dataset in Section 6.2. We elaborate on the proposed solution to the problem, and the results of the experiment in Section 6.3 and 6.4 respectively. We finally conclude this chapter with a discussion of the results in 6.5.

6.1 Observations Pertaining to the Problem Statement

According to the definition in UD[1], AUX is used as a common POS tag for verbal auxiliaries, as well as non-verbal TAME markers. The class of copulas are also included in this list.

This definition of auxiliaries is a bit different from Shopen [2007] which separates the two classes of auxiliaries and copulas in different categories. The work also points out the correlation between the position of an inflected auxiliary in relation to the verb, and other word properties of the language, as first pointed by Greenberg [1963]. In his work, Greenberg notes that the position of an inflected auxiliary in relation to the verb is generally the same as the position of verb in relation to an object. It is important to note that this generalization only holds for the inflected auxiliaries, and thus languages where the auxiliaries are not inflected are automatically ruled out from the consideration. Shopen points out the well-known exception to this generalization in case of verb-second languages like those of German.

While the generalization made by Greenberg is a very good marker for possible identification of inflected auxiliaries, the requirement of identification of auxiliaries in noninflected form still remains as a problem. This problem can however, be mitigated in part by the usage of the list of tokens identified as auxiliary in a given language, as was started in UDv2.4 with the help of a validator (cf. Level 5 checks in `validate.py`[2] file). It must also be pointed out that since Greenberg did not extend this generalization to SVO languages, the generalization only holds for languages with VSO and SOV dominant word-order languages. Combining that with verb-second languages, the generalization can not be used globally across all the languages.

When the copulas are included in the definition of AUX, the already difficult problem of separating AUX and VERB becomes even harder. In many languages, auxiliaries are a subset of verbs, with respect to specific usages. In other words, the same token can act as

[1]https://universaldependencies.org/u/pos/AUX_.html
[2]https://github.com/UniversalDependencies/tools

a verb or an auxiliary, depending upon the usage. The list of copula in many languages is also a subset of verbs, called as copulative verbs. However, as Shopen notes, there are cases of languages where the copula are not verbal in nature. The function of a copula can be realized by other means as well. The most common of these, viz. juxtaposition (example language- Ilocano), and use of predicators (example language- `bm`) are listed in the work, where they may be combined with existing copulative verbs in the grammar of the language.

In essence, while the class `AUX` in UD includes the copulative verbs, predicators, and other non-verbal TAME markers, the class `VERB` is composed of open class categories of verbs.

6.2 Dataset Definition

This experiment was initially tried on UDv2.3, but failed terribly. With the release of UDv2.4, this experiment was tried again, keeping the dataset treebank same, but changing the neural network model architecture. In the current documentation, we will use `hi`-hdtb treebank from UDv2.4.

There are a few reasons for the choice of the language for the experiment. In `hi`, we can more often than not draw a clear line of distinction between auxiliary as defined by UD, and the verbs. While the auxiliaries undergo inflection, and also include predicators and other TAME markers, they are restricted to a few tokens which rarely, if at all, are used as independent verbs. The factors as listed above, combined with the author's native fluency in the language makes it an ideal candidate for this experiment.

6.3 Experiment

We approach the problem at hand as a classification problem, specifically as a Sequence Labelling based NER problem. As part of this measure, we convert the data from the entire `hi` data to a format that suits the task[3].

There exist two tag formats for NER, namely IOB and IOBES. While IOB is composed of 3 tags- Inside, Outside, Begin; the IOBES tagset extends the IOB tagset by adding End and Singleton tags. The IOBES tagset helps with the better annotation of the data, as it provides more information. For example, in IOB tagset, the singleton entities are labelled with 'B' tag, without any following elements covered by 'I' tag. In IOBES, the tag 'S' is used to specify a single element being tagged. Similarly, the end of a sequence is not marked explicitly by IOB tagset, but is done with 'E' tag in IOBES format.

To convert our data into the desired format, we use the following methodology. All the instances marked as `AUX` are labelled as "S-aux", and all the instances marked as `VERB` are labelled as "S-verb". The rest of the tokens are labelled with 'O' tag. We do not consider contiguous tokens as a continuous chain, and thus not use either of 'I', 'B' or 'E' tags at

[3]Code available at `https://github.com/Akshayanti/aux_verb.git`

all. This is also done so as to have better control over each token that the model learns to tag, thereby increasing the granularity of the data, while keeping it simple.

For the task of NER, as well as POS Tagging, Flair embeddings [Akbik et al., 2018] were the SOTA at the time of performing this experiment. The embeddings were shown to outperform several models available at the time, across multiple NLP tasks, and therefore were the natural choice for this experiment. However, there are several hyper-parameters that can be tuned with respect to the models. We decided to tune the hyper-parameters with their corresponding choices as listed in Table 6.1. The best choice for the different hyper-parameters is also listed in the same table.

Hyper-Parameter	**Choices**	**Tuned Value**
Embeddings	**Stack1:** Forward and Backward Flair Embedding trained on `hi`-newswire **Stack2:** Word Embedding for `hi`, Forward and Backward Flair Embedding trained on `hi`-newswire	Stack2
Use CRF?	True, False	True
Use RNN?	True, False	True
RNN Layers	1, 2, 4	2
Size of Hidden Layer	32, 64, 128, 256	256
Dropout	Uniform Distribution in [0.0, 0.5]	0.25
Learning Rate	0.05, 0.1, 0.15, 0.2, 0.25	0.1

Table 6.1: Hyper-Parameters for Neural Network

Since we are trying to correct the gold standard itself, we are very liable to run into a cold start problem. To counter this problem, we perform a k-fold cross-validation on the data, with k=10. We concatenate the different splits of the treebank (train, dev, test) and then split the data into 10 folds, with test set in each fold disjoint with other test data in other folds. We then proceed to convert each of those folds into the IOBES tagset as we did with the unmodified data.

With the optimized parameters, we train models on each fold of the data. The trained model instances are then used to predict the test set in each fold as well. The output of the evaluation writes the predictions with the associated confidence in the predicted label. We here identify the 6 kind of patterns as listed in Table 6.2.

Category	**Original**	**Prediction**
aux_TP	S-aux	S-aux
O_TP	O	O
verb_TP	S-verb	S-verb
aux-O	O	S-aux
	S-aux	O

Continued on next page

Category	**Original**	**Prediction**
aux-verb	S-aux S-verb	S-verb S-aux
verb-O	O S-verb	S-verb O

Table 6.2: Categories of Error Patterns

Since we also have confidence scores associated with each prediction, we focus on a set of error patterns within certain bounds on the confidence scores. Figure 6.1 shows the distribution of confidence scores for instances where the predicted label matches the original label, with the associated confidence value lower than 0.80. For these categories, we focus on the subset where the confidence score is lower than 0.67. The idea is that since there are 3 categories, a prediction with a confidence lower than $\frac{2}{3}$ is liable to be erroneous. For the instances where there is a mismatch between the predicted label and the originally annotated label, we focus on instances with the confidence in prediction higher than 0.995. The idea in this case is that if the model is really sure about the prediction, the original annotation might be erroneous, and is worth looking into.

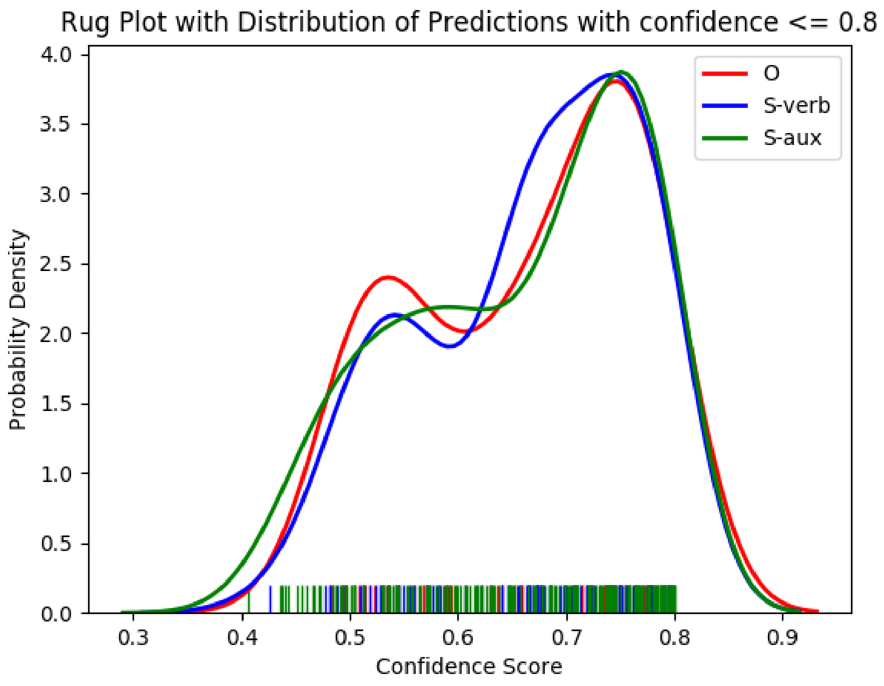


Figure 6.1: Rug plot with Distribution of Predictions with low confidence score

Having identified instances within each category that have confidence scores within the relevant bound, these instances were manually annotated to see if they were actually mislabelled patterns or not. We can summarize the entire experiment in the form of algorithm as defined in Algorithm 7.

Algorithm 7 Experiment to Identify Mislabelled `AUX` and `VERB` tags

```
Input: $data \leftarrow$ UDv2.4 treebank
 1: Convert $data.train$, $data.test$ and $data.dev$ to IOBES format
 2: Optimize Sequence Labelling NER model configurations for the $data$
 3: $model.config \leftarrow$ best sequence labelling NER model configuration
 4: $data.complete \leftarrow data.train + data.dev + data.test$
 5: {The different splits of the data concatenated together}
 6: $iter.id \leftarrow$ fold of $data.complete$, numbered as $id$
 7: {Performed 10-fold cross-validation to split $data.complete$}
 8: $model \leftarrow$ NER model with $model.config$ configuration
 9: for $id$ in $\{1, \dots , 10\}$ do
10:     $model.id \leftarrow model$ trained on $iter.id.train$ data
11:     $model.id.test \leftarrow$ Prediction of $model.id$ on $iter.id.test$ data
12: end for
13: $identified.pure \leftarrow$ Instances identified as True Positive across all $model.id.test$
14: {Confidence score $\leq 0.6700$}
15: $identified.cross \leftarrow$ Instances identified as False Positive or False Negative across all
    $model.id.test$
16: {Confidence score $\geq 0.9950$}
17: Manual Annotation of $identified.pure$ and $identified.cross$
```

6.4 Results

The output of running the NER tagger on test data within each of the fold returns the predictions of the model, and an associated confidence score value. Also, owing to multi-class classification, the model performance is expressed in form of confusion metrics for each class `AUX`, `VERB` along with the metrics like Precision, Recall, Accuracy, F1 Score.

The metrics corresponding to the best performing model on the original treebank is listed in Table 6.3. When the models were trained on each of the folds, keeping the architecture of the best model, there was no loss in performance of the trained models (metric considered- micro averaged F1 score).

As mentioned in previous section, we focused on the instances of the tagged data with confidence scores in particular bounds. Table 6.4 lists the number of instances that were focused on in each category (as defined in Table 6.2). The table also lists the number of instances that were identified as mislabelled, following the annotation procedure.

Label	Precision	Recall	Accuracy	F1 Score
`AUX`	98.89	99.50	98.40	99.19
`VERB`	99.32	98.87	98.20	99.09

Averaging	Accuracy	F1 Score
Micro	98.29	99.14
Macro	98.30	99.14

Table 6.3: Metrics of Best Model trained over original `hi` data

Category	Focused	Mislabelled	Percentage
aux_TP	83	3	3.61
O_TP	25	5	20.00
verb_TP	45	10	22.22
aux-O	10	9	90.00
aux-verb	42	23	54.76
verb-O	20	11	55.00
Overall	225	61	27.11

Table 6.4: Results of Manual Annotation

6.5 Discussion of the Results

Metric	Count
Sentences	16 647
Words	351 704
Tagged `AUX`	26 030
Tagged `VERB`	33 753

Table 6.5: Statistics for `hi` data

Table 6.5 lists the counts of sentences and the number of `AUX` and `VERB` tags in the entire `hi`-hdtb treebank. Of the total number of tags listed in either category, we are able to focus on just 225 instances where we might be able to identify the problems. Even out of those 225, the success ratio is less than 30%. While certain patterns are more reliable than others (the case where predicted labels don't match the annotated labels), the numbers are not significant enough for the process to be automated.

7. Future Work Recommendations

This chapter discusses in brief the other problems that have been recognised within the scope of UD. None of these works mentioned in this chapter were undertaken in this study. For future researchers interested in tackling more problems with respect to UD, this chapter could be a good point of reference.

7.1 Ellipsis

The problem with annotation of Elliptical Structures is big enough to warrant a discussion of its own in UD Annotation Guidelines[1],[2].

Droganova and Zeman [2017] analyzed the elliptical constructions in UDv2.0 by principally using `orphan` relations[3] as a way to identify the cases of non-promoted dependents with promoted dependents. While this helps in identifying only a certain number of cases, it fails to identify the cases where the dependents are promoted.

In Enhanced Dependencies, `orphan` is replaced by placing a null node to indicate the elided token. However, as we discuss later, Enhanced Dependencies are not available for all languages or even all treebanks in the same language. Thus, the identification and correction of erroneous elliptical constructions remains a problem that needs to be solved within the scope of basic dependency graphs in UD.

7.2 Function Words and Associated Dependency Relations

Conjunctions are identified by two POS tags, viz. `SCONJ`, `CCONJ`. The associated dependency relations for the two POS tags are `mark`, and `cc` respectively. While these are the usually associated dependency relations, the boundary between the two is fuzzy. In the sense, it is possible for a token to be marked by `SCONJ`, and have a `cc` dependency relation (similarly for `CCONJ` and `mark`). Added to this are the cases where the tokens marked by another POS tag can act as conjunctions. Consider the following example from `en`-ParTUT (UDv2.3), where `PART` acts as a conjunction, and thus the `mark` deprel associated to it.

Example 12. Ukraine's constitutional structure is for Ukraine's citizens alone to decide.

Furthermore, both the POS tags in question (`SCONJ`, `CCONJ`) can have other dependency relations attached to them as well. As such, it is difficult (and nonsensical) to limit the deprels for a particular POS tag to occur with a particular deprel (especially in this case). However, there can be still some processes we can observe (and correct). For example, if a

[1]`https://universaldependencies.org/u/overview/enhanced-syntax.html#ellipsis`
[2]`https://universaldependencies.org/u/overview/specific-syntax.html#ellipsis`
[3]`https://universaldependencies.org/u/dep/orphan.html`

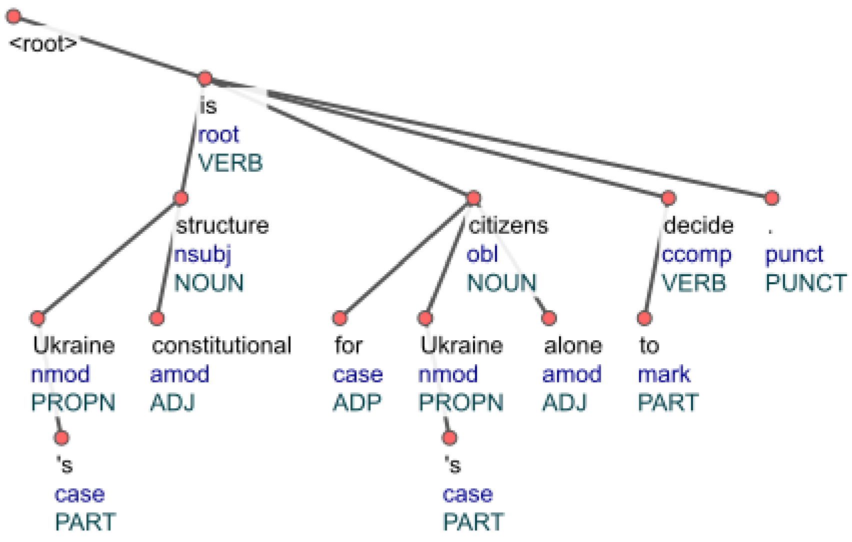


Figure 7.1: Tree for Example 12 showing association of `PART` with `mark` deprel

particular token occurs more with the `mark` deprel, but is consistently labelled as `CCONJ`, the annotation should be taken a closer look at, and a possible disparity identified.

7.3 nmod4obl

In UDv1, `nmod` relation was used for nominals modifying either predicates or other nominals. Following a change in guidelines in v2, the deprel was restricted to modifying nominals. Furthermore, a new relation `obl` (oblique) was introduced for oblique dependents of predicates.

To put it simply, this conversion implied the following in an equation format, where $\texttt{x}_{vi}$ refers to the dependency relation `x` as used in version i of UD treebanks:

$$\texttt{nmod}_{v1} = \texttt{nmod}_{v2} \cup \texttt{obl}$$

Let us again refer to the trees in Figure 2.1 and Figure 2.2, with respect to the token *specification* this time. In the example sentence, we see that the token is:

1. Linked to the wrong head in Figure 2.1. Instead of being linked to *conforms*, the token is wrongly attached to *closely*.

2. Is wrongly labelled as `nmod` in Figure 2.1, when it should have been labelled as `obl` as in Figure 2.2.

This kind of error is referred to as combined head identification and labelling error in Alzetta et al. [2017]. In the same work, the authors note that this error contributes to around 7 % of total discovered errors in the newspaper section of the Italian UD Treebank

(IUDT). In the work, the authors attribute this error pattern to annotation inconsistency internal to the treebank. Given that this error is largely stemming from the change in guidelines, changing guidelines can be argued as a cause of this error, rather than what is proposed by the authors in their work. Added to the difficulty of correct identification of head, it is not always possible to isolate the error in dependency label. Although a significantly important error, this is not covered in the scope of the current research. Nonetheless, this is an important error that should be taken care of in future.

7.4 Punctuation

The UD Annotation guidelines on punctuation are simple and straightforward[4]. There are discrepancies when it comes to implementation of the guidelines. Some of them are listed as below:

1. It is difficult to identify the next conjunct in case of missing `CCONJ` and `SCONJ` tags. The information about the next conjunct should be deduced semantically in most cases.

2. Raising a punctuation is a problem that goes with the previous instance since it's not always clear at what level the punctuation must attach to.

3. For nested punctuation, different languages use different sets of nested punctuation pairs, specifically with respect to quotation marks. As such, the treatment of paired punctuation pairs needs to be handled in a language-specific manner.

The `fixpunct.py` block in Udapi-python [Popel et al., 2017] tries to take care of significant number of edge cases in different UD treebanks. However, a more concrete solution is needed for the problems aforementioned.

7.5 UD and Enhanced Dependencies

Enhanced Dependencies can be understood simply as an additional layer of annotation of dependencies in UD, which essentially marks all the dependencies. The Enhanced Dependencies usually aim to cover aspects which can be missed by the regular annotation scheme, due to limitations like each node having exactly one head. However, not all of the languages, or their treebanks have been annotated with the Enhanced Dependencies so far. While they have been deemed to be useful in multiple cases (like that of ellipsis, as mentioned before), their full potential might not have been realized so far.

In our experiment on `conj_head` (cf. Section 5), we did not work with the problem of conjunction sandwiches. It is very likely that such constructions which are difficult to be

[4]`https://universaldependencies.org/u/overview/specific-syntax.html#punctuation`

recognized by the regular dependencies can be searched for rather easily with the Enhanced Dependencies.

We leave it as another open problem for future research to identify cases which are more difficult to handle with regular dependencies, while trying to use Enhanced Dependencies. As an add-on to the task, it can also be tested if some algorithms mentioned in the research can be improved upon/discarded, when Enhanced Dependencies are used.

7.6 Unspecified Dependencies - `dep` deprel

According to the UD definition of `dep` deprel[5], the deprel is reserved for cases when a more precise relation cannot be found. This can be either owing to the sentence splitting in treebanks of some languages, or owing to the limitation in parsing software. Nonetheless, the relation should be avoided as much as possible.

Noticing that some treebanks follow sentence splits where the parts of sentences might be labelled as different sentences (as in the example of a list), the deprel in question is more liable to be used in such instances. However, looking at the data in UDv2.4, some languages have more than 1% of the tokens marked with such relation (Examples being `ko`, `ur`, `ja`-BCCWJ, `it`-PoSTWITA, `hi`-HDTB, `gl`-CTG, `cs`-PDT, among others). While these might be all true positives in other languages, a significantly higher count of `dep` is more troublesome and is less likely to be all true positives in such cases.

An experiment can be performed on such instances where the data without any `dep` deprel is used as a training set to parse the instances with the deprel in question and then the results verified. Nonetheless, the cases of tokens marked with deprel in question need to be reduced in some languages. As such, we leave it as a problem for future researchers to tackle.

[5]`https://universaldependencies.org/u/dep/dep.html`

Conclusion

In the research, we introduced a new evaluation method for checking the similarity of different treebanks within the same language and tried to correct the problematic instances identified by different error miners before. We proposed an evaluation metric in this document, which we are hopeful would be helpful to future researchers. We tried fixing some of the problems identified by previous researchers. While some of the attempts at the solutions have been successful, the others still need refinement and additional work to complete them, owing to their time requirements.

One major advantage of an iterative process with respect to UD treebanks is how individual error types can be focused on in each iteration. With the UD validator (cf. Level 5 checks in `validate.py`[6] file) identifying and notifying the development teams of the individual errors, the process no longer suffers from a cold start problem.

It is important to note here that the different problems identified in the document seldom occur in isolation. As such, many of the problems can be intertwined with each other, resulting in error propagation at an exponential scale. Having said that, very often finding the right error and correcting it also propagates the corrections. Consider the example of experiment on `conj_head`. Correction of this error instance in the specific case of `eu` also corrected the case of non-projective associations in the trees.

Of the multiple problems discussed in the scope of this document, there might still be some problems that would have escaped the eye. There is a high chance that with the incoming iterations, more and more of the experiments discussed in the document would be rendered obsolete, and will not be required.

As the cost of storage falls lower, the size of the treebanks would increase. Essentially, at one point it might be impossible for human annotators to be part of the error-identification and error-correction process for the entire treebank. The current work was primarily aimed at finding the methods that don't need human annotators in the pipeline, and can be relied upon to fix the errors across different languages in a reliable manner. The research has been in some aspect successful at that front.

There are still a considerable number of problems that have been identified, but which could not be corrected in the scope of this research, one prime example being that of `nmod4obl`. The author hopes that the future researchers will be able to tackle the problems in a greater capacity, and possibly improve upon the methods already discussed in this research.

[6]`https://github.com/UniversalDependencies/tools`

Bibliography

Bhasha Agrawal, Rahul Agarwal, Samar Husain, and Dipti M. Sharma. An Automatic Approach to Treebank Error Detection Using a Dependency Parser. In *International Conference on Intelligent Text Processing and Computational Linguistics*, volume 7816, pages 294–303, 03 2013. doi: 10.1007/978-3-642-37247-6_24.

Alan Akbik, Duncan Blythe, and Roland Vollgraf. Contextual string embeddings for sequence labeling. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1638–1649, 2018.

Héctor Martínez Alonso and Daniel Zeman. Universal Dependencies for the AnCora treebanks. *Procesamiento del Lenguaje Natural*, (57), 2016.

Héctor Martínez Alonso, Željko Agić, Barbara Plank, and Anders Søgaard. Parsing universal dependencies without training. *arXiv preprint arXiv:1701.03163*, 2017.

Chiara Alzetta, Felice Dell'Orletta, Simonetta Montemagni, and Giulia Venturi. Dangerous relations in dependency treebanks. In *Proceedings of the 16th International Workshop on Treebanks and Linguistic Theories*, pages 201–210, 2017.

Chiara Alzetta, Felice Dell'Orletta, Simonetta Montemagni, and Giulia Venturi. Universal Dependencies and Quantitative Typological Trends. A Case Study on Word Order. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC-2018)*, 2018.

Bharat Ram Ambati, Rahul Agarwal, Mridul Gupta, Samar Husain, and Dipti Misra Sharma. Error Detection for Treebank Validation. In *Proceedings of the 9th Workshop on Asian Language Resources*, pages 23–30, 2011.

Collin F Baker, Charles J Fillmore, and John B Lowe. The berkeley framenet project. In *Proceedings of the 17th international conference on Computational linguistics-Volume 1*, pages 86–90. Association for Computational Linguistics, 1998.

Alena Böhmová, Jan Hajič, Eva Hajičová, and Barbora Hladká. The Prague dependency treebank. In *Treebanks*, pages 103–127. Springer, 2003.

Adriane Boyd, Markus Dickinson, and W Detmar Meurers. On detecting errors in dependency treebanks. *Research on Language and Computation*, 6(2):113–137, 2008.

Sabine Buchholz and Erwin Marsi. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the tenth conference on computational natural language learning*, pages 149–164. Association for Computational Linguistics, 2006.

Wanxiang Che, Yijia Liu, Yuxuan Wang, Bo Zheng, and Ting Liu. Towards better UD parsing: Deep contextualized word embeddings, ensemble, and treebank concatenation. *arXiv preprint arXiv:1807.03121*, 2018.

Michael John Collins. A new statistical parser based on bigram lexical dependencies. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 184–191. Association for Computational Linguistics, 1996.

Daniel Zeman, Ondřej Dušek, David Mareček, Martin Popel, Loganathan Ramasamy, Jan Štěpánek, Zdeněk Žabokrtský, and Jan Hajič. HamleDT: Harmonized Multi-Language Dependency Treebank. *Language Resources and Evaluation*, 48(4):601–637, 2014. ISSN 1574-020X.

Daniël De Kok, Jianqiang Ma, and Gertjan Van Noord. A generalized method for iterative error mining in parsing results. In *Proceedings of the 2009 workshop on grammar engineering across frameworks (GEAF 2009)*, pages 71–79, 2009.

Miryam de Lhoneux and Joakim Nivre. Should Have, Would Have, Could Have. Investigating Verb Group Representations for Parsing with Universal Dependencies. In *Proceedings of the Workshop on Multilingual and Cross-lingual Methods in NLP*, pages 10–19, 2016.

Marie-Catherine de Marneffe, Timothy Dozat, Natalia Silveira, Katri Haverinen, Filip Ginter, Joakim Nivre, and Christopher D. Manning. Universal Stanford dependencies: A cross-linguistic typology. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*, pages 4585–4592, Reykjavik, Iceland, May 2014. European Languages Resources Association (ELRA). URL `http://www.lrec-conf.org/proceedings/lrec2014/pdf/1062_Paper.pdf`.

Marie-Catherine de Marneffe, Matias Grioni, Jenna Kanerva, and Filip Ginter. Assessing the Annotation Consistency of the Universal Dependencies Corpora. In *Proceedings of the Fourth International Conference on Dependency Linguistics (Depling 2017)*, pages 108–115, 2017.

Felice Dell'Orletta, Giulia Venturi, and Simonetta Montemagni. Linguistically-driven Selection of Correct Arcs for Dependency Parsing. *Computación y Sistemas*, 17(2):125–136, 2013.

Markus Dickinson and W. Detmar Meurers. Detecting Errors in Part-of-speech Annotation. In *Proceedings of the Tenth Conference on European Chapter of the Association for Computational Linguistics - Volume 1*, EACL '03, pages 107–114, Stroudsburg, PA, USA, 2003a. Association for Computational Linguistics. ISBN 1-333-56789-0. doi: 10.3115/1067807.1067823. URL `https://doi.org/10.3115/1067807.1067823`.

Markus Dickinson and W. Detmar Meurers. Detecting Inconsistencies in Treebanks. *IEEE Transactions on Learning Technologies - TLT*, 01 2003b.

Markus Dickinson and W. Detmar Meurers. Detecting Errors in Discontinuous Structural Annotation. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 322–329, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics. doi: 10.3115/1219840.1219880. URL `https://doi.org/10.3115/1219840.1219880`.

Kira Droganova and Daniel Zeman. Elliptic Constructions: Spotting Patterns in UD Treebanks. In *Proceedings of the NoDaLiDa 2017 Workshop on Universal Dependencies (UDW 2017)*, pages 48–57, 2017.

Kira Droganova, Olga Lyashevskaya, and Daniel Zeman. Data Conversion and Consistency of Monolingual Corpora: Russian UD Treebanks. In *Proceedings of the 17th International Workshop on Treebanks and Linguistic Theories (TLT 2018), December 13–14, 2018, Oslo University, Norway*, number 155, pages 52–65. Linköping University Electronic Press, 2018.

Joseph H Greenberg. Some universals of grammar with particular reference to the order of meaningful elements. *Universals of language*, 2:73–113, 1963.

Jiří Havelka. Beyond Projectivity: Multilingual Evaluation of Constraints and Measures on Non-Projective Structures. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 608–615, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/P07-1077`.

Katri Haverinen, Jenna Nyblom, Timo Viljanen, Veronika Laippala, Samuel Kohonen, Anna Missilä, Stina Ojala, Tapio Salakoski, and Filip Ginter. Building the essential resources for Finnish: the Turku Dependency Treebank. *Language Resources and Evaluation*, 48(3):493–531, Sep 2014. ISSN 1574-0218. doi: 10.1007/s10579-013-9244-1. URL `https://doi.org/10.1007/s10579-013-9244-1`.

Tuomo Kakkonen. Dependency treebanks: methods, annotation schemes and tools. In *Proceedings of the 15th Nordic Conference of Computational Linguistics (NODALIDA 2005)*, pages 94–104, Joensuu, Finland, May 2006. University of Joensuu, Finland.

Francesco Mambrini and Marco Passarotti. Non-projectivity in the Ancient Greek dependency treebank. In *Proceedings of the second international conference on dependency linguistics (Depling 2013)*, pages 177–186, 2013.

Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. The Penn Treebank: Annotating Predicate Argument Structure. In *Proceedings of the Workshop on Human Language Technology*, HLT '94, pages 114–119, Stroudsburg, PA, USA, 1994. Association for Computational Linguistics. ISBN 1-55860-357-3. doi: 10.3115/1075812.1075835.

Marie-Catherine de Marneffe, Miriam Connor, Natalia Silveira, Samuel R. Bowman, Timothy Dozat, and Christopher D. Manning. More Constructions, More Genres: Extending Stanford Dependencies. In *Proceedings of the Second International Conference on Dependency Linguistics (DepLing 2013)*, pages 187–196, Prague, Czech Republic, August 2013. Charles University in Prague, Matfyzpress, Prague, Czech Republic. URL `https://www.aclweb.org/anthology/W13-3721`.

Ryan McDonald, Joakim Nivre, Yvonne Quirmbach Brundage, Yoav Goldberg, Dipanjan Das, Kuzman Ganchev, Keith Hall, Slav Petrov, Hao Zhang, T Oscar, et al. Universal dependency annotation for multilingual parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 92–97, 2013.

Joakim Nivre and Chiao-Ting Fang. Universal Dependency Evaluation. In *Proceedings of the NoDaLiDa 2017 Workshop on Universal Dependencies, 22 May, Gothenburg Sweden*, number 135, pages 86–95. Linköping University Electronic Press, 2017.

Joakim Nivre, Cristina Bosco, Jinho Choi, Marie-Catherine de Marneffe, Timothy Dozat, Richárd Farkas, Jennifer Foster, Filip Ginter, Yoav Goldberg, Jan Hajič, Jenna Kanerva, Veronika Laippala, Alessandro Lenci, Teresa Lynn, Christopher Manning, Ryan McDonald, Anna Missilä, Simonetta Montemagni, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Maria Simi, Aaron Smith, Reut Tsarfaty, Veronika Vincze, and Daniel Zeman. Universal Dependencies 1.0, 2015. URL `http://hdl.handle.net/11234/1-1464`. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Joakim Nivre, Mitchell Abrams, Željko Agić, Lars Ahrenberg, Gabrielė Aleksandravičiūtė, Lene Antonsen, Katya Aplonova, Maria Jesus Aranzabe, Gashaw Arutie, Masayuki Asahara, Luma Ateyah, Mohammed Attia, Aitziber Atutxa, Liesbeth Augustinus, Elena Badmaeva, Miguel Ballesteros, Esha Banerjee, Sebastian Bank, Verginica Barbu Mititelu, Victoria Basmov, John Bauer, Sandra Bellato, Kepa Bengoetxea, Yevgeni Berzak, Irshad Ahmad Bhat, Riyaz Ahmad Bhat, Erica Biagetti, Eckhard Bick, Agnė Bielinskienė, Rogier Blokland, Victoria Bobicev, Loïc Boizou, Emanuel Borges Völker, Carl Börstell, Cristina Bosco, Gosse Bouma, Sam Bowman, Adriane Boyd, Kristina Brokaitė, Aljoscha Burchardt, Marie Candito, Bernard Caron, Gauthier Caron, Gülşen Cebiroğlu Eryiğit, Flavio Massimiliano Cecchini, Giuseppe G. A. Celano, Slavomír Čéplö, Savas Cetin, Fabricio Chalub, Jinho Choi, Yongseok Cho, Jayeol Chun, Silvie Cinková, Aurélie Collomb, Çağrı Çöltekin, Miriam Connor, Marine Courtin, Elizabeth Davidson, Marie-Catherine de Marneffe, Valeria de Paiva, Arantza Diaz de Ilarraza, Carly Dickerson, Bamba Dione, Peter Dirix, Kaja Dobrovoljc, Timothy Dozat, Kira Droganova, Puneet Dwivedi, Hanne Eckhoff, Marhaba Eli, Ali Elkahky, Binyam Ephrem, Tomaž Erjavec, Aline Etienne, Richárd Farkas, Hector Fernandez Alcalde, Jennifer Foster, Cláudia Freitas, Kazunori Fujita, Katarína Gajdošová, Daniel Galbraith, Marcos Garcia, Moa Gärdenfors, Sebastian Garza, Kim Gerdes, Filip Ginter, Iakes Goenaga, Koldo Gojenola,

Memduh Gökırmak, Yoav Goldberg, Xavier Gómez Guinovart, Berta González Saavedra, Matias Grioni, Normunds Grūzītis, Bruno Guillaume, Céline Guillot Barbance, Nizar Habash, Jan Hajič, Jan Hajič jr., Linh Hà Mỹ, Na-Rae Han, Kim Harris, Dag Haug, Johannes Heinecke, Felix Hennig, Barbora Hladká, Jaroslava Hlaváčová, Florinel Hociung, Petter Hohle, Jena Hwang, Takumi Ikeda, Radu Ion, Elena Irimia, Ọlájídé Ishola, Tomáš Jelínek, Anders Johannsen, Fredrik Jørgensen, Hüner Kaşıkara, Andre Kaasen, Sylvain Kahane, Hiroshi Kanayama, Jenna Kanerva, Boris Katz, Tolga Kayadelen, Jessica Kenney, Václava Kettnerová, Jesse Kirchner, Arne Köhn, Kamil Kopacewicz, Natalia Kotsyba, Jolanta Kovalevskaitė, Simon Krek, Sookyoung Kwak, Veronika Laippala, Lorenzo Lambertino, Lucia Lam, Tatiana Lando, Septina Dian Larasati, Alexei Lavrentiev, John Lee, Phương Lê Hồng, Alessandro Lenci, Saran Lertpradit, Herman Leung, Cheuk Ying Li, Josie Li, Keying Li, KyungTae Lim, Yuan Li, Nikola Ljubešić, Olga Loginova, Olga Lyashevskaya, Teresa Lynn, Vivien Macketanz, Aibek Makazhanov, Michael Mandl, Christopher Manning, Ruli Manurung, Cătălina Mărănduc, David Mareček, Katrin Marheinecke, Héctor Martínez Alonso, André Martins, Jan Mašek, Yuji Matsumoto, Ryan McDonald, Gustavo Mendonça, Niko Miekka, Margarita Misirpashayeva, Anna Missilä, Cătălin Mititelu, Yusuke Miyao, Simonetta Montemagni, Amir More, Laura Moreno Romero, Keiko Sophie Mori, Tomohiko Morioka, Shinsuke Mori, Shigeki Moro, Bjartur Mortensen, Bohdan Moskalevskyi, Kadri Muischnek, Yugo Murawaki, Kaili Müürisep, Pinkey Nainwani, Juan Ignacio Navarro Horñiacek, Anna Nedoluzhko, Gunta Nešpore Bērzkalne, Lương Nguyễn Thị, Huyền Nguyễn Thị Minh, Yoshihiro Nikaido, Vitaly Nikolaev, Rattima Nitisaroj, Hanna Nurmi, Stina Ojala, Adédayọ̀ Olúòkun, Mai Omura, Petya Osenova, Robert Östling, Lilja Øvrelid, Niko Partanen, Elena Pascual, Marco Passarotti, Agnieszka Patejuk, Guilherme Paulino Passos, Angelika Peljak Łapińska, Siyao Peng, Cenel-Augusto Perez, Guy Perrier, Daria Petrova, Slav Petrov, Jussi Piitulainen, Tommi A Pirinen, Emily Pitler, Barbara Plank, Thierry Poibeau, Martin Popel, Lauma Pretkalniņa, Sophie Prévost, Prokopis Prokopidis, Adam Przepiórkowski, Tiina Puolakainen, Sampo Pyysalo, Andriela Rääbis, Alexandre Rademaker, Loganathan Ramasamy, Taraka Rama, Carlos Ramisch, Vinit Ravishankar, Livy Real, Siva Reddy, Georg Rehm, Michael Rießler, Erika Rimkutė, Larissa Rinaldi, Laura Rituma, Luisa Rocha, Mykhailo Romanenko, Rudolf Rosa, Davide Rovati, Valentin Roșca, Olga Rudina, Jack Rueter, Shoval Sadde, Benoît Sagot, Shadi Saleh, Alessio Salomoni, Tanja Samardžić, Stephanie Samson, Manuela Sanguinetti, Abigail Walsh Sarah McGuinness, Dage Särg, Baiba Saulīte, Yanin Sawanakunanon, Nathan Schneider, Sebastian Schuster, Djamé Seddah, Wolfgang Seeker, Mojgan Seraji, Mo Shen, Atsuko Shimada, Hiroyuki Shirasu, Muh Shohibussirri, Dmitry Sichinava, Natalia Silveira, Maria Simi, Radu Simionescu, Katalin Simkó, Mária Šimková, Kiril Simov, Aaron Smith, Isabela Soares Bastos, Carolyn Spadine, Antonio Stella, Milan Straka, Jana Strnadová, Alane Suhr, Umut Sulubacak, Shingo Suzuki, Zsolt Szántó, Dima Taji, Yuta Takahashi, Fabio Tamburini, Takaaki Tanaka, Isabelle Tellier, Guillaume Thomas, Liisi Torga, Trond Trosterud, Anna Trukhina, Reut Tsarfaty, Francis Tyers, Sumire Uematsu, Zdeňka Urešová, Larraitz Uria, Hans Uszkoreit, Sowmya Vajjala, Daniel van Niekerk, Gertjan van Noord, Viktor Varga, Eric Villemonte de la Clergerie,

Veronika Vincze, Lars Wallin, Jing Xian Wang, Jonathan North Washington, Maximilan Wendt, Seyi Williams, Mats Wirén, Christian Wittern, Tsegay Woldemariam, Tak-sum Wong, Alina Wróblewska, Mary Yako, Naoki Yamazaki, Chunxiao Yan, Koichi Yasuoka, Marat M. Yavrumyan, Zhuoran Yu, Zdeněk Žabokrtský, Amir Zeldes, Daniel Zeman, Manying Zhang, and Hanzhi Zhu. Universal Dependencies 2.4, 2019. URL `http://hdl.handle.net/11234/1-2988`. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Slav Petrov, Dipanjan Das, and Ryan McDonald. A Universal Part-of-Speech Tagset. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC-2012)*, pages 2089–2096, Istanbul, Turkey, May 2012. European Languages Resources Association (ELRA). URL `http://www.lrec-conf.org/proceedings/lrec2012/pdf/274_Paper.pdf`.

Martin Popel, Zdeněk Žabokrtskỳ, and Martin Vojtek. Udapi: Universal API for universal dependencies. In *Proceedings of the NoDaLiDa 2017 Workshop on Universal Dependencies (UDW 2017)*, pages 96–101, 2017.

Rudolf Rosa and Zdenek Zabokrtsky. Klcpos3-a language similarity measure for delexicalized parser transfer. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 243–249, 2015.

Benoît Sagot and Éric de La Clergerie. Error mining in parsing results. In *Proceedings of the 21st international conference on computational linguistics and 44th annual meeting of the association for computational linguistics*, pages 329–336, 2006.

Timothy Shopen. *Language Typology and Syntactic Description*, volume 1, pages 40–59. Cambridge University Press, 2 edition, 2007. ISBN 0-511-36671-X. doi: 10.1017/CBO9780511619427.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer Sentinel Mixture Models. *CoRR*, abs/1609.07843, 2016.

Milan Straka and Jana Straková. UDPipe, 2016. URL `http://hdl.handle.net/11234/1-1702`. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Milan Straka, Jan Hajič, Jana Straková, and Jan Hajič jr. Parsing Universal Dependency Treebanks using Neural Networks and Search-Based Oracle. In *Proceedings of Fourteenth International Workshop on Treebanks and Linguistic Theories (TLT 14)*, December 2015.

Gertjan Van Noord. Error mining for wide-coverage grammar engineering. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 446. Association for Computational Linguistics, 2004.

Erik Velldal, Lilja Øvrelid, and Petter Hohle. Joint UD Parsing of Norwegian Bokmål and Nynorsk. In *Proceedings of the 21st Nordic Conference on Computational Linguistics*, pages 1–10, Gothenburg, Sweden, May 2017. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/W17-0201`.

Daniel Zeman. Reusable Tagset Conversion Using Tagset Drivers. In *Proceedings of the Language Resources and Evaluation Conference*, LREC, 2008. ISBN 2-9517408-4-0.

Daniel Zeman, David Mareček, Jan Mašek, Martin Popel, Loganathan Ramasamy, Rudolf Rosa, Jan Štěpánek, and Zdeněk Žabokrtský. HamleDT 2.0, 2014. URL `http://hdl.handle.net/11858/00-097C-0000-0023-9551-4`. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Daniel Zeman, Martin Popel, Milan Straka, Jan Hajič, Joakim Nivre, Filip Ginter, Juhani Luotolahti, Sampo Pyysalo, Slav Petrov, Martin Potthast, Francis Tyers, Elena Badmaeva, Memduh Gokirmak, Anna Nedoluzhko, Silvie Cinková, Jan Hajič jr., Jaroslava Hlaváčová, Václava Kettnerová, Zdeňka Urešová, Jenna Kanerva, Stina Ojala, Anna Missilä, Christopher D. Manning, Sebastian Schuster, Siva Reddy, Dima Taji, Nizar Habash, Herman Leung, Marie-Catherine de Marneffe, Manuela Sanguinetti, Maria Simi, Hiroshi Kanayama, Valeria de Paiva, Kira Droganova, Héctor Martínez Alonso, Çağrı Çöltekin, Umut Sulubacak, Hans Uszkoreit, Vivien Macketanz, Aljoscha Burchardt, Kim Harris, Katrin Marheinecke, Georg Rehm, Tolga Kayadelen, Mohammed Attia, Ali Elkahky, Zhuoran Yu, Emily Pitler, Saran Lertpradit, Michael Mandl, Jesse Kirchner, Hector Fernandez Alcalde, Jana Strnadová, Esha Banerjee, Ruli Manurung, Antonio Stella, Atsuko Shimada, Sookyoung Kwak, Gustavo Mendonça, Tatiana Lando, Rattima Nitisaroj, and Josie Li. CoNLL 2017 shared task: Multilingual parsing from raw text to universal dependencies. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–19, Vancouver, Canada, August 2017. Association for Computational Linguistics. doi: 10.18653/v1/K17-3001. URL `https://www.aclweb.org/anthology/K17-3001`.

Daniel Zeman, Jan Hajič, Martin Popel, Martin Potthast, Milan Straka, Filip Ginter, Joakim Nivre, and Slav Petrov. CoNLL 2018 shared task: multilingual parsing from raw text to universal dependencies. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–21, 2018.

List of Figures

List of Tables

List of Abbreviations

- **CLAS**- Content-based Labelled Attachment Score
- **CoNLL**- Conference on Computational Natural Language Learning
- **GS**- Gold Standard
- **LAS**- Labelled Attachment Score
- **LISCA**- LInguistically-driven Selection of Correct Arts
- **LTR**- Left To Right written order language
- **MWE**- Multi-Word Entity
- **NER**- Named Entity Recognition
- **POS**- Part Of Speech
- **RTL**- Right To Left written order language
- **SOTA**- State Of The Art
- **TAME**- Time, Aspect, Modality, Evidentitality
- **UAS**- Unlabelled Attachment Score
- **UD**- Universal Dependencies

A. Appendix

A.1 Terminology Pertaining to UD

This appendix is meant primarily for the offline/hard copy readers of the document. A better (and official) explanation of the terms can be accessed online[1,2].

UD uses an extension of CONLL-X format [Buchholz and Marsi, 2006], referred to as CONLL-U format. The CONLL-U format is used for the annotation procedure, with three types of lines. Each line is delimited by LF character as line break, written in UTF-8 encoding. The details of the line types are as follows:

1. **Blank Line**: A line without any content, used as a separator for annotations of different sentences in the treebank.

2. **Comment Line**: A line starting with hash (#) symbol, typically contains details about the annotated sentence. The details that are common across all treebanks are 'sent_id' (a unique ID associated with each sentence in the treebank), and 'text' (the text of the annotated sentence). The comment can also include any other details like paragraph id, document id, etc.

3. **Word Line**: Each Word Line contains the annotation of a single word, in a 10-column TSV (tab-separated values) format. The columns, in order, and their explanation are as follows:

   (a) **ID**: Word Index in the sentence, starts at 1. Can be a ranged value for fused tokens and multiword tokens; decimal value for empty nodes. The ID of a token can be only greater than 0.

   (b) **FORM**: Word Form, as it appears in the sentence.

   (c) **LEMMA**: Lemma or Stem of Word Form.

   (d) **UPOS**: The Universal POS tag of the word, as per UD Tagset.

   (e) **XPOS**: The language-specific POS tag of the word. Generally comes from the original tagset that was converted into UD.

   (f) **FEATS**: List of morphological features from UD feature inventory, or a language specific version thereof.

   (g) **HEAD**: Head of the current word in dependency relation. Contains 'ID' of the parent word, or 0 if the parent word is 'Root' (explained later).

   (h) **DEPREL**: Universal Dependency Relation, extendable with language specific extension thereof (explained later).

[1] `https://universaldependencies.org/format.html`

[2] `https://universaldependencies.org/u/overview/morphology.html`

(i) **DEPS**: Enhanced Dependency Relation in form of head:deprel pairs.

(j) **MISC**: Any other annotation.

Of the different columns (referred to as Fields), there are associated restrictions, briefed as follows:

- Fields must not be empty. An unspecified value is represented by an underscore (_) symbol.
- Fields other than FORM and LEMMA cannot contain space characters.
- UPOS, HEAD, DEPREL are not allowed to be left unspecified.

There are some additional points with respect to UD Annotation that must be clarified.

1. For the dependency tree, UD annotates the global root of a sentence as a token with ID=0, referred to as ROOT. The root in the sentence is always a singular unit, and is a direct child of this ROOT node.

2. A dependency relation is expressed in a format that combines the universal deprel and language specific part of deprel with a colon mark (:). The language specific extension is optional, but is present in a lot of cases nonetheless. We refer to the universal relation as udeprel, and the language specific extension as xdeprel. Following example illustrates the same.

   Example 13. In DEPREL Field value as `acl:relcl`, `acl` is the universal dependency relation (referred to as udeprel, as per Udapi nomenclature) while `relcl` is the language specific extension of `acl` udeprel (referred to as xdeprel, as per Udapi nomenclature).

   As mentioned earlier, we refer to udeprel when we talk about deprels in this document, unless otherwise stated.

A.2 List of Language Codes

This appendix contains the list of languages, along with their identification codes, as used in the different treebanks of UDv2.4. A full list of ISO 693-2 language codes can also be accessed online[3].

Table A.1 indicates languages where the ISO code (ISO 693-1 or ISO 693-2) is used as an identifier. If a language is not identified by its ISO code, it is listed in Table A.2.

Note:

- * against a language name indicates language not present in UDv2.3.
- + against a language name indicates added data from UDv2.3 to UDv2.4. The addition can be in the form of entire treebank, or train/dev data for a given treebank.

Code	Language Name
af	Afrikaans
akk	Akkadian
am	Amharic
ar	Arabic
be	Belarusian
bg	Bulgarian
bm	Bambara
br	Breton
ca	Catalan
cop	Coptic
cs	Czech
cu	Old Church Slavonic
cy	Welsh*
da	Danish
de	German+
el	Greek
en	English
es	Spanish
et	Estonian
eu	Basque
fa	Persian
fi	Finnish
fo	Faroese
fr	French+
ga	Irish

Continued on next page

[3]https://www.loc.gov/standards/iso639-2/php/code_list.php

Code	Language Name
gl	Galician
got	Gothic
grc	Ancient Greek
he	Hebrew
hi	Hindi
hr	Croatian
hu	Hungarian
hsb	Upper Sorbian
hy	Armenian$^+$
id	Indonesian
it	Italian$^+$
ja	Japanese
kk	Kazakh
ko	Korean
krl	Karelian*
la	Latin
lt	Lithuanian$^+$
lv	Latvian
mr	Marathi
mt	Maltese
myv	Erzya
no	Norwegian$^+$
nl	Dutch
pl	Polish
pt	Portuguese
ro	Romanian
ru	Russian$^+$
sa	Sanskrit
sk	Slovak
sl	Slovenian
sme	North Sami
sr	Serbian
sv	Swedish
ta	Tamil
te	Telugu
th	Thai
tl	Tagalog
tr	Turkish
ug	Uyghur
uk	Ukrainian

Continued on next page

Code	**Language Name**
ur	Urdu
vi	Vietnamese
wo	Wolof*
yo	Yoruba
yue	Cantonese
zh	Chinese

Table A.1: Languages in UDv2.4, identified with their ISO Codes

Code	**Language Name**
aii	Assyrian*
bxr	Buryat
fro	Old French
gun	Mbya Guarani*
kmr	Kurmanji
kpv	Komi Zyrian
lzh	Classical Chinese*
orv	Old Russian*
pcm	Naija
qhe	Hindi-English
swl	Swedish Sign Language
wbp	Warlpiri

Table A.2: Languages in UDv2.4, without defined ISO Codes

A.3 Multiple Treebanks in Languages (UDv2.4)

Table A.3 contains the different languages containing the different treebanks. The second column of the table corresponds to the count of the different treebanks, and the last column contains the name of the treebanks. Notice that PUD treebanks are not included in the counts, or the additional treebanks. A list of PUD treebanks can be accessed in Appendix A.4.

Language	Count	Treebank Names
ar	2	PADT, NYUAD
cs	4	CAC, CLTT, FicTree, PDT
de	3	GSD, HDT, LIT
en	5	ESL, EWT, GUM, LinES, ParTUT
es	2	AnCora, GSD
et	2	EDT, EWT
fi	2	FTB, TDT
fr	5	FQB, FTB, GSD, ParTUT, Sequoia
gl	2	CTG, TreeGal
grc	2	Perseus, PROIEL
gun	2	Dooley, Thomas
it	4	ISDT, ParTUT, PoSTWITA, VIT
ja	3	BCCWJ, GSD, Modern
ko	2	GSD, Kaist
kpv	2	IKDP, Lattice
la	3	ITTB, Perseus, PROIEL
lt	2	ALKSNIS, HSE
nl	2	Alpino, LassySmall
no	3	Bokmaal, Nynorsk, NynorskLIA
pl	2	LFG, PDB
pt	2	Bosque, GSD
ro	2	Nonstandard, RRT
ru	3	GSD, SynTagRus, Taiga
sl	2	SSJ, SST
sv	2	LinES, Talbanken
tr	2	GB, IMST
zh	3	CFL, GSD, HK

Table A.3: Multiple Treebanks in Different Languages, UDv2.4

A.4 PUD Treebanks

Table A.4 contains a list of languages which contain a PUD treebank. Notice that PUD treebanks contain only the test set, and are devoid of train and dev data.

Code	**Language Name**
ar	Arabic
cs	Czech
de	German
en	English
es	Spanish
fi	Finnish
fr	French
hi	Hindi
id	Indonesian
it	Italian
ja	Japanese
ko	Korean
pl	Polish
pt	Portuguese
ru	Russian
sv	Swedish
th	Thai
tr	Turkish
zh	Chinese

Table A.4: Languages with PUD Treebanks, UDv2.4

A.5 Treebanks in UDv2.4, sans train/dev Data

The Table A.5 lists treebanks (in alphabetical order) with either of train or dev data being unavailable. Note that all PUD treebanks, generated for CONLL-2018 Shared Task, do not have either of train or dev data, and are therefore not included in this table.

Treebank Name	**Unavailable Data**
Akkadian-PISANDUB	train, dev
Amharic-ATT	train, dev
Assyrian-AS	train, dev
Bambara-CRB	train, dev
Breton-KEB	train, dev
Buryat-BDT	dev
Cantonese-HK	train, dev
Chinese-CFL	train, dev
Chinese-HK	train, dev
Erzya-JR	train, dev
Estonian-EWT	dev
Faroese-OFT	train, dev
French-FQB	train, dev
Galician-TreeGal	dev
German-LIT	train, dev
Irish-IDT	dev
Japanese-Modern	train, dev
Karelian-KKPP	train, dev
Kazakh-KTB	dev
Komi_Zyrian-IKDP	train, dev
Komi_Zyrian-Lattice	train, dev
Kurmanji-MG	dev
Mbya_Guarani-Dooley	train, dev
Mbya_Guarani-Thomas	train, dev
Naija-NSC	train, dev
North_Sami-Giella	dev
Old_Russian-RNC	train, dev
Sanskrit-UFAL	train, dev
Slovenian-SST	dev
Tagalog-TRG	train, dev
Turkish-GB	train, dev
Upper_Sorbian-UFAL	dev
Warlpiri-UFAL	train, dev
Welsh-CCG	train, dev
Yoruba-YTB	train, dev

Table A.5: Unavailable Data in UDv2.4 Treebanks