Visualization as an aid to Text Processing

Nikola Miroslavov Nikolov

4th Year Project Report
Computer Science
School of Informatics
University of Edinburgh

2016

Abstract

Computational tools for carrying out text processing and linguistic analysis can benefit enormously from good visualisation support. This applies both to novice and experienced users. However, there is relatively little established practice in building appropriate tools for visualisation, especially compared to other scientific domains, although there is a wealth of visual metaphors for depicting linguistic structure.(Taken from the project description)

Table of Contents

Chapter 1

Introduction

The general idea behind this report is to explain the thought process instigating a system intended to help people, through visualisation to study and research dependency parsing. Chapter 1 will talk about the motives behind building a tool that uses visualisation to aid learners and researchers understand dependency parsing, why we have chosen dependency parsing as the focus of this system and a brief overview of the system itself. Chapter 2 will provide the necessary background knowledge and Chapter 3 will justify the design decisions and the constrains of the technologies used.

1.1 Why is visualisation important

There is almost an universal agreement that visualisation can immensely aid learning and understanding.

> *The power of the unaided mind is highly overrated. Without external aids, memory, thought, and reasoning are all constrained. But human intelligence is highly flexible and adaptive, superb at inventing procedures and objects that overcome its own limits. The real powers come from devising external aids that enhance cognitive abilities. How have we increased memory, thought, and reasoning? By the invention of external aids: It is things that make us smart.*[11, Norman 1993, Chapter 3]

Visualisation can help us see things from a different perspective and unravel mysteries which were not obvious to us before. There are many studies that show concrete evidence that our brains process information quicker and with a greater ease when we can use visualisation as a media of communicating information. For instance a study was done on university students involving multiplication of large numbers with and without pencil and paper. The results show a staggering five-fold decrease in the amount of time the students needed to do the computations when they had something to write with.[6] The main reason is not that the computations were hard, but holding the partial results in memory can be confusing. This idea is true not only for computations, but for natural language processing too. When presented with a problem that

requires a long, step by step process to resolve, it is only natural to use visual aid to ease our thought process. Having said that we will adopt this idea when tackling the conceptually challenging dependency parsing.

1.2 Why Dependency Parsing

This section will focus only on the reasons behind choosing dependency parsing as the main focus of our tool, rather than going into details about what dependency parsing is - Section 2.4 will talk about the fundamentals behind it. There are two main reasons for choosing dependency parsing in particular - the technique is gaining popularity in machine translation[9], speech recognition and information extraction [8]. One of the main reasons for this is that dependency grammars can much better accommodate free or flexible word order than phrase structure based grammars making it much batter at dealing larger amount of topologically diverse languages. [12]. Another reason is that there have been a sizeable advancement in the automatic dependency parsers using machine learning and annotated corpora. However dependency parsing is a conceptually hard and somewhat uncharted area when it comes to innovative learning environments.

Natural language processing is already armed with an abundance of tools for text visualisation in web browser environments.[3] However there are limited to almost no such systems targeted specifically to explain and support dependency parsing. Given that we are presented with the unique opportunity to explore the limits of web-browser environments combined with the Natural Language Tool Kit (NLTK) to produce an inventive visualisation tool for this approach to parsing.

1.3 The system

The intent of this project is to create a web-based system which through visualisation can aid people in understanding or researching dependency parsing. The motivation behind the idea is no other than the fact that there is indeed a lack of such systems, and a system of this sort can prove invaluable to the teaching process. Virtual learning environments can greatly decrease the amount of research and exercises needed to be done by the students when learning about dependency parsing. Although the project does not meet all of the criteria to be considered as a functioning virtual learning environment as such environments are vast and complex in the sheer amount of content presented to the learner and the included features, it can moreover be regarded as a first step, or a single module of a system of that type. Chapter 3 will focus entirely on how the system was build and the evolution of the project from the initial idea to the end results achieved.

Chapter 2

Background

2.1 Syntactic Structures

There are two main views of syntactic structures that are used when parsing natural language text using a statistical model - constituent and dependency structures.[7] The constituent structure (or also more widely recognised as phrase structure) revolves around combining words into constituents which ultimately form the whole sentence like in 2.1.

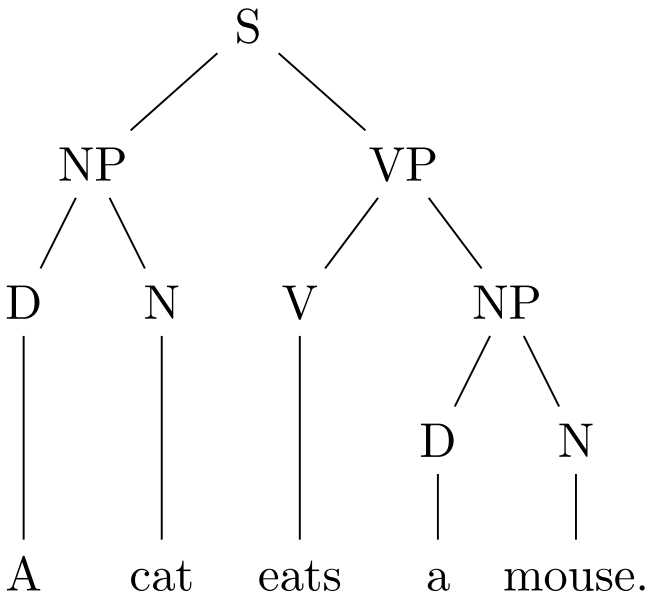

Figure 2.1: Constituency based parse tree of the sentence ”A cat eats a mouse” [4]

In syntactic analysis we usually refer to a group of words to be a constituent if they can serve as a single unit in a hierarchical structure. Generally this type of syntactic structure is found more suitable for languages with a fixed word order since there is a much more clearer constituency structure. Dependency structures, on the other hand, aim to show which word depends on which other word in the sentence and in what way.

A word is considered to be a dependant of another word if it modifies or is an argument of that word. Even though both structures represent a hierarchical ordering of the elements in the sentence, unlike the constituency structure, the dependency structure is concerned with the relations between the elements in the sentence rather than the grouping of words forming units of higher order.[10]
Since the work reported here focuses mainly on dependency structures we well not be looking further in constituency structures. However the ideas and practices behind dependency structures and dependency parsing will be furthermore explored in the next several sections.

2.2 Dependency Syntax and Grammar

The focus of this section will be to introduce terminology connected with dependency syntactic structures. The majority of the terminology and practices mentioned in this section should be considered as the most widely and commonly used, and not the only way of representation and understanding. Lucien Tesnire wrote in 1959: *"Every word in a sentence is not isolated as it is in the dictionary. The mind perceives connections between a word and its neighbours. The totality of these connections forms the scaffold of the sentence. These connections are not indicated by anything, but it is absolutely crucial that they be perceived by the mind; without them the sentence would not be intelligible."*[13] In his theory of syntax, Lucien Tesnire, talks about connections between words. Those connections are nowadays more commonly known as *dependencies*. This widely spread perception of the connections as dependencies forges the term Dependency Grammar.
The general idea behind dependency syntax is to depict connections between words using binary asymmetric relations (*"arrows"*) to show if they are modifiers or arguments of other words. The common convention is to use *"typed"* arrows - write on the arrows what is the grammatical relation between the words. These arrows originate from the head (governor or superior) and connect to a dependent (modifier or inferior). Ordinarily dependencies form trees of acyclic manner. The finite verb is regarded as the center of the sentence and all of the other words are either directly or indirectly dependent on it. In the parse tree in Figure 2.2 we can observe that the words *"Bills"*, *"were"* and *"by"* are inferior to the verb *"submitted"* thus the arrows commence from the verb and point at its dependants.To enhance the visual perception of the tree a widely spread practice is to add one pseudo node at the beginning of the tree often named *"ROOT"* which points at the head of the sentence - in this case the verb *"submitted"*.

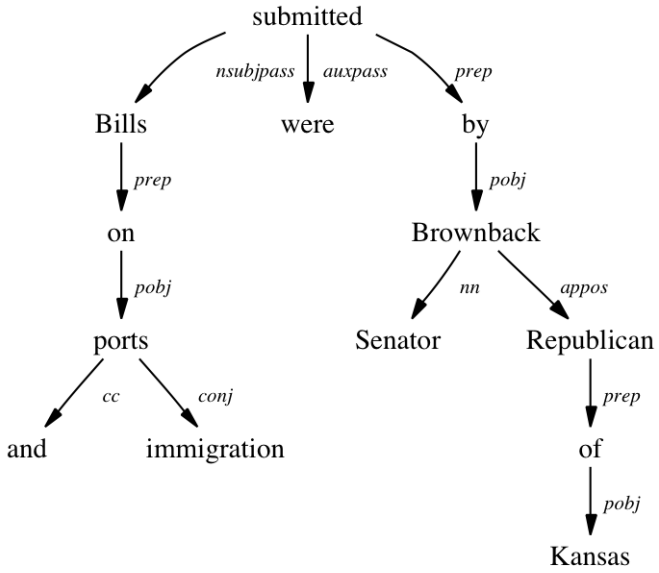

Figure 2.2: Tree representation of the dependency connections in the sentence " Bills on ports and immigration were submitted by Senator Brownback, Republican of Kansas" [1]

2.3 Dependency Graphs

Even though a hierarchical structure of the trees can be used for representing dependencies, a more flattened approach can provide greater insight to the untrained eye as it is further intuitive. In this report we will be using dependency graphs as the main form of illustration. Dependency graphs are directed graphs depicting dependencies between the nodes. The nodes in these graphs are the words themselves and the word ordering is kept. These graphs can take the form of trees like Figure 2.3, however for the purpose of this report we will be using only their flattened counterparts.

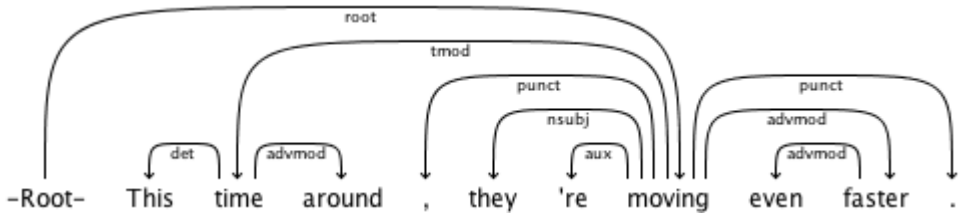

Figure 2.3: Dependency parsing of the sentence "This time around, they're moving even faster." [2]

Dependency graph G obeys several formal conditions:

1. G is **connected** - for every node i there is a node j such that $i \rightarrow j$ or $j \rightarrow i$
2. G is **acyclic** - if $i \rightarrow j$ then not $j \rightarrow^* i$, where $\rightarrow^*$ means *"path"*

3. G obeys the **single-head constraint** - if $i \rightarrow j$ then not $k \rightarrow j$ for any $k \neq i$
4. G is **projective** - if $i \rightarrow j$ then $i \rightarrow^* k$ for any k that $i < k < j$ or $j < k < i$

For a full derivation look at [5, Page 47].

When using the format used in Figure 2.3 for longer sentences the projectivity of the graphs might not be obvious but since the aim is to preserve the linear ordering of the words. However if the words are re-ordered this property it can clearly be seen that this rule is kept at all times.

2.4 Dependency parsing

Chapter 3

The System so far

3.1 The Server

Since we are creating a web-based environment there will be several main topics to be covered, first of which is going to be the server. Since the project revolves around natural language processing and will be highly associated with the Natural Language Tool Kit, the platform chosen for the server side of the system is *Flask*. *Flask* is a micro-framework(a micro-framework is a framework that has little to no dependence on external libraries) for Python that is ideal for the quick build up and easy support of small to medium sized web applications. It takes advantage of a template language - certain features can be directly transferred from one web-page to another easily without the need for repetition of code. The server is ran locally and at the moment it cannot be accessed online.

Chapter 4

Suggested Timeline and current Progress

This chapter is meant to explain my current progress. The things included in here can be reworked and included in the final report.

4.1 Flask and D3.js

So far we have tried two different ways of visualising the dependency graphs - with a normal parse trees showing clear hierarchical structure and a more conventional way of drawing dependency trees - the flat structure from 2.3. Even though we have tried two different layouts for the trees, we have not changed they library used to draw them - D3.js. D3 provides incredible flexibility to what can be drawn on the SVG pane and it can provide a huge variety of possible future implementations of features.

The parse tree variant, that was later on scrapped, worked in a completely different manner than the dependency graph alternative.

Here I will give example of the main differences between the two, which later on in the final report will be extended and discussed in a larger detail.

- The first system intended to take as an input a sentence typed by the user. It was able to take the input from the user, however this presented a rather hard problem - we needed a reliable dependency parser that would parse the said sentence and then return the information in a format we would be able to create the dependency tree from.

  At this point we decided to adopt a different way of doing things - since this was going to be mainly a system used by students to learn dependency parsing we could only provide visualisation for pre-set sentences. This would give little to no customization by the user, but we would be able to tailor the system perfectly to the examples and possibly add more feature explaining in detail the reasons for the connections between the words and the grammar rules used to derive

these connections. Even though this approach might have made a good basis for people to learn about dependency parsing, the lack of robustness made it look more like a detailed guide rather than a responsive system that adapted to the particular needs of the user.

As a contrast the second version of the system did not take a user input sentence, but rather a .txt file that contained a CoNLL grammar for an already parsed sentence. Since there is an abundance of already parsed sentences, containing a large variety of different syntactic structures and lengths, included in the Treebank database in NLTK this approach would present the user with a vast variety of possible sentences that can be visualised. This means that every user can choose to look at an example that he/she is unsure of and needs more clarification. This is exactly the flexibility that our first implementation lacked and can be considered as the right step to the final goal of the project.

- The dependency graph class in Python's NLTK is a vital part of our second implementation, however we did not take advantage of it during the early version of the project. This class provides information about the sentence, extracted directly from a ConLL structure. It provides an easier way of manipulating the graphs themselves and the extraction of information about the types of connections and word tags is simpler, and more importantly already done, completely eliminating the necessity of a parser. The dependency graph object is constructed from a ConLL structure, justifying why we require the users to upload a file containing ConLL grammar. The files are uploaded, parsed and then removed from the server for security reasons. At the moment we are constructing the sentence straight from the grammar, and we are not removing the punctuation which can make the visualisation a bit more overcrowded. Even though punctuation can make a difference in the parsing, a possible feature can be included to get rid of the punctuation in the sentence and change the numeration of the nodes in the dependency graph object.

- As mentioned before the early version of the system used a tree structure for the visualisation. Even though it might be sufficient enough in explaining what and why was happening, the convention for dependency parsing is to use flat, linear structure and this would make the users less confused in the long run. The notion of hierarchy is lost in the process, however the preservation of word order is much more natural to the untrained eye. This approach also makes it a lot easier to see if the parse is projective or non-projective. A tree is projective if the arrows do not cross when the words are in linear order, each word and its dependants form a contiguous substring.

- Just as a note, Bootstrap was added to the website, to better the visual appeal of the system and to give it a navigation bar for easier maneuvering. Bootstrap also brings a nice structure and consistency to the site.

- A feature that will most probably be added next to our existing system is to show the type of connections between words written on the connecting arrows. This is a fairly simple process. The only challenge will be to make ensure that the text is visible and drown on the arrow not in the arrow.

- Colour coding can be added to the arrows, depending on the type of connection they represent. Even though this is not necessary it can aid the learners when understanding the difference between the connection types.
- We can also add tooltips providing additional information on why were the connections made. This can be rather tricky since when constructing the dependency graph we are taking the information from the ConLL, which does not provide information of the sort.

4.2 Suggested Timeline

The last weeks of January and the beginning of February can be left for further improvement of the system, adding new features and improving on the previous ones.

Afterwards, until the end of February I will try to get conceptually familiar with Bookeh and IPython if needed, and think more about evaluation. And March will be entirely write up of the project.

Bibliography

[1] The stanford natural language processing group. http://nlp.stanford.edu/software/stanford-dependencies.shtml.

[2] The stanford natural language processing group, software. http://nlp.stanford.edu/software/.

[3] Text visualisation tools. http://textvis.lnu.se/.

[4] Text visualisation tools. https://en.wikipedia.org/wiki/File:Acat.svg.

[5] Dependency parsing, tutorial at coling-acl, sydney. http://stp.lingfil.uu.se/~nivre/docs/ACLslides.pdf, 2006.

[6] Stuart K Card, Jock D Mackinlay, and Ben Shneiderman. *Readings in information visualization: using vision to think*. Morgan Kaufmann, 1999.

[7] Dan Jurafsky Christopher Manning. Syntactic structure: Constituency vs dependency. https://class.coursera.org/nlp/lecture/161.

[8] Aron Culotta and Jeffrey Sorensen. Dependency tree kernels for relation extraction. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 423. Association for Computational Linguistics, 2004.

[9] Michel Galley and Christopher D Manning. Quadratic-time dependency parsing for machine translation. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 773–781. Association for Computational Linguistics, 2009.

[10] Igor Mel'uk. *Dependency Syntax: Theory and Practice*. State University of New York Press, Albany, first edition, 1988.

[11] Donald A Norman. *Things that make us smart: Defending human attributes in the age of the machine*. Basic Books, 1993.

[12] Dependency Parsing. Synthesis lectures on human language technologies. *Sandra Kubler, Ryan McDonald, Joakim Nivre*.

[13] Lucien Tesnire. *Elemnts de syntaxe structurale*. 2nd edition edition, 1959.