Natural Language Processing Homework 4

Harry Qi, Zike Hu

Question 1

(a) An interesting sentence:

All the faith he had had had had no effect on the outcome of his life.

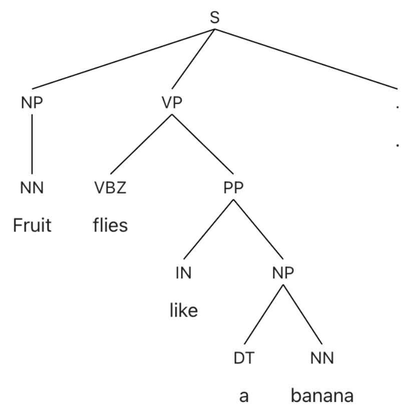
We have tried AllenNLP Parser and Berkeley Neural Parser on this sentence; what is interesting about this sentence is that these two parsers gave us two different sentence structures, where AllenNLP Parser thinks it is (All the faith (he had)) (had(had(had no effect on the outcome of his life))) and Berkeley Neural Parser thinks it is (All the faith (he had had had))(had no effect on the outcome of his life); these are two different explanation of tense whether he "had" all the faith or he "had had had" all the faith so that the tense explanation to "effect" is also different ("had had had" no effect or "had" no effect).

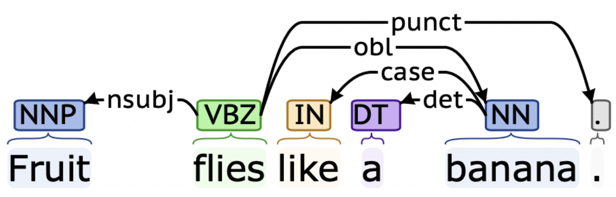
(b) The wrong parser:

Fruit flies like a banana.

We have tried all three parsers, but all of them misunderstand this sentence. They all think *Fruit* is a Noun, flies is a Verb, and meanwhile, *like a banana* is a Noun Phrase, in which *like* is regarded as a Preposition.

However, in fact, fruit cannot fly. Therefore, the correct understanding of this sentence is that the insect, *Fruit flies*(Noun Phrase)*, like*(Verb) *a banana*(Noun Phrase).



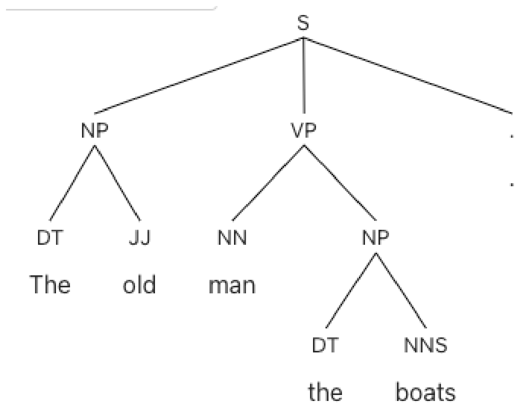


This is because both of these structures are right in syntax. However, the latter one makes sense in semantics. So, we think that when a parser tries to parse this kind of sentence, it might not choose the most "authentic" sentence structure based on their semantic meaning.

(c) "Adversarial" sentences:

1) The young man the ticket booth.

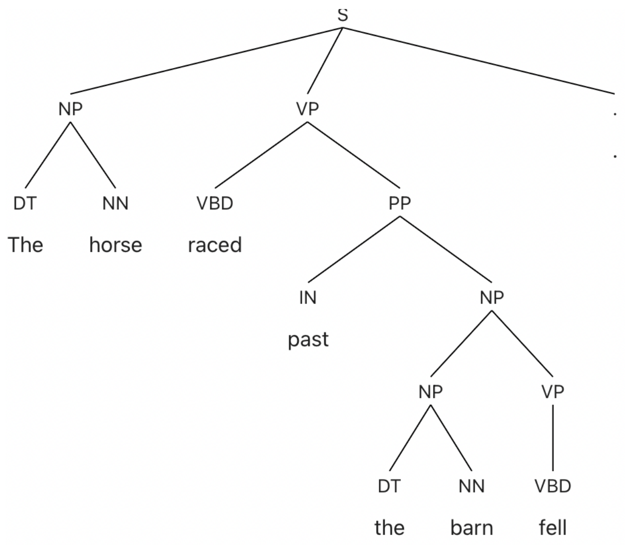
Explanation: It seems that we do not have any verbs in the sentence, but "man" could be a verb that means to serve in the force or complement of, so it is grammatically correct.



2) The horse raced past the barn fell.

Explanation: The sentence is designed with multiple verbs and the predicate *fell* is far from the subject *The horse.* On top of that, the clause *raced past the barn* omits the conjunction *that*, therefore, the subject *The horse* is followed by the verb in the clause directly.

True understanding: The horse (that was raced past the barn) fell.

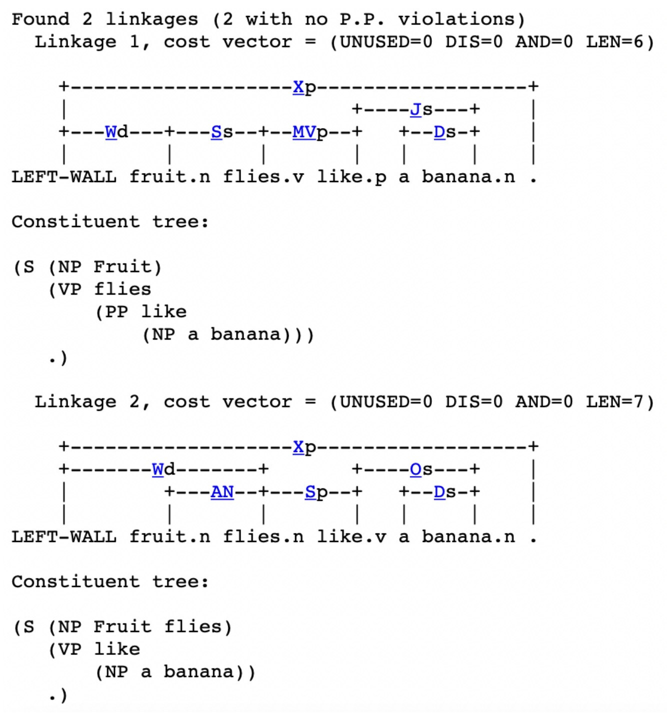


Question 2.

(a) Using all example sentences from question 1, we have found some interesting things by using **Dependency Grammar** and **Link Grammar** (the other two won't work):

(i) We have tried to use both parsers to parse the sentences we used in question in 1(a): *All the faith he had had had had no effect on the outcome of his life*. The Link Grammar parser returns 12 different meanings! But, the Dependency Grammar parser only returns one meaning that we did not interpret in question 1(c).

(ii) The example from 1(b) *Fruit flies like a banana.*
The Dependency Grammar Parser's result is the same as the previous model. However, the Link Grammar Parser performs much better than them. This parser is able to find out the correct parsing as expected and shows all the potential parsing matching the syntax.

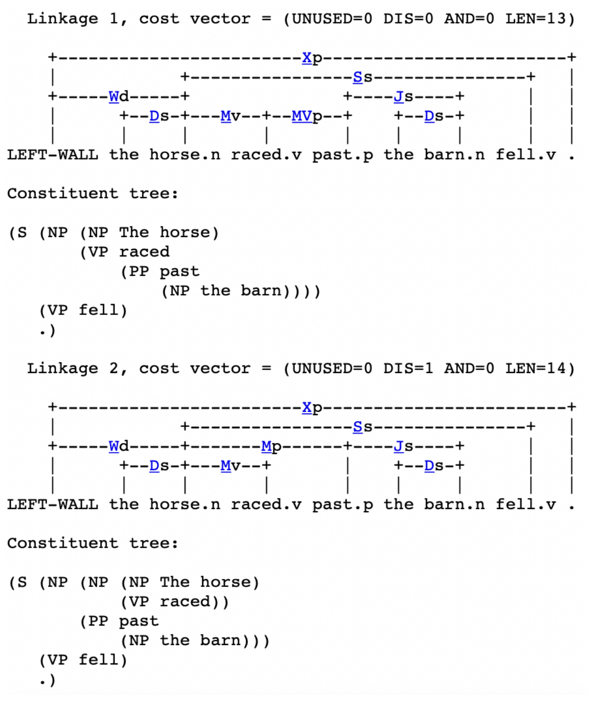


```
Found 2 linkages (2 with no P.P. violations)
  Linkage 1, cost vector = (UNUSED=0 DIS=0 AND=0 LEN=6)

    +-------------------Xp------------------+
    |                        +----Js---+    |
    +---Wd---+---Ss--+--MVp--+   +--Ds-+    |
    |        |       |       |   |     |    |
LEFT-WALL fruit.n flies.v like.p a banana.n .

Constituent tree:

(S (NP Fruit)
   (VP flies
       (PP like
           (NP a banana)))
   .)

  Linkage 2, cost vector = (UNUSED=0 DIS=0 AND=0 LEN=7)

    +-------------------Xp------------------+
    +-------Wd-------+       +----Os---+    |
    |        +---AN--+---Sp--+   +--Ds-+    |
    |        |       |       |   |     |    |
LEFT-WALL fruit.n flies.n like.v a banana.n .

Constituent tree:

(S (NP Fruit flies)
   (VP like
       (NP a banana))
   .)
```

(iii) Playing with the first example from question 1(c), "*The young man the ticket booth*.", we found that Link Grammar could parse the correct meaning without returning any other meaning like simply combining two noun phrases together as one sentence.

(iv) 1(c) *The horse raced past the barn fell.*

The performance of Dependency Grammar Parser is surprisingly good. It not only parses the right predicate *fell*, but also raises two different parsings that both make sense, which we did not find out before.



```
  Linkage 1, cost vector = (UNUSED=0 DIS=0 AND=0 LEN=13)

    +------------------------Xp------------------------+
    |            +----------------Ss---------------+   |
    +-----Wd-----+               +----Js----+      |   |
    |      +--Ds-+---Mv--+--MVp--+    +--Ds-+      |   |
    |      |     |       |       |    |     |      |   |
LEFT-WALL the horse.n raced.v past.p the barn.n fell.v .

Constituent tree:

(S (NP (NP The horse)
       (VP raced
           (PP past
               (NP the barn))))
   (VP fell)
   .)

  Linkage 2, cost vector = (UNUSED=0 DIS=1 AND=0 LEN=14)

    +------------------------Xp------------------------+
    |            +----------------Ss---------------+   |
    +-----Wd-----+-------Mp------+----Js----+      |   |
    |      +--Ds-+---Mv--+       |    +--Ds-+      |   |
    |      |     |       |       |    |     |      |   |
LEFT-WALL the horse.n raced.v past.p the barn.n fell.v .

Constituent tree:

(S (NP (NP (NP The horse)
           (VP raced))
       (PP past
           (NP the barn)))
   (VP fell)
   .)
```

(b) We have used **Stanford Parser (Both)** and **Dependency Grammar Parser (Chinese)** to parse Chinese and Japanese.

(i) Japanese

We found that Dependency Grammar Parser could parse most sentences correctly, while sometimes it would misunderstand some words properties.

For example, 父は、生の魚を売ることを生業にしている, meaning that (my) father makes living by sellig raw fish, "生の魚" here means raw fish so that "生の" should be a adjective, while parser thinks "生" is a noun (while it is usually not a noun) and "の" as an adpositional phrase; the parser only interpret "生の" together as a case of fish being alive, which is technically right but not in exact as the authentic usage.

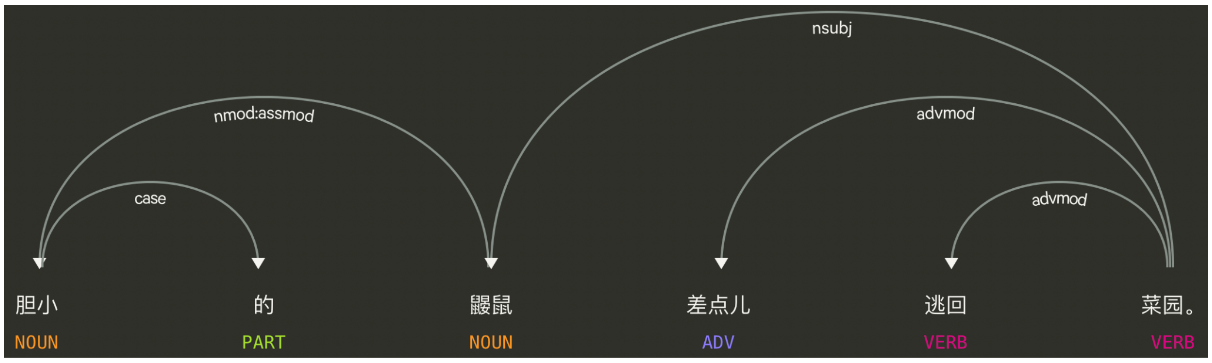
Besides, "は" is equivalent to "be" in English, but the parser interpreted it as adpositional phrase as well, which is confusing (should be a verb).

Finally, "生業にしている" here is equivalent to "making live"; "している" is rather a present progressive tense of "do", where its original form "do" is "する"; however, the parser split "している" into three part "し"(Verb) + "て" (subordinating conjunction) + "いる"(Verb), which is kind of weird, but it could be somehow make sense given how the overall Japanese structures were formed.

Compared to English, there are some novel phrase categories in Japanese, like AUX(auxiliary phrase), for supporting this language context.

(ii) Chinese

There are many mistakes in the results of Dependency Grammar Parser. For example, it even parses the Noun Phrase 菜园 as a Verb as the picture shows below.



However, the Stanford Parser's performances are much better. Although there are still some mistakes when encountering sentences with complicated grammars or expressions, they perform really well on the sentence with the grammar rules different from English.

The styles of the parse trees in Chinese look similar in English, but empirically, the number of tree layers in English is more than in Chinese. And sometimes there's no verb in Chinese.

We think the reasons for those differences might lie in that Mandarin Chinese is an isolating language, i.e. one-to-one correspondence words/morphemes. Therefore, there is no space between different words. Thus, we need to segment the words from sentences first, which might cause some ambiguities. Mistaking 菜园 as Verb Phrase might be on account of that. In addition, the sentences in Chinese cannot contain the predicate. Thus, Dependency Grammar Parser seems like it is using English grammar to parse Chinese sentences, which leads to wrong parsing.

Question 3.

(a) To keep track of the item's best parse and total weight of this best parse, we added backpointer, which is initialized to be list of None that has length that is equal to the rule's right hand side, and weight, initialized to be the rule's weight unless it is a children of previous item (in that case weight would be the rule's weight + the weight got from the previous item), to each item. Each item has its own backpointer so that even if we delete the duplicate item from the column, the new item with less weight would not be affected.

(b) Our parse.py should take no more than O(n^3) space complexity and O(n^2) space complexity. Before pushing a new item, we first check the self.col[position]._index, which is a dictionary, and check if there is a matched item in the same column (agenda) of the parse chart, then compare the weight. If the new item carries lighter weight, we would mark the old item's flag to False, which means we no longer need to process this item later; if the weight is heavier, we do not push it to the column. If the new item is not present in self.col[position]._index, meaning this is a new item, we then push it as usual.

As explained, if the new item, before pushing, is already existed in the _index dict and has less weight, we would just push it to this column; it only takes O(1) time to check and push it, which is append to the __item list and add the item to the __index dictionary. It is necessary because if we only need to search four rules in a column but we have 1000 items, the worst case is that we need to iterate through all 1000 items to get these four rules, which is inefficient.

Question 4.

(a) Parse analysis.

John is happy.

```
(ROOT (S (NP (NPR (NNP John)))
        (VP (VBZ is)
           (ADJP-PRD (JJ happy)))
        (PUNC. .)))
34.22401061796059
```

As the parse shown above, the result of parsing seems fair and follows the grammar strictly. However, because of the limitation of the grammar, the parse and comprehension of the sentences seems a little bit complicated and disalternative.

(b) Speedups analysis.

E.1 Batch duplicate check

We add item index to our _lhs_index dictionary when we predict a nonterminal, so if we have another same nonterminal to predict later, and it will be already existed in the dictionary, so we do not need to predict all expansions for this nonterminal again.

E.5 Indexing customers

We create a dictionary to store the next symbol of the item(key) and its index in Agenda(value), then we also create a function to take the item through its index. Therefore, the time complexity declines from O(n) to O(1), when we search the customers for attaching methods.

Although we used the PyPy JIT compiler for speedup, we still ran parse.py over 1 hour. After adding these speedups methods, the run time of our program decreased sharply. Eventually, we only ran for only around 16 mins.