Computational Linguistics: Dynamic and Statistical Parsing

Raffaella Bernardi

CIMeC, University of Trento

e-mail: bernardi@disi.unitn.it

Contents

1. Done and to be done

We have seen:

- Top-down and bottom-up parsing
- Problem of top-down with Left recursive rules
- Back-tracking
- Depth vs. Breath first
- Overgeneration

Today, we will look into:

- Left corner parser
- Probabilistic parsers
- Demo with NLTK on left corner parser

2. Dynamic Programming

To cope with amibiguity efficiently, several algorithms do not derive the same sub-analysis by the same set of steps more than once.

They do so by storing derived sub-analysis in a well-formed substring table or **chart** and retrieving entries from the table as needed rather than recomputing them.

This is an instance of a general technique known as **dynamic programming**.

2.1. Re-known chart parsing algorithms

- CKY (Cocke-Kasami-Younger) (Kasami 1965; Younger 1967, Cocke 1970). Bottom-up. Demo: http://martinlaz.github.io/demos/cky.html
- Earley (Earley 1968). Top-down.
- Left-corner parsing

2.2. Left-corner parsing: Using both bottom-up and top-down approaches

We have seen that using a pure **top-down approach, we are missing some important information provided by the words** of the input string which would help us to guide our decisions.

However, similarly, using a pure **bottom-up approach, we can sometimes end up in dead ends** that could have been avoided had we used some bits of top-down information about the category that we are trying to build.

The key idea of **left-corner parsing** is to **combine top-down processing with bottom-up processing** in order to avoid going wrong in the ways that we are prone to go wrong with pure top-down and pure bottom-up techniques.

2.3. Left Corner of a rule

The left corner of a rule is the first symbol on the right hand side.

For example,

- np is the left corner of the rule $s \rightarrow$ **np** vp, and
- iv is the left corner of the rule $vp \rightarrow$ **iv**.
- Similarly, we can say that "vincent" is the left corner of the lexical rule $pn \rightarrow$ **vincent**.

2.4. Left Corner parser

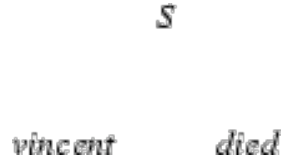
A left-corner parser starts with a **top-down prediction** fixing the category that is to be recognized, like for example s. Next, it takes a **bottom-up step** and then alternates bottom-up and top-down steps until it has reached an s.

1. The bottom-up processing steps work as follows. Assuming that the parser has just recognized a noun phrase, it will in the next step look for a rule that has an np as its left corner.
2. Let's say it finds $s \rightarrow np\ vp$. To be able to use this rule, it has to recognize a vp as the next thing in the input string.
3. This imposes the top-down constraint that what follows in the input string has to be a verb phrase.
4. The left-corner parser will continue alternating bottom-up steps as described above and top-down steps until it has managed to recognize this verb phrase, thereby completing the sentence.

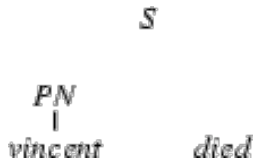
2.5. Example

Now, let's look at how a left-corner recognizer would proceed to recognize "vincent died".

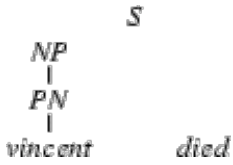
1. Input: vincent died. Recognize an s. (Top-down prediction.)



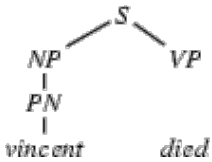
2. The category of the first word of the input is pn. (Bottom-up step using a lexical rule $pn \rightarrow$ **vincent**.)



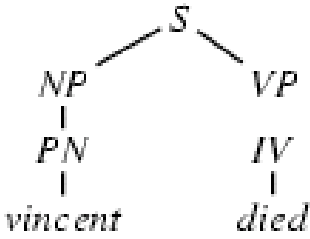
3. Select a rule that has *pn* at its left corner: $np \rightarrow$ **pn**. (Bottom-up step using a phrase structure rule.)



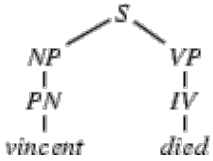
4. Select a rule that has *np* at its left corner: $s \rightarrow$ **np** *vp* (Bottom-up step.)



5. Match! The left hand side of the rule matches with , the category we are trying to recognize.



6. Input: died. Recognize a *vp*. (Top-down prediction.)
7. The category of the first word of the input is *iv*. (Bottom-up step.)
8. Select a rule that has *iv* at its left corner: $vp \rightarrow iv$. (Bottom-up step.)



9. Match! The left hand side of the rule matches with *vp*, the category we are trying to recognize.

Make sure that you see how the steps of bottom-up rule application alternate with top-down predictions in this example. Also note that this is the example that we used earlier on for illustrating how top-down parsers can go wrong and that, in contrast to the top-down parser, the left-corner parser doesn't have to backtrack with this example.

2.6. What did we improve and what not?

This left-corner recognizer handles the example that was problematic for the pure top down approach **much more efficiently**.

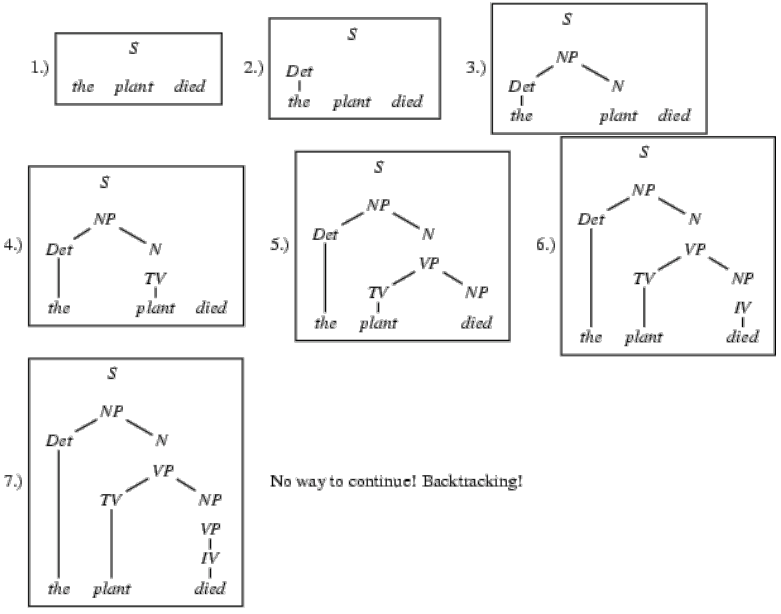
It finds out what is the category of "vincent" and then doesn't even try to use the rule $np \rightarrow det\ n$ to analyze this part of the input. Remember that the top-down recognizer did exactly that.

But there are **no improvement** on the example that was problematic for the **bottom-up** approach: "the plant died". Just like the bottom up recognizer, the left-corner recognizer will first try to analyze "plant" as a transitive verb.

Let's see step by step what the left-corner recognizer defined above does to process "the plant died" given the grammar.

Try it first your self.

2.7. Solution


1.)
S
the plant died
2.)
S
Det
the plant died
3.)
S
NP
Det N
the plant died
4.)
S
NP
Det N
TV
the plant died
5.)
S
NP
Det N
VP
TV NP
the plant died
6.)
S
NP
Det N
VP
TV NP
IV
the plant died
7.)
S
NP
Det N
VP
TV NP
VP
IV
the plant died
No way to continue! Backtracking!

2.8. Comments

So, just like the bottom-up recognizer, the left-corner recognizer chooses the wrong category for "plant" and needs a long time to realize its mistake.

However, the left-corner recognizer provides the information that the constituent we are trying to build at that point is a "noun". And **nouns can never start with a transitive verb** according to the grammar we were using.

If the recognizer uses this information, it would notice immediately that the lexical rule relating "plant" to the category transitive verb cannot lead to a parse.

2.9. Left Corner Table

The solution is to record this information in a table.

This left-corner table **stores which constituents can be at the left-corner of which other constituents**.

For the little grammar of the problematic example the left-corner table would look as follows:

```
s ---> np vp
np ---> det n
vp ---> iv
vp ---> tv np
tv ---> plant
iv ---> died
det ---> the
n ---> plant
```

s	np, det, s
np	det, np
vp	iv, tv, vp
det	det
n	n
iv	iv
tv	tv

3. Statistical Parsing

Chart parsers are good in representing ambiguities in an efficient way, but they don't resolve them.

A probabilisitic parser computes the probability of each internpretation and choose the most probable interpretation.

Most modern parsers are probabilistic.

3.1. Probabilistic CFG (PCFG)

A CFG in which its re-writing rules are associated with a probability.

$$A \rightarrow \beta \quad [p]$$

where p expresses the probability that the given non-terminal A will be expanded to the sequence β. It is the probability of a given expansion (right-hand-side) β given the left-hand-side (LHS) A:

$$P(RHS|LHS)$$

The sum of all the possible expansions of a non-terminal must be 1.

3.2. Example of PCFG

```
S -> NP VP           [.80]
S -> Aux NP VP       [.15]
S -> VP              [.05]
NP -> Pronoun        [.35]
NP -> Proper-Noun    [.30]
NP -> Det Nominal    [.20]
NP -> Nominal        [.15]
etc..
```

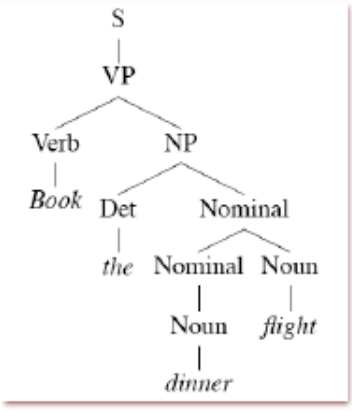
3.3. Probability of a parse tree

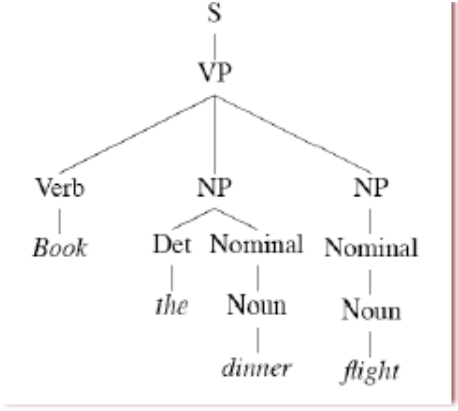
The probability of a particular parse tree T is defined as the product of the probabilities of all the n rules used to expand each of the n non-terminal nodes in the parse tree T:

$$P(T, S) = \prod_{i=1}^{n} P(RHS|LHS)$$

3.4. Example of the probability of parse trees

"Book the dinner flight" can be parsed into two ways: (a) book a flight that serves dinner (left) vs. "book a flight on behalf of the dinner"





3.5. Example of the probability of parse trees

Given the grammar below, the left tree is the more probable one:

	Rules		P		Rules		P
S	→	VP	.05	S	→	VP	.05
VP	→	Verb NP	.20	VP	→	Verb NP NP	.10
NP	→	Det Nominal	.20	NP	→	Det Nominal	.20
Nominal	→	Nominal Noun	.20	NP	→	Nominal	.15
Nominal	→	Noun	.75	Nominal	→	Noun	.75
				Nominal	→	Noun	.75
Verb	→	book	.30	Verb	→	book	.30
Det	→	the	.60	Det	→	the	.60
Noun	→	dinner	.10	Noun	→	dinner	.10
Noun	→	flights	.40	Noun	→	flights	.40

$$P(T_{left}) = .05 * .20 * .20 * .20 * .75 * .30 * .60 * .10 * .40 = \mathbf{2.2 \times 10^{-6}}$$

$$P(T_{right}) = .05 * .10 * .20 * .15 * .75 * .75 * .30 * .60 * .10 * .40 = \mathbf{6.1 \times 10^{-7}}$$

3.6. Learning PCFG rule probabilities: Treebank

A syntactically annotated corpus is called a **treebank**.

Several treebanks have been created generally by the use of a parser to automatically parse each sentence, followed by the use of linguists to hand-correct the parses.

Treebanks are useful also to evaluate parsers.

3.7. Problems with PCFGs: Poor independence assumptions

Poor independence assumptions: CFG rules impose an independence assumption on probabilities, resulting in poor modeling of structural dependencies across the parse tree.

For example, in English NPs that syntactic subjects are far more likely to be pronouns vs. NPs that are syntactic objects are far more likely to be non-pronominal (e.g. a proper nouns).

In Switchboard corpus, Subject 91% pronoun vs. 9% non-pronoun. Object 34% pronoun vs. 66% non-pronoun; though the probabilities of the re-writing rules from NP are:

$$NP \rightarrow DT\ NN[.28]$$
$$NP \rightarrow PRN[.25]$$

PCFGs don't have a way to represent that the 2nd rule in subject position should go up to .91, while in object position the probability of the 1st rule should go up to

.66.

3.8. Problems with PCFGs: Lack of lexical conditioning

Lack of lexical conditioning: CFG rules don't model syntactic facts about specific words, leading to problems with subcategorization ambiguities, preposition attachment, and coordinate structure ambiguities.

(a) Workers [dumped [sacks]$_{NP}$ [into a bin]$_{PP}$]$_{VP}$ vs. (b) Fishermen caught [tons of herring]$_{NP}$

The preference of (a) VP attachment vs. (b) NP attachment could be required by the affinity between the noun **dumped** and the preposition **into** that is greater than the one between **sacks** and the preposition **into**. While **tons** occurs with **of** more often than what **caught** does.

We would need a parser that deals with this lexical dependency statistics for different verbs and prepositions.

4. Re-known parsers

- Collins, 1999
- Charniak parser, 1997
- Stanford Parser (Klein and Manning 2003) Online demo: http://nlp.stanford.edu:8080/parser/
- DG: Standford Dependency Parser (D. Manning et ali. 2003-2014)
- CCG: C&C parsers and the CCG bank: http://groups.inf.ed.ac.uk/ccg/software.html

And of course with Neural Network Models:

- DG: Danqi Chen and Christopher D Manning. 2014. A Fast and Accurate Dependency Parser using Neural Networks. Proceedings of EMNLP 2014
- CCG: Misra and Artzi, EMNLP 2016 Neural Shift Reduce Parser for CCG Semantic Parsing

5. Parsers evaluation

Readings:

"Parser Showdown at the Wall Street Corral: An Empirical Investigation of Error Types in Parser Output", Jonathan K. Kummerfeld, David Hall, James R. Curran, and Dan Klein, EMNLP 2012

and

https://tech.grammarly.com/blog/the-dirty-little-secret-of-constituency-

6. Conclusion

Back to our Goals:

1. provide students with an overview of the field with focus on the syntax-semantics interface;
2. bring students to be aware on the one hand of **several lexicalized formal grammars**, [**Done**]
3. on the other hand of **computational semantics** models and be able to combine some of them to capture the natural language syntax-semantics interface; [**next block of classes**]
4. evaluate several applications with a special focus to Interactive Question Answering and Language and Vision Models;
5. make students acquainted with writing scientific reports. (Reading, Summarize, DIscussion, Proposals) [Started]