FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Akshay Aggarwal

Consistency of Linguistic Annotation

Institute of Formal and Applied Linguistics

Supervisor of the master thesis:	RNDr. Daniel Zeman, PhD Koldo Gojenola, PhD
Study programme:	Computer Science
Study branch:	Computational Linguistics

Prague 2020

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.

A very special note of thanks is extended to Chiara Alzetta of ItaliaNLP Lab for her constant help in an experiment. From the discussion about the algorithm, to actually running it, the experiment on lisca tools would not have been possible without her valuable inputs.

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.

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 and a half years, my family and friends. Special thanks are due to my friends in Prague for helping me keep my sanity in place.

Title: Consistency of Linguistic Annotation

Author: Akshay Aggarwal

Institute: Institute of Formal and Applied Linguistics

Supervisors: RNDr. Daniel Zeman, PhD, Institute of Formal and Applied Linguistics; Koldo Gojenola, PhD, Computer Languages and Systems, University of Basque Country (UPV-EHU), Spain

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 the Universal Dependencies Project in a language neutral manner, the solutions being reliable enough to not need a human annotator in the pipeline.

Keywords: Annotation Universal Dependencies Error Mining Morphology Language Independent Syntax

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 corpus, 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 function-word head to content-word head structure, the entire dependency structures need to be modified, thus introducing problems. Such

[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`

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 to millions 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 tradition differs from the guidelines.

3. In case of low-resource languages, it might be difficult to find a knowledgeable 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

[5]

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.

Although there already existed annotation schemes that were used as de facto standards, with the example of The Stanford dependencies [Marneffe et al., 2013],

[5] A rather more detailed history of UD Project can be accessed on UD homepage at `https://universaldependencies.org/introduction.html#history`. This section deals with a shorter version thereof.

Google universal tagset [Petrov et al., 2012], HamleDT [Zeman et al., 2014a], 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., 2014b], 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 $x_1 \neq x_0$.

    - 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.

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

    - 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 February 2019, when UDv2.3 was the latest release. As such, most of the experiments contained herein were first developed for UDv2.3. However, with the release of UDv2.4, they 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. Similarly, the newer experiments were carried out entirely on UDv2.5 Zeman et al. [2019] data.

Nonetheless, there are some experiments that work well for UDv2.4 and UDv2.5 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 from Chapter 2. In the subsequent chapters (Chapters 4 - 7), 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 8 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 9, 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	Output
Regular	Lorem ipsum
Italic	*Lorem ipsum*
Bold	**Lorem ipsum**
Tag	`Lorem ipsum`
Tag2	`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 UDv2.5, 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, almost 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 Žabokrtský [2015] show that KL-Divergence score of POS trigrams can be effectively used for source selection for POS Tagging in a delexicalised cross-language model transfer scenario. 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 asymmetric. 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:

   $$\theta_{POS} = KL_{cpos^3}(A, B) + KL_{cpos^3}(B, A) \leq \theta_2 \tag{2.4}$$

   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 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 2.1.

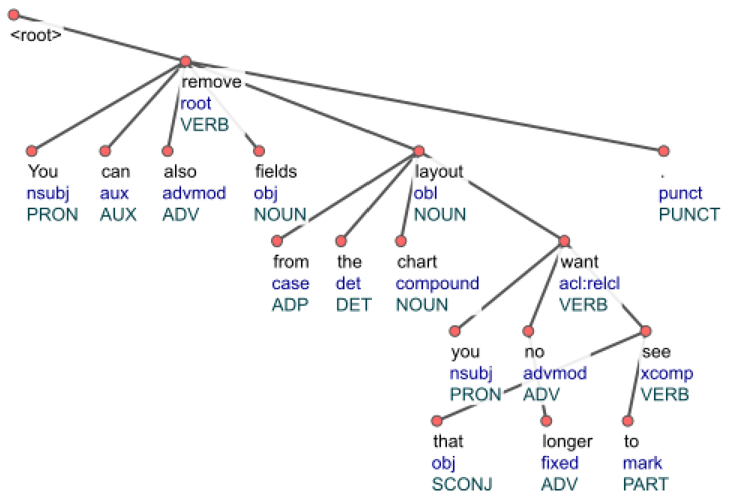


Figure 2.1: Sample Non-projective Tree

In the graph, notice the edge going from *see* to *that*[1]. 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 $i \to j$. We use $i < j$ to denote the node i preceding 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:

[1]It is worth noting that the token is mis-tagged as `SCONJ`, when it should be tagged as `PRON`

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.

We study the counts of non-projective structures in different UD treebanks, and then also attempt to remove the cases of false non-projectivities in Chapter 5 where we disregard the non-projectivities introduced/caused by punctuation marks, and focus on the remaining cases.

2.3 Problems Caused by Change of Guidelines in UDv2

A summarized version of changed guidelines from UDv1.x to UDv2 can be accessed online[2]. 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]`https://universaldependencies.org/v2/summary.html`

2.3.1 Head Identification Error In Conjunctions

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 conjunctions are attached to the immediately succeeding conjunct in UDv2, as opposed to UDv1 where they were attached to the first conjunct.

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 (referred to as `conj_head` henceforth) 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 6 where we discuss about it in detail.

2.3.2 Auxiliaries in UD

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.

2.4 Open Problems

2.4.1 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[3] 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[4]. 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 8 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[5].

[3]`https://universaldependencies.org/u/pos/all.html#aux-auxiliary`

[4]`https://github.com/UniversalDependencies/docs/issues/275`

[5]`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 scouting for error patterns within the scope of the treebank, also known as automatic error mining methods.

3.1.1 Variation Nuclei and Variations Thereof

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 constituency-based 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 sentence. This again is not very optimal for large treebanks.

3.1.2 LISCA

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, fudapthere 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 discussed in more detail in Chapter 7 where we also discuss the factors affecting the optimal choice of k-value.

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
sv	2	LinES, Talbanken

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

4.2 Tuning Parameter on LAS Scores

For given treebanks A and B, we defined θ_{LAS} in Definition 2 as

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

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

A few properties of the metric θ_{LAS} can be mentioned here.

1. In an ideal scenario, the metric θ_{LAS} should not be negative. Since we are training and evaluating on the same dataset, we expect $LAS_{x,x}$ to be considerably high. In case a treebank has a lower score when trained and tested on itself, either the parsing algorithm fails to capture the linguistic data (verifiable by training and evaluating on treebanks of same language, if possible), or the annotation procedure is very likely to be non-uniform across the given treebank. The latter can also be because of the different genres of data present therein.

2. The exact value of the metric is the greater value of the two associations. Simply speaking, if treebanks P, and Q are being evaluated, the maximal value from either of P or Q being evaluated on the other becomes the metric score.

3. In general, the lower the value of the metric, the better.

In this section, we shall try to optimize θ_1 value, given θ_{LAS} values for different treebanks as given in Table 4.2, reported in form of confusion matrices as in Alonso and Zeman [2016]. The horizontal rows mark the treebank the parser was trained on, while the vertical column marks the treebank on which the parser was tested. Following a look at the θ_{LAS} values, we try to address the different factors that might affect the metric value. Eventually, we try to find the optimal value range for θ_1 in order for the considered treebanks to be considered harmonious with each other, with respect to θ_1 parameter.

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

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

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

[a]Annotated Data not in correct CONLL-U format, UD 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 Parsed with GS Tokenisation with GS Tags

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. If we look at `ko` parser scores, we can see that the parsers perform rather poorly as compared to other treebanks when training and testing data remains the same. This is very likely because of the treebanks not following consistent annotation scheme within itself.

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 PDT data. Therefore, we need to still regulate for train data size disparities.

Treebank	Sentences	Tokens
CAC	23 478	471 594
CLTT	860	26 225
FicTree	10 160	133 137
PDT	68 495	1 171 190

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.

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 homogenous in nature. By homogenous, 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.

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

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.

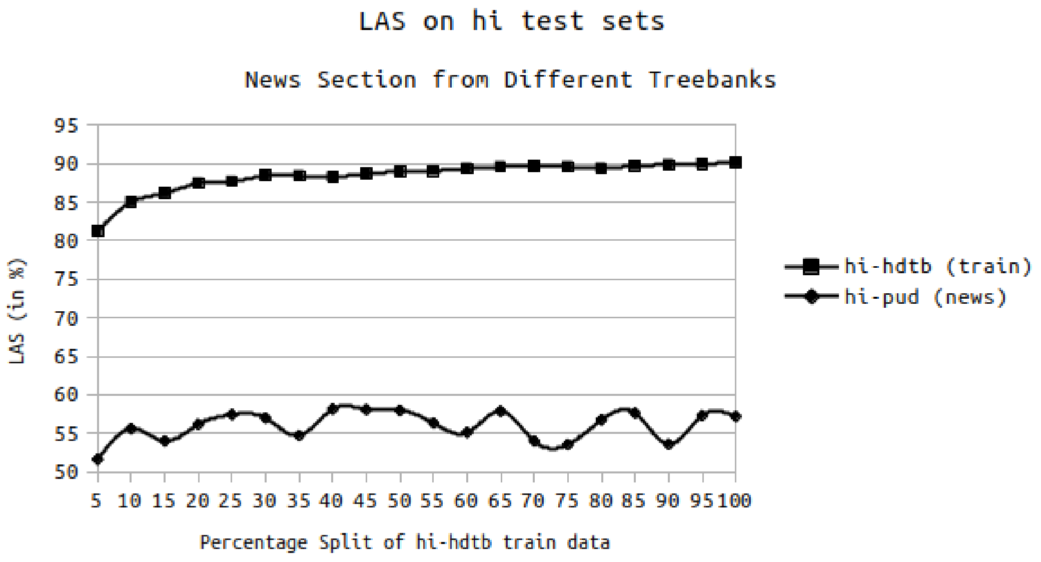


Figure 4.1: LAS for size disparity

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 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 when measured in terms of number of tokens in the train data, we can bound θ_{LAS} by an upper bound given by $\theta_{1,size}$ as below.

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

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

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 4. We are given two treebanks, containing the same genre of data, differing in the size. Let us assume the two given parameters as follows, with the size given in terms of tokens in train data:

$$\begin{gathered} size(A) = 1200000\text{tokens in train data} \\ size(B) = 10000\text{tokens in train data} \\ \alpha = 120 \\ LAS(A, A) = 92 \\ LAS(A, B) = 65 \\ LAS(B, B) = 91 LAS(B, A) = 75 \\ \theta_{LAS} = max(92 - 65, 91 - 75) = max(27, 16) = 27 \end{gathered}$$

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

$$\begin{gathered}
\zeta = \lfloor \alpha - 80 \rfloor = \lfloor 120 - 80 \rfloor = \lfloor 40 \rfloor = 40 \\
\gamma = \lfloor \alpha - \zeta - 60 \rfloor = \lfloor 120 - 40 - 60 \rfloor = \lfloor 20 \rfloor = 20 \\
\epsilon = \lfloor \alpha - \zeta - \gamma - 40 \rfloor = \lfloor 120 - 20 - 20 - 40 \rfloor = \lfloor 20 \rfloor = 20 \\
\delta = \lfloor \alpha - \zeta - \gamma - \epsilon - 20 \rfloor = \lfloor 120 - 40 - 20 - 20 - 20 \rfloor = \lfloor 20 \rfloor = 20 \\
\beta = \lfloor \alpha - \zeta - \gamma - \epsilon - \delta \rfloor = \lfloor 120 - 40 - 20 - 20 - 20 \rfloor = \lfloor 20 \rfloor = 20 \\
\theta_{1,size} = \frac{10}{32} + \frac{20}{16} + \frac{20}{8} + \frac{20}{4} + \frac{20}{2} = 0.3125 + 1.25 + 2.5 + 5 + 10 = 19.0625 \\
\theta_{LAS} \geq \theta_{1,size} = 19.0625
\end{gathered}$$

In this case, since the calculated value of θ_{LAS} is more than the permissible limit, the two treebanks are not harmonised with respect to each other.

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 5. Consider two treebanks from before again. Let us restate the given parameters as follows:

$$\begin{gathered}
\alpha = 120 \\
\theta_{LAS} = 27
\end{gathered}$$

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

$$\begin{gathered}
\frac{\zeta}{32} = 1.25 \\
\frac{\gamma}{16} = 1.25 = \frac{\zeta}{32}
\end{gathered}$$

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 ζ 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

It is very well known that there exists significant difference between genres. 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. Intuitively, some genres are more similar than others. For the same effect, LAS were computed between equal instances[3] of different genres.

Genre	**Sentence Count**
blog	136
fiction	7 252
news	6 744
nonfiction	1 273
social	526
spoken (conversational)	789
academic	51
legal	11
spoken (media)	158
spoken (prepared)	306

Table 4.8: Genre Distribution in UDv2.4 `pl`-LFG treebank

`pl`-LFG treebank in UDv2.4 contains data from 10 different genres[4]. The sentence counts of different genres are shown in Table 4.8. Of the different kind of data in *spoken* genre, we keep only the *conversational*, discarding the others from consideration. We also remove *academic* and *legal* data from our consideration owing to a considerably low number of sentences. The genres we work with are marked in bold in the table. For the experiment, we first split `pl`-LFG treebank into the constituent genres, discarding those that are not marked in bold in Table 4.13. For the data from considered genres, we proceed as follows:

1. For a given genre, we randomly sample 125 instances from the genre data. We refer to this resultant split as working data for the genre.
2. For the given genre, train a parser on the data.
3. For the given genre, report the LAS when compared with all other genres in form of confusion matrices.

We would like to point out that the number of instances are really low in this experiment. For a better experiment, it would be an ideal case to have

[3]in terms of number of sentences

[4]For understanding of what genre category involves exactly what kind of data, refer to the github page of the treebank at `https://github.com/UniversalDependencies/UD_Polish-LFG`

a bigger treebank with more genres, and the genre of each sentence specified. However, since we don't have that ideal treebank as of yet, we work with the current treebank. We report the LAS in Table 4.9.

	blog	**fiction**
blog	94.03	65.23
fiction	58.95	90.34

	blog	**nonfiction**
blog	92.54	68.17
nonfiction	65.25	94.03

	blog	**news**
blog	94.03	61.85
news	67.64	92.07

	blog	**social**
blog	94.03	63.83
social	60.31	91.87

	blog	**spoken**
blog	94.03	62.31
fiction	42.62	94.71

	social	**fiction**
social	91.87	62.84
fiction	62.86	90.34

	social	**nonfiction**
social	91.87	59.81
nonfiction	68.81	92.54

	social	**news**
social	91.87	56.62
news	71.00	92.07

	social	**spoken**
social	91.87	61.20
spoken	50.97	94.71

	fiction	**nonfiction**
fiction	90.34	62.15
nonfiction	73.41	92.54

	fiction	**news**
fiction	90.34	60.07
news	74.20	92.07

	fiction	**spoken**
fiction	90.34	72.60
spoken	52.50	94.71

	nonfiction	**news**
nonfiction	92.54	65.58
news	69.40	92.07

	nonfiction	**spoken**
nonfiction	92.54	71.49
spoken	46.06	94.71

	news	**spoken**
news	92.07	73.99
spoken	39.37	94.71

Table 4.9: LAS Scores for Genre Optimization

It is worth noting that we trained the UDPipe Parser with default configuration. From the table, there seems to be almost no coherence at all. The difference ranges from almost 50 points (case of *fiction* and *spoken*) to the best case of just a bit under 20 points (case of *fiction* and *nonfiction*). Let us try to make sense of data in individual cases.

In case of *spoken* genre, the parser, when trained on *spoken* data performs well on itself, but fails to perform well on any other genre. From this, we can guess that this genre is different from any other considered genre. Even so, the variability in the scores when parser trained on *spoken* data is used to evaluate other genres is of about 12 points.

It doesn't make sense to allocate around 40 points to data coming from different genre as a safe option. Going that way, a treebank with a singular genre would be different by 120 points from a treebank with 3 different genres. Using the same principle, we refuse to over-generalize and thus, don't optimise for the genre, owing to the lack of understanding of correlation of parsing score with genre based similarity.

We speculate that the data fails to show any patterns owing to not having enough baseline. In the sense, we need a better metric to asses the similarity of two genres (cf. Section 4.3.2) before the current metric can be optimised for genre based disparities.

To be sure that the effect is caused by genres, and not because of small size of the training data being used, we ran the experiment for a 100 times, without a significant change in the scores reported.

4.2.3 Other Factors

We took into account the two major factors that causes 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 don 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 the average number of tokens (defined on a per-language basis). 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 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

The presence of projective structures has been known to affect the parsing quality. The greater the difference in number of non-projective structures, the greater the difference in the parser performance. Minimising 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.

4.2.4 Brief Discussion on the metric

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 Based on POS Distribution

We briefly discussed on the metric KL_{cpos^3} in Section 2.1 as follows:

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

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.

For given treebanks A and B, we defined θ_{POS} in Definition 2 as

$$\theta_{POS} = KL_{cpos^3}(A, B) + KL_{cpos^3}(B, A) \tag{2.4}$$

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

A few properties of the metric θ_{POS} can be mentioned here.

1. While there is no upper limit on the theoretical value of the metric, it can never attain a negative value. The metric gets a zero-value when the distribution of coarse POS tags[5] is almost the same across both the treebanks in consideration.
$$\theta_{POS} \geq 0 \tag{4.2}$$

2. The given metric is symmetric in nature.

3. In general, the lower the value of the metric, the better.

In this section, we shall try to optimize θ_2 value, given θ_{POS} values for different treebanks as given in Table 4.10[6]. Following a look at the θ_{POS} values, we try to address the different factors that might affect the metric value. Eventually, we try to find the optimal value range for θ_2 in order for the considered treebanks to be considered harmonious with each other, with respect to θ_2 parameter.

[5]Since we are working with UPOS value fields in annotation, it can be referred to as UPOS tag for the sake of convenience and clarity. For the remainder of this section, we would use coarse POS tags and UPOS tag interchangeably.

[6]Code available at `https://github.com/Akshayanti/theta_POS`

Treebank1	**Treebank2**	θ_{POS}
es-AnCora	es-GSD	0.352
et-EDT	et-EWT	0.415
fi-FTB	fi-TDT	1.195
gl-CTG	gl-TreeGal	0.714
grc-Perseus	grc-PROIEL	4.641
ko-GSD	ko-Kaist	2.56
lt-ALKSNIS	lt-HSE	0.876
nl-Alpino	nl-LassySmall	0.667
pl-LFG	pl-PDB	0.625
pt-Bosque	pt-GSD	0.412
ro-Nonstandard	ro-RRT	1.364
sl-SSJ	sl-SST	2.787
sv-LinES	sv-Talbanken	0.441

fr	GSD	ParTUT
ParTUT	0.676	-
Sequoia	0.249	0.523

la	ITTB	Perseus
Perseus	1.106	-
PROIEL	3.763	3.901

ru	GSD	SynTagRus
SynTagRus	0.567	-
Taiga	1.027	0.631

no	Bokmaal	Nynorsk
Nynorsk	0.095	-
NynorskLIA	2.291	2.375

cs	CAC	CLTT	FicTree
CLTT	1.453	-	-
FicTree	1.138	2.657	-
PDT	0.373	1.935	1.006

en	EWT	GUM	LinES
GUM	0.256	-	-
LinES	0.41	0.449	-
ParTUT	0.619	0.432	0.545

it	ISDT	ParTUT	VIT
ParTUT	0.133	-	-
VIT	0.122	0.195	-
PoSTWITA	1.67	1.478	1.764

Table 4.10: θ_{POS} Scores for Different Treebanks per language, UDv2.4

4.3.1 Optimization for Size Disparity

In the definition of $KL_{cpos^3}(tgt, src)$, it is worth noting that the metric is defined on distributions of trigrams found in *tgt*, and *src*. As such, the size of a definition would not be a major driving factor in the KL_{cpos^3} scores between *tgt*, and *src*. The metric θ_{POS}, being the sum of KL_{cpos^3} scores, is therefore not affected significantly even in case of a size disparity of the distribution of upos tags in the two treebanks.

To reaffirm the belief that the size of treebanks does not affect the θ_{POS} metric, we conducted a small experiment, using data from UDv2.4 as follows:

1. We split data[7] from hi-hdtb treebank into 20 different parts, with each

[7]train + dev + test data

part containing {5, 10, ..., 100}% of the data[8] in the original treebank. It is worth noting that the parts were split in an increasing manner, i.e the part with x% of original data was present in entirety in the part with $(x + 5)$% of the original data. The parts are labelled as `hi_percentage` for easier reference.

2. We split `hi`-pud treebank into two parts, each part containing instances of only one genre out of wiki and news. We identify the parts as `pud-wiki` and `pud-news` respectively.

3. We calculate θ_{POS} scores for each `hi_percentage` part of the treebank with `pud-news` and report the results in Table 4.11.

4. We calculate the coverage of different UPOS trigrams among each treebank pair (`pud-news`, `hi_percentage`), and report the higher value in either direction in Table 4.12. We calculate the coverage by counting the number of UPOS trigrams that were present in both *src* and *tgt* while estimating $KL_{cpos^3}(tgt, src)$, expressed as a percentage of the total number of trigrams in *tgt*.

Part Name	θ_{POS}	Part Name	θ_{POS}
`hi_5`	0.757	`hi_55`	0.823
`hi_10`	0.776	`hi_60`	0.824
`hi_15`	0.785	`hi_65`	0.826
`hi_20`	0.795	`hi_70`	0.83
`hi_25`	0.796	`hi_75`	0.834
`hi_30`	0.807	`hi_80`	0.834
`hi_35`	0.815	`hi_85`	0.837
`hi_40`	0.816	`hi_90`	0.84
`hi_45`	0.821	`hi_95`	0.839
`hi_50`	0.82	`hi_100`	0.842

Table 4.11: θ_{POS} Scores And Effect of Size Disparity

At first glance, there seems to be an increase in the θ_{POS} scores with the increase in size of the split off the `hi`-HDTB treebank. However, once we look at the coverage data, we can start making sense of the increase in metric value. Upto a certain size limit, the proportion of the UPOS trigrams of source are not reflected in either of the treebank, causing the distributions to be further apart. However, as the size increases, there are increasingly more patterns common to both the treebanks, thus ensuring the metric value does not fluctuate and that it stabilizes. Therefore, size is not a metric that needs to be factored in for comparing the values of θ_{POS} metric, but the coverage is the important factor. The metric is based on calculating the expectation values, and therefore requires no tuning in this regard.

[8]in terms of number of sentences

Part Name	Coverage (in %)	Part Name	Coverage (in %)
`hi_5`	62.96	`hi_55`	85.471
`hi_10`	71.3	`hi_60`	86.099
`hi_15`	75.605	`hi_65`	86.368
`hi_20`	78.117	`hi_70`	86.547
`hi_25`	79.91	`hi_75`	86.726
`hi_30`	81.256	`hi_80`	86.906
`hi_35`	82.87	`hi_85`	87.085
`hi_40`	83.677	`hi_90`	87.354
`hi_45`	84.215	`hi_95`	87.444
`hi_50`	85.022	`hi_100`	87.713

Table 4.12: Coverage of UPOS trigrams over treebanks and `pud-news` data

4.3.2 Optimization for Genre Distribution

It is very well known that there exists significant difference between genres. Even if there are equal number of instances for different genres of data such that the data belongs to the same treebank, there would still exist some disparity in the parsing and tagging of different trees with respect to each other. Intuitively, some genres are more similar than others. For the same effect, KL_{cpos^3} scores were computed between equal instances[9] of different genres.

Genre	**Sentence Count**
blog	136
fiction	7 252
news	6 744
nonfiction	1 273
social	526
spoken (conversational)	789
academic	51
legal	11
spoken (media)	158
spoken (prepared)	306

Table 4.13: Genre Distribution in UDv2.4 `pl`-LFG treebank

`pl`-LFG treebank in UDv2.4 contains data from 10 different genres[10]. The sentence counts of different genres are shown in Table 4.13. Of the different kind of data in *spoken* genre, we keep only the *conversational*, discarding the others from consideration. We also remove *academic* and *legal* data from our consideration owing to a considerably low number of sentences. The genres we work with are marked in bold in the table. For the experiment, we first split `pl`-LFG treebank into the constituent genres, discarding those that are not marked in bold in Table 4.13. For the data from considered genres, we proceed as follows:

[9]in terms of number of sentences

[10]For understanding of what genre category involves exactly what kind of data, refer to the github page of the treebank at `https://github.com/UniversalDependencies/UD_Polish-LFG`

1. For a given genre, we randomly sample 125 instances from the genre data. We refer to this resultant split as working data for the genre.

2. For the given genre, compute θ_{POS} score of the working data of this genre with the working data of other genres.

3. Report the computed θ_{POS} scores.

We repeat the above steps for a 100 times, generating randomly sampled data for each genre, every time[11]. We report the average of the computed scores from all the runs in Table 4.14.

Genres	**blog**	**fiction**	**news**	**nonfiction**	**social**
fiction	0.511 ±0.033	-	-	-	-
news	0.446 ±0.027	0.468 ±0.038	-	-	-
nonfiction	0.461 ±0.027	0.41 ±0.034	0.453 ±0.034	-	-
social	0.594 ±0.045	0.5 ±0.05	0.631 ±0.061	0.501 ±0.048	-
spoken	1.184 ±0.068	0.84 ±0.074	1.178 ±0.095	0.994 ±0.081	0.824 ±0.056

Table 4.14: θ_{POS} Scores (± sd) for Genre Optimization (Averaged over 100 runs)

Intuitively, the pair of genres with the smaller value of θ_{POS} metric are more similar to each other than those contained in the pair with a higher metric value. From the table, it is evident that the *spoken* genre is least similar to any other considered genre, followed by *social* and *blog*.

There are a large number of genres that are not covered in our approach. However, we can come up with a theoretical limit nonetheless. We can describe our coverage of genres in 3 broad categories, viz. print media (*fiction*, *news*, *nonfiction*), social media (*blog*, *social*) and spoken data (*spoken*). There exists an overlap between the categories. However, any genre that is not included can almost always be classified within the realm of the categories defined above. It is important to note that the categories should be considered as points in the continuous range, and not as discreet ones in themselves. From there, we can establish the upper limits on the variability of θ_{POS} score as in Table 4.15.

Category 1	**Category 2**	**Limit**
Print	Social	0.8
Print	Spoken	1.4
Social	Spoken	1.4

Table 4.15: Limits on θ_{POS} for Genre-based Disparity

[11]For *blog* data containing 136 sentences, the data can be sampled in $\binom{136}{125} > 4.8 \times 10^{15}$ ways. The count of sentences in randomly sampled data was purposefully fixed to 125, to allow increased variability of trigrams in the data without suffering from overfitting caused due to lack of data.

4.4 Combining Optimized Values

In our experiments, we tried to optimize the two parameters θ_1 and θ_2 based on size and genre variations, by considering only one at a time. As mentioned earlier, this is not always possible since the premise of the genre-distribution based study is the isolation of individual genres. The isolation is not always possible. Also, while some genres are intuitively extremely different (like wiki data and internet slangs), some others are not very much so (like wikipedia and nonfiction). In case of the experiment for θ_{POS} based on genre distribution, the different distribution of UPOS in a different language might affect the scores for the gathered limits to be deemed useless.

Also is important to note that the genre-distribution and size-disparity can occur 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 distribution, 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. Experiment2: Mining False Non-Projective Structures

5.1 Observations About Problem Statement

5.1.1 Projectivization vs False Non-Projectivity

Projectivization can best be defined as making a non-projective structure artificially projective. This could be for the sake of training a parser, or to form a phrase-based constituency tree, among others. Nivre and Nilsson [2005] discuss the projectivization procedure for a non-projective edge using a process called edge lift. In the process, once a non-projective edge is detected, it is re-attached to the grandparent node as per the original attachment. The authors also define a minimal projective transformation to remove non-projective edges from a tree in an iterative manner. To do so, the algorithm starts with the node that has the least distance between the head and the dependent, and then performs lift operation on it, repeating the process until the entire tree is projectivized. The authors use this approach to find an optimal parser for parsing non-projectivities in the languages. A udapy-python block on the method of Nivre and Nilsson is also under development[1] at the time of writing this document.

While there have been several publications on how to develop parsers that can parse non-projective trees efficiently ([Hall and Novák, 2005], [Nivre, 2007], [Gómez-Rodríguez et al., 2009], among others), there are also multiple works in an attempt to understand the linguistic features that cause non-projectivities ([Mannem et al., 2009], [Mambrini and Passarotti, 2013], [Uresova et al., 2016], among others). The need for automatic error-correction methods for treebanks was realised early on, thanks to Boyd et al. [2008]. Since then, the focus of such techniques for dependency treebanks has been limited to either determining the direction of association of edges, determining right candidate node for association (also intersects with dependency parsing), or determining the correct association between the nodes. While a search for false non-projective structures would also employ all of the above, this is the first work that focuses on the particular problem of finding false non-projective structures.

Mannem et al. [2009] inspect the causes of non-projectivity in Hyderabad Dependency Treebank[2] (HyDT) for `hi` where they define the notion of rigidity of a non-projective structure as the possibility of reordering the construction without losing on the meaning of a sentence (not taking into consideration the discourse and topic-focus). The authors look at major causes of non-projectivities in the treebank, and classify each of the cause as rigid/non-rigid based on if it is possible to remove non-projectivity from the structure through reordering. Out of the total number of non-projective structures in the treebank, only around a 20% of them could be classified as rigid structures, which couldn't be projectivized. The authors rely on their knowledge of the language to classify the underlying causal linguistic phenomenon (of non-projectivity) as rigid/non-rigid, without offering

[1] `https://github.com/udapi/udapi-python/tree/master/udapi/block/transform`
[2] Later adapted into UD as `hi`-hdtb treebank

any (data driven) rules to determine the rigidity of a structure. The authors also define the notion of *naturalness* of a projectivized construction, based on if the speakers of language would prefer a projective or non-projective structure. Again, this is also not defined in an empirical manner, but is open to subjectivity of the speaker of the language. Although the definition of rigidity can come handy in context of free word-order languages, it is not a metric that can be employed to all languages universally. Extending definitions given by Mannem et al. and based on their classification, we would like to differentiate between a few different kinds of non-projective structures as below, while still going by the definition of *rigidity* and *natural*-ness as given by the authors.

1. Not rigid, and **does not** require reordering for natural projectivity
2. Not rigid, and **does** require reordering for natural projectivity
3. Not rigid, and no natural projectivity possible
4. Rigid, with natural projectivity possible
5. Rigid, and no natural projectivity possible

Given that we are not interested in reordering the nodes of a tree in a given treebank, the second case as mentioned above does not warrant consideration in the present research. For the exact same reason, we can combine remaining cases to get the following two cases, referred to as condensed cases henceforth:

1. **Natural Projectivity Possible**: A given edge can occur non-projectively, as well as projectively. A very common example of this can be seen in poetry, or in elliptical constructions where the choice of construction can make an edge projective or non-projective.
2. **Natural Projectivity Not Possible**: There are a few constructions that seem to always introduce non-projectivity. In `hi`, the tendency is exhibited by token कि (*ki*; that) as pointed out by Mannem et al. [2009].

Different treebanks in UD do not annotate the multiple possible structures of a given sentence, but rather the most frequently used one. Therefore, a given sentence (or a sequence of tokens contained therein) can be categorised in either of the two condensed cases. In this experiment, we seek to identify the cases of the first kind where an edge has been annotated as non-projective, when a projective edge is more likely. Going the reverse way, we also want to identify the cases where a given edge in second case has been annotated as a projective edge. To do so, we use the concept of variation nuclei as used in Boyd et al. [2008]. We have not yet defined the *natural*-ness of a construction so far, in terms of data-driven approach. We do not go over the concept here, but elaborate on it alongwith the discussion of variation nuclei in Section 5.4.

5.1.2 Terms Related to Non-Projectivity

In a directed edge $i \rightarrow j$, we refer to i as head, and j as dependent. Unless mentioned otherwise, we refer to directed edges, with the ordered pair (i, j) denoting $i \rightarrow j$. We refer to the subtree, rooted at node i as $Subtree(i)$. To specify that node i precedes node j in tree, we denote it in the form of $i < j$. A node x is said to be in gap of non-projective edge if (i, j) if $x \in (i, j)$ & $x \notin Subtree(i)$. Alternatively, the node x is said to belong to gap of (i, j), denoted as $x \in Gap(i, j)$.

Edge degree is defined as the number of nodes in a tree T that are in gap. The edge-degree of a tree is defined as the maximum edge-degree from the individual edge's edge-degree. While this is based on a per-edge basis, a gap degree is defined as the number of times a non-projective edge's gap is broken. A very trivial way of calculating gap degree of an edge (i, j) is to iterate through the nodes in order. Let us say, we have node $x \in (i, j)$. If x alongwith its immediate neighbours are all in $Subtree(i)$ or if none of them is, the gap is not broken. Every time the condition is not met, the gap is said to be broken. The gap degree of an edge is the number of times the gap is broken. Similar to the definition of edge-degree of a tree, gap degree of a tree is the maximum gap degree among all edges therein.

5.1.3 Relaxations to Non-Projectivity

The condition of non-projectivity is a strict constraint for natural languages, exhibited by very few constructions in most languages of the world. To better account for linguistic processes, several relaxations to the definition of non-projectivity were defined. A discussion of all such relaxations is out of scope of this work. However, we define 3 most widely used relaxations here.

1. **Planarity**
   The given tree is said to be planar, if it does not have any edges that overlap. Formally speaking, given two undirected edges $i1 \leftrightarrow j1$ and $i2 \leftrightarrow j2$; if $i1 < i2 < j1 < j2$ or $i1 > i2 > j1 > j2$, the edges are said to overlap. Therefore, a given tree is called non-planar if there exists a pair of edges $i1 \leftrightarrow j1$ and $i2 \leftrightarrow j2$ such that the edges overlap.

2. **Ill-Nestedness**
   It is easier to define the condition of ill-nestedness rather than to define the well-nestedness. A given (sub)tree is called ill-nested, if for given undirected edges $i1 \leftrightarrow j1$ and $i2 \leftrightarrow j2$; $i1 \in Gap(i2, j2)$ & $i2 \in Gap(i1, j1)$. It is worth noting that projective trees are always well-nestedness, but a well-nested tree is not always projective.

3. **Mild Non-Projectivity**
   A tree is said to be mildly non-projective if

   (a) It is well-nested.

   (b) The gap degree of the tree is bound by any constant k. Essentially, gap degree of tree $\leq k$.

5.1.4 Genre Association with Non-Projectivity

Mambrini and Passarotti [2013] show in their work on `grc` that non-projectivity might be affected by genre distribution. In particular, poetic style is liable to contain more non-projective structures than prose. The claim about genre distribution affecting projectivity is also supported by Yadav et al. [2017], where they look at different genres (news and conversations) using different parameters to account for lack of non-projective structures in the conversational genre, than in news genre. Although more research is needed on the topic, we try to base our experiments while restricting ourselves to the genre distributions that are more or less alike (cf. Section 5.2 and discussion of `support treebanks`).

5.1.5 Punctuation Induced Non-Projectivity

A punctuation node can induce non-projectivity in either of the two ways as mentioned below:

1. Non-projective attachment of a punctuation node.

2. Non-projectivity caused by only punctuation node(s) in gap.

According to UD guidelines, a punctuation node should be attached to the surrounding dependent unit. However, it is not always possible to identify the correct dependent where the node should be attached. Consider the following example from `en`-lines UD v2.5 treebank, and the associated dependency tree in Figure 5.1, with specific reference to the punctuation mark immediately following the token marked in bold. While the punctuation token could have been correctly marked to either of *right* or *said*, it is attached to *'s* causing non-projectivity.

Example 6. That's **right**, said Quinn.

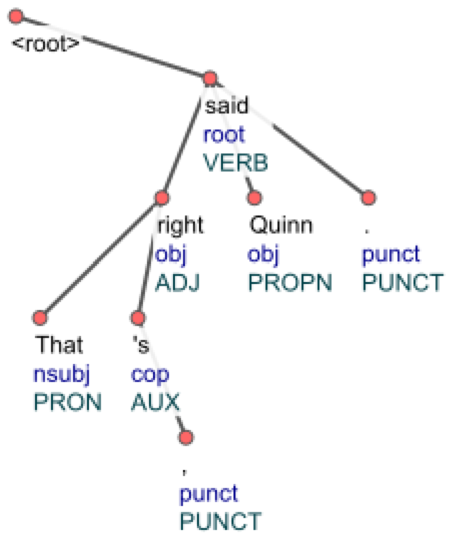


Figure 5.1: Punctuation Node Attached Non-Projectively

Similarly, the punctuation node(s) can induce non-projectivity, by attaching itself to the wrong node. Consider the following example from `en`-ewt UDv2.5

treebank, and the associated dependency tree in Figure 5.2, with specific reference to the punctuation mark immediately following the token in bold. A faulty association of this punctuation induces non-projectivity in another node.

Example 7. Analyst Team 1 : **Coach** : Lisa Gilette

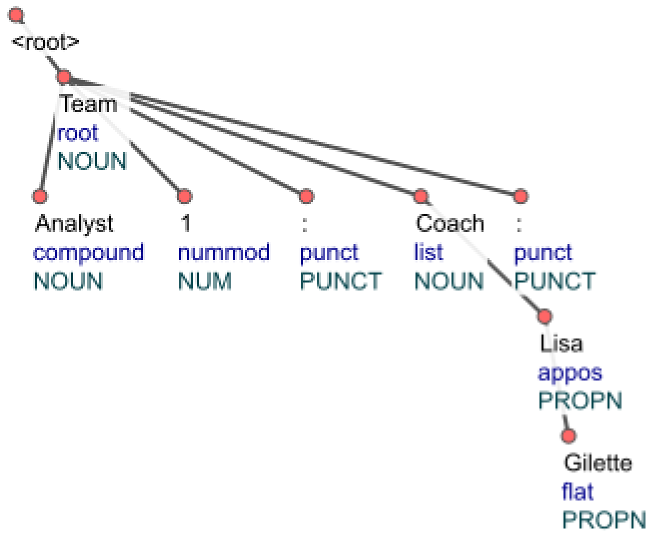


Figure 5.2: Punctuation Node Causing Non-Projectivity

In our treatment of the data, we discard a non-projective edge $i \rightarrow j$ if any of the following conditions are met, citing it as Punctuation Induced Non-Projectivity.

1. Either of head (node i), or dependent (node j) is a punctuation node.

2. The nodes in $Gap(i, j)$ consist of **only** punctuation node(s).

The rest of the chapter is organised as follows. Section 5.2 talks about the used dataset for the experiment. We split the analysis of the data in two parts, analysing the non-mild non-projective structures in Section 5.3, and then discussing the concept of variation nuclei and the selection (and querying process) of variation nuclei in mild non-projective structures in Section 5.4. We continue with an evaluation of the findings of the experiments in Section 5.5. Section 5.6 concludes.

5.2 Dataset

The experiment was started on UDv2.3, and eventually brought over to UDv2.4. The experiment(s) on non-projective structures in a treebank were tried over an extended period of time, and as the newer versions of UD were released, the dataset for the experiment changed. The presented version of the experiments was conducted over UDv2.5 [Zeman et al., 2019]. As with other experiments that changed datasets in the chronology of the research, the number of problematic cases have diminished over the newer versions of UD. Nonetheless, the problem of identifying the incorrect cases of non-projectivity in a treebank is still relevant, especially in case of treebank(s) of languages where non-projectivity is already established as a common phenomenon[3] (Mambrini and Passarotti [2013], Bhat and Sharma [2012], Havelka [2007]).

The `fixpunct` block[4] of udapi-Python [Popel et al., 2017] attempts to rehang the punctuation nodes. This also takes care of the punctuation nodes that cause non-projectivity, or are attached non-projectively. Since the procedure of lifting the punctuation nodes is not deterministic, there are times when the udapi block makes the situation worse, by introducing non-projectivities in the given data[5].

UDv2.5 contains 157 treebanks in 90 languages. Of those treebanks, we disregarded PUD treebanks entirely from the consideration, owing to their small size (1000 sentences). In the remaining treebanks, there exist considerable number of non-projectivities induced due to punctuation (cf. Section 5.1.5). The `fixpunct` block is applied on these treebanks to eliminate cases of punctuation induced non-projectivity.

Having treated the treebanks with udapi-python block, there were 2 categories of treebanks that were chosen as experimental data. The first category, hereafter referred to as `working treebanks`, contains top 10 treebanks (ranked in order of percentage composition of non-projective trees) which satisfy the following conditions:

1. Total number of trees $\geq 10,000$

2. Number of non-projective trees $\geq 10\% \cdot$ Total number of trees

The second category is hereafter referred to as `support treebanks`. This category consists of the treebanks that provide added instances of data for treebanks found in `working treebanks`. For a treebank $T_w \in$ `working treebanks`, a candidate treebank T_c can be included in the list of `support treebanks` if it satisfies the following conditions:

1. Language of $T_w =$ Language of T_c

2. Total number of trees in $T_c \geq 5,000$

3. Number of non-projective trees in $T_c \geq 5\% \cdot$ Total number of trees in T_c

4. Genre composition of $T_c \subseteq$ Genre composition of T_w

Having considered the conditions for each category, we get a list of treebanks as can be seen in Table 5.1. The table also shows the counts of trees which fail the relaxations to the conditions of non-projectivity as well (cf. Section 5.1.3). The two categories of treebanks, viz. `working treebanks` and `support treebanks` are separated by a horizontal line in the table. In all instances where the two categories are maintained separate, the results of the treebanks shall

[3]As mentioned earlier, there exists an association between the genre of the text and the counts of non-projective structures therein. Although we do not discard that association, in this context we refer to the languages where the probability of finding non-projectivities in high, regardless of the genre.

[4]`https://github.com/udapi/udapi-python/blob/master/udapi/block/ud/fixpunct.py`

[5]One such example can be seen in case of `cs_pdt` in Table 5.4 where the number of non-projective edges is increased, post application of the udapi block on the treebank

be included in the same manner, with a horizontal line in between. Table 5.2 and Table 5.3 show the gap degree and edge degree statistics respectively for the experimental data. The effect of application of udapi-python block can be seen in Table 5.4 where the counts of non-projective edges are listed for the treebanks in the experimental data. Table 5.5 shows the counts of all non-projective structures, and their relaxations in UDv2.5.

Treebank	**# Trees**	**Non-Proj.**		**Non-Planar**		**Ill-Nested**	
		Trees	%	Trees	%	Trees	%
`grc`_perseus	13 919	8 683	62.38	1 098	7.89	126	0.91
`grc`_proiel	17 080	6 409	37.52	392	2.30	38	0.22
`la`_ittb	21 011	7 680	36.55	338	1.61	35	0.17
`la`_proiel	18 411	5 227	28.39	448	2.43	38	0.21
`ko`_kaist	27 363	5 875	21.47	83	0.30	0	0
`fro`_srcmf	17 678	2 726	15.42	290	1.64	82	0.46
`orv`_torot	16 944	2 575	15.20	71	0.42	4	0.02
`nl`_alpino	13 578	1 953	14.38	128	0.94	0	0
`hi`_hdtb	16 647	2 264	13.60	116	0.70	13	0.08
`cs`_cac	24 709	3 143	12.72	50	0.20	14	0.06
`cs`_pdt	87 913	10 145	11.54	159	0.18	47	0.05
`nl`_lassysmall	7 338	446	6.08	25	0.34	1	0.01

Table 5.1: Non-Projectivity and Relaxations in Experimental Data
% is calculated on the total number of trees (# Trees)

Treebank	**Gap Degree**				
	gd0	gd1	gd2	gd3	gd4+
`grc`_perseus	45.57	50.59	3.72	0.12	-
`grc`_proiel	66.70	31.18	2.04	0.08	<0.01
`la`_ittb	68.74	30.20	1.06	<0.01	-
`la`_proiel	77.31	21.70	0.93	0.05	<0.01
`ko`_kaist	84.54	15.36	0.10	-	-
`fro`_srcmf	89.11	10.58	0.29	0.02	<0.01
`orv`_torot	87.94	11.77	0.29	<0.01	-
`nl`_alpino	90.48	9.31	0.20	-	<0.01
`hi`_hdtb	89.49	10.19	0.28	0.04	<0.01
`cs`_cac	92.41	7.52	0.07	-	-
`cs`_pdt	92.81	7.06	0.12	<0.01	<0.01
`nl`_lassysmall	95.97	3.95	0.08	-	-

Table 5.2: Gap Degrees (in %) in Experimental Data
% is calculated on the total number of edges

Treebank	Edge Degree				
	ed0	ed1	ed2	ed3	ed4+
grc_perseus	37.62	35.70	12.37	6.19	8.12
grc_proiel	62.48	23.58	6.74	3.06	4.14
la_ittb	63.45	25.47	4.63	2.52	3.93
la_proiel	71.61	17.33	5.04	2.60	3.41
ko_kaist	78.53	6.02	3.35	2.48	9.61
fro_srcmf	84.58	5.39	4.18	2.40	3.45
orv_torot	84.80	9.84	2.67	1.29	1.39
nl_alpino	85.62	6.47	3.87	1.86	2.18
hi_hdtb	86.4	4.05	3.07	2.62	3.85
cs_cac	87.28	6.58	3.12	1.33	1.68
cs_pdt	88.46	5.41	2.82	1.26	2.05
nl_lassysmall	93.92	2.89	1.46	0.76	0.97

Table 5.3: Edge Degrees (in %) in Experimental Data
% is calculated on the total number of edges

Treebank	# Edges	Non-Proj. Edges$^+$		Non-Proj. Edges*	
		No.	%	No.	%
grc_perseus	202 989	20 574	10.14	18 635	9.18
grc_proiel	213 999	11 247	5.26	11 247	5.26
la_ittb	353 035	11 104	3.15	10 689	3.03
la_proiel	200 163	9 481	4.74	9 481	4.74
ko_kaist	350 090	8 928	2.55	8 821	2.52
fro_srcmf	170 741	3 712	2.17	3 712	2.17
orv_torot	149 780	3 961	2.64	3 961	2.64
nl_alpino	208 470	2 779	1.33	2 753	1.32
hi_hdtb	351 704	2 680	0.76	2 680	0.76
cs_cac	494 383	3 827	0.77	3 827	0.77
cs_pdt	1 506 484	12 310	0.82	12 387	0.82
nl_lassysmall	98 033	582	0.59	576	0.59

Table 5.4: Effect of application of udapi-python block on Experimental Data
$^+$: Counts before the application
*: Counts after the application
% is calculated on the total number of edges (# Edges)

Treebank	# Trees	Non-Proj.		Non-Planar		Ill-Nested	
		Trees	%	Trees	%	Trees	%
af_afribooms	1934	432	22.3371	19	0.9824	1	0.0517
aii_as	57	0	0.0	0	0.0	0	0.0
akk_pisandub	101	7	6.9307	0	0.0	0	0.0
am_att	1074	26	2.4209	0	0.0	0	0.0
ar_nyuad	19738	122	0.6181	0	0.0	0	0.0
ar_padt	7664	638	8.3246	19	0.2479	11	0.1435

Continued on next page

Treebank	**# Trees**	**Non-Proj.**		**Non-Planar**		**Ill-Nested**	
		Trees	%	Trees	%	Trees	%
ar_pud	1000	38	3.8	1	0.1	0	0.0
be_hse	637	46	7.2214	0	0.0	0	0.0
bg_btb	11138	342	3.0706	2	0.018	1	0.009
bho_bhtb	254	35	13.7795	7	2.7559	1	0.3937
bm_crb	1026	33	3.2164	0	0.0	0	0.0
br_keb	888	24	2.7027	1	0.1126	1	0.1126
bxr_bdt	927	145	15.6419	12	1.2945	1	0.1079
ca_ancora	16678	746	4.473	5	0.03	0	0.0
cop_scriptorium	1575	206	13.0794	0	0.0	0	0.0
cs_cac	24709	3143	12.7201	50	0.2024	14	0.0567
cs_cltt	1125	163	14.4889	7	0.6222	6	0.5333
cs_fictree	12760	1455	11.4028	32	0.2508	3	0.0235
cs_pdt	87913	10098	11.4864	157	0.1786	47	0.0535
cs_pud	1000	104	10.4	2	0.2	1	0.1
cu_proiel	6338	1034	16.3143	32	0.5049	5	0.0789
cy_ccg	956	18	1.8828	1	0.1046	0	0.0
da_ddt	5512	1185	21.4985	55	0.9978	19	0.3447
de_gsd	15590	1451	9.3072	24	0.1539	4	0.0257
de_hdt	189928	12871	6.7768	588	0.3096	37	0.0195
de_lit	1922	150	7.8044	10	0.5203	1	0.052
de_pud	1000	137	13.7	6	0.6	1	0.1
el_gdt	2521	142	5.6327	0	0.0	0	0.0
en_esl	5124	208	4.0593	7	0.1366	4	0.0781
en_ewt	16622	767	4.6144	22	0.1324	6	0.0361
en_gum	5427	410	7.5548	10	0.1843	1	0.0184
en_lines	5243	459	8.7545	24	0.4578	13	0.2479
en_partut	2090	39	1.866	2	0.0957	1	0.0478
en_pronouns	285	5	1.7544	0	0.0	0	0.0
en_pud	1000	45	4.5	1	0.1	0	0.0
es_ancora	17680	928	5.2489	5	0.0283	0	0.0
es_gsd	16013	937	5.8515	16	0.0999	2	0.0125
es_pud	1000	45	4.5	1	0.1	0	0.0
et_edt	30972	993	3.2061	9	0.0291	3	0.0097
et_ewt	1662	111	6.6787	2	0.1203	1	0.0602
eu_bdt	8993	2983	33.1702	424	4.7148	92	1.023
fa_seraji	5997	401	6.6867	25	0.4169	1	0.0167
fi_ftb	18723	1444	7.7124	150	0.8012	73	0.3899
fi_pud	1000	36	3.6	0	0.0	0	0.0
fi_tdt	15136	931	6.1509	9	0.0595	0	0.0
fo_oft	1208	33	2.7318	2	0.1656	1	0.0828
fr_fqb	2289	75	3.2765	1	0.0437	0	0.0
fr_ftb	18535	2019	10.8929	69	0.3723	21	0.1133
fr_gsd	16342	428	2.619	6	0.0367	0	0.0
fr_partut	1020	45	4.4118	0	0.0	0	0.0
fr_pud	1000	17	1.7	0	0.0	0	0.0

Continued on next page

Treebank	# Trees	Non-Proj.		Non-Planar		Ill-Nested	
		Trees	%	Trees	%	Trees	%
fr_sequoia	3099	66	2.1297	0	0.0	0	0.0
fr_spoken	2789	340	12.1907	8	0.2868	1	0.0359
fro_srcmf	17678	2726	15.4203	290	1.6405	82	0.4639
ga_idt	1763	272	15.4282	22	1.2479	9	0.5105
gd_arcosg	2193	259	11.8103	14	0.6384	8	0.3648
gl_ctg	3993	0	0.0	0	0.0	0	0.0
gl_treegal	1000	113	11.3	7	0.7	2	0.2
got_proiel	5401	949	17.5708	32	0.5925	5	0.0926
grc_perseus	13919	8890	63.8695	1275	9.1601	150	1.0777
grc_proiel	17080	6409	37.5234	392	2.2951	38	0.2225
gsw_uzh	100	4	4.0	0	0.0	0	0.0
gun_dooley	1046	0	0.0	0	0.0	0	0.0
gun_thomas	98	4	4.0816	0	0.0	0	0.0
he_htb	6216	472	7.5933	6	0.0965	0	0.0
hi_hdtb	16647	2264	13.6	116	0.6968	13	0.0781
hi_pud	1000	257	25.7	16	1.6	1	0.1
hr_set	9010	810	8.99	20	0.222	9	0.0999
hsb_ufal	646	73	11.3003	2	0.3096	0	0.0
hu_szeged	1800	488	27.1111	38	2.1111	17	0.9444
hy_armtdp	2502	179	7.1543	4	0.1599	0	0.0
id_gsd	5593	291	5.2029	11	0.1967	2	0.0358
id_pud	1000	13	1.3	0	0.0	0	0.0
it_isdt	14167	196	1.3835	9	0.0635	5	0.0353
it_partut	2090	42	2.0096	2	0.0957	2	0.0957
it_postwita	6713	86	1.2811	2	0.0298	2	0.0298
it_pud	1000	8	0.8	0	0.0	0	0.0
it_twittiro	1424	17	1.1938	1	0.0702	0	0.0
it_vit	10087	353	3.4996	18	0.1784	7	0.0694
ja_bccwj	57109	163	0.2854	1	0.0018	0	0.0
ja_gsd	8186	0	0.0	0	0.0	0	0.0
ja_modern	822	5	0.6083	0	0.0	0	0.0
ja_pud	1000	0	0.0	0	0.0	0	0.0
kk_ktb	1078	130	12.0594	3	0.2783	1	0.0928
kmr_mg	754	130	17.2414	5	0.6631	4	0.5305
ko_gsd	6339	1006	15.87	22	0.3471	3	0.0473
ko_kaist	27363	5938	21.7008	89	0.3253	0	0.0
ko_pud	1000	66	6.6	0	0.0	0	0.0
koi_uh	49	1	2.0408	0	0.0	0	0.0
kpv_ikdp	117	3	2.5641	0	0.0	0	0.0
kpv_lattice	210	4	1.9048	1	0.4762	0	0.0
krl_kkpp	228	45	19.7368	3	1.3158	0	0.0
la_ittb	21011	7771	36.9854	357	1.6991	39	0.1856
la_perseus	2273	1094	48.1302	201	8.8429	64	2.8157
la_proiel	18411	5227	28.3906	448	2.4333	38	0.2064
lt_alksnis	3642	441	12.1087	7	0.1922	1	0.0275

Continued on next page

Treebank	**# Trees**	**Non-Proj.**		**Non-Planar**		**Ill-Nested**	
		Trees	%	Trees	%	Trees	%
lt_hse	263	38	14.4487	2	0.7605	1	0.3802
lv_lvtb	13643	888	6.5088	7	0.0513	0	0.0
lzh_kyoto	15115	0	0.0	0	0.0	0	0.0
mdf_jr	65	2	3.0769	0	0.0	0	0.0
mr_ufal	466	28	6.0086	1	0.2146	1	0.2146
mt_mudt	2074	81	3.9055	1	0.0482	0	0.0
myv_jr	1550	79	5.0968	4	0.2581	3	0.1935
nl_alpino	13578	1961	14.4425	129	0.9501	0	0.0
nl_lassysmall	7338	447	6.0916	25	0.3407	1	0.0136
no_bokmaal	20044	1495	7.4586	32	0.1596	0	0.0
no_nynorsk	17575	1361	7.744	27	0.1536	4	0.0228
no_nynorsklia	5250	495	9.4286	37	0.7048	3	0.0571
olo_kkpp	125	17	13.6	2	1.6	2	1.6
orv_rnc	604	189	31.2914	10	1.6556	3	0.4967
orv_torot	16944	2575	15.1971	71	0.419	4	0.0236
pcm_nsc	948	6	0.6329	0	0.0	0	0.0
pl_lfg	17246	111	0.6436	3	0.0174	1	0.0058
pl_pdb	22152	1390	6.2748	20	0.0903	2	0.009
pl_pud	1000	52	5.2	0	0.0	0	0.0
pt_bosque	9365	2862	30.5606	307	3.2782	72	0.7688
pt_gsd	12078	684	5.6632	11	0.0911	6	0.0497
pt_pud	1000	33	3.3	0	0.0	0	0.0
qhe_hiencs	1898	192	10.1159	7	0.3688	4	0.2107
ro_nonstandard	15843	819	5.1695	9	0.0568	1	0.0063
ro_rrt	9524	864	9.0718	21	0.2205	10	0.105
ro_simonero	491	54	10.998	4	0.8147	1	0.2037
ru_gsd	5030	318	6.3221	10	0.1988	2	0.0398
ru_pud	1000	24	2.4	0	0.0	0	0.0
ru_syntagrus	61889	4658	7.5264	58	0.0937	13	0.021
ru_taiga	3264	277	8.4865	12	0.3676	5	0.1532
sa_ufal	230	40	17.3913	3	1.3043	0	0.0
sk_snk	10604	347	3.2724	4	0.0377	2	0.0189
sl_ssj	8000	960	12.0	11	0.1375	2	0.025
sl_sst	3188	144	4.5169	1	0.0314	0	0.0
sme_giella	3122	338	10.8264	21	0.6726	5	0.1602
sms_giellagas	36	2	5.5556	0	0.0	0	0.0
sr_set	4384	172	3.9234	5	0.1141	1	0.0228
sv_lines	5243	305	5.8173	13	0.2479	4	0.0763
sv_pud	1000	38	3.8	0	0.0	0	0.0
sv_talbanken	6026	181	3.0037	0	0.0	0	0.0
swl_sslc	203	67	33.0049	6	2.9557	0	0.0
ta_ttb	600	9	1.5	0	0.0	0	0.0
te_mtg	1328	2	0.1506	0	0.0	0	0.0
th_pud	1000	28	2.8	0	0.0	0	0.0
tl_trg	55	0	0.0	0	0.0	0	0.0

Continued on next page

Treebank	**# Trees**	**Non-Proj.**		**Non-Planar**		**Ill-Nested**	
		Trees	%	Trees	%	Trees	%
tr_gb	2802	28	0.9993	0	0.0	0	0.0
tr_imst	5635	646	11.4641	65	1.1535	26	0.4614
tr_pud	1000	149	14.9	4	0.4	0	0.0
ug_udt	3456	172	4.9769	1	0.0289	0	0.0
uk_iu	7060	547	7.7479	9	0.1275	1	0.0142
ur_udtb	5130	1158	22.5731	98	1.9103	27	0.5263
vi_vtb	3000	87	2.9	1	0.0333	0	0.0
wbp_ufal	55	6	10.9091	0	0.0	0	0.0
wo_wtb	2107	63	2.99	1	0.0475	1	0.0475
yo_ytb	100	9	9.0	0	0.0	0	0.0
yue_hk	1004	126	12.5498	13	1.2948	5	0.498
zh_cfl	451	4	0.8869	0	0.0	0	0.0
zh_gsd	4997	117	2.3414	1	0.02	0	0.0
zh_gsdsimp	4997	0	0.0	0	0.0	0	0.0
zh_hk	1004	43	4.2829	0	0.0	0	0.0
zh_pud	1000	7	0.7	0	0.0	0	0.0

Table 5.5: Non-Projectivity and Relaxations in UDv2.5 (% of # Trees)

5.3 Structures with Gap Degree > 2

Nivre [2006] points out that limiting the gap degree to at most 2 excludes less than 0.5% of the dependency graphs. Looking at the data in Table 5.2, we can see that the number of trees with a gap degree ≥ 3 is less than 0.15%. This essentially means that we can focus on the experimental data in 2 parts, one where the trees have a gap degree of at most 2, and the others with gap degree of at least 3.

5.3.1 Not Mild Non-Projective Structures

Having defined the notion of mild non-projectivity earlier in Section 5.1.3, we specify the limit on gap degree as $k = 4$. We find that there are roughly 0.02% of the total number of non-projective trees (11 trees in 4 treebanks) that fail to satisfy one or both of the conditions (cf. Table 5.6). A manual evaluation of such cases indicates errors in annotation, and therefore a false non-projectivity. It is worth noting that in some of the aforementioned cases, the entire tree might not be projective after the erroneous non-projective edge is removed. In some cases, this could be owing to the presence of another valid non-projective edge in the tree. In other cases, this could be due to the non-projective edge containing more gaps in the wrong annotation, and the correction decreasing the gap degree.

Treebank	# Non Proj. Trees	# Gap Degree ≥ 3	
		Mild	Not Mild
grc_perseus	8 683	13	3
grc_proiel	6 409	14	-
la_ittb	7 680	1	-
la_proiel	5 227	6	4
ko_kaist	5 875	-	-
fro_srcmf	2 726	2	2
orv_torot	2 575	1	-
nl_alpino	1 953	1	-
hi_hdtb	2 264	5	2
cs_cac	3 143	-	-

Table 5.6: Mildness of Non-Projectivity in High Gap Degree Trees

5.3.2 Mild Non-Projectivity

The counts of mild non-projective trees in different treebanks is shown in Table 5.6. The mild non-projective trees being few in number were also manually annotated by native speakers of the language or an annotator comfortable with the language. In manual evaluation, we found around 25% of trees contained genuine instances of non-projectivity, with no possibility of a reduction in the gap degree of the tree.

5.4 Structures with Gap Degree of at max 2

Having analyzed high gap-degree trees, we are left with trees of gap-degree ≤ 2. For analyzing the high count of such trees, we adapt the concept of variation nucleus, as adapted for usage in case of dependency trees by Boyd et al. [2008], tuning it specifically for the current problem at hand.

5.4.1 Variation Nucleus

For our problem statement, we are primarily concerned with the edges that are non-projective in nature. To that effect, we develop the variation nuclei for the concerned edge. In Boyd et al. [2008], the authors use deprels and a NIL relation to combine together the different possible annotations between the nodes of a given edge. While that approach is useful to find the inconsistencies in labelling, it is not helpful in determining if a particular edge is more likely to be projective or not. We are interested in looking at the edge's occurrence in the different trees in the treebank. Once all such occurrences are found, we can check if the majority of them are attached projectively or not, and then take a decision based on the general consensus. In a data-driven approach, we define the *natural*-ness of an edge by looking at the edge in different trees, and going by the majority association type (dependent attached to head projectively or non-projectively). To get a clear consensus that is not readily affected by addition of data, we need to define some parameter for the consensus. We get to definition of such a constraint in Section 5.5, after we have defined the variation nucleus, and have formally outlined the variation nucleus query methodology.

Although gap degree is considered as a tighter constraint, there have been no works based on inspection of the gap nodes (nodes in the gap of a non-projective edge) to the best of our knowledge. For the current specific problem, we hypothesise that inspecting the gap nodes could be advantageous. The primary advantage of inspecting gap nodes is in languages with a limited free word order. If the token in edge is not the main cause for the non-projectivity, it should be because of what comes in between the edge. In Mambrini and Passarotti [2013], an inspection of nodes in gap revealed the presence of clitics very often responsible for non-projectivity in `grc`. In such a case, the gap node would definitely add as a determining factor to query for non-projective associations of the edges. To that effect, in addition to the head and dependent of an edge, we also select nodes from gap to add to our variation nucleus.

5.4.2 Variation Nucleus: Definition and Constraints

For browsing through different possible attachments of the same dependent to the same head, we always need the two ends of an edge as primary fixed element to form a query. For the given edge, if we search by keeping a particular deprel or UPOS tag as the constant factor, the query results would essentially explode in count. The approach of using deprels is referred to as "dependency heuristic" by Boyd et al.. We mentioned earlier how the variation nuclei as proposed by Boyd et al. was evaluated on UD treebanks in de Marneffe et al. [2017] for three languages (`en`, `fi`, `fr`). They found that the technique of using lemma works for languages that are not morphologically rich, but fails otherwise. Owing to the different forms possible for a given lemma in morphologically rich languages, we still advocate for the lemma based approach.

Limiting the gap degree to 2 in the considered trees does not restrict the number of gap nodes. Therefore, it is not possible (and/or useful) to add every gap node to the variation nucleus. We call the set of gap nodes that compose the variation nucleus as gap nucleus. We select nodes from gap nodes to add to gap nucleus by the process of elimination using Algorithm 1.

Algorithm 1 get_vn_from_gap()

```
Input: Set of gap nodes $G$, Head of non-projective edge $H$
 1: Set of candidate nodes $V = G$
 2: for all $x$ in $G$ do
 3:     if $x == ROOT$ then
 4:         DO NOTHING
 5:     else if $H$ in $x.descendants$ then
 6:         for all $z$ in $x.descendants$ do
 7:             if $z$ in $H.descendants$ then
 8:                 DO NOTHING
 9:             else
10:                 $V.remove(z)$
11:             end if
12:         end for
13:     else
14:         for all $z$ in $x.descendants$ do
15:             if $z$ in $V$ then
16:                 $V.remove(z)$
17:             end if
18:         end for
19:     end if
20: end for
21: return $V$
```

Proceeding as per Algorithm 1, we are left with two kind of nodes in the resultant gap nucleus. The first kind consists of all the nodes that are leaf nodes, i.e. they have no children; and the second kind consists of the nodes who do not have their parents in the gap. We are interested in looking at gap nucleus that is sufficiently large enough to provide context of the gap. As such, we place the restriction that the length of gap nucleus should be at least 2 before it can be added to the variation nucleus. Given a sufficiently large gap nucleus, it is very unlikely that all the nodes present in the gap nucleus would occur in another tree's gap nucleus as well. Thus, we create combinations of elements in the gap nuclei, ranging from choice of 1 node at a time, until all the nodes are utilised. These combinations of nodes from the gap nuclei constitute the sub-nuclei, and are used to generate variation nuclei of their own. We refer to the largest variation nucleus generated by the gap nucleus as principal variation nucleus.

A considerable difference in selection of our variation nucleus from that of Boyd et al. is the usage of non-fringe heuristic, which essentially requires same immediate neighbours for the ends of the edge in every queried instance. Given a non-projective edge (i, j), if we denote the immediate neighbours of the tokens (in word order) as $i-1, i+1$ and $j-1, j+1$ for the nodes i and j respectively, non-fringe heuristic requires a retrieved instance of the variation nucleus to have $i-1, i+1$ and $j-1, j+1$ as neighbours of nodes i, and j respectively for the retrieval to be valid. For the task of mining for false non-projectivity, we argue that the approach is counter-productive in two ways:

1. In the established research on linguistic analysis of non-projective structures (cf. [Mambrini and Passarotti, 2013], [Bhat and Sharma, 2012], among oth-

ers), there have been identified instances of sole tokens being responsible for causing non-projectivity. Once the non-fringe heuristic is used, the precision in query of the responsible node decreases, resulting in lower number of verifiable instances.

2. The concept of looking at the neighboring nodes works efficiently when one needs to identify inconsistencies as classes, without pinpointing to the error kind. This approach works for sequence labelling based tasks such as mining for tagging inconsistencies or dependency based inconsistencies, but not when a particular edge has to be scanned through.

Owing to the above mentioned reasons, we do not use non-fringe heuristic and therefore rely on using lemmas to encompass the edge in considerably more lenses than would be possible with using wordforms, for example.

5.4.3 Querying for Variation Nucleus

For this part of the experiment, we now use the `support treebanks` as well. For each generated variation nucleus, we first create an index to list the source of the sentence ID that resulted in the nucleus. The outline pattern for querying a given variation nucleus (principal, or otherwise) in cases of experiment where the dependent is kept in variation nucleus is as outlined in Algorithm 2. In the variation nucleus, the first and the last element in the nucleus are the head and dependent of the non-projective edge respectively.

5.5 Evaluation of Experiments on Variation Nuclei

We summarise the counts of all the unique variation nuclei[6] generated per language in Table 5.7.

Lang. Code	**# Non Proj. Edges**	**# Variation Nuclei**
`grc`	29 882	4 669
`la`	20 170	2 532
`ko`	8 821	4 360
`fro`	3 712	1 291
`orv`	3 961	653
`nl`	2 753	489
`hi`	2 680	831
`cs`	3 827	749

Table 5.7: Counts of Unique Variation Nuclei in Experimental Dataset

We did not define on the constraints to determine the consensus earlier in Section 5.4.2. We define two rules to determine consensus here. We only inspect

[6]The longest subsequence is considered. If *(A,B,C,D)* is the first variation nucleus, and *(A,B,C)* is second variation nucleus that were queried for, the smaller sequence, even if generated from a different source tree, is not included in the counts.

Algorithm 2 pattern_query()

```
Input: Tree $T$ to perform query on, Variation Nucleus as a list $V$
if $T.id$ not in $V.source$ then
    {Make sure both head and dependent are in tree, and are linked to each other}
    $head \leftarrow$ first element in $V$
    $dependent \leftarrow$ last element in $V$
    $gapNucleus \leftarrow$ remaining elements in $V$
    if $head.lemma$ in $T$ then
        if $dependent.lemma$ in $T$ and $dependent.parent == head$ then
            {Start iterating through nodes in gap nucleus now}
            $fromNode = head.ord$
            $toNode = dependent.ord$
            if $fromNode > toNode$ then
                swap($fromNode, toNode$)
            end if
            for all $x$ in $gapNucleus$ do
                if $x.lemma$ not in $T$ or not ($fromNode < x.ord < toNode$) then
                    return
                end if
            end for
            {Program didn't terminate, all nodes in gap nucleus present}
            if $dependent$ attached projectively to $head$ then
                $globalCounter[V][projective] += 1$
                $globalList[V][projectiveIDs].append(T.id)$
            else
                $globalCounter[V][nonprojective] += 1$
                $globalList[V][nonprojectiveIDs].append(T.id)$
            end if
        end if
    end if
end if
```

the edges that satisfy both the conditions. In the event that either condition is not satisfied, we discard the edge from our consideration.

1. A given variation nucleus might show up in multiple trees since it is being queried for across the treebank(s). A given variation nucleus might appear just 10 times across the queried treebanks though, for example. In this case, we cannot deduce anything concrete about the (non-)projectivity of the edge. In a data driven approach, we can only talk about patterns that manifest themselves multiple times. For our experiment, we set the minimum threshold at 0.005% of the treebank size threshold. Since we chose treebanks that had at least 10,000 trees, we set the lower limit at 50. In essence, a variation nucleus should appear in at least 50 trees in the queried treebank(s) before it can be inspected.

2. In an ideal scenario, there are edges that are always associated projectively, or non-projectively. In this scenario, our algorithm should identify 0 trees

where the edge is marked non-projective, implying there is never a projective manner of association for the edge, and vice-versa. However, it is likely that we will have instances that show up as both projective, and non-projective. If we go by the crude definition of majority consensus, a ratio of $\frac{50.1}{49.9}$ is also technically a consensus. However, this is not a strong majority, and so we need to define a stricter constraint to be able to reach a definitive consensus. Intuitively, an edge that shows up 90% of the time as either of projective/non-projective and only 10% of the time otherwise is likely to be an error, regardless of the rarity of non-projective structures in the language. Using the same intuition, we define the majority consensus to be 9:1. This essentially means that we inspect an edge iff it shows up in either category at least 90% of the time.

5.6 Results and Conclusion

The total number of variation nuclei that were retrieved for each language when the dependent was kept are shown in Table 5.8. The *Retrieved VN* refers to the count of nuclei that were found in other trees when queried for.

Language	**# VN Queried**	**Retrieved VN**
`grc`	4 669	243
`la`	2 532	188
`ko`	4 360	2
`fro`	1 291	5
`orv`	653	21
`nl`	489	37
`hi`	831	44
`cs`	749	37

Table 5.8: Query Results of Variation Nuclei with dependent not removed
VN = Variation Nucleus/Nuclei

In our experiment, we found the constraints to be very restrictive. Of all the retrieved variation nuclei, we could find just two nuclei (1 in `cs` and 1 in `la`), that were able to satisfy all constraints. However, the edges found by the nuclei were found to be wrongly annotated as non-projective. We also found 2 instances of ideal scenario (with both constraints satisfied), where the variation nucleus was associated only with non-projective edges throughout the queried pool of data.

Since the approach didn't work for any of the 10 languages, we inspected the cause of low retrieval of variation nuclei for the generated queries. In case of `ko` data, the lemma data contains plus (+) symbol to annotate the fusion of different lemma. Since it is possible for different tokens to fuse together, the algorithm could not handle such case, and thus the low retrieval for variation nuclei in this case. In case of `fro`, the data does not contain any lemmas and so lowercase forms were used instead. Considering inflectional morphology of the language, the algorithm could not with simply the forms, and so the retrieval was extremely low.

6. Experiment 3: `conj_head`

As discussed in Section 2.3.1, 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 6.1, allowing us to define our effective dataset in Section 6.2. We elaborate on our proposed solution to the problem, and the explanation of the algorithm used in the experiment in Section 6.3 and 6.4 respectively. We finally evaluate our experiment in Section 6.5.

6.1 Observations About 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 POS 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 POS tag (cf. Section 9.3). We take a look at the different issues associated with the problem that makes it a difficult one to solve in following subsections.

6.1.1 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 putrah paramadurmedhasah vasushakti ugrashaktih* ***anantashaktishca*** *iti babhuvuh .*
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 (*v*). The semicolon sign delimits the different examples, even when written together.

Example 9.
Text (he): התפוח והכלב; ורד; הייתי בטיול ובגדול נהנתי
Translit: *htfvch* ***vhchlv****;* ***vrd****; T vtvl* ***vvgdvl*** *nhnt*;
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.

6.1.2 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*, making *A, B* part of the example as asyndetic.
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.

6.1.3 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)

The first two examples cannot be distinguished in their dependency trees within the current annotation framework of UD. Only the last example can be distinguished from the first two. This poses a problem by introducing ambiguity into the semantic structure of the sentence as in case of shared dependents[2].

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.

6.1.4 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 following example from `en`-lines treebank, and the corresponding dependency tree in Figure 6.1. The token of interest is written in bold.

Example 12. They can move the fields that are displayed in the row, column, **and** data area of the PivotTable list, or add or remove fields from the list.

Notice that conjunction *and* should be correctly linked to *data*, but is rather attached non-projectively to *add*, since the token is marked as `conj`. In the example above, 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`
[2]`https://universaldependencies.org/u/dep/conj.html#conj-conjunct`

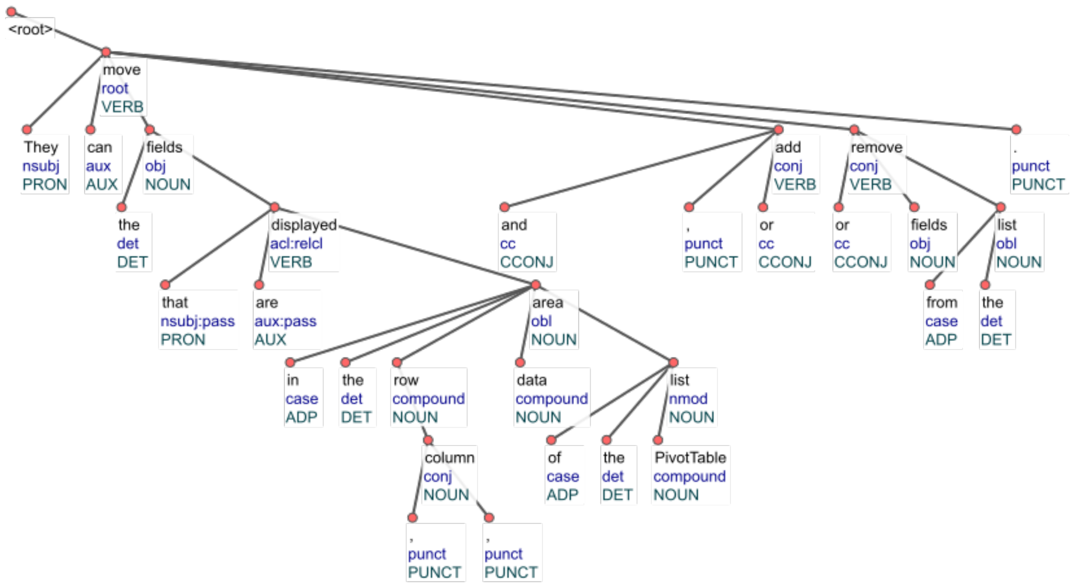


Figure 6.1: Possible Case of Conjunction Sandwich

6.2 Dataset

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.

6.3 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.

In CONLL-U format (cf. Appendix A.1.1), the tokens are ordered top-down. This essentially means that regardless of whether the language is LTR or RTL, the tokens in both ordered in same manner in CONLL-U format. Thus, the algorithm for identification, and correction of the direction of attached dependency needs not be modified for either of the considered languages, owing to their RTL or LTR manner of orthography. Table 6.1 lists the counts of instances identified as attached in the wrong direction, in comparison to the total number of instances.

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

Table 6.1: Counts of Coordinating Conjunctions attached in wrong direction

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 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 6.2, and 6.3. The token D is the technical root of the sentence, or a node in the tree of such that nodes A, B and C belong to subtree of node D. Node B is the conjunction token of interest, with node C being the correct conjunct it should be attached to. Association with node A in the shown (sub-)trees is the faulty association that needs to be corrected. Note that any other possible attachments to either of node A, node C or node D are not shown in the figures. However, that does not rule out the possibility of each of nodes A, C and D being heads of subtrees.

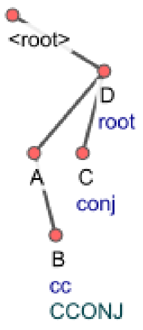
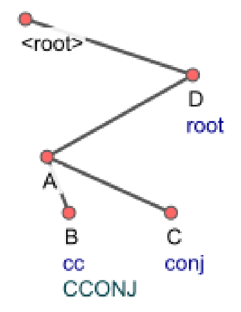
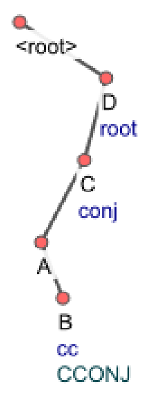


Figure 6.2: Possible Wrong Attachments of a Coordinating Conjunction: Correct conjunct as parent's sibling



(a) Correct conjunct as sibling



(b) Correct conjunct as grandparent

Figure 6.3: Possible Wrong Attachments of a Coordinating Conjunction

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 6.2 first, failing which we try to solve it with respect to the case as in Figure 6.3a, and eventually as in Figure 6.3b. 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.

6.4 Algorithm

We start with defining some wrapper functions in Algorithm 3 and 4. 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 attached directly to the technical root of the tree, thereby making sure there is just one root node at any instance.

Algorithm 3 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
```

Algorithm 4 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
```

Having defined our wrapper functions, we start by trying to attach the conjunction at the same level to other siblings in Algorithm 5. 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 5 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 6. 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 6 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 7.

Algorithm 7 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 8 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
```

6.5 Evaluation and Results

We implement the algorithm in form of a Udapi-python [Popel et al., 2017] block[3]. The runtime of the block for the data is as mentioned in Table 6.2, 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 6.2: 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 evaluation was done manually by the native speakers of the language. While evaluating, correct head association of the conjunction, and the direction of dependency were the two factors taken into account. The scores can be seen in Table 6.3.

Language	Corrected Instances
`af`	95
`ar`	97

Table 6.3: Results of Experiment on `conj_head`, evaluated with 100 random samples

With the defined algorithm in previous subsection, we were able to correct a significant number of flagged error cases, just by identifying the direction of dependency. 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 6.4.

[3]Code alongwith manually annotated data available at `https://github.com/Akshayanti/conj_head.git`

grc	PROIEL	Perseus
PROIEL	75.84	32.02
Perseus	43.23	70.54

grc	PROIEL	Perseus
PROIEL	77.86	32.14
Perseus	43.55	70.70

Table 6.4: LAS, Before, and 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.

7. Experiment 4: LISCA and Cold Start Problem

We discussed about LISCA briefly in Section 3.1.2. Therein, we also mentioned how in an ideal case, the test and train data should belong to the same register. We also discussed on the problem of cold-start, and we briefly mentioned how we sought to use *k*-fold Cross Validation as a workaround.

In the following subsections, we shall elaborate on the experiment with LISCA. The main objective of the experiment is to seek a workaround for the cold-start problem.

7.1 Dataset

The experiment was conducted entirely on data from UDv2.4[1]. In particular, the experiment was conducted and is reported on the `hi`-HDTB treebank. The motivation behind the limiting of the dataset to a particular language's was threefold. Firstly, the treebank in question is limited to news genre. The lack of variability in the genre satisfies the initial condition for the LISCA algorithm viz. the data belonging to the same genre/register. Secondly, the treebank has a medium size as can be seen in Table 7.1, with a total of 16,000+ sentences. This essentially means that while trying for the different values of *k* in *k*-fold cross validation, larger values could be experimented with. The checks with the different values of *k* was also considered important to determine the optimal value of k, if any. Lastly, the author has `hi` as their native language, making it easier for them to analyse the given data, thus reducing the source of ambiguity during the manual annotation process.

Split	Sentences
dev	1 659
test	1 684
train	13 304
Total	**16 647**

Table 7.1: Size of `hi`-HDTB treebank

7.2 Data Preparation for k-fold Cross Validation

We start the process by concatenating the different splits of the treebank into a single treebank. The resultant treebank contains 16,647 sentences. Next, we downsample this data to 16,000 sentences. This downsampled data becomes our functional dataset for the experiment. The downsampling is needed to allow

[1]Code alongwith manually annotated data is available at `https://github.com/Akshayanti/lisca_coldstart`

for the different values of k to work. While the data if downsampled to 16,640 instances would have also worked, we chose to set the count to 16,000 sentences for empirical reasons.

The experiments were conducted on 3 different values of k. The chosen values were $k = \{2, 4, 8\}$. When the values of $k \geq 10$ were considered, the resulting data folds became smaller enough to not yield satisfactory results. Having made the choice of k values, the effective dataset was then split into folds corresponding to the k value.

For each value of k, there were total of k runs that were made. In each run, one of the folds was designated as a test fold, and the remaining $k - 1$ folds were designated as the training fold such that every fold had been designated as a test fold exactly once. For each run, there is exactly one testing fold and one training fold, labelled as run data. The LISCA algorithm for each run data was run by Dell'Orletta et al. separately, wherein the training fold was used to train the algorithm, and then the arcs in test fold given scores as per the trained model for the run data. The results for the different runs of a given k value were then analysed. Algorithm 9 summarises the procedure involved so far.

Algorithm 9 Data Preparation for k-fold Cross Validation

```
Input: Downsampled hi-HDTB Treebank $T$
 1: for all $k$ in {2, 4, 8} do
 2:     $T.folds \leftarrow \{T.1, ..., T.k\}$ subject to conditions:
 3:     $T = \bigcup\{T.1, ..., T.k\}$ {Condition 1}
 4:     $\bigcap\{T.x1, T.x2\} = \phi \ \forall\{T.x1, T.x2\} \in T.folds$ {Condition 2}
 5:     for $run$ in 1, ..., k do
 6:         $fold.test \leftarrow T.run$
 7:         $fold.training \leftarrow T - T.run$
 8:         $lisca.run \leftarrow$ trained LISCA on $fold.training$
 9:         Rank arcs in $fold.test$ using $lisca.run$
10:     end for
11: end for
```

7.3 Analysis of LISCA Scores

The evaluation of a trained LISCA model on a given test data generates several types of statistics. As mentioned earlier, the output data was split into 10 equal bins in descending order of arc scores, with an additional bin for the remnants in Alzetta et al. [2017]. The generated data from LISCA shows a number of statistics pertaining to each of the bins. Some of the statistics include POS distribution, deprel distribution, POS and deprel distribution, syntactic link length distribution, among others. In addition to these statistics, the generated data also marks each arc in the test data with a confidence score. While the statistics are a useful feature, the cross validation process renders it unhelpful for the analysis of the entire data. Having said that, we focused on the confidence scores for each arc, as generated by different runs for a given value of k.

k-value	Absolute Minimum	Absolute Maximum	0-score arcs	Total arcs
2	0.00	1.96 E-07	3 487	336 054
4	0.00	1.93 E-07	2 620	336 083
8	0.00	1.91 E-07	2 319	366 087

Table 7.2: Statistics for Arc Scores

Table 7.2 shows the maximum and minimum confidence scores allotted for any arc for individual k values. As the number of folds increase, the overall maximum arc score decreases. Also, the number of arcs with a 0-score decrease in count. For analysis, we combine the data from different runs for a given k value, and then analyse them. We do not pay attention to the non-zero arc scores, but limit our focus to the zero-valued scores. The idea behind this analysis is that the lowest scores being 0, all the 0-scored arcs will occupy the 9th, 10th or the 11th bins, as per the original binning procedure. Note that the most cases of erroneous instances were found to be concentrated in the aforementioned bins. This is used as basis for the further analysis of the data.

For the analysis, we concentrated on 0-scored arcs as mentioned before. As evident from the data, the number of 0-scored arcs was the minimal for $k = 8$. As the count of 0-scored arcs increased with a decreasing k value, the 0-scored arcs discovered for a higher k value were present for the lower k value as well. When the k value was decreased from $k = 8$ to $k = 4$, all the 0-scored arcs found in the former were also present in the latter. Similarly, the arcs found in case of $k = 2$ was an addition to the arcs found when $k = 4$, and as stated above, when $k = 8$. As such, the lower the k-value, the more different kinds of error the algorithm is able to pick up. Having noted this, we analysed the effect of each k-value in the following manner. For the 0-scored arcs that were common to $k = \{2, 4, 8\}$, 200 randomly chosen arcs (out of 2319) were evaluated manually. Out of the arcs local to $k = \{2, 4\}$, 100 were randomly chosen for manual evaluation. Finally, out of the remaining arcs local to $k = 2$, another 100 were randomly chosen for manual evaluation. Table 7.3 shows the typology of errors that were discovered during the manual annotation process.

Typology of Error	$k = \{2, 4, 8\}$	$k = \{2, 4\}$	$k = 2$
acl4amod	1	0	2
amod4xcomp	5	2	2
auxAsHead	0	0	2
nmod4obl	3	0	5
obl4advcl|acl	1	1	1
advmod4amod	1	1	0
nsubj4obl	3	3	1
obl4xcomp	3	1	4
xcomp4advmod	0	0	1
Wrong Head	13	2	4
Tree Error	23	1	3
Random Error	32	10	10
No Error	115	79	65
Total Errors	85 (42.5 %)	21 (21 %)	35 (35 %)

Table 7.3: Error Typologies for different folds in k-fold cross validation

7.4 Typologies of Errors

Of the typologies of errors as mentioned in Table 7.3, the ones indicated in bold are the errors that have been previously pointed out by Alzetta et al. [2017]. Of the other discovered error types, none except `nsubj4obl` occur in significant counts. As such, we discuss briefly about the following typologies in this section: `nsubj4obl`, Wrong Head, Tree Error.

7.4.1 `nsubj4obl`

Oblique Dependent marked as Nominal Subject: This error type refers to the cases where the oblique nominal dependent of a nominal subject is incorrectly marked as nominal subject. This is a relatively obscure error pattern, but nonetheless the pattern is exhibited across the data. In the example given below, *Bhartiya Janta Party* is an oblique dependent of *Trinamool Congress*, but has been marked wrongly as the nominal subject, as can be seen in Figure 7.1

Example 13.
Text (`hi`): भारतीय जनता पार्टी से दूरी बनाते हुए तृणमूल कांग्रेस ने ...
Translit: *bhartiya janata party se doori banaate hue Trinamool Congress ne ...*
Lit.: Bhartiya Janta Party from-*Dat.* dissociating Trinamool Congress *Nom. ...*
Translated: Dissociating itself from Bhartiya Janta Party, Trinamool Congress ...

7.4.2 Wrong Head

While Alzetta et al. mention head labelling error as a sub-type of the error pattern(s), we distinguish the error type from others. In our case, we use the label to identify the cases when the associated deprel is wrong because of the selection of wrong head. The error type refers to the case where the selection

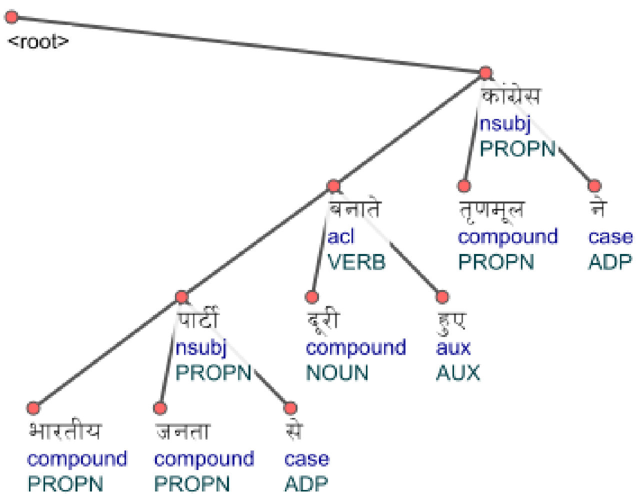


Figure 7.1: Dependency Tree for Example 13 Showing nsubj4obl Error Type.

of wrong head is detrimental in choice of a deprel, and where the case of deprel assignment is suspected on the basis of the relational head. This is an important distinction because there exist significant number of parsers that assign more weight to the relationships, and decide on the associated deprels accordingly.

7.4.3 Tree Error

This error type can be considered as an extreme case of 'Wrong Head' error. In the case of dependency trees, where multiple tokens suffer from 'Wrong Head' error, it is not possible to evaluate the tree owing to disrupted tree structure. The disruptions in the tree structure can be exhibited in form of non-projectivity, multiple deprel errors, multiple head errors, among others. Essentially speaking, a tree marked with 'Tree Error' requires re-annotation before any analysis can be performed on it.

7.5 Results and Discussion

Looking at the data obtained from Table 7.3, we achieve $\approx$ 42% precision in identification of errors when we select $k = 8$. The precision drops to $\approx$ 35% when the value of k is lowered to $k = 4$. As the value of k is lowered to $k = 2$, the precision rises again to $\approx$ 40%. While a very high or a very low value of k can be considered optimal, we argue that the size of dataset is the determining factor in selection of value of k as discussed below:

1. For the case of a very low k value, there is a slight drop in precision. This implies that the results of evaluation are likely to contain a lot of random errors, and false negatives. However, for instances where the dataset is not particularly large, a very low k value is more optimal.

2. For the case of a very high k value, there is the problem associated with the size of the dataset. If the dataset is not big enough, the algorithm might not have been trained well enough to avail the best precision. As such, the

choice of a very high k value is optimal for cases when the dataset is large enough to allow more fragmentation.

While there were no significant error patterns that were discovered with this experiment, we tackled an essential problem that often came in the way of using LISCA. Originally, the authors of the algorithm proposed that there is another gold-standard that is needed for training LISCA, in order to correct a silver-standard. However, we show that there is no need for another gold-standard, and that the algorithm can be trained on the silver-standard data itself in case of a missing gold-standard. This is essential since for low-resource languages where the errors need to be mined, and corrected; there is more often than not, a lack of gold-standard data. The algorithm can still be used for such cases, by selecting a low/high value of k for k-fold cross validation procedure.

8. Negative Experiment: `AUX` vs. `VERB`

We discussed in brief the problem of separating instances labelled `AUX` and `VERB` in Section 2.4.1 earlier. We shall treat this problem in this section, with a glance through some of the observations on the problem in Section 8.1, followed by the definition of the working dataset in Section 8.2. We elaborate on the proposed solution to the problem, and the results of the experiment in Section 8.3 and 8.4 respectively. We finally conclude this section with a discussion of the results in 8.5.

8.1 Observations About 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. 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 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

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

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.

8.2 Dataset Definition

This experiment was initially tried on `hi`-hdtb treebank from UDv2.3, but failed terribly. With the release of UDv2.4, this experiment was tried again, keeping the dataset treebank same, but changing the model architecture et al.

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.

8.3 Experiment

We approach the problem at hand as a classification problem, specifically as a Sequence Labelling based NER problem. In the experiment, we create a tagger that tags the data in either of the three categories as `AUX`, `VERB` or neither of the two. Since the training data needs to contain the information on what instances to mark in either category and also differentiate tokens not marked as either UPOS tag, we label the data using NER tag format. Given the training data and the lack of a platinum standard, we would be evaluating the tagger on the training data itself using k-fold cross validation. During the predictions, we also get confidence score associated with each predicted tag. By analysing the confidence score of each prediction and comparing it with the already annotated data, we should be able to point out the anomalies. As part of this measure, the first step in the experiment requires that 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.

[3]Code alongwith manually annotated data is available at `https://github.com/Akshayanti/AUX-vs-VERB-UDv2.4`

To convert our data into IOBES 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 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.

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 be 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 8.1. The best choice for the hyper-parameter are 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 8.1: 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 the model on each fold of the data. The trained model instances are then used to predict the test set in each fold as well. As mentioned earlier, the output of an evaluation writes the predictions with the associated confidence in the predicted label. We here identify 6 categories of error patterns, based on the predicted label and the original label for the data, as listed in Table 8.2.

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
aux-verb	S-aux S-verb	S-verb S-aux
verb-O	O S-verb	S-verb O

Table 8.2: 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 8.1 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.

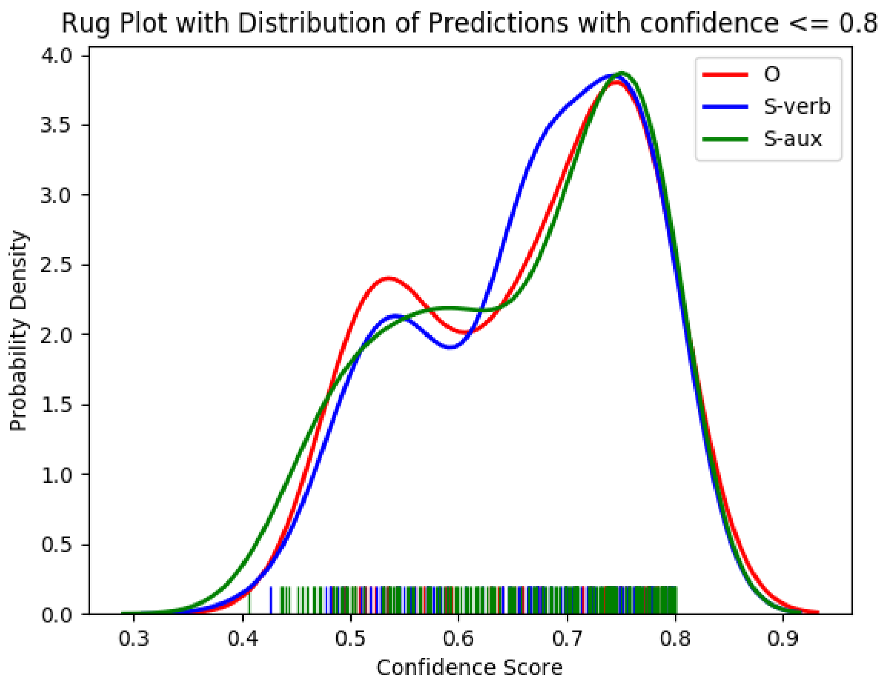


Figure 8.1: Rug plot with Distribution of Predictions with low confidence score

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 10.

Algorithm 10 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, ... , 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 $identified.cross$
```

8.4 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 8.3. 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).

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

Table 8.3: Metrics of Best Model trained over original `hi` data

As mentioned in previous section, we focused on the instances of the tagged data with confidence scores in particular bounds. Table 8.4 lists the number of instances that were focused on in each category (as defined in Table 8.2). The table also lists the number of instances that were identified as mislabelled, following the annotation procedure.

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 8.4: Results of Manual Annotation

8.5 Discussion of the Results

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

Table 8.5: Statistics for `hi` data

Table 8.5 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 identified instances, just a bit over 25% are actually erroneous. While certain patterns are more reliable than others (the case where predicted labels don't match the annotated labels), the recall for the experiment is very low. Owing to such low recall, the current approach at tagging the data to identify the erroneous instances is not reliable enough for the process to be automated.

9. 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.

9.1 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, cf. Section 9.2), their full potential might not have been realized so far.

In our experiment on `conj_head` (cf. Chapter 6), we did not work with the problem of conjunction sandwiches. It is very likely that such constructions which are difficult to be 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 or discarded when Enhanced Dependencies are used.

9.2 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 discussed earlier, 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

[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`

problem that needs to be solved within the scope of basic dependency graphs in UD.

9.3 Function Words and Associated deprels

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) and the associated tree in Figure 9.1, where `PART` acts as a conjunction, and thus the `mark` deprel associated to it.

Example 14. Ukraine's constitutional structure is for Ukraine's citizens alone to decide.

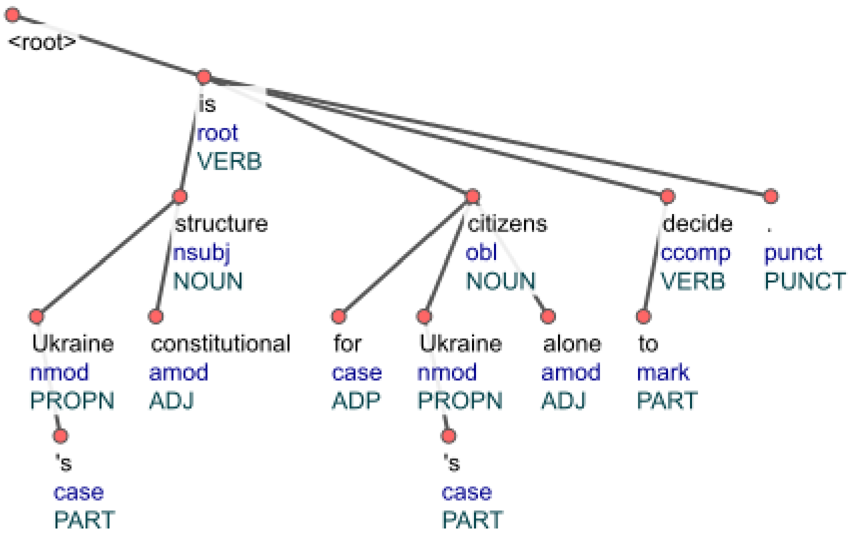


Figure 9.1: Tree for Example 14 showing association of `PART` with `mark` deprel

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 might still be some processes we can observe (and correct). For example, if a 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.

9.4 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:

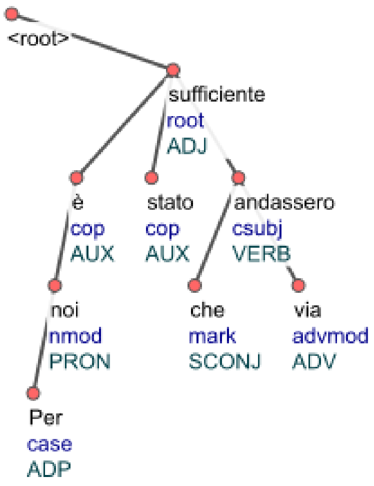
1. The definition of `AUX` was extended to include copula verbs, and non-verbal TAME (Time, Aspect, Mood, Evidentiality. Might/might not include Voice and Polarity) particles.

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

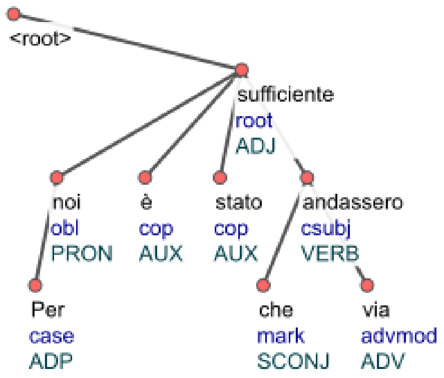
3. The relation `auxpass` was removed from UDv2.0, making it as a subcategory of the larger `aux` relation, in the form of `aux:pass`. Essentially speaking, `auxpass` was demoted to a sub-category of `aux` relation.

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 9.2, taken directly from Alzetta et al. [2017][4].

Example 15.
Text (it): Per noi è stato sufficiente che andassero via
Lit.: For us it-has been enough that they-went away



(a) **O**



(b) **C**

Figure 9.2: Example tree 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,

[4]In the original example, **noi** (us) was annotated as a dependent of both **Per** (for), and **è** (has-been). Under UD representation, there can not be more than one head for any given node. As such, we believe it was an error in the paper and not in their data. In this case, we show the corrected dependencies.

classified as a head identification error, contributes to around 13 % of the total discovered erroneous instances. Since it is difficult to separate and identify the instances marked correctly as `AUX` (cf. Chapter 8 for the experiment on attempt at differentiation between `AUX` and `VERB` tags), the attempt at the solution for this problem was not worked at.

9.5 `nmod4obl`: Confusion of `nmod` and `obl` Relations

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:

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

This essentially meant that there were two changes that had to be made, while the data was being modified to fit UDv2 guidelines. In case a relation was to be changed from $\texttt{nmod}_{v1}$ to `obl`, there were often 2 changes that needed to be made.

1. The token's parent had to be changed, owing to the difference in definition of the deprel `nmod` in UDv2 and UDv1.
2. The token's deprel had to be changed from `nmod` to `obl`. Although a seemingly simple change, the difficulty in the correct implementation stems from the lack of automatic rules to differentiate between erroneous attachments of the deprels involved.

The above two errors often occur together, and are 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.

9.6 Punctuation

The UD Annotation guidelines on punctuation are simple and straightforward[5]. There are discrepancies when it comes to implementation of the guidelines. Some

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

of them are listed as below:

1. It is difficult to identify the next conjunct in case of missing `CCONJ` and `SCONJ` tags as in case of asyndetic conjunctions (cf. Section 6.1.2). In such cases, the information about the next conjunct should be deduced semantically in most cases. We saw a similar case in Section 6.1.4 where the next conjunct is not clear, owing to other (more suitable) deprel(s) being used in place of `conj` deprel.

2. Re-attachment of a punctuation node 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.

9.7 Unspecified Dependencies - `dep` deprel

According to the UD definition of `dep` deprel[6], 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.

[6]`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`[7] 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` in Chapter 6. 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` (cf. Section 9.5). 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.

[7]`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.

Riyaz Bhat and Dipti Sharma. Non-Projective Structures in Indian Language Treebanks. *Edições Colibri*, 2012.

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.

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.

Carlos Gómez-Rodríguez, David Weir, and John Carroll. Parsing mildly non-projective dependency structures. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, pages 291–299. Association for Computational Linguistics, 2009.

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

Keith Hall and Václav Novák. Corrective modeling for non-projective dependency parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology*, Parsing '05, page 42–52, USA, 2005. Association for Computational Linguistics.

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.

Prashanth Mannem, Himani Chaudhry, and Akshar Bharati. Insights into non-projectivity in hindi. In *Proceedings of the ACL-IJCNLP 2009 student research workshop*, pages 10–17. Association for Computational Linguistics, 2009.

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. Constraints on Non-Projective Dependency Parsing. In *11th Conference of the European Chapter of the Association for Computational Linguistics*, 2006.

Joakim Nivre. Incremental non-projective dependency parsing. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 396–403, 2007.

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 and Jens Nilsson. Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 99–106. Association for Computational Linguistics, 2005.

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 Žabokrtský, 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 Zdeněk Žabokrtský. 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.

Zdenka Uresova, Eva Fucikova, and Jan Hajic. Non-projectivity and valency. In *Proceedings of the Workshop on Discontinuous Structures in Natural Language Processing*, pages 12–21, San Diego, California, June 2016. Association for Computational Linguistics. doi: 10.18653/v1/W16-0902. URL `https://www.aclweb.org/anthology/W16-0902`.

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`.

Himanshu Yadav, Ashwini Vaidya, and Samar Husain. Understanding constraints on non-projectivity using novel measures. In *Proceedings of the Fourth International Conference on Dependency Linguistics (Depling 2017)*, pages 276–286, Pisa,Italy, September 2017. Linköping University Electronic Press. URL `https://www.aclweb.org/anthology/W17-6531`.

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, 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, 2014a. ISSN 1574-020X.

Daniel Zeman, David Mareček, Jan Mašek, Martin Popel, Loganathan Ramasamy, Rudolf Rosa, Jan Štěpánek, and Zdeněk Žabokrtský. HamleDT 2.0, 2014b. 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.

Daniel Zeman, Joakim Nivre, Mitchell Abrams, Noëmi Aepli, 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, Colin Batchelor, 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, Tatiana Cavalcanti, Gülşen Cebiroğlu Eryiğit, Flavio Massimiliano Cecchini, Giuseppe G. A. Celano, Slavomír Čéplö, Savas Cetin, Fabricio Chalub, Jinho Choi, Yongseok Cho, Jayeol Chun, Alessandra T. Cignarella, Silvie Cinková, Aurélie Collomb, Çağrı Çöltekin, Miriam Connor, Marine Courtin, Elizabeth Davidson, Marie-Catherine de Marneffe, Valeria de Paiva, Elvis de Souza, 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, Olga Erina, Tomaž Erjavec, Aline Etienne, Wograine Evelyn, 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, Bernadeta Griciūtė, Matias Grioni, Normunds Grūzītis, Bruno Guillaume, Céline Guillot-Barbance, Nizar Habash, Jan Hajič, Jan Hajič jr., Mika Hämäläinen, 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, Markus Juutinen, Hüner Kaşıkara, Andre Kaasen, Nadezhda Kabaeva, Sylvain Kahane, Hiroshi Kanayama, Jenna Kanerva, Boris Katz, Tolga Kayadelen, Jessica Kenney, Václava Kettnerová, Jesse Kirchner, Elena Klementieva, 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, Kyung-Tae Lim, Maria Liovina, 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, Sarah McGuinness, Gustavo Mendonça, Niko Miekka, Margarita Misirpashayeva, Anna Missilä, Cătălin Mititelu, Maria Mitrofan, Yusuke Miyao, Simonetta Montemagni, Amir More, Laura Moreno Romero, Keiko Sophie Mori, Tomohiko Morioka, Shinsuke Mori, Shigeki Moro, Bjartur Mortensen, Bohdan Moskalevskyi, Kadri Muischnek, Robert Munro, 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, Atul Kr. Ojha, 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, Jason Phelan, Jussi Piitulainen, Tommi A Pirinen, Emily Pitler, Barbara Plank, Thierry Poibeau, Larisa Ponomareva, Martin Popel, Lauma Pretkalniņa, Sophie Prévost, Prokopis Prokopidis, Adam Przepiórkowski, Tiina Puolakainen, Sampo Pyysalo, Peng Qi, Andriela Rääbis, Alexandre Rademaker, Loganathan Ramasamy, Taraka Rama, Carlos Ramisch, Vinit Ravishankar, Livy Real, Siva Reddy, Georg Rehm, Ivan Riabov, 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, 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, Aline Silveira, 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, Andrius Utka, Sowmya Vajjala, Daniel van Niekerk, Gertjan van Noord, Viktor Varga, Eric Villemonte de la Clergerie, Veronika Vincze, Lars Wallin, Abigail Walsh, 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, Manying Zhang, and Hanzhi Zhu. Universal dependencies 2.5, 2019. URL `http://hdl.handle.net/11234/1-3105`. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

List of Figures

List of Tables

List of Abbreviations

- **GS**- Gold Standard
- **LAS**- Labelled Attachment Score
- **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].

A.1.1 CONLL-U Format

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).

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

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

(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.

A.1.2 UD Annotation

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 16. 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 UDv2.5. A full list of ISO 639-1 and ISO 639-3 language codes can also be accessed online[3].

Table A.1 indicates languages where the ISO codes (ISO 639-1 or ISO 639-3) is used as an identifier. The only exception is `qhe`, which is reserved for local use

Note:

- * against a language name indicates language not present in UDv2.4.

Code	Language Name
af	Afrikaans
aii	Assyrian
akk	Akkadian
am	Amharic
ar	Arabic
be	Belarusian
bg	Bulgarian
bho	Bhojpuri*
bm	Bambara
br	Breton
bxr	Buryat
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
fro	Old French
ga	Irish
gd	Scottish Gaelic*
gl	Galician
got	Gothic
grc	Ancient Greek
gsw	Swiss German*

Continued on next page

[3]`https://iso639-3.sil.org/code_tables/639/data/all`

Code	Language Name
gun	Mbya Guarani
he	Hebrew
hi	Hindi
hr	Croatian
hu	Hungarian
hsb	Upper Sorbian
hy	Armenian
id	Indonesian
it	Italian*
ja	Japanese
kk	Kazakh
kmr	Kurmanji
ko	Korean
koi	Komi Permyak*
kpv	Komi Zyrian
krl	Karelian
la	Latin
lt	Lithuanian
lv	Latvian
lzh	Classical Chinese
mdf	Moksha*
mr	Marathi
mt	Maltese
myv	Erzya
no	Norwegian
nl	Dutch
olo	Livvi*
orv	Old Russian
pcm	Naija
pl	Polish
pt	Portuguese
ro	Romanian*
ru	Russian
sa	Sanskrit
sk	Slovak
sl	Slovenian
sme	North Sami
sms	Skolt Sami*
sr	Serbian
sv	Swedish
swl	Swedish Sign Language
ta	Tamil
te	Telugu
th	Thai
tl	Tagalog
tr	Turkish

Continued on next page

Code	Language Name
ug	Uyghur
uk	Ukrainian
ur	Urdu
vi	Vietnamese
wbp	Warlpiri
wo	Wolof
yo	Yoruba
yue	Cantonese
zh	Chinese*

Table A.1: Languages in UDv2.5, identified with their ISO Codes

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

Table A.2 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.2: Multiple Treebanks in Different Languages, UDv2.4

A.4 PUD Treebanks

Table A.3 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.3: Languages with PUD Treebanks, UDv2.4

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

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