Differences in Dependency Parsing across Languages

Isabel Sharp

May 15, 2017

1 Introduction

Perhaps the most common syntactic parsing of sentences is phrase-structure parsing, which involves breaking down a sentence into sub-constituents that further break down into the parts of speech representing the actual words of the sentence. Dependency parsing is a different parsing method that involves linking words via binary grammatical relations called dependency relations. Both methods of parsing have their advantages, and both can be used to parse any language.

This project examines dependency parsing for several different languages. The main goal of the project is to determine the effectiveness of dependency parsing across languages; in other words, does dependency parsing lend itself to some languages more so than others. In order to evaluate such a question, sample sentences from several different languages are parsed into dependency graphs. These parses are evaluated both for accuracy and efficiency of the parse. Based on these results, some analysis of the parsing algorithm itself will determine the effectiveness of the method for each language, and whether it might be valuable to explore different parsing techniques, or even small modifications to the parsing algorithm, that would lead to improved, language-specific parsers.

2 Dependency Parsing

2.1 Description

On a high level, dependency parsing consists of finding links between words in a sentence. Each pair of linked words are related by a dependency relation. In these relation pairs, one word is a “head”, and the other a “dependent”. In this context, the dependent word is dependent on the head word. One word in the sentence does not depend on any other words; this word is the root of the sentence. Each of these relations is a binary relation, meaning only two words can be represented by a single dependency. Forming a dependency parse of a sentence consists of finding these relations between the words in the sentence.

Somewhat similarly to phrase-structure parsing, dependency parses are often visualized by a graphical representation that shows the relations between the words in the sentence. The similarities to constituent parsing end here, however; dependency parsing does not break the sentence down into sub-constituents. Instead, the individual words in the sentence are related through the dependency relations. The typical graphical representation of a dependency parse shows the words in a sentence, with an arc for each relation. The arc begins at the head word in the pair, and ends at the dependent word. These arcs can be unlabeled, but it is often useful to label the arcs with the name of the relation that allows the two words to be connected. Consider the following example sentence.

(1) I prefer the morning flight through Denver

A typical dependency graph for this sentence would look like figure 1, with the sentence displayed across the page, and the arcs for each dependency drawn above the words, each labeled with the rule that allows the arc to exist. However, it is possible to visualize a dependency parse in a manner more similar to a normal phrase-structure parsing of a sentence. Figure 2 shows this: on the left is the dependency parse of the same sentence as figure 1, and on the right is a phrase-structure parse of the sentence. Both are drawn in a tree structure. Since dependency parsing does not break a sentence into constituents, the trees are not very similar. In a dependency parse visualized this way, the tree illustrates the relative importance of the words in the sentence. The root of the sentence is at the top of the tree. Words higher up in the tree are closer to being the root of the sentence–or, there are less "steps" between them and the root of the sentence. The words at the bottom of the tree branches are complete dependents; no words depend on them in the sentence.

Figure 1: A typical dependency graph for (1)

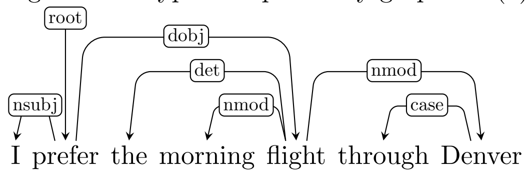
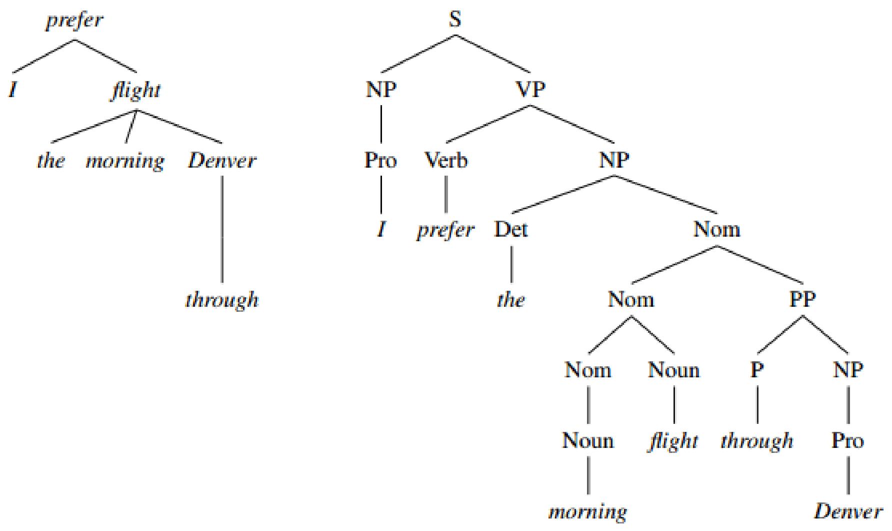


Figure 2: Tree visualizations for dependency and phrase-structure parses of (1)



2.2 Advantages

One of the major advantages to dependency parsing over normal phrase-structure parsing is the proximity of related words in the tree form of the parse. This is illustrated clearly in figure 2. Consider the predicate *prefer*. The arguments to this verb, *I* and *flight*, are directly linked to the word in the dependency parse. In the phrase-structure parse, however, there are several intermediate levels of constituent labels to traverse between the arguments and the verb. Similar behavior is exhibited by modifiers. The words *morning* and *Denver* modify *flight*. In the dependency parse tree, these words are directly below *flight*, which makes this relationship intuitive. In the constituent tree, this relationship is much more difficult to visualize, as there are extra levels of the tree between these related words, obscuring their direct link.

Related to these direct links is the semantic value of dependency parsing. As previously discussed, in a dependency parse tree, the arguments of a predicate are directly below the predicate. This mirrors the semantic relationship between predicates and arguments. While a dependency parse might not exactly match the logical semantic translation of a sentence, it provides a better approximation than most phrase-structure parses. These relationships could be directly extracted to be used for semantic analysis of sentences.

Another advantage to dependency parsing is that it can be used for morphologically complex languages. In this case, instead of finding links between only the words in a sentence, the parser can relate different parts of the word as well; for example, stems and affixes. This of course requires some modification to the parsing algorithm, but is far more simple to do than a similar extension to a phrase-structure parsing algorithm would be.

Perhaps the most important advantage to dependency parsing is its usefulness for languages with free word order. In normal phrase-structure parsing, a separate rule is needed for each possible word order. This expands exponentially as sentences become more and more complex. The parsing also becomes much slower, as it is necessary for the parser to iterate through each rule to determine in which order the words in the sentence are, and then to separate the sentence into sub-constituents based on that order. With dependency parsing, words are categorized based only on their dependency relations. Thus, if an adjective depends on a noun, it does not matter on which side of the noun the adjective is placed; given a simple relation that connects the two words, the parser will be able to handle either case. Along these lines, dependency parsing allows two words to be connected from anywhere in the sentence. This is also most useful in free word order languages, where related words may not be within close proximity to one another. In a phrase-structure representation of this kind of sentence, the relationship between the two words would be very difficult to visualize; however, since a dependency parse allows two words to be related regardless of their position in the sentence, as long as there exists a dependency relation connecting them, the relationship would be clear from the parse.

It is evident from the description of dependency parsing that sub-constitutents are eliminated. This makes the visualization of a dependency parse much more simple than that of a typical phrase-structure parse. While this is not a big concern for simple sentences, phrase-structure trees for long and complex sentences can often be confusing to analyze, and also can be difficult to evaluate in terms of correctness. By eliminating these constituents to provide a more simple parsing strategy, dependency parsing is easier to analyze. In most cases, it is not necessary to even draw a tree to visualize the results of the parse; a simple graph such as that in figure 1 is easy enough to evaluate. Another advantage to the elimination of sub-constituents in the parsing is that sentence fragments can easily be parsed by a dependency parser with little to no modification of the parsing algorithm. This is helpful when only a portion of a sentence needs to be analyzed. Most phrase-structure parsers require that the entire sentence be parsed, which is often more information and effort than is needed. There are, of course, situations in which the constituent structure of a sentence is necessary, in which case dependency parsing would be eliminating important information. However, there are many applications to parsing in which that information is not necessary to achieve the desired result, and in these cases, dependency parsing provides a more simple means to an end.

2.3 Project Usage

2.3.1 Reason for Dependency Parsing

In theory, the comparison of parsing across languages could be done with phrase-structure parsing instead of dependency parsing. However, the differences in the two parsing methods would result in significant differences in this comparison. In phrase-structure parsing, as discussed above, sentences are broken down into constituents, and this continues until the word level. This means that a specific rule must be written for each different possible sentence word order, and the sentence is parsed in a relatively straightforward way by matching the words to the rules in which they fit, and following these rules until the entire sentence is parsed. Ambiguity can occur in phrase-structure parsing, but it is often resolved quickly. In fact, it is usually only in extreme cases such as garden path sentences that a possible parse of a sentence is rejected at the "last minute", or just before the sentence is entirely parsed.

Since dependency parsing relies only on relationships between the individual rules in the sentence, there is much more room for ambiguity as the parse progresses. This is why it is possible for different languages to have vast differences in parsing. In phrase-structure parsing, rules are language-specific, so as long as a parser knows which language a sentence comes from, it could conceivably choose the correct rules fairly easily, meaning all languages would parse in a similar amount of time. An exception to this is for languages with free word order, where each possible constituent breakdown must be spelled out by a different rule. However, this project only considers languages with a fairly fixed word order, so this is not a major concern. Conversely, with dependency parsing there are often multiple available choices for the parser at each step, and it is only much later in the sentence that these choices lead to an incorrect or impossible parse. Because of this, dependency parsing of the same sentence in different languages can take varying amounts of steps. The project aims to highlight this by using dependency parsing across different languages and examining the number of steps taken.

2.3.2 Dependency Relations

This project uses the dependency relations in (2). The head of the pair is shown on the left side of the arrow and the dependent word is on the right. These relations are the same across all languages considered. What differs, however, is the direction of the relation. For example, an adjective always depends on the noun it is modifying. In English, the adjective appears before the noun, so there is a left arc from the noun to adjective in the final set of dependencies. In Spanish, however, the word order concerning adjectives is different: a noun precedes the adjectives that modify it. This means that the final dependency graph would show a right arc from the noun to the adjective. Thus, even though the rule that links adjectives to their noun heads is the same for English and Spanish, the resulting arcs in the dependency graphs are not necessarily the same.

(2) The dependency relations used in this project:

- advmod: Adjective $\rightarrow$ Adverb
- amod: Noun $\rightarrow$ Adjective
- aux: Verb $\rightarrow$ Auxiliary
- case: Noun $\rightarrow$ Preposition
- det: Noun $\rightarrow$ Determiner

- dobj: Verb $\rightarrow$ Noun
- iobj: Verb $\rightarrow$ Noun
- nsubj: Verb $\rightarrow$ Noun
- nummod: Noun $\rightarrow$ Number
- root: Root $\rightarrow$ Verb

The parser implemented by the project does not label the dependency relations. Adding the labels requires the parser to search through the relations in more detail to select the appropriate relation for the exact situation. It also requires the parser to be able to determine the difference between constituents such as direct objects and indirect objects. While the labels add valuable information about the relationship between the words to the final dependency parse, it is more simple and straightforward to not include the labels in the parsing algorithm. Thus, for the purposes of this projects, all dependency relations are unlabeled except that of the root to the head of the sentence.

3 Parsing Algorithm

There are several different methods or algorithms to produce a dependency parse of a sentence. Some of these methods use techniques such as constraint satisfaction or statistical lexical dependencies. Machine learning is often applied to dependency parsing to improve and guide the parser. In this project, however, an extension to basic shift-reduce parsing will be used. In particular, the method used to produce the dependency parse is Nivre's Parsing Algorithm.

3.1 Nivre's Algorithm

The dependency parsing algorithm proposed by Nivre restricts the results of the parse to rooted trees. This means that there is a single designated root node that is the head of the tree: it has no incoming arcs. There is a unique path from the root to every word in the sentence. Furthermore, each word in the sentence has exactly one incoming arc. These constraints guarantee that each word has a single head, and that the entire sentence parses into a single, connected structure.

In Nivre's algorithm, words are stored in two places: an input list, and a stack. The words in the input list are processed from left to right, and are pushed onto the stack once they are a part of a dependency arc. The stack contains words that have been partially processed (meaning they are part of a dependency relation, but could still be considered for other relations). Each arc must also be kept track of; these are kept in a set, and new arcs are added to the set when a dependency is realized by the function. The bulk of the algorithm consists of four main functions. Each of these functions performs a single operation on the words in the input list or the stack. Also necessary to the algorithm is a function which keeps track of the state of the parse, and determines which of the four functions to apply next. This function is often called the guide function, or the oracle.

3.1.1 Shift

The shift function is the most simple of the four main functions. It takes the first word in the unprocessed input word list, and adds it to the stack.

3.1.2 Reduce

The reduce function removes a word from the stack when it is done being processed. Before the word can be removed, it must be the case that the word already has a head (or, there is a dependency in the set that has the word as its dependent). This is required because, after being reduced, the word is finished being processed. Thus, in order to enforce the condition that every word in the sentence is part of one connected graph, the word must have a head before it is finished being processed.

3.1.3 Left Arc

The left arc function creates an arc between two words in the sentence. This is a left arc, meaning it has as a head a word somewhere in the sentence, with a second word as a dependent that is to the left of the head. The dependent word need not be the word immediately to the left of the head. The left arc function makes as a head the first word in the list of unprocessed words, and as the dependent the top word on the stack. The arc itself is stored as a relation in the set. The stack is then reduced. There is one condition on the left arc function that must be followed. In order for a left arc to occur, the first word in the stack must not already have a head in the set of dependencies. This is a fairly logical condition. If it were not enforced, it could be the case that the top of the stack already had a head in some other relation. The left arc would make the first word in the input list the head for this word as well; this would mean that the word at the top of the stack would have two heads. This is not acceptable in a dependency parse, so the condition must be followed.

3.1.4 Right Arc

The right arc function is a kind of parallel to the left arc function, with the direction of the arc reversed. It creates a right arc between two (not necessarily consecutive) words in the sentence. A right arc has the head word to the left of the dependent word. The right arc function makes as a head the word on the top of the stack, and as a dependent the first word in the list of unprocessed words. This arc is stored as a relation in the set, and then the dependent word is moved out of the unprocessed list and to the top of the stack. Similar to the left arc, there is a condition on the right arc function. The word at the beginning of the unprocessed list cannot already have a head in the dependency relations. For the same reasoning as with the left arc, this would cause the word to have two heads, which is not allowed.

3.1.5 Oracle

The oracle function is perhaps the most important part of the parsing algorithm; it is also the least specified. The oracle must choose the correct function of the four main functions to apply to the current state of the parse. The oracle must also make sure that it is following the specified conditions on the reduce, left arc, and right arc functions. The order in which one of those functions is chosen is very important to the parsing efficiency. Nivre experimented with three possible methods to determine in which order the oracle should try the functions. This project uses a constant priority for the transitions: left arc $>$ right arc $>$ reduce $>$ shift. This means that a left arc will always be attempted before a right arc (as long as the left arc conditions are met), and on down to a shift. This is somewhat intuitive because a left arc removes a word from the stack, meaning a word becomes fully processed and can be eliminated. This decreases the size of the remaining set of words that have to be processed. On the other hand, the shift operation can be applied at any point to any word, so it would not be logical to attempt the shift operation before the other operations, since it would always occur.

Developing the oracle function was the most complicated part of the implementation of Nivre's parsing algorithm. Since ambiguity can occur when parsing, it is necessary to keep track of the state of the parser at every transition. Executing a left arc on the first step might be an incorrect choice, but the parser will not know this until much later in the parse when there are no more allowed functions, but there are still unprocessed words. Thus, it is necessary for the oracle to maintain a list of parser states. These states include the input list, the stack, and the set of dependency arcs at any given point in the parse. By using this saved state, when the parser encounters a failed parse, it can backtrack one step and try another valid set of steps, repeating this process until a solution is found or all possible parsing orders are attempted.

3.2 Extension to Track Efficiency

One final extension to Nivre's parsing algorithm was required for the project. The intent of the project was to examine the differences in parsing efficiency across the languages, so it was necessary to keep track of how many steps were executed by the parser before finding a valid parse for the sentence. This was accomplished using reference variables. One was increased at each function call from the oracle to one of the four main functions. The other was increased each time the oracle encountered an invalid parse and was forced to backtrack using a saved state. Using these variables, it was possible to examine the efficiency of the parser for each sentence analyzed.

4 Language Choices

This project is very small-scale, and as a result only concentrates on three languages, all from the Indo-European family. The parser is developed for a Germanic language, English, and two Romance languages: Spanish and Portuguese. All three of these languages are SVO-ordered, which made the parsing algorithm easier, both to implement and to analyze. The main purpose of evaluating these three languages was to see how similar languages are within the same sub-family, and whether languages within the same sub-family parse more similarly than a language from a different sub-family. Spanish and Portuguese have nearly identical sentence structures, but they differ enough from English to make a difference in the parsing.

As explained in the dependency relations section, the same relations hold across these three languages. What is different is the direction of the arcs permitted for each language. These differences come from differences in word order between the languages. Since all three languages are SVO, the basic word order is the same. There are, however, small differences that cause the parsing algorithm to work differently across the languages. Although Spanish is inherently SVO, it has a generally more free word order than English. For example, the indirect object can be placed at the end of the sentence (like English), but it can also appear initially in the sentence. This is demonstrated in (3) - (6).

(3) English: I am going to tell you a story

(4) Portuguese: vou lhe contar uma história

(5) Spanish: te voy a contar un cuento

(6) Spanish: voy a contar un cuento a ti

Sentences (5) and (6) would be translated to the same sentence in English (specifically, (3)), and indeed have the same meaning, but both word orders are valid, and both are used. Although it is possible to mirror the word order of (6) in English:

(7) English: I am going to tell a story to you

this construction is non-typical and would not be frequently used. Thus, the parser must be able to handle input sentences in both word orders for the different languages, but in particular for Spanish.

Another case of English having a different word order than the chosen Romance languages is that of adjectives. In English, adjectives come before the noun they are modifying. In Spanish and Portuguese, however, the adjective follows the noun. This is demonstrated in (8) - (10).

(8) English: the red book

(9) Portuguese: o livro vermelho

(10) Spanish: el libro rojo

These three languages were chosen for the project because they have similar enough word order to not require writing three completely different parsing algorithms in order to produce the dependency parse. However, they have small differences that can be highlighted by the analysis of the parsing efficiency. The chosen languages also allow the comparison of the challenges when parsing two somewhat different languages as opposed to two very similar languages.

5 Results

It seems likely from the discussion of word order across the three languages that there are cases in which the parser should take the same number of steps to build a parse, and cases in which the number of steps should vary slightly. What is perhaps not clear is how much less efficient a parse will be from even the slightest variation in word order. Several examples are now analyzed that highlight the range of differences in parsing efficiency across simple sentences.

5.1 Example 1

A first analysis is given of a sentence for which all three languages take the same effort to parse. (Again, "effort" here denotes number of calls from the oracle to the four main functions outlined in the parsing algorithm.) Consider sentences (11) - (13):

(11) English: I like the trees

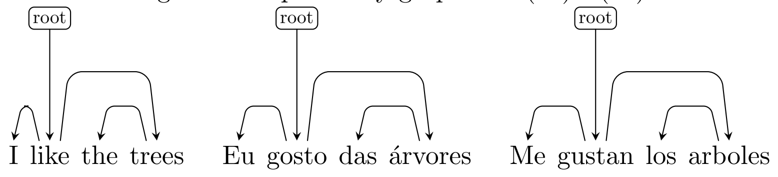
(12) Portuguese: Eu gosto das árvores

(13) Spanish: Me gustan los arboles

When presented with any of these sentences, the parser takes 14 steps to execute the parse, where one step represents one call to a core function. Define *backtracking* to be a point where the parser is forced to abandon a parse because the end state is not yet reached, but there are no available options left. When parsing any of (11) - (13), the parser is forced to backtrack twice.

It is also necessary to check the correctness of the parse to ensure that the three examples are indeed being parsed in the right way. The dependency relations given as output from the parse can easily be converted into a dependency graph and analyzed. The graphs for sentences (11)-(13) are shown in figure 3. All three graphs in figure 3 have the same dependency relations, and these relations are correct given the dependency rules listed in (2). Although the parser is not equipped to handle dependency tags, the tags on the dependencies in figure 3 would be the same across the languages. This example shows that there are sentences for which the parser is just as efficient for Spanish or Portuguese as it is for English.

Figure 3: Dependency graphs for (11) - (13)



5.2 Example 2

Although sentences (11) - (13) show that it is possible for the parser to handle different languages with no change in efficiency, this is not always the case. In fact, parsing only a slightly more complex sentence brings extreme differences in parsing efficiency. An analysis of the English sentence (14) and its multi-language parses will demonstrate this.

(14) English: John gave you a red book

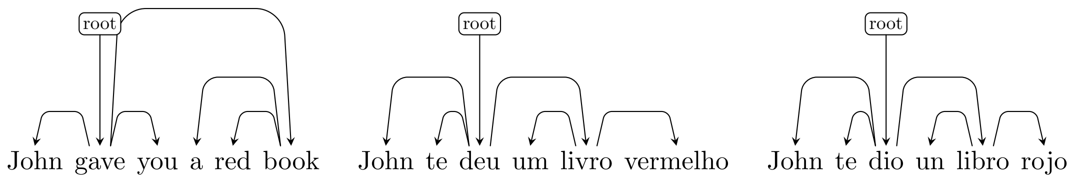
The sentence (14) takes 21 steps to parse, and the parser has to backtrack three times. A translation of (14) into Portuguese and Spanish that takes the most natural word order in those languages would produce the sentences (15) and (16). Note that there are differences in the syntax of these sentences; the adjective follows the noun, instead of preceding it like English, and the indirect object precedes the verb, instead of immediately following it as occurs in (14).

(15) Portuguese: John te deu um livro vermelho

(16) Spanish: John te dio un libro rojo

For both (15) and (16), the parser takes 24 steps to complete the parse of the sentence, backtracking six times. Although the number of additional steps required is not much higher for Portuguese and Spanish than for English, there is still a noticeable difference in the parsing efficiency. As in Example 1, figure 4 shows that the dependency graphs created by the parser are very similar. The differences here between English and the Romance languages can be accounted for by the differences in word order; the dependency relations between corresponding words are still the same.

Figure 4: Dependency graphs for (14) - (16)



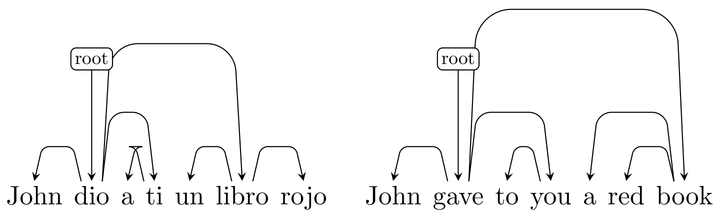
It is possible to make the word order closer to that of English; however, this vastly slows the parser. In Spanish, a preposition is required if the indirect object directly follows the verb. This construction can be seen in (17). The corresponding English translation (that includes the preposition) is in (18).

(17) Spanish: John dio a ti un libro rojo

(18) English: John gave to you a red book

When parsing (17), 51 steps are required, and the parser is forced to backtrack 18 times. The difference in parsing the two English sentences ((14) and (18)) is significant as well: the addition of the preposition increases the number of steps to 42 and the number of backtracking occurrences to nine. Although this is approximately twice as much effort as the simpler, non-preposition English sentence, it is still much more efficient than the Spanish phrase. The dependency graphs for these two sentences, shown in figure 5, are more similar than those for the more natural word ordered sentences of figure 4, but still show a small difference due to the adjective order.

Figure 5: Dependency graphs for (17) and (18)



There is one more word order that should be considered for this sentence: having the indirect object at the end. These constructions are shown in (19) - (21):

(19) English: John gave a red book to you

(20) Portuguese: John deu um livro vermelho para te

(21) Spanish: John dio un libro rojo a ti

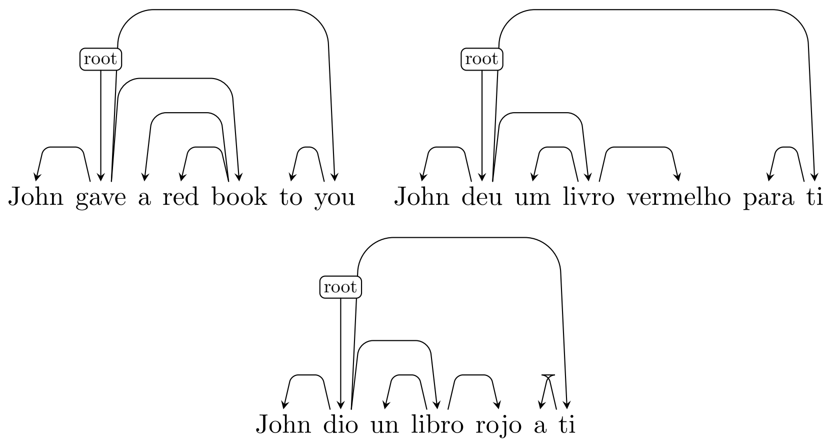
The only difference in syntactic structure between these sentences is the adjectival order. It is the same for the Romance languages, but differs for English. This brings about an expected result: the parser takes 40 steps with eight backtracking occurrences to parse (19), but 63 steps with 18 backtracking occurrences to parse (20) and (21). This shows that a small difference in word order can bring about large decreases in efficiency. The dependency parses for these sentences, shown in figure 6, are again almost identical, with the exception of the adjective order.

5.3 Analysis of Results

Although only parses of simple sentences were analyzed, it is clear that there is a significant difference in parsing efficiency across the languages. This result leads to the conclusion that the parsing algorithm is best suited for English sentence constructions. Even the simple word order reversal required for adjectives in Spanish and Portuguese as opposed to English caused the parser to take many more steps and backtrack more times in a single parse. These results imply that it would be worthwhile to develop a more language-specific rendition of a dependency parsing algorithm if much work were to be done with languages that are not English. As more and more complicated sentences are analyzed, these parsing inefficiencies would grow quickly, which could potentially cause problems for a program in terms of time and space constraints.

The results from Examples 1 and 2 also show that languages seem to be more similar within a sub-family than across sub-families. The Romance languages Spanish and Portuguese

Figure 6: Dependency graphs for (19) - (21)


root
John gave a red book to you
root
John deu um livro vermelho para ti
root
John dio un libro rojo a ti

parsed in an identical manner on the example sentences, while English had slight differences that caused the variations in efficiency. It is reasonable to assume that this phenomena could extend to other languages; Romance languages such as Italian and Romanian are likely to parse more similarly to Spanish and Portuguese than they are to English.

6 Extensions

It is clear that this project does not go very far into detail, both with the level of sentence complexity allowed by the parser, and the spread of languages tested. In fact, significant results could only really be obtained by vast expansion of these two areas. Another important extension considers parser efficiency: both in general, and in language-specific cases.

At its current state, the parsing algorithm handles only very simple sentences from the three languages. This is necessary to develop a baseline parser that works for several languages. However, more significant differences in the languages only come to light when more complex sentence structure is analyzed. The dependency relations used in the project were fairly simple relations between parts of speech that come up in most sentences. However, there are many other dependency relations that would be necessary for a more well-developed parser. The Universal Dependencies database contains relations for language phenomena such as copula, compound words, relative clauses, and conjunctions. It is only when examining sentences containing these more complex constructions that the more interesting differences between the languages are highlighted. Along this same subject, the simple act of adding more words to the parser's lexicon would increase the effectiveness of the project. The differences between the languages are best shown through repeated examples, so being able to form more sentences would better show how the parsing strategy changes across the languages.

As the project is so small-scale, it was developed only to handle three different languages. By choosing two languages from the same sub-family and one from a different sub-family, it was possible to examine how the similarities in parsing change as familial boundaries are crossed. However, only looking at three different languages, even well-chosen ones, does not provide much insight into how this problem would scale, both within language families and across families. In order to analyze the true differences in dependency parsing across languages, it would be necessary to work with languages outside the Indo-European family. The project would also have to consider many more languages, in order to get a complete sense of how the parsing differs across languages. Some thought would be required as to whether or not to consider languages that are not inherently SVO in order; it is clear that these sentences would parse differently than the SVO languages that were considered, but this difference in parsing is not the main consideration of the project. In order to extend to these languages, a completely different parsing algorithm, in particular the guide function, would be necessary. However, there are many languages that are SVO and yet still have enough subtle differences in word order that their parses would be different; it is these languages that the project could most easily be extended to handle.

Finally, the parser could be better optimized for different languages. This project examined the effectiveness of the current parsing algorithm for these three languages. By examining a larger set of results, it would be possible to determine a better way to parse sentences from each language. Perhaps the most useful extension to this project would be language-specific oracles. The oracle function is essentially what determines which parsing order to try. Having a language-specific oracle would mean that the parse order could be optimized for each language. This would improve parsing efficiency and would be more likely to return the correct parse in cases of ambiguity. On a less language-specific note, it might be possible that the parsing algorithm is not even as effective as it could be for English. In this case, a reasonable course of action would be to try different parse orders and guide functions to improve the search strategies and decision-making of the current parser. This extension would not have to be language-specific, but would be valuable to increasing parsing efficiency and effectiveness.

7 Bibliography

Jurafsky, Daniel, and James H. Martin. *Speech and Language Processing.* 2017. Accessed 14 Apr. 2017

Nivre, Joakim. "Universal Dependencies V2." *Universal Dependencies.* LINDAT/CLARIN, 2014. Web. 14 Apr. 2017.

Nugues, Pierre M. *Language Processing with Perl and Prolog.* Springer Berlin Heidelberg, 2014. Accessed 14 Apr. 2017.