NLP Assignment 2 Report

authors & contributors:
Petrus Kiendys, Amin Abu-Sneineh,
Oskar Berntorp, Azddin Bezberkan

Carrying out the experiments

Choose two (reasonable) sentences from the training set, and draw the tree structure by hand
Answer: We chose the following two sentences:

- “Individuell beskattning av arbetsinkomster” (line 1 from the files containing bracketing formatted phrase structure trees)
- “Kommunalskatteavdraget slopas” (line 17 from the files containing bracketing formatted phrase structure trees)

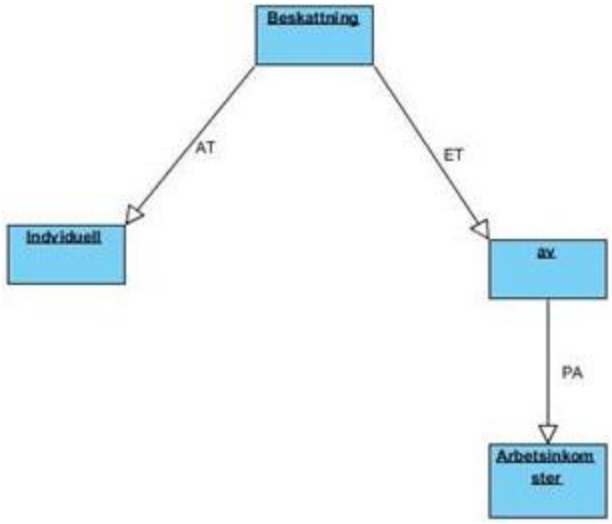

Fig. 1: Sentence 1 presented as a dependency tree

Fig. 2: Sentence 2 presented as a dependency tree

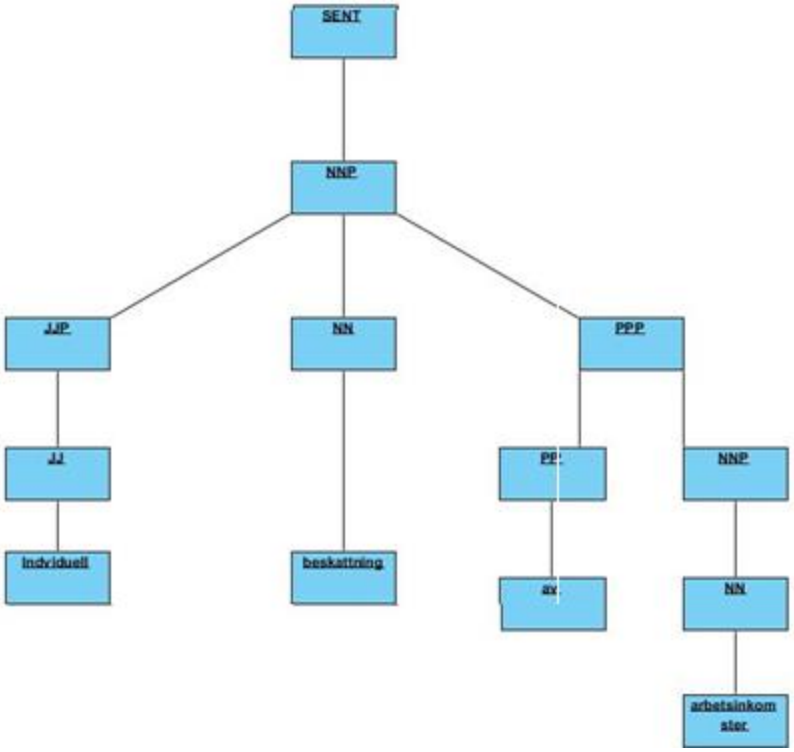

Fig. 3: Sentence 1 presented as a cfg-pos style tree

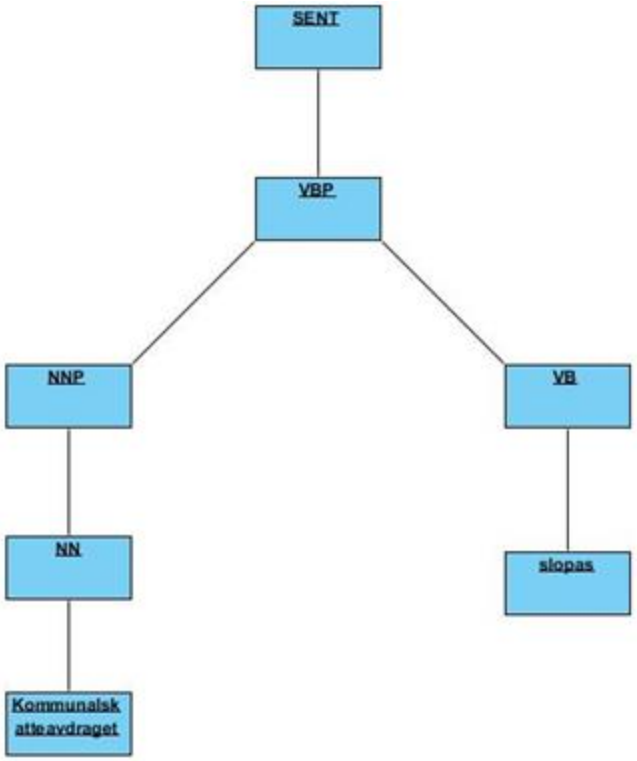

Fig. 4: Sentence 2 presented as a cfg-pos style tree

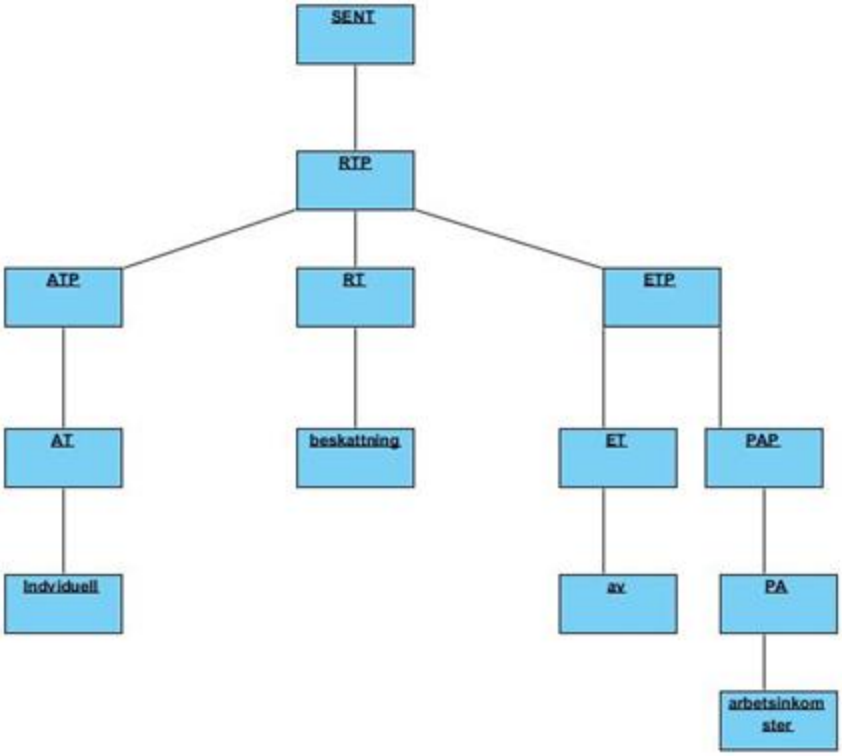

Fig. 5: Sentence 1 presented as a cfg-dep style tree

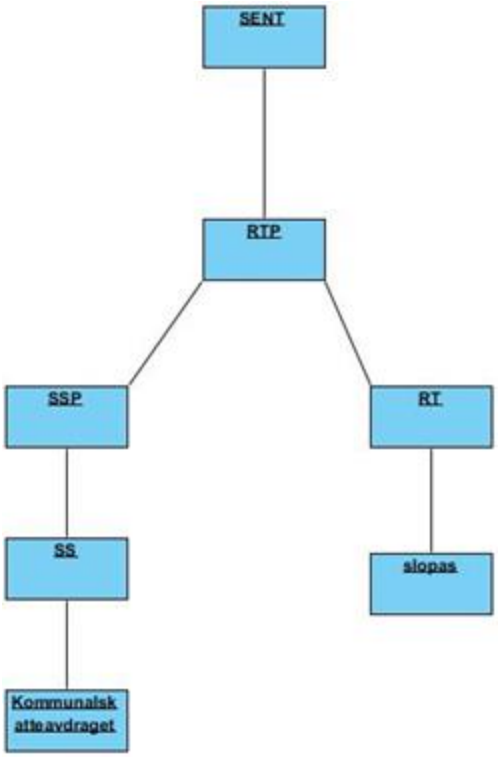

Fig. 6: Sentence 2 presented as a cfg-dep style tree

Does this algorithm do a good job of creating phrase-structure trees from a linguistic point of view?

Answer: It does not (or rather, the pseudocode describing it is too vague and does not define all operations that need to be executed in order to generate the same phrase-structure trees that are presented in the .cfg files). On the following images we can see sentence 1 being transformed from a dependency tree into a phrase-structure tree:

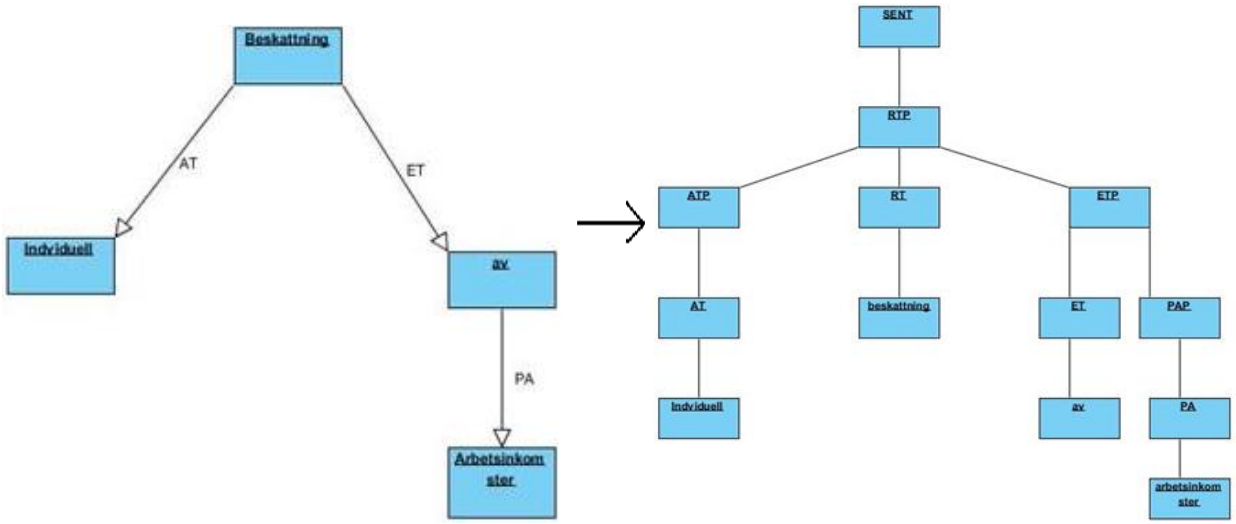

Fig. 7: Transformation of sentence 1 from a dependency tree to a phrase-structure tree through Algorithm 1 (using undefined operations)

The following images show the general idea of how dependency trees are presented as phrase-structure trees according to Algorithm 1:

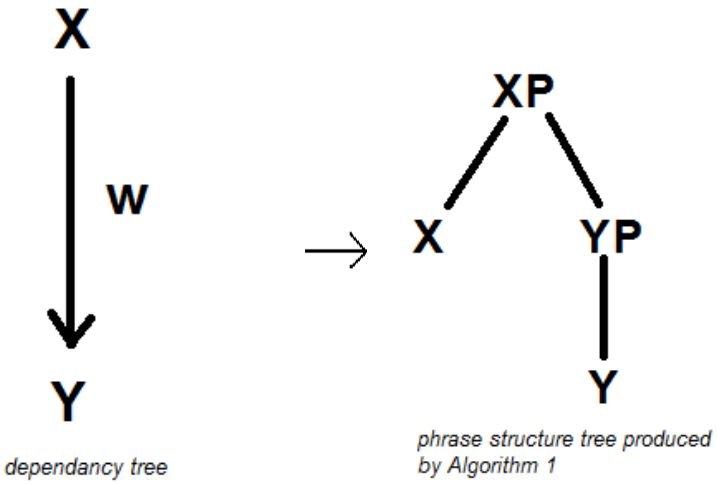

Fig. 8: General description of transformation/presentation of a phrase-structure tree from a dependency tree through Algorithm 1

As we can see, by comparing the tree produced by algorithm 1 to the phrase structure trees of the bracketing formatted files, the algorithm is too simplistic and does a poor job of creating proper phrase structure trees.
It's important to note that phrase-structure trees adhere to constituency relations and dependency trees adhere to dependency relations. As such it is hard to transform a dependency tree to a phrase structure tree with simplicity.
Furthermore, the algorithm only specifies how nodes should be manipulated in the input dependency tree and does not specify how the dependency label that describe the relationship between each node should be presented on the phrase-structure tree. Simply adding a "P" suffix to a non-terminal node does not properly solve this issue as the label contains a word/terminal that was transformed from the dependency tree rather than a having a label consisting of a phrase (such as VB, NP, etc.).
As I explained earlier, some operations are not well-defined. The operation of "change node into a constituent node, and add a "P" suffix to its label" is what comes to mind. Does this operation entail taking the dependency relation label (placed on edges between vertices) and applying it as the value/label of the node?
Lastly, appending the SENT node/label for the whole tree is also omitted from the pseudocode.

With all of this said, despite the fact that the pseudocode might be too vague or incomplete to describe the process of obtaining phrase-structure trees (contained in the .cfg files) from dependency trees (contained in the .conll files), assuming that it does [describe the complete process], it does a pretty good job at creating phrase-structure trees.
The relationships between the nodes presented in the dependency tree are still intact and present in the phrase-structure tree. Due to structuring nodes with constituents we can see that the phrase-structure trees are more fine-grained and as a result will produce more grammar rules.

Can you think of any improvements? Explain why this would be an improvement.
Answer: I'm not sure if there's any improvements that can be applied as far as to present a dependency tree in the structure of a phrase-structure tree.
If the sole goal of this task is to present a dependency tree as a phrase-structure tree then there is one "optimization" that comes to mind. Removing P-suffixed nodes/labels would produce a more coarse-grained phrase-structure tree that would bare more resemblance to a dependency tree as dependency trees are more compact/coarse-grained than constituency-based trees.
However, if the task of presenting a dependancy tree in the structure of a phrase-structure tree also entails adhering to the paradigm of constituency-based trees then the "optimized" or trimmed phrase-structure tree is faulty.
As a side-note this would reduce the amount of grammar rules generated by rule counters.

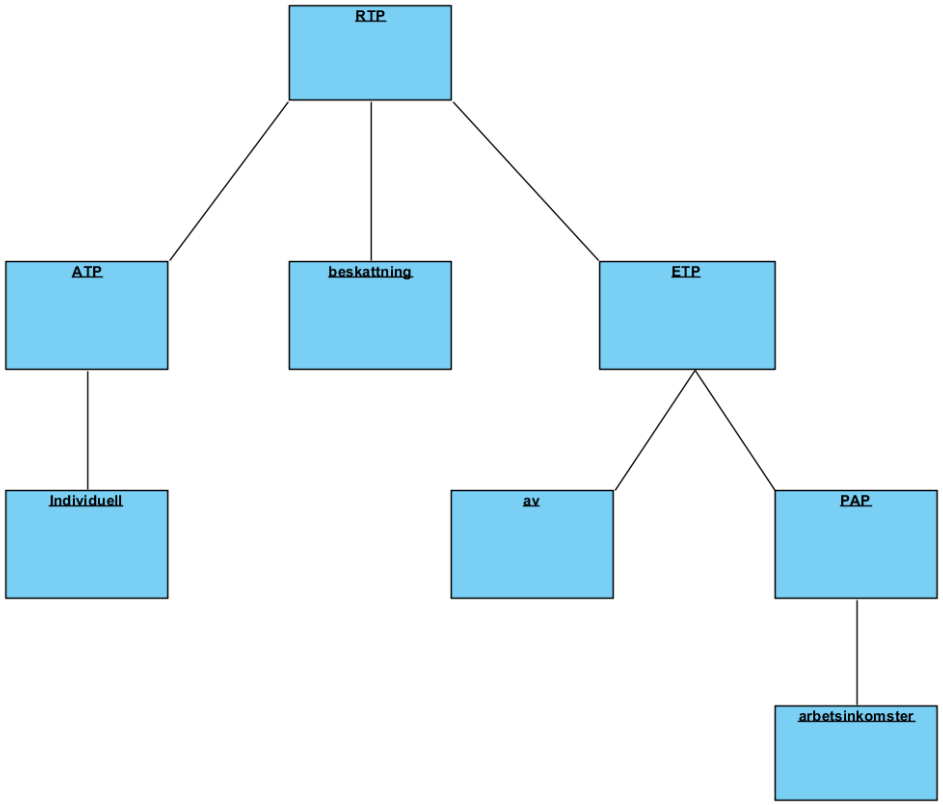

Fig. 9: Transformation/presentation of a phrase-structure tree from a dependency tree through Algorithm 1 with suggested optimization

How many different rules (grammar rules and lexicon rules) do extract for each parsing model?
Answer:
For parsing model dep we extracted 4489 grammar rules and 20618 lexicon rules.
For parsing model pos we extracted 3373 grammar rules and 13920 lexicon rules.

*Why do you think there such a big difference in number of rules between the two models (i.e., compare counts extracted from [**cfg-pos**] with those extracted from [**cfg-dep**])?*
Answer: The key difference between generating grammar rules for the two different models is that parsing model dep always assigns a RTP label to the root node whereas parsing model pos assigns a part-of-speech tag. The predetermined construction of an RTP label for the root node will produce more grammar rules.
As for the difference in number of lexicon rules, this can be attributed to two main causes. First off, there is a greater number of unique labels for dependancy relations (field 8 of .conll file) than there are unique labels for part-of-speech tags (field 4 of .conll file). Secondly, there are more unique rules produced by the dep parsing model than by the pos parsing model. The following lines serve as examples of this:
line 12267 of lexicon-pos.txt: rollen NN 4
line 12334 of lexicon-dep.txt: rollen SS 1 PA 1 OO 2

Have a look at the output of the parser. Do you notices anything overtly incorrect?
Answer: Yes, there are a lot of sentences provided with test_input.txt that could not be parsed based on the grammar and lexicon rules that we trained BitPar on. These sentences/lines start with “No parse for: [SENT]” where [SENT] is the sentence that could not be parsed.

Evaluation

There is a lot of output of evalb. At the end of the output, is the overall summary and you are mainly interested in precision, recall and f1-measure. How do your parser models do?
Answer: Our parser models performed differently, the different results are presented below:

- Parsing model pos
  - precision: 33.25
  - recall: 33.15
  - f1-measure: 33.20
- Parsing model dep
  - precision: 46.34
  - recall: 45.87
  - f1-measure: 46.10

Given this data, we can conclude that our parser models are not doing too well. BitPar parsing results typically have an f1-measure of around 70% according to your instructions, however you have also specified that this f1-measure is achieved when parsing English on Penn treebank data. Due to the fact that we’re working with Talbanken data, the results may differ.

Error Analysis

Use the explanations on what the dependency and pos labels are for Talbanken. Consider the previous discussion in your report and try to explain the parse results (this should be at least a half page of explanations).

Answer: First off, let's take a look at the number of sentences that are evaluated for both models. We're sending in 1145 sentences for both models including invalid blank sentences which will be skipped during the evaluation process. The parse of pos contains 612 valid sentences which will be evaluated whereas the parse of dep only contains 102 sentences which will be evaluated and compared to the sentences of the goldfile.

We can see that the dep parsing model performs better than the pos parsing model. The precision, recall and f1-measure evaluation metrics are about 13% higher for the dep parsing model. This is of course due to the previously mentioned fact that there is only a small subset of sentences in the dep goldfile that will actually be evaluated and compared to the candidate parse.

When we take a look at the other metrics of the evaluation we notice something strange. The pos parsing model has a tagging accuracy of 72.37, significantly higher than the tagging accuracy of 38.10 for the dep parsing model. This further validates our previous claim that the amount of unique labels for the pos parsing model is by far less than the amount of unique labels for the dep parsing model. This results in BitPar accurately assigning part-of-speech tags to the parsed pos candidate file, whereas the parsing of the dep candidate file is a bit more difficult as there are more rules and labels to choose from. We should note that the -v flag of the BitPar command tells BitPar to perform a Viterbi parse and will output the most probable parse, perhaps it is the effect of this that we see in the difference of the tagging accuracy.

Looking at the metric for "complete match" we can see that the evaluation of the pos parsing model has a 4.58 complete match, while the evaluation of the dep parsing model has a 31.37 complete match. Again this is mainly due to the fact that the dep parse model contains 102 valid sentences that will be evaluated out of a total 1145 (which could not be parsed).

The crossing metrics can also be analyzed in a similar manner, however all of the metrics of the dep parsing model seem to be a consequence of the fact that only 102 valid sentences are compared and 1043 invalid blank sentences are skipped. This inflates the evaluation metrics of dep and makes it hard to reason and compare the different models with each other.