Parsing
Assignment 3

Anshul Tomar 170070007
Anwesh Mohanty 170070009
Parth Shettiwar 170070021

September 2020

Contents

1 Constituent Parsing to Dependency Parsing

1.1 Approach

We did our implementation using the Allen Parser. As described in class, the method was to find the head word in each sub-tree and then propagate it upwards. The head was found using rule basis. We compiled a comprehensive list of rules to determine the head everytime. Finally, if suppose, the tags are not matching any of the current rules, then we used the principle that English is Head initial language and hence propagated the first word of the sub-tree as the head upwards and rest all as modifiers. In this case, we keep the relation of head to modifier as "R".

```
Rules["VB CC VB"] = [[0],[[0, 1, 'cc'],[0, 2, 'conj']]]
Rules["VBP NP"] = [[0],[[0, 1, 'dobj']]]
Rules["VBP S SBAR"] = [[0],[[0, 1, 'xcomp'],[0, 2, 'dep']]]
Rules["VBG NP"] = [[0],[[0, 1, 'dobj']]]
Rules["NNP NNP"] = [[0],[[0, 1, 'dobj']]]
Rules["NP JJ"] = [[1],[[1, 0, 'npadvmod']]]
Rules["CD NNS"] = [[1],[[1, 0, 'num']]]
Rules["NP ADJP"] = [[0],[[0, 1, 'amod']]]
Rules["DT NN"] = [[1],[[1, 0, 'det']]]
Rules["DT JJ NN"] = [[2],[[2, 1, 'amod'],[2, 0, 'det']]]
Rules["NNP CD"] = [[0],[[0, 1, 'num']]]
Rules["IN NP"] = [[0],[[0, 1, 'pobj']]]
Rules["VB NP PP NP"] = [[0],[[0, 1, 'pobj'], [0, 2, 'prep'], [0, 3, 'tmod']]]
Rules["MD VP"] = [[1],[[1, 0, 'aux']]]
Rules["VBZ VP"] = [[1],[[1, 0, 'aux']]]
Rules["VBZ NP"] = [[0],[[0, 1, 'dobj']]]
Rules["JJ NNS"] = [[1],[[1, 0, 'amod']]]
Rules["VB NP"] = [[0],[[0, 1, 'dboj']]]
Rules["NP VP"] = [[1],[[1, 0, 'nsubj']]]
Rules["DT JJ NN NN"] = [[3],[[3, 0, 'det'], [3, 1, 'amod'], [3, 2, 'amod']]]
Rules["NP NP"] = [[0],[[0, 1, 'nmod']]]
Rules["PP NP"] = [[0],[[0, 1, 'pobj']]]
Rules["VBD NP"] = [[0],[[0, 1, 'nobj']]]
Rules["TO VP"] = [[1],[[1, 0, 'aux']]]
Rules["VBP S"] = [[0],[[0, 1, 'xcomp']]]
Rules["DT NNS"] = [[1],[[1, 0, 'det']]]
Rules["VBD PP"] = [[0],[[0, 1, 'prep']]]
```

Figure 1: The rules list

The above diagram shows all the rules used while determining the head of a subtree. The following example will clear the assignment of arrays shown in image:

Rules["VBP NP"] = [[0],[0,1,"dobj"]]

In this assignment, the first 0 tells the location of head, so here, VBP becomes the head when VBP and NP comes together. The second entry, i.e. [[0,1,"dobj"]], tells that a graph edge is there between first position and directed towards second position of the subtree.So here, the edge will be from VBP to NP. Also, "dobj" tells the relation between the two.

1.2 Algorithm

1. Get the constituent Parse tree from Allen Parser
2. Initialise a dependency parse tree with the words as nodes.
3. Get a sub tree of height 3 from bottom of Constituent Parse tree.
4. Find the head of this subtree from the rules as shown in Image.
5. Update the dependency parse tree by making an edge from the head to modifiers found from above step, as well as, updating the edge names, by the relations got by the Rule list.
6. Change the constituent parse tree by replacing the subtree by the head.
7. Go to step 3 and repeat until no subtree of height 3 is found.

1.3 Results

For showing graph we used graphviz library. We tried our parse tree with sentences and tried to compare them with the dependency parse tree obtained from Allen Parser. As many a times, the Allen parser gives erroneous graphs, there was no point calculating accuracy for our algorithm. We first started with easy sentences:
Sentence: The boy runs fast

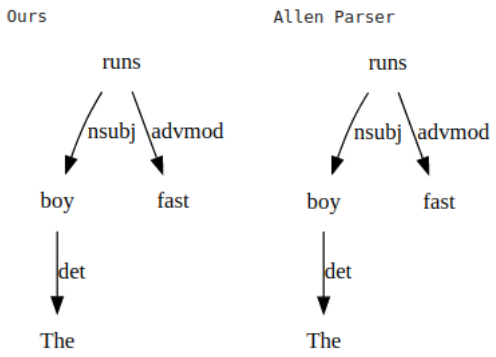


Figure 2: Left:Ours, Right:Allen Parser

Sentence: The car crashed into the tree

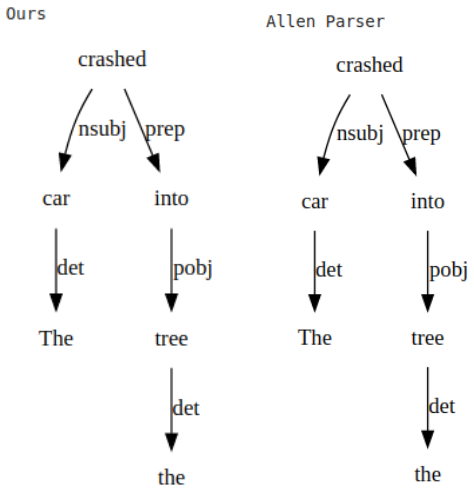


Figure 3: Left:Ours, Right:Allen Parser

We then tried some hard and bigger sentences.
Sentence: The quick brown fox jumps over the lazy dog

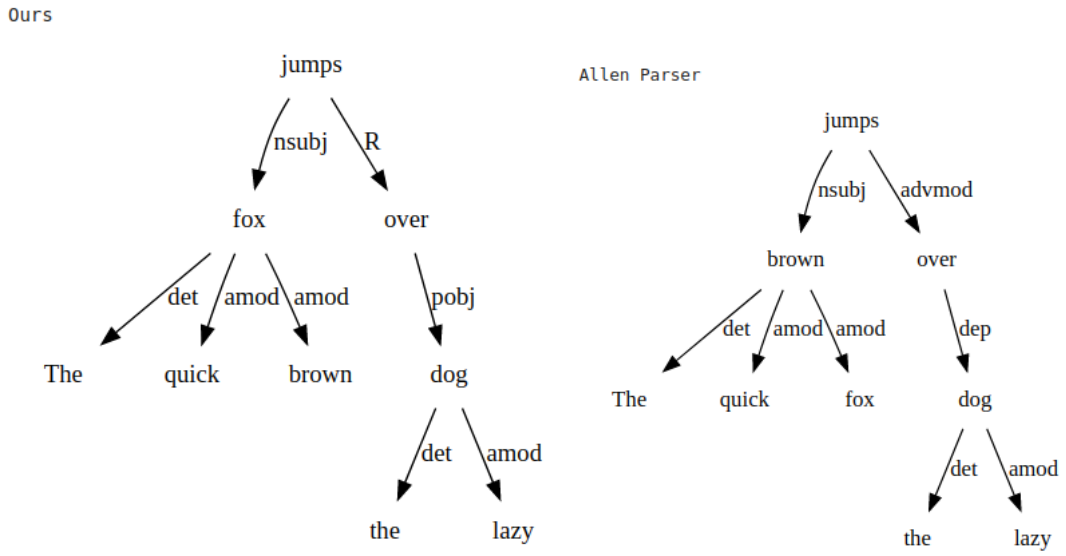


Figure 4: Left:Ours, Right:Allen Parser

In this case, in fact our parser performed better than Allens, as can bee seen, the allen parser has considered ”brown” as the nsubj relation to jumps, which is wrong. Our parser gives the correct parse tree. At the same time, due to limited set of rules, it has given the relation jumps-¿over as R, which is there when the relation is not found in the Rule list. However, using the property of head initialness of English language, we got the correct final parse tree.

1.4 Error Analysis

The Allen Parser has lots of inaccuracies and doesnt give correct dependency parse tree always. Hence we manually checked our accuracy with stanford CoreNLP parse trees for various sentences. We divided the analysis into 2 parts:1) Accuracy on 3-4 word length sentenes, 2)5-9 word length sentences. Following are the number of sentences on which it gave correct parse tress.

Number of words	Number of sentences tested	Number of correct parse tress
3-4	10	9
5-8	10	6

Table 1: Error analysis of CP to DP conversion

Since we have written a limited number of rules, hence sometimes for higher number of words cases, our algorithm produces an incorrect parse tree.

- Since our parser is completely rule based, it will give perfect parse tress for sentences which have the rules we have listed . Right now, we have written rules for all the active voice sentences, hence it works well in them.
- For longer sentences it doesnt perform well, as we have more rules to be written explicitly.
- For interrogative, exclamatory or passive voice sentences, rules are yet to be written.
- For adding any rule, look at the original constituent parse tree and dependency tree (printed in the notebook), and look at the sub-tree where our dependency parse tree doesn't match groundtruth dependency tree and formulate the rule as per the convention mentioned earlier and add it in the second last block of notebook.

1.5 Strengths

- Many a times, our parser performs better than Allen parser.
- The time taken for producing the parse tree is minimal.
- With enough time given, our parser can produce accurate results if all rules are listed down.

1.6 Limitations

- Since its rule based, we can only type in limited set of rules, and hence cant cover all possibilties at any given day.
- Further, since we consider only a subtree of height 3 at any moment, and not care about the effect of far away words, this limits our performance further.
- Also many a times, allen parser itself gives a wrong constituent parse tree, in that case, our final generated dependency parse tree is also wrong.

2 Dependency Parsing to Constituent Parsing

2.1 Approach

Implemented this using spacy module. Used this because it directly gives each and every word's parent's, children, pos in a dependency parse tree in the form of tokens which makes the recursion part much easier. Now given a recursion tree we first find out the location of the root as the left part of the root will be a noun phrase and the right part will constitute the verb phrase. Let the root's index be i. Now that we have separated the sentence into two parts NP(0 - i-1) and VP(i-n), we recursively find out the constituency parse tree.

Noun phrase

- Select "nsubj" dependency and call NP(nsubj)
- Let's say our function is NP(word).
- The base case is:- No child Return nltk.Tree(word.pos_,[word.text])
- If it is punctuation we ignore it
- Recursion:- For all the children of the word if child.i > word.i :- right_tree.append(NP(child)) else :- left_tree.append(NP(child))
- Return nltk.Tree(L,merge(left_tree,right_tree))
- Where L is decided by label of the first word.

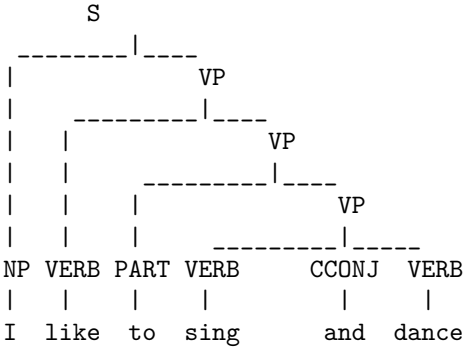
Verb Phrase

- Select "root" dependency and call VP(root)
- Let's say our function is VP(word).
- The base case is:- No child Return nltk.Tree(word.pos_,[word.text])
- If it is punctuation we ignore it
- Recursion:- For all the children of the word if child.i > word.i :- right_tree.append(VP(child)) elif (child.dep_ == 'aux'):- left_tree.append(VP(child))
- Return nltk.Tree(L,merge(left_tree,right_tree))
- Where L is decided by POS of the first word.

2.2 Results

It gives correct parse trees for active voice sentences and sentences not containing negative words like "not". Some examples are as given below.

```
Sentence :- I like to sing and dance
```



```
Sentence :- Fast dog caught the person
                    S
           _________|_____
          |               VP
          |           _____|___
          NP         |         NP
          |          |      ___|____
          NP         |     |        VP
        ____|___     |     |        |
    ADJ      NOUN  VERB  DET      NOUN
     |        |     |     |        |
    Fast     dog  caught the     person

Sentence :- The quick brown fox jumps over the dog.

                   S
          ___________|_______________
         |                           VP
         |                    _______|___
         |                   |           PP
         |                   |     ________|___
         NP                  |    |            NP
         |                   |    |         ___|___
         NP                  |    |        |       VP
    _____|___________        |    |        |       |
DET  ADJ   ADJ  PROPN  VERB ADP      DET     NOUN
 |    |     |     |     |    |        |       |
The quick brown  fox  jumps over     the     dog
```

2.3 Error Analysis

Same sentences that were used for CP to DP have been used here. Sentences used:-

```
list_sent = ['Three people laughed always',
    'I play football',
    'I like to drink',
    'Car is running fast',
    'India is best country',
    'We work hard',
    'I trust you',
    'Winners never make excuse',
    'You helped me',
    'Strive for excellence']
list_sent2 = [
    'I like to sing and dance',
    'Fast dog caught the person',
    'The quick brown fox jumps over the lazy dog',
    'Big vehicles block the city road',
    'I am testing the accuracy of parse tree',
    'Blunders have occured when we are careless',
    'I pretend to laugh always',
    'It is a sunny day and I was sleeping',
    'There was a jaguar behind the car',
    'The fast athlete won the race'
    ]
```

Number of words	Number of sentences tested	Number of correct parse tress
3-4	10	8
5-8	10	4

Table 2: Error analysis of DP to CP conversion

The most common mistakes are when the sentence is in passive voice, the nsubj dependency comes after the verb or the nsubj is completely missing from the sentence.

Limitations Not yet able to convert when nsubj does not come before the verb (like passive sentences, only verb phrases, etc). Also whenever a negative word comes it does not show up in the tree. Some examples are given below:-

```
Sentence :- This should not be the case.


    this nsubj be 0 [] DET
    should aux be 1 [] VERB
    not neg be 2 [] PART
    be ROOT be 3 [this, should, not, case] AUX
    the det case 4 [] DET
    case attr be 5 [the] NOUN
                 S
      ___________|___
     |               VP
     |       ________|___
     |      |            VP
     |      |       _____|___
     |      |      |         NP
     |      |      |       __|___
     |      |      |      |      VP
     |      |      |      |       |
     NP   VERB   AUX    DET     NOUN
     |      |      |      |       |
    this should  be     the     case
```

For passive sentences the code gives an error saying it could not find the noun phrase of the sentence. For example :- ”Work was not completed by him”.