Natural Language Processing: Assignment 3

Johannes Gontrum

1 Annotation: Differences

1.1 Verb-object relations

A consistent error throughout my annotation choosing the wrong label for a relation between a verb and its objects. I annotated under the assumption that *nsubj* should be used for relations from a verb to a noun that is the subject. As stated in the documentation: "A nominal subject (nsubj) is a nominal which is the syntactic subject and the proto-agent of a clause."[1]. *iobj* on the other hand is used for all other nouns that are required arguments of the verb: "The indirect object of a verb is any nominal phrase that is a core argument of the verb but is not its subject or (direct) object." [2], whereas I consequently chose *obj* to connect the verb to any other noun.

While I was correct about the usage of *nsubj* and *iobj*, *obj* should only be used for other core arguments of the verb: "The object of a verb is the second most core argument of a verb after the subject." [3]. I made the mistake to apply it also for relations like "attacked" → "April" in the context "the victim had attacked the subject in April". But clearly, "April" is not an object of the verb, and more a temporal adverb. Therefore the relation *obl* should be used in these circumstances: "The obl relation is used for a nominal (noun, pronoun, noun phrase) functioning as a non-core (oblique) argument or adjunct. This means that it functionally corresponds to an adverbial attaching to a verb, adjective or other adverb."[4].

1.2 Multiword expressions

In multiword expression (MWE) of nouns that are the object of a verb, I was unsure which of the nouns should be the dependent in the verb-subject dependency. I intuitively decided to always use the first noun and connect the other nouns in the expression with the *compound* relation. When evaluating my annotation, I realized, however, that this is not the intended way. The documentation differentiates different kinds of MWEs: If the MWE is e.g. "[..] as well as [...]", the relation *fixed* should be used: "It used for certain fixed grammaticized expressions that behave like function words or short adverbials"[5]. For nouns MWEs without a clear head, *flat* is the best choice: "[...] these expressions do not have any internal syntactic structure and that the structural annotation is in principle arbitrary."[6]. An example here are names like "Hilary Rodham Clinton". In contrast, phrases like "apple pie" have an internal hierarchical structure ("pie" is the head, because "apple" specifies the kind of pie), so we use the *compound* relation here, "[...] which applies to endocentric (headed) MWEs". [7].

As an example, I annotated "spoke to CNN Style" as *CNN* → *Style* — *compund* and *spoke* → *CNN* — *obl*, whereas the gold standard reversed the order: *Style* → *CNN* — *compound* and *spoke* → *Style* — *obl*. However, after reading the documentation in detail, I believe that neither annotation is correct. Instead, I suggest that "CNN Style" describes a name, so the *flat* label would be adequate: *CNN* → *Style* — *flat* and *spoke* → *CNN* — *obl*.

1.3 Embedded clauses

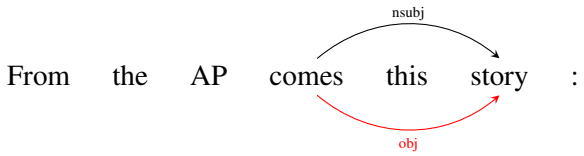
In my annotation of the sentences, I always used *ccomp* to connect the predicate of an embedded clause to the main predicate. For example in "Mr Osborne signed up with a US speakers agency after being sacked in July" I introduced the dependency *signed* → *sacked* — *ccomp*. Similar to my confusion about the *obj* vs. *obl* labels mentioned in 1.1, *ccomp* should, however, be used if the clause of the predicate is an object of the main

[1] http://universaldependencies.org/u/dep/nsubj.html
[2] http://universaldependencies.org/u/dep/iobj.html
[3] http://universaldependencies.org/u/dep/obj.html
[4] http://universaldependencies.org/u/dep/obl.html
[5] http://universaldependencies.org/u/dep/fixed.html
[6] http://universaldependencies.org/u/dep/flat.html
[7] http://universaldependencies.org/u/dep/flat.html

predicate: ”A clausal complement of a verb or adjective is a dependent clause which is a core argument.” [8]. The documentation gives the example ”He says you like to swim” with *says* → *like* — *ccomp*, because ”you like to swim” is an argument of ”say”. In my example mentioned before, the relation *advcl* is used in the gold standard: ”An adverbial clause modifier is a clause which modifies a verb [...]” [9]. ”after being sacked [...]” again describes the temporal circumstances of the first clause, so *advcl* is clearly the correct annotation here.

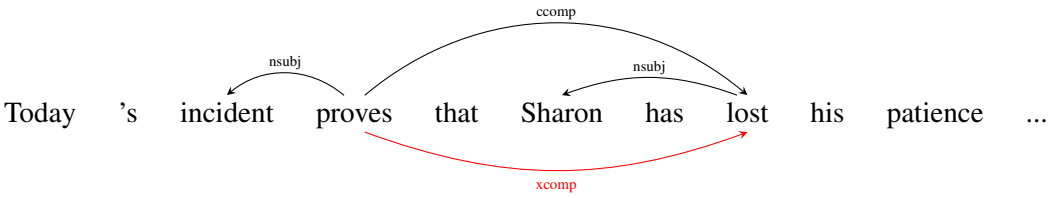
2 Parsing: Differences

2.1 Verb-object relations



In this sentence, the noun ”story” is the only object of the ”come”, which is also the subject. It should, therefore, be annotated as *comes* → *story* — *nsubj*. The parser, however, labels the relation as *obj*. This is clearly an error because it produces a dependency graph that does not contain a single subject. I wonder if the parser made this mistake because it is unusual for the subject to appear after the predicate, a position where objects are more frequently found.

2.2 Embedded clauses

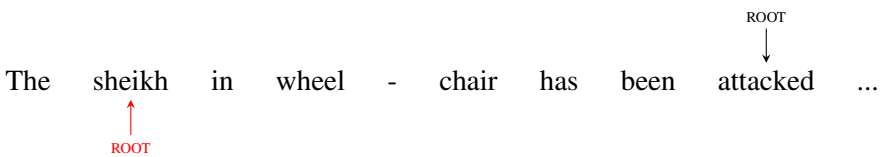


As discussed in 1.3, the relation *ccomp* has to be used if a clause is the argument of a predicate. It then uses the main predicate as the head and the verb of the embedded sentence as the dependent: *proves* → *lost* — *ccomp*. This is also the way, the gold annotation interprets this sentence. The parser recognizes that this relation describes the connection to an embedded sentence, it uses, however, the wrong label: *xcomp*.

As stated in the documentation, an important difference between *ccomp* and *xcomp* is the subject of the sentence they link to. *ccomp* referes to a verb in an embedded clause that has its **own** subject, while *xcomp* is used, if the subject of the main clause and the embedded clause are the **same**: ”However, if the subject of the clausal complement is controlled (that is, must be the same as the higher subject or object, with no other possible interpretation) the appropriate relation is xcomp.”[10].

But in our case, ”incident” is obviously the subject of the clause around the predicate ”prove”, while ”Sharon” is the subject of the ”lost” clause. Consequently, *ccomp* is the correct relation, not *xcomp*.

2.3 Wrong root

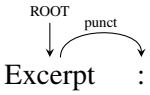


One of the most devastating errors is to choose the wrong root word (the word that is the dependent of the ”root” dependency) because it controls the main structure of the whole sentence. This happened in the sentence

[8] http://universaldependencies.org/u/dep/ccomp.html
[9] http://universaldependencies.org/u/dep/advcl.html
[10] http://universaldependencies.org/u/dep/xcomp.html

above. While it seems intuitive for a human that "attacked" should be the root, the parser mistakenly selected the noun "sheikh" instead.

Though I believed this to be a very obvious case, I was surprised how poorly documented it is. How do we define the root of a sentence? The universal dependencies guidelines merely state "The root grammatical relation points to the root of the sentence." [11], but does not give clear instructions how to identify it. Intuitively, I would always see a verb as the root of the sentence, however, there are cases like:



Here the noun "Excerpt" must be the root. Also, UD guidelines state that the dependents of the root relation are verbs in only 56% of the cases, with nouns in the second place (17%) [12].

Also, literature (see Kübler et al. (2009), Jurafsky and Martin (2009)) does not formally define this case. The only difference is that in the examples in Kübler et al. (2009), *PRED* is used as the label of the *ROOT* $\rightarrow$ *verb* relation.

3 Arc-eager transition sequence

The table containing the transitions can be found in Table 1.

When constructing the set of arcs in an arc-eager way, one should create arcs as soon as possible. My general strategy was to use the SHIFT operation whenever no arc transition could be made. I also noticed that a REDUCE transition was required after the right-arc transition connected the verb with its objects. This is because the topmost word in the stack would not be used anymore and must, therefore, be removed. I also enumerated all words by their index in the sentence to avoid confusion with the two words "for".

4 Context-free grammars

In my grammar in Figure 1, I have added 'SS' (line 2), a super-start-symbol, that connects a sentence with the punctuation. I had to make this change because I construct an embedded sentence structure for sentence four, which introduced an unwanted ambiguity for the attachment of the final stop. The embedded sentence is then attached to a CP (line 4) which will be integrated into a VP (line 6). The construction rule $S \rightarrow NP\ VP\ PP$ is linguistically questionable because this structure does not make it clear to which phrase a PP is connected. Ignoring this, I added the rule to describe sentence five and to avoid even more PP-ambiguities.

The sentence "Global warming has caused a change in the pattern of the rainy seasons." can be described with two different trees given the grammar in Figure 1. Walking the tree top-down, we first apply the previously mentioned super-start-symbol rule to connect the sentence with its punctuation. The sentence consists in this case of an *AP* (here: adjective-phrase) and a verb phrase *VP*. The *AP* is constructed by combining the adjective 'global' and the noun 'warming'. The *VP* that is the root of the sister tree has again a *VP* and an *NP* as its children. I decided to combine an auxiliary verb and a full verb ("has" and "caused") to a *VP* to avoid adding too many non-terminal symbols. Up to this point, both trees in Figure 2 and 3 are identical.

In the tree in Figure 2 the *NP* however covers a different structure. Here, it splits the span it describes in two almost equally long parts: "a change in the pattern" and "of the many seasons", both with trees of similar depth. The first part attaches the *PP* "in the pattern" directly to the *NP* "a change" (blue) to form an *NP*, while it higher up attaches its sister PP_1 (orange) to it.

In comparison, this structure is deeper in the tree in Figure 3: Here, PP_1 again embeds the nominal phrase NP_2, but it is governed by the first PP, PP_0, that also covers the span "the pattern", described by NP_1. NP_0 on the other hand is higher up in the tree and a sibling to the PP_0 structure.

The reason for this typical ambiguity is the recursive production rule $NP \rightarrow NP\ PP$. If we have multiple constituences *NP* and *PP*, we have multiple ways to apply the rule: Either having the first shallow *NP* (the red NP_0 in the current example) as a direct child under the main *NP* and a *PP* that also embeds the second *PP* (orange PP_1 here). Or, as shown in the tree in Figure 2, we recursively apply the same rule again and embed both NP_0 and PP_0 as the first child.

[11] http://universaldependencies.org/u/dep/root.html
[12] http://universaldependencies.org/u/dep/root.html

Transition	Stack	Buffer	Arcs
	([ROOT$_0$],	[He$_1$, ..., .$_9$],	$\emptyset$)
SH $\Rightarrow$	([ROOT$_0$, He$_1$],	[worked$_2$, ..., .$_9$],	$\emptyset$)
LA$_{\text{nsubj}}$ $\Rightarrow$	([ROOT$_0$],	[worked$_2$, ..., .$_9$],	$A_1 = \{(\text{worked}_2, \text{nsubj}, \text{He}_1)\}$)
RA$_{\text{root}}$ $\Rightarrow$	([ROOT$_0$, worked$_2$],	[for$_3$, ..., .$_9$],	$A_2 = A_1 \cup \{(\text{ROOT}_0, \text{root}, \text{worked}_2)\}$)
SH $\Rightarrow$	([ROOT$_0$, worked$_2$, for$_3$],	[the$_4$, ..., .$_9$],	A_2)
SH $\Rightarrow$	([ROOT$_0$, ..., the$_4$],	[BBC$_5$, ..., .$_9$],	A_2)
LA$_{\text{det}}$ $\Rightarrow$	([ROOT$_0$, worked$_2$, for$_3$],	[BBC$_5$, ..., .$_9$],	$A_3 = A_2 \cup \{(\text{BBC}_5, \text{det}, \text{the}_4)\}$)
LA$_{\text{case}}$ $\Rightarrow$	([ROOT$_0$, worked$_2$],	[BBC$_5$, ..., .$_9$],	$A_4 = A_3 \cup \{(\text{BBC}_5, \text{case}, \text{for}_3)\}$)
RA$_{\text{obl}}$ $\Rightarrow$	([ROOT$_0$, worked$_2$, BBC$_5$],	[for$_6$, ..., .$_9$],	$A_5 = A_4 \cup \{(\text{worked}_2, \text{obl}, \text{BBC}_5)\}$)
RE $\Rightarrow$	([ROOT$_0$, worked$_2$],	[for$_6$, ..., .$_9$],	A_5)
SH $\Rightarrow$	([ROOT$_0$, ..., for$_6$],	[a$_7$, ..., .$_9$],	A_5)
SH $\Rightarrow$	([ROOT$_0$, ..., a$_7$],	[decade$_8$, .$_9$],	A_5)
LA$_{\text{det}}$ $\Rightarrow$	([ROOT$_0$, ..., for$_6$],	[decade$_8$, .$_9$],	$A_6 = A_5 \cup \{(\text{decade}_8, \text{det}, \text{a}_7)\}$)
LA$_{\text{case}}$ $\Rightarrow$	([ROOT$_0$, worked$_2$],	[decade$_8$, .$_9$],	$A_7 = A_6 \cup \{(\text{decade}_8, \text{case}, \text{for}_6)\}$)
RA$_{\text{obl}}$ $\Rightarrow$	([ROOT$_0$, ..., decade$_8$],	[.$_9$],	$A_8 = A_7 \cup \{(\text{worked}_2, \text{obl}, \text{decade}_8)\}$)
RE $\Rightarrow$	([ROOT$_0$, worked$_2$],	[.$_9$],	A_8)
RA$_{\text{punct}}$ $\Rightarrow$	([ROOT$_0$, worked$_2$, .$_9$],	[],	$A_9 = A_8 \cup \{(\text{worked}_2, \text{punct}, .9)\}$)

Table 1: Transition table for the sentence "He worked for the BBC for a decade .".

```
# Grammar
SS -> S Punct
S -> NP VP | AP VP | NP VP PP
CP -> Comp S
NP -> Pronoun | Det Noun | Noun Noun | Noun Conj Noun | Det AP
 ↪  | Det NP | Noun NP | NP PP
VP -> Verb Verb | Aux Verb | Verb Noun | Adv Verb | Verb PP PP
 ↪  | Verb NP | VP NP | VP CP
PP -> Prep NP
AP -> Adj Noun

# Lexicon
Noun -> 'BBC' | 'decade' | 'CNN' | 'Style' | 'experience' | 'warming'
 ↪  | 'change' | 'pattern' | 'seasons' | 'Davis' | 'Cup' | 'part'
 ↪  | 'scheme' | 'money' | 'sponsorship' | 'advertising'
Verb -> 'worked' | 'spoke' | 'caused' | 'wonder' | 'played' | 'makes'
Aux -> 'has'
Adj -> 'Global' | 'rainy'
Adv -> 'also'
Comp -> 'whether'
Pronoun -> 'He' | 'She' | 'I'
Det -> 'the' | 'The' | 'a'
Prep -> 'for' | 'to' | 'about' | 'of' | 'in' | 'through'
Conj -> 'and'
Punct -> '.' | ','
```

Figure 1: A context-free grammar that describes the syntactic structure of the first five sentences.

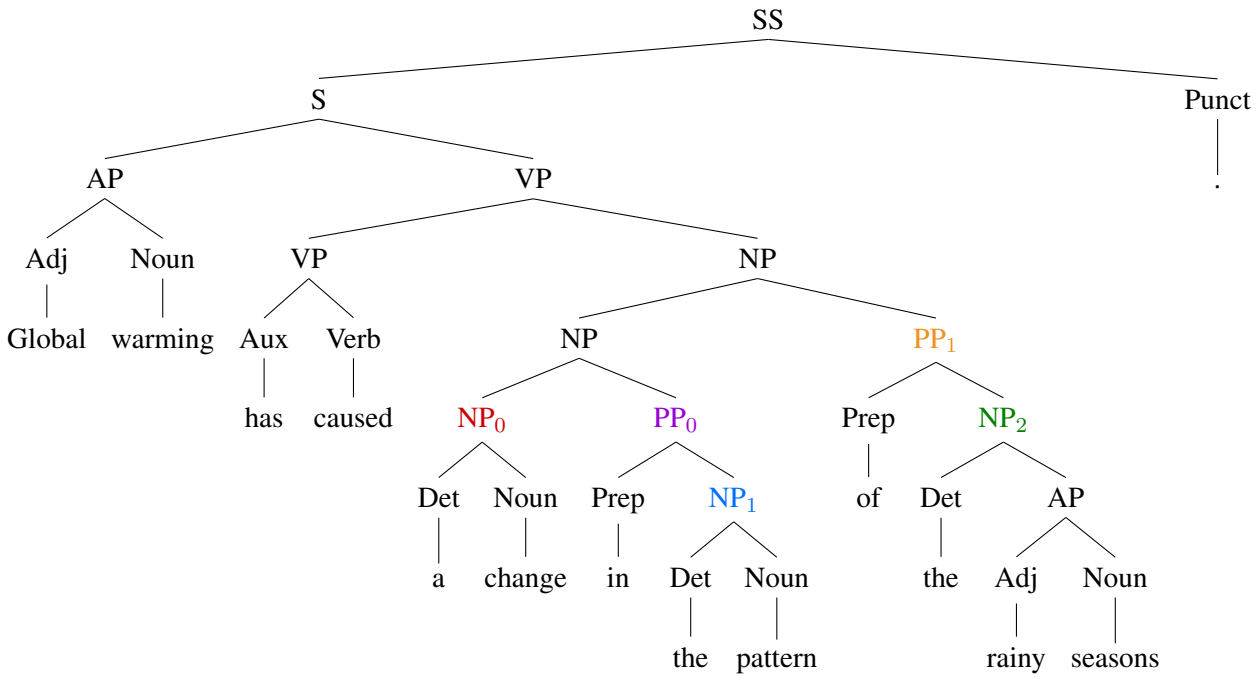


Figure 2: The first syntactic tree for the sentence ”Global warming has caused a change in the pattern of the rainy seasons.”. Indizes next to non-terminals are for illustrative purposes only and not part of the grammar.

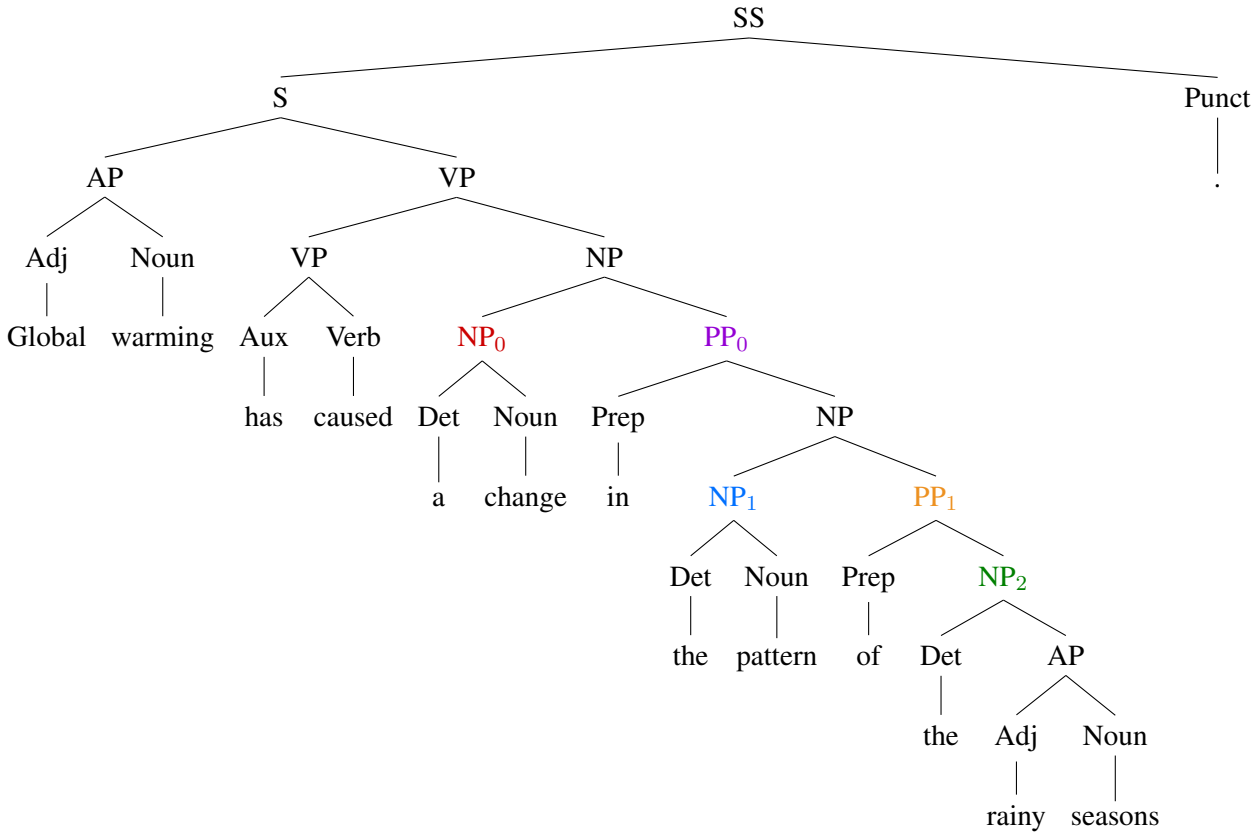


Figure 3: The second syntactic tree for the sentence ”Global warming has caused a change in the pattern of the rainy seasons.”. Indizes next to non-terminals are for illustrative purposes only and not part of the grammar.

	0	1	2	3	4	5	6	7	8
0	$\{\text{Pron}_{\text{init}}\}$							$\{\text{S}_{\langle 0,0\rangle,\langle 1,7\rangle}\}$	$\{\textbf{SS}_{\langle 0,7\rangle,\langle 8,8\rangle}\}$
1		$\{\text{Verb}_{\text{init}}\}$						$\{\text{VP}_{\langle 1,1\rangle,\langle 2,7\rangle}\}$	
2			$\{\text{Prep}_{\text{init}}\}$		$\{\text{PP}_{\langle 2,2\rangle,\langle 3,4\rangle}\}$			$\{\text{VP1}_{\langle 2,4\rangle,\langle 5,7\rangle}\}$	
3				$\{\text{Det}_{\text{init}}\}$	$\{\text{NP}_{\langle 3,3\rangle,\langle 4,4\rangle}\}$				
4					$\{\text{Noun}_{\text{init}}\}$				
5						$\{\text{Prep}_{\text{init}}\}$		$\{\text{PP}_{\langle 5,5\rangle,\langle 6,7\rangle}\}$	
6							$\{\text{Det}_{\text{init}}\}$	$\{\text{NP}_{\langle 6,6\rangle,\langle 7,7\rangle}\}$	
7								$\{\text{Noun}_{\text{init}}\}$	
8									$\{\text{Punct}_{\text{init}}\}$

Table 2: Parse chart for the sentence "He worked for the BBC for a decade ." with the grammar in Figure 4. Back pointers are encoded in the subscripts of the non-terminals: $\langle a, b\rangle\langle c, d\rangle$ describes that the current symbol was constructed out of the symbol in the cell a, b (row a, column b) and the symbol in c, d. Please note that back pointers must also specify the non-terminal in the cell they point to when the parse-chart describes ambiguous derivations. Empty cells contain the empty set $\emptyset$.

5 Parsing CFGs with the CKY algorithm

To parse the sentence "He worked for the BBC for a decade.", I used the baseline grammar from lab 10, instead of my own in Figure 1. However, to apply the CKY algorithm with the given grammar, it must first be converted into Chomsky normal form (CNF). A context-free grammar in CNF is required by the parsing algorithm because it iterates over the all possible ways a span could be split into two parts. Hence, production rules with non-terminals on the right-hand side (RHS) must have an arity of two. To transform the baseline grammar into CNF, I introduce the super-start symbol again, split a rule with a longer RHS into two by adding a new non-terminal (*VP1*) and resolve the chain rule in *NP* $\rightarrow$ *Pron* by replacing every occurrence of *NP* with *Pron*, after duplicating the original rule. Please note that in contrast to the algorithm in Jurafsky and Martin (2009), my indices are off by one, because I find it easier to implement and think about charts and lists when they start by 0.

The CKY algorithm works by moving diagonally through the chart cells. It starts in cell $0, 0$ (see Table 2) and moves down to $8, 8$, adding the PoS-Tag to the cell (according to the terminal rules) for the word at the given index in the sentence. In each further iteration, the diagonal line gets shorter, as we move towards the topmost cell $0, 8$. We can interpret each cell at the position j, i over the span of the sentence they describe. A non-terminal in the given cell would cover the words from the index j to the index i. For each of these cells, we try to find a point $j < k \leq i$, where we can split it. We then see if we can find a rule that has a symbol in the first cell (j, k) as the first symbol on its RHS and a symbol from cell $k + 1, i$ as the second one. For each of these rules, we add their left-hand side to the current cell.

We do so for the first time when moving down the second diagonal from $0, 1$ to $7, 8$: In cell $3, 4$ we discover that we can split it into $3, 3$ and $4, 4$ and find the symbols *Det* and *Noun* in the cells. Since the rule *NP* $\rightarrow$ *Det Noun* exist, *NP* is added to $3, 4$. We repeat this for $6, 7$ as well. The next diagonal is shorter again, but it also gives us more possibilities to split the growing spans. in $2, 4$, we examine the cells $2, 2$ and $3, 4$, as well as $2, 3$ and $4, 4$. We use the previously built *NP* to form a *PP* here. Next is *PP* in $5, 7$ and *VP1* in $2, 7$. The next non-empty cell is the *VP* in $1, 7$ and the sentence symbol *S* in $0, 7$. Eventually, we check all the ways we can split the whole sentence into two parts for the last cell in $0, 8$. Here we combine *S* and the punctuation into the super-start-symbol *SS*. Since this cell covers the whole sentence and also contains the start symbol of the grammar, we know that we have successfully recognized the input.

6 Pet project: Implementing an arc-eager dependency parser

While working on the assignment, I became very interested in dependency parsing. For me, the best way to understand an algorithm is to implement it myself, so I decided to build a small parser. While the instructions for the different transitions are stated down very clearly in Kübler et al. (2009) and thus easy to write down in Python, my parser was still without any knowledge of what order to apply these transitions. While one can write a small grammar for a constituency parser in a few minutes to test it, I found it frustrating that this does

not seem to work so easily for a transition based system. I, therefore, decided to try and train a model that can predict the right transition given a configuration.

As described in Kübler et al. (2009), one has to transform the dependency structure in a treebank into a sequence of configurations. For each sentence, one has to start with the initial configuration and then chose the right transition, leading to the next configuration. Since we already know how the graph is constructed, we can derive these decisions. Since the oracle definition in Kübler et al. (2009) lacks the *Reduce* function and is probably not for an arc-eager parser, I found working instructions in Goldberg and Nivre (2012). Given a corpus reader, a suitable graph- and configuration class, I was able to generate the desired output.

For training a classifier, I decided to use *scikit-learn*. For now, I concentrate on predicting the right transition, ignoring labels. After comparing the performance of a few classifiers using cross-validation (*support vector machines, logistic regression, random forest classification*), I chose logistic regression. Its training time is tolerable (a few minutes for the training set) and it performed with an accuracy of 0.6979 against the random baseline of 0.25.

After training a model, I integrated it as the oracle function o in my parser to predict the best transition for the current configuration. However, I realized that either my model is very bad or one needs to add sanity checks for the transitions. My first model, for example, failed to create even the first transition, because it predicted a left-arc with *ROOT* as the dependent. All in all, I am of course disappointed that my parser performed very badly, but I am still eager to improve my model and have a look at other, more mature, implementations of a transition based dependency parser.

If you are interested, my implementation can be found here: `https://github.com/jgontrum/uu_lt_nlp_17/blob/master/parsing.ipynb`.

6.1 Problems

Of course, I do not expect a good performance of a small parser written in a hurry. But I see a few points that I want to improve in the future. For one, I only predict the transition, not the label, for a given configuration. I'm wondering if it makes more sense to train a second model for the label classification or if a combination of transition and label in one model would work better. Also, the classification algorithm I chose is probably neither the best nor are its parameters optimized. I expect to improve the accuracy of the parser here. Also, to properly evaluate the output of my parser, it must be written in the CONLL format again. Due to the limited time, I did not implement a proper output function.

Generally, more feature engineering is needed to further improve the performance of the parser. Right now, I only rely on the top three words on the stack and the top five words on the buffer, neglecting even PoS tags or dependency information. I think, that by working on the words themselves, I often run into an OOV problem. Using the part-of-speech tags here instead is something I want to investigate in the future.

References

Yoav Goldberg and Joakim Nivre. A dynamic oracle for arc-eager dependency parsing. In *COLING*, pages 959–976, 2012.

Daniel Jurafsky and James Martin. *Speech and language processing: An introduction to natural language processing*. Prentice Hall, 2009.

Sandra Kübler, Ryan McDonald, and Joakim Nivre. Dependency parsing. *Synthesis Lectures on Human Language Technologies*, 1(1):1–127, 2009.

```
# Grammar
SS -> S Punct
S -> NP VP
S -> Pron VP
NP -> Det Noun
VP -> Verb VP1
VP1 -> PP PP
PP -> Prep NP
PP -> Prep Pron

# Lexicon
Noun -> 'BBC' | 'decade'
Verb -> 'worked'
Pron -> 'He'
Det -> 'the' | 'a'
Prep -> 'for'
Punct -> '.'
```

Figure 4: A context-free grammar in Chomsky normal form that can be used to parse the sentence ”He worked for the BBC for a decade .”.