A Comparative Approach to Query an English Dependency Treebank: SPARQL vs. TüNDRA Web Tool

Anita Guseinova
anita.guseinova@student.uni-tuebingen.de

Fidan Can
fidan.can@student.uni-tuebingen.de

Abstract

This study aims to query the treebank **UD_English−ParTUT**, which is originally in CoNLLU format, from Universal Dependencies (2014-2021) with the query language SPARQL by comparing it to the TüNDRA, which is a web tool for treebank research and stands for Tübingen aNnotated Data Retrieval Application (Martens, 2013). It is based on TIGERSearch query, and contains the aforementioned treebank. To be able to reach the same results as this web tool outputs, we, as the researchers, first converted the treebank dataset with **CoNLLU-RDF** (Chiarcos and Fäth, 2017) to Turtle format (.ttl) to use SPARQL on it. After the conversion, some more changes were done in the resulting .ttl file with Python so that the SPARQL queries can also return the sentences where the words we are interested are available. Later on, to have a comparative approach, we ran queries on **UD_English−ParTUT** on TüNDRA web tool, and aimed to figure out the corresponding SPARQL queries. Overall, it was seen that while some certain queries on TüNDRA were relatively more easily done, the corresponding SPARQL queries of them were more challenging. The paper will give a much deeper insight into all the steps taken along with the necessary clarifications regarding the terminologies, their rationales and roles in our research in the following sections.

1 Introduction

Before delving into the methodologies and tools we use, it can be ideal to provide a background so that the readers can follow along the project in a more comprehensive way; and thus, it seems necessary to introduce some terminologies. With the separate subsections, the fundamental concepts can be seen as follows:

1.1 Universal Dependencies

It is a project for grammar annotation such as part of speech and syntactic dependencies for treebanks, and it is open to contribution; thus, there are 300 contributors producing approximately 200 treebanks in over 100 different languages. UD aims to facilitate language learning and the development of multi-lingual parsing, and its philosophy lies in creating universal categories in order that it can give mutual guidelines across languages when it also provides adjustments specific to a certain language structure. (Universal Dependencies , 2014-2021).

To briefly touch upon the basic structure of UD Treebanks, de Marneffe et al. (2021) state in their article that a clause includes a main predicate expressing an action or a state, and often, there are nominals included in those actions or states, from which it can be deduced that languages follow a hierarchical structure. To make it clearer, clauses can be made of modifiers, nominals, and clauses; as for nominals, they can be made of the aforementioned three phrasal units; and we can also see that modifiers can include modifiers.

To represent this hierarchical structure in UD, dependency grammar view is taken, which means that a linguistic structure has a head and other elements contained are the dependents of this head. Such a relation between head and the other linguistic items are shown by arrows going from the head to the dependents (de Marneffe et al., 2021)

1.2 CoNLLU

CONLL-U is a type of format that is followed to annotate data at sentence and at word/token level. Annotations in this format have the following features: Word lines include the annotations of a token or say a word in 10 different fields which are separated by a single tab.
Comment lines begin with the hash symbol, #.

Each of those 10 fields (ID, FORM, LEMMA, UPOS, XPOS, FEATS, HEAD, DEPREL, DEPS, MISC) stands for a different attribute of the word/-token. Some of the fields are as follows:
LEMMA: Root or stem of the word.
ID: It stands for the word/token index in a sentence.
FEATS: It represents the morphological characteristics such as singularity/plurality of a noun (CoNLL-U Format , 2020).
Our dependency treebank, UD_English-ParTUT was also initially in CoNLLU format.

1.3 TüNDRA Web Tool

After giving a small introduction to CoNLLU format, it is a good opportunity for us to give the readers some background on the web tool for treebank research, TüNDRA, as this web tool makes up one of the most important parts of our project to create corresponding SPARQL queries based on those of TüNDRA tool.

To begin with, Martens (2013) states that TüNDRA is an acronym for Tübingen aNnotated Data Retrieval Application and an application for searching treebanks, which are regarded as corpora with linguistic annotations to show the relationships between the different parts of the linguistic elements for research studies. This web tool is partly based on the TIGERSearch, a search engine to extract information from a database which is in a graph structure (TIGERSearch , 2011-2020). "The query language of TIGERSearch incorporates elements ... A restricted kind of attribute-value structures are available" (TIGERSearch , 2011-2020). TüNDRA also supports querying dependency treebanks in addition to constituency trees which are in a hierarchical form.

As our treebank, *UD_English-ParTUT*, is a dependency treebank in CoNLLU format, it was quite suitable to use this tool to query the treebank after getting introduction/tutorial to the syntax of TüNDRA web tool's query syntax based on TIGERSearch. The tutorial to how to query a treebank can be found in `https://weblicht.sfs.uni-tuebingen.de/Tundra/tutorial`. It is also worth stating that our treebank, which we got from Universal Dependencies and then converted to .ttl for SPARQL queries, was already available on this web application.

In addition, TüNDRA outputs the results of the queries in a table format and also shows the context, namely the sentences where the linguistic items we are interested in appear. Visualization is also provided after running a query. In the case of dependency treebank, one can see the certain element/-s in context, and they can be seen as highlighted in the visualization of dependency tree structure (e.g., arrows going from the head to the dependents) with their relation to the other elements.

As it was mentioned earlier, TIGERSearh query language which the TüNDRA is based on provide a syntax structure for attributes, and values being written in square brackets: a very basic example is as "[attribute = value]" (TIGERSearch , 2011-2020) Further, it is possible to use regular expressions, and specific to this query language, there are some special operators as ".", "!", "<" and "&" (Martens, 2013).

Since we will be already explaining our queries in the section,"Comparison of Queries", it may not be convenient to repeat them here. For more information on the query syntax, you can refer to the TüNDRA Tutorial available on `https://weblicht.sfs.uni-tuebingen.de/Tundra/tutorial` (Martens, 2013).

1.4 RDF & Turtle

Due to the limitation of the paper length and the scope of the project, we will try to briefly mention the relationship between RDF and Turtle as well.

According to RDF 1.1 Turtle (2014) RDF stands for Resource Description Framework, and it is a "labeled graph data format for representing information in the web" (SPARQL Query Language for RDF, 2013). It is most of the time used to represent social networks, personal information in addition to providing a way of integration over separate information resources (SPARQL Query Language for RDF, 2013).

Related to RDF, Turtle, which stands for **Terse RDF Triple Language**, is a textual syntax for RDF among other syntaxes such as JSON-LD . Turtle is compatible with N-Triples format in addition to the triple syntax of SPARQL. In short, it can be said that Turtle is a representation of RDF graphs. That is, a Turtle document/textual syntax lets us to write an RDF graph.

Moreover, RDF 1.1 Turtle (2014) states that an RDF graph is comprised of triples which are made of subject, predicate and object respectively, and a basic triple structure can be visualised as a sequence of these triples, which are separated by a

white space. To have a closer look at RDF triples for an RDF graph representation, it is seen that:
the **subject** can be an IRI (Internationalized Resource Identifier) or a blank node (RDF does not make a reference to the internal structure of blank nodes).
the **predicate** is an IRI.
the **object** can be an IRI, literal (e.g. strings, numbers, dates, etc.) or a blank node.

As it will be related to our project and the queries, it is also worth mentioning that the subjects often have more than one predicate: a sequence of predicates and objects, separated by ';', following a subject. This represents a series of RDF Triples with that specific subject, and each predicate and object allocated to one triple (RDF 1.1 Turtle , 2014)

1.5 SPARQL for RDF

SPARQL, standing for **SPARQL Protocol and RDF Query Language** "allows users to query information from databases that can be mapped to RDF" (What is SPARQL? , 2022). That is, it is an RDF query language.

SPARQL enables users to navigate relationships in RDF graph data through graph pattern matching, and these relationships can be discovered through some basic matching, joins, unions and so on (What is SPARQL? , 2022). Note that in the previous section, it was clarified that RDF graphs are made of triples which are **subject**, **predicate** and **object**.

We aim to explain some of the basics of SPARQL query syntax here to give a bit of insight to the readers; however, for those interested in learning about it more can refer to SPARQL Query Language for RDF or What is SPARQL? .

As SPARQL is a query language for RDF graphs, its syntax is based on triples. Feigenbaum (2009) defines its basic structure which can be seen as follows:
URIs (Uniform Resource Identifier) represent the resources (for the subject and predicate), and they can be shortened by defining prefixes (represented with **PREFIX** in SPARQL) while objects can be literals such as numbers and strings. Feigenbaum (2009) also touches upon what a SPARQL query is made of, respectively by describing 4 main categories:

1) Prefixes to abbreviate URIs:
PREFIX foo: <**http://example.com/resources/**>

2) An outputting clause which expresses what kind of information we want to return from the query: **SELECT**

3) The condition pattern which expresses what we want to query: **WHERE { ... }**.

4) Modifiers to modify the output: **ORDER BY** can be given as an example.

Variables in SPARQL stand for the triples we want to query on, and "?" is put in front of the variable names (e.g. ?subj, ?pred, ?obj).

One of the important query clauses in SPARQL for this paper which is worth mentioning is **FILTER** as it will later on appear when we compare the SPARQL and TüNDRA Web Tool queries (TIGERSearch):

FILTER restricts the query output to those which the filter expression evaluates to TRUE for (SPARQL Query Language for RDF, 2013). For instance, one can make use of regular expressions to define a specific word pattern in the FILTER statement.

The 3rd section of the paper will give a clearer look into SPARQL queries, and as mentioned earlier, readers are encouraged to refer to the resources suggested above or look at the references to get more information on SPARQL as it seems not convenient to include a whole query language structure here.

2 Methodology

2.1 Treebank

For the purpose of our study, we used the dependency treebank, UD_English-ParTUT , which we first downloaded from Universal Dependencies to convert it to .ttl file format. UD_English-ParTUT is a treebank that was developed at the University of Turin. It includes different kinds of texts such as talks and Wikipedia articles in addition to other kinds as legal texts (UD_English-ParTUT , n.d.)

One of the main reasons why we chose this treebank is that it is available both on Universal Dependencies and on TüNDRA (Martens, 2013). That is a very important criterion for us: as we were not able to download treebanks from TüNDRA, a treebank which is available on both platforms was found to be a way to go so that we could later on compare the SPARQL queries with those of TüNDRA web tool. Therefore, UD_English-ParTUT was accessed through Universal Depen-

dencies . Another reason is that as both researchers have different native languages, a treebank in English was preferred over others.

2.1.1 Conversion from CoNLLU to Turtle

Having had the treebanks in hand, we ended up having the representation of the UD_English-ParTUT treebank in the CoNLLU format. Additionally, those files were split into three different .conllu files by default: train, dev and test. First, these three files were merged into one .conllu file as we did not need training, development and test sets.On the contrary, we needed the whole data in one file so that we could then compare it with the same treebank available on TüNDRA (Martens, 2013). After putting these three files into one, we used CoNLLU-RDF rendering tool (Chiarcos and Fäth, 2017) to convert the .conllu file to .ttl file to work with SPARQL. The resulting file is called *en_partut.ttl* in our GitHub repository, which is accessible via `https://github.com/tnitn/SPARQL-project` (Can and Guseinova, 2022) Conversion was not the only thing needed to work on this .ttl file. There was one more change made which will be explained in the following section.

2.1.2 Adjustments to the Turtle File

TüNDRA (Martens, 2013) allows users to query a treebank, and it does not only output the queried elements in a table view with its attributes but also shows them in a sentence context where they belong to. To be able to achieve the same goal: namely, being able to return sentences with SPARQL, a Python program was written, which is the file *turtle_changer.py* in our GitHub repository: please see Can and Guseinova (2022).

Normally, the file, *en_partut.ttl* does not provide a direct reference from a word token to the sentence/-s it belongs to; for that reason, *turtle_changer.py* was created. This python code adds to every subject (subject in terms of triples), which contains an object nif:Word or an object nif:Sentence, another pair of a predicate-object, namely "conll:SENT" which were retrieved from a rdfs:comment predicate.

To be more precise, the function *def create_mod_ttl(graph, sentences, dest_name = "trial.ttl"), in* turtle_changer.py handles the aforementioned case and gives us the ideal .ttl file format we aimed to end up with. As a result of this adjustment to the .ttl file, *en_partut.ttl*, we could achieve to have the wanted .ttl format: the resulting file is now called *en_partut_adapted.ttl* and can be again accessed on our Github repository which also includes *turtle_changer.py* (Can and Guseinova, 2022).

It is important to note that this file can be used for any .ttl file generated by the aforementioned conllu-rdf tool by Chiarcos and Fäth (2017).

2.2 Steps into the SPARQL Queries

Having had the final version of the .ttl format for our dependency treebank, we needed to run SPARQL queries on it. The way we chose our queries was an important step as we aimed to find the corresponding SPARQL queries of the Tündra queries which can be found on the tabs "Tutorial" or "Query Help" of the TüNDRA Web Tool (see Martens (2013)).

In total, we have written 12 queries for TIGERSearch to run on TüNDRA Web Tool and 12 SPARQL corresponding queries to run on Apache Jena Fuseki . The following section, "Comparison of Queries" will let the readers have a closer look in them.

Moreover, to create templates for our SPARQL queries with the possibility to change the exact instance searched, we wrote another Python program, *sparql_queries.py*. To see those templates and for more information regarding the code , please refer to this file in our GitHub repository, `https://github.com/tnitn/SPARQL-project` (Can and Guseinova, 2022) as all the comments for the written Python methods are available in this file.

3 Comparison of Queries

Query 1: Lemma

Retrieves **all the sentences** that contain the lemma "I". This is a specific example to get the lemma "I". Any other lemma can also be queried with the same syntax.

TIGERSearch - 1

[lemma = "I"]

SPARQL - 1

Since we are trying to retrieve the sentences with the LEMMA "I", we return *?sent* in the SELECT statement by putting the condition for the LEMMA in the WHERE syntax.

```
PREFIX conll: <http://ufal.mff.c
↪ uni.cz/conll2009-st/task-des
↪ cription.html#>

SELECT ?sent
WHERE {
  ?s conll:LEMMA "I";
     conll:SENT ?sent .
}
```

Query 2: Words containing a certain pattern

Retrieves **all the sentences** that contain word(s) finishing with "able". Any other sentence including a word starting with, ending with or containing any sequences of characters can also be queried with the same syntax after some simple modifications done with regular expressions. Note that in RegEx, **.** stands for any character and * stands for 0 or more times of those characters.

TIGERSearch - 2

[word = /.*able/]

SPARQL - 2

Note that the dollar symbol in ".*able$" stands for the end of a line (which in case of *?word* contains of the word only). Different from the previous query, we use the FILTER syntax to use a regular expression to specify the wanted word which is comprised of a specific sequence.

```
PREFIX conll: <http://ufal.mff.c
↪ uni.cz/conll2009-st/task-des
↪ cription.html#>

SELECT ?sent
WHERE {
  ?s conll:WORD ?word;
     conll:SENT ?sent .
  FILTER regex(?word, ".*able$")
}
```

Query 3: Word1 OR Word2

Retrieves **all the sentences** that contain either the word "food", or word "drink" (or both). With the same syntax, any other words can be queried rather than these two.

TIGERSearch - 3

Note the regular expression for grouping: "()". Also note that "|" stands for expressing "or".

[word = ("food" |"drink")]

SPARQL - 3

A very important syntax appearing here different from the previous SPARQL queries is the UNION syntax, which is used to combine graph patterns in order that any of the alternatives may match.

```
PREFIX conll: <http://ufal.mff.c
↪ uni.cz/conll2009-st/task-des
↪ cription.html#>

SELECT ?sent1 ?sent2
WHERE {
  {?s conll:WORD "food";
      conll:SENT ?sent1 . }
  UNION
  {?s conll:WORD "drink";
      conll:SENT ?sent2 .}
}
```

Query 4: POS & LEMMA

Retrieves **all the sentences** that contain word(s), part of speech of which is NOUN and lemma of which starts with "un". Any other POS and lemma starting with a specific sequence of characters can be queried with the same syntax.

TIGERSearch - 4

It is possible to query more than one attribute in a single square bracket with the "&" operator.

[pos = "NOUN" & lemma = /un.*/]

SPARQL - 4

Note that the caret symbol in "^un.*" stands for the beginning of a line (which in case of *?lemma* contains the lemma only).

```
PREFIX conll: <http://ufal.mff.c
↪ uni.cz/conll2009-st/task-des
↪ cription.html#>

SELECT ?sent
WHERE {
  ?s conll:POS_COARSE "NOUN";
     conll:LEMMA ?lemma;
     conll:SENT ?sent .
  FILTER regex(?lemma, "^un.*")
}
```

Query 5: Adjacent Words: Word1 AND Word2

Retrieves **all the sentences** that contain word "the" followed by the word "whole": that is, the sentences where "whole" comes **right after** the word

”the”. Any other sentence with specified adjacent words can be queried with the same syntax.

TIGERSearch - 5

Note the ”.” in between the square brackets. It can be interpreted as ”and”, and it is used to show the adjacency between the words being queried.

[word = “the”] . [word = “whole”]

SPARQL - 5

Note that word boundary in RegEx is symbolized with \b. Also note that *regex* in FILTER requires doubling all the backslash characters.

```
PREFIX conll: <http://ufal.mff.c⌋
 ↪  uni.cz/conll2009-st/task-des⌋
 ↪  cription.html#>
PREFIX nif: <http://persistence.⌋
 ↪  uni-leipzig.org/nlp2rdf/onto⌋
 ↪  logies/nif-core#>

SELECT ?sent
WHERE {
  ?s a nif:Sentence;
     conll:SENT ?sent
  FILTER (regex(?sent, "\\bthe
   ↪  whole\\b"))
}
```

Query 6: Word1, Another Word, Word2

Retrieves **all the sentences** that contain the word “the”, one word in between, and the word “world”. Any other sentence including a word between 2 specified words can also be queried - this is a specific example we have queried.

TIGERSearch - 6

In the previous query, it was clarified that ”.” in between the square brackets are for querying the adjacent words; however, if we are interested in a word sequence which is not necessarily adjacent, we put a distance between the words: see ”2” after ”.” which means **only one word in between** like in ”the whole world”.

[word = ”the”] .2 [word = ”world”]

SPARQL - 6

Note that in the FILTER line, the regular expression includes \\w+ this time when compared to the 5th query. This regular expression means ”any word character” and we need ”+” to say that there should be at least 1 word character and a space in between the two words.

```
PREFIX conll: <http://ufal.mff.c⌋
 ↪  uni.cz/conll2009-st/task-des⌋
 ↪  cription.html#>
PREFIX nif: <http://persistence.⌋
 ↪  uni-leipzig.org/nlp2rdf/onto⌋
 ↪  logies/nif-core#>

SELECT ?sent
WHERE {
  ?s a nif:Sentence;
     conll:SENT ?sent
  FILTER (regex(?sent, "\\bthe
   ↪  \\w+ world\\b"))
}
```

Query 7: Word1, Several Other Words, Word2

Retrieves **all the sentences** that contain the word “he”, one or two words in between, and the word “to”. Any other sentence including any specific number of words between 2 specified words can also be queried - this is a specific example we have queried.

TIGERSearch - 7

Comparing to the previous query, we now have a range of words that can be in between the two specified words, and ”.2,3” means **one or two words in between** like ”he managed to” or ”he managed not to”.

[word = ”he”] .2,3 [word = ”to”]

SPARQL - 7

Here RegEx is used again, and the number of words in between, written like ”\w+ ”, must be either 1 or 2: ”{1,2}”.

```
PREFIX conll: <http://ufal.mff.c⌋
 ↪  uni.cz/conll2009-st/task-des⌋
 ↪  cription.html#>
PREFIX nif: <http://persistence.⌋
 ↪  uni-leipzig.org/nlp2rdf/onto⌋
 ↪  logies/nif-core#>

SELECT ?sent
WHERE {
  ?s a nif:Sentence;
     conll:SENT ?sent
```

```
  FILTER (regex(?sent,
   ↪  "\\bhe\\b(\\w+
   ↪  ){1,2}to\\b"))
}
```

Query 8: Word1, Unspecified Number of Other Words, Word2

Retrieves **all the sentences** that contain the word "he", unspecified number of words in between, and the word "to". Any other sentence with two words being by an unspecified number of other words can be queried with the same syntax.

TIGERSearch - 8

RegEx asterisk sign, "*" comes in handy here and is used to allow any number of words in between given two.

[word = "he"] .* [word = "to"]

SPARQL - 8

In this query we are only interested in "he" and "to" being two complete words with any characters to be allowed to be in between them.

```
PREFIX conll: <http://ufal.mff.c⌋
 ↪  uni.cz/conll2009-st/task-des⌋
 ↪  cription.html#>
PREFIX nif: <http://persistence.⌋
 ↪  uni-leipzig.org/nlp2rdf/onto⌋
 ↪  logies/nif-core#>

SELECT ?sent
WHERE {
  ?s a nif:Sentence;
     conll:SENT ?sent
  FILTER (regex(?sent,
   ↪  "\\bhe\\b.*\\bto\\b"))
}
```

Query 9: POS1, Another Word, POS2

Retrieves **all the sentences** that contain the word a word with a pos "DET" followed by a word with a pos "PROPN". Any other two words with specific POSes, or even a word with specific POS followed by a specific lemma, can be queried the similar way after simple modifications in both languages.

TIGERSearch - 9

[pos = "DET"] . [pos = "PROPN"]

SPARQL - 9

In this query we make use of an ID field in order to find all the POSes that stand one after another, and filter out only the ones that are from the same sentence.

```
PREFIX conll: <http://ufal.mff.c⌋
 ↪  uni.cz/conll2009-st/task-des⌋
 ↪  cription.html#>
PREFIX xsd: <http://www.w3.org/2⌋
 ↪  001/XMLSchema#>

SELECT ?sent1
WHERE {
  ?s1 conll:POS_COARSE "DET";
        conll:ID ?id1;
        conll:SENT ?sent1.
  ?s2 conll:POS_COARSE "PROPN";
        conll:ID ?id2;
        conll:SENT ?sent2.
  FILTER (xsd:integer(?id1)+1 =
   ↪  xsd:integer(?id2) &&
   ↪  ?sent1 = ?sent2)
}
```

Query 10: Word2 as the Head of Word1

Retrieves **all the sentences** that contain the word "we", the head of which is the word ""see". Any other head-relation queries could be written in their place - this is a specific example we have queried.

TIGERSearch - 10

Sign ">" stands for head relations in TIGERSearch.

[word = "see"] > [word = "we"]

SPARQL - 10

SPARQL syntax makes the query quite straightforward, as the subject of the given "head" is reused to find all the words having this subject as the object of the predicate "conll:HEAD".

```
PREFIX conll: <http://ufal.mff.c⌋
 ↪  uni.cz/conll2009-st/task-des⌋
 ↪  cription.html#>

SELECT ?sent
WHERE {
  ?s conll:WORD "see" .
  ?s1 conll:HEAD ?s;
      conll:WORD "we";
      conll:SENT ?sent
}
```

Query 11: POS2 as the Head of Word1 with Edge

Retrieves **all the sentences** that contain the word ”whole”, the head of which is the word with a pos ”NOUN” and the edge ”amod”. Any other head-relation queries could be written in their place - this is a specific example we have queried.

TIGERSearch - 11

Add the edge name near ”>” to search for a specific edge in TIGERSearch.

[pos = ”NOUN”] >amod [word = ”whole”]

SPARQL - 11

The query is very similar to the previous one, the only difference is the restrictions added on the possible edge.

```
PREFIX conll: <http://ufal.mff.c⌋
 ↪  uni.cz/conll2009-st/task-des⌋
 ↪  cription.html#>

SELECT ?sent
WHERE {
  ?s conll:POS_COARSE "NOUN" .
  ?s1 conll:HEAD ?s;
         conll:WORD "whole";
         conll:EDGE "amod";
         conll:SENT ?sent
}
```

Query 12: Negation Example

Retrieves **all the sentences** that contain a word, a pos of which is **not** “adp”, the head of which is a word with a pos “propn”.

TIGERSearch - 12

Add the ”!” symbol for negation in front of the greater than symbol, ”>”

[pos=”PROPN”] !> [pos=”ADP”]

SPARQL - 12

This query is similar to two previous ones. The difference here is that FILTER without ReGex is used here to filter out the part of speech that is to be avoided.

```
PREFIX conll: <http://ufal.mff.c⌋
 ↪  uni.cz/conll2009-st/task-des⌋
 ↪  cription.html#>

SELECT ?sent
WHERE {
  ?s conll:POS_COARSE "PROPN";
     conll:SENT ?sent .
  ?s1 conll:HEAD ?s;
      conll:POS_COARSE ?pos;
  FILTER (?pos != "ADP")
}
```

4 Conclusion

Overall, it was seen that reproducing TüNDRA queries in SPARQL to query a dependency treebank became possible with the necessary adjustments done to the CoNLLU files so as to end up with a Turtle file consisting of triples which were required to run SPARQL queries. Therefore, in this aspect it can be said that this project proved convenience and achieved its goal: that is, being able to query a treebank with SPARQL by comparing it to the corresponding TIGERSearch query, which Tündra Web Tool is based on, to output the same results. This also gave us the opportunity to see how these two different query syntaxes work.

TüNDRA Web Tool seems to have different kinds of advantages in terms of its query syntax and the way it outputs the results. The TIGERSearch query that this web tool is based on seemed pretty clear and less complicated to us when compared to SPARQL. The tutorial that TüNDRA offers (Martens, 2013) made it very easy for us to learn how to query the treebanks there since the the structure of queries are not intricate. Though we say that it was easier syntax-wise compared to SPARQL, some may still disagree if they are more experienced with SPARQL.

As for the way how TüNDRA Web Tool outputs the query results, it can be said that it is remarkably clear: For instance, it provides a table for each word in a sentence with their attributes and the statistics of the query output. One can also click on ”show all attributes” to see all the attributes of a token. In addition to this, it also outputs another table to show the sentence contexts where the tokens in interest are included. Therefore, it is pretty useful to have a better understanding of what has been queried. As for the last point regarding the output representation, users can see the visual output of the treebanks: in our case, it would be the visual representation of a dependency treebank where the head and dependents are connected to each other with arrows by showing the relationship between

them. The queried item is highlighted on the visual representation of the tree, as well. In short, TüNDRA provides users with different types of output, which makes the query search more comprehensible. For the SPARQL queries, Apache Jena Fuseki (2011–2022) was used, and when it was compared to TüNDRA, it was seen that visualization of the output is not available.

4.1 Limitations

This research study was not without challenges and limitations. First of all, finding a treebank which is available both on TüNDRA (Martens, 2013) and Universal Dependencies (Universal Dependencies , 2014-2021) was a bit challenging as it was not possible to download treebanks from TüNDRA so that we could later on convert it to .ttl format. Therefore, alternatively we had to look through mutual treebanks, and then downloaded one from Universal Dependencies .

Secondly, converting the dependency treebank from .conllu format to .ttl format was not enough on its own to work later with SPARQL. To clarify, as TüNDRA Web tool outputs also the sentences which contain specific items being queried, we wanted to achieve the same goal with SPARQL; therefore, the file *turtle_changer.py* (see Can and Guseinova (2022)) was created so that the words could refer to the sentences they belong to.

Third, as opposed to what TüNDRA offers, we did not get a visual representation of our SPARQL queries on Apache Jena Fuseki . If we had gotten it, it could have facilitated the comprehensive understanding of our treebank better.

4.2 Further Research

A further research study can be based on the visualization of SPARQL queries, and another comparative study can be conducted as it was not in the scope of this project. That is, we suggest that a tool for query output visualization can also be developed to make the representation of SPARQL query results more efficient, and that tool can be compared to what Apache Jena Fuseki offers.

References

Apache Jena Fuseki . 2011–2022. Apache jena fuseki.

Fidan Can and Anita Guseinova. 2022. Sparql-project.

Christian Chiarcos and Christian Fäth. 2017. Conll-rdf: Linked corpora done in an nlp-friendly way. *Universal Dependencies*, pages 74–88.

CoNLL-U Format . 2020. Conll-u format.

Lee Feigenbaum. 2009. SPARQL By Example.

Marie-Catherine de Marneffe, Christopher D. Manning, Joakim Nivre, and Daniel Zeman. 2021. Universal dependencies. *Computational Linguistics*, pages 255–308.

Scott Martens. 2013. TüNDRA: A Web Application for Treebank Search and Visualization. In *In: Proceedings of The Twelfth Workshop on Treebanks and Linguistic Theories (TLT12)*, pages 133–144, Sofia.

RDF 1.1 Turtle . 2014. Rdf 1.1 turtle.

SPARQL Query Language for RDF. 2013. Sparql query language for rdf.

TIGERSearch . 2011-2020. Tigersearch.

UD_English-ParTUT . n.d. Ud_english-partut.

Universal Dependencies . 2014-2021. Universal dependencies.

What is SPARQL? . 2022. What is sparql?