GRAMMARS & PARSING

Shallow parsing
Probabilistic parsing
Dependency parsing

Natural Language Processing
Prof. Licia Sbattella
Finite-state Parsing (Shallow Parsing)

Company Name: Bridgestone Sports Co
Verb Group: said
Noun Group: Friday
Noun Group: it
Verb Group: had set up
Noun Group: a joint venture
Preposition: in
Location: Taiwan
Preposition: with
Noun Group: a local concern
Conjunction: and
Noun Group: a Japanese trading house
Verb Group: to produce
Noun Group: golf clubs
Finite-state Parsing

- Sometimes full parsing is not required
  - E.g. information extraction often do not need to extract all the structure of the sentence
  - Recognition of base phrases
    - Selection of such base phrases can depend on the application
- Use subset of CFG, without recursion
  - e.g.: \( \text{NP} \rightarrow \text{NP} \text{ VP} \) not allowed
  - Rules translated in a set of FSAs
  - Very efficient parsing

\[
\text{AdjP} \rightarrow \text{Ordinal} \\
| \{ \{ \text{Q-er} | \text{Q-est} \} \{ \text{Adj} | \text{Vparticle} \} + \\
| \{ \text{N[sing, !Time-NP]} (“-“) \{ \text{Vparticle} \} \\
| \text{Number (“-“)} \{ \text{“month” | “day” | “year”} \} (“-“) “old”\}
\]
Finite-state Parsing

- E.g.: FASTUS (Appelet e Israel 1997)
- Limited recursion can be handled by means:
  - Automata cascade
    - Each level handles a “recursion” level
    - E.g.: NP → NP and NP
      NP on the left treated differently than NP on the right
- Recursive Transition Networks (RTNs)
  - Are not true FSAs
  - Arcs with non-terminal symbols handled as subroutines
    (returns when the non-terminal has been parsed)
Eliminate parsing ambiguities

- Probabilistic methods
- Adding probabilities to grammars
- Parsers consider probability of the analyzed configurations
  - Returning the most probable configuration
Probabilistic CFGs

- The probabilistic model
  - Assigns probabilities to parse trees
- Takes into account probabilities in the model
- Parsing with probabilities
  - Simple adaptations of the dynamic programming algorithms
  - The goal is to find the tree associated with the maximum probability
The probabilistic model

- Adds probabilities to grammar rules
- Summing probabilities associated to the set of rules expanding the same non-terminal symbol, the result is 1

\[ VP \rightarrow \text{Verb} \quad 0.55 = \quad P(\text{Verb} \mid VP) \]
\[ VP \rightarrow \text{Verb NP} \quad 0.40 = \quad P(\text{Verb NP} \mid VP) \]
\[ VP \rightarrow \text{Verb NP NP} \quad 0.05 = \quad P(\text{Verb NP NP} \mid VP) \]

1
The probabilistic model

- The parse tree defines the grammar rules
- Parse tree probability
  - Calculated as the product of probabilities associated to the rules involved in tree derivation
- Probability of a word sequence (phrase)
  - Is the probability of the associated parse tree, if no ambiguities exist
  - Is the sum of probabilities associated to parse trees, if ambiguities exist
- Probabilities are calculated by means of an annotated database (treebank):
  \[
P(a \rightarrow b) = P(b | a) = \frac{C(a \rightarrow b)}{\sum_{\gamma} C(a \rightarrow \gamma)} = \frac{C(a \rightarrow b)}{C(a)}
\]
Parsed Corpora: Treebanks

- The Penn Treebank
- Treebanks can be used to derive a CFG/PCFG
  - CFG: just collect the set of productions used in the treebank
  - PCFG: add probabilities to the collected productions, as explained before
- Issues:
  - The grammar generalize poorly to sentences that are not included in the treebank
Probabilistic Parsing

- Probabilistic parsing needs
  - A grammar
  - A vast and robust dictionary with POS
  - A parser

- Dynamic programming algorithm: CKY (Cocke-Kasami-Younger) — Ney91 – Collins99 – Aho & Ullman72
  - Assigns probabilities to constituents when they are completed and put in the table
  - Bottom-up parser: uses maximum probability for going towards the top

- Other dynamic programming algorithm: Viterbi parsing
Example

- \( P(T_a) = 1.5 \times 10^{-6} \)

- \( P(T_b) = 1.7 \times 10^{-6} \)

<table>
<thead>
<tr>
<th>Rules</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>S \rightarrow Aux NP VP</td>
<td>.15</td>
</tr>
<tr>
<td>NP \rightarrow Pro</td>
<td>.40</td>
</tr>
<tr>
<td>VP \rightarrow V NP NP</td>
<td>.05</td>
</tr>
<tr>
<td>NP \rightarrow Nom</td>
<td>.05</td>
</tr>
<tr>
<td>NP \rightarrow PNoun</td>
<td>.35</td>
</tr>
<tr>
<td>Nom \rightarrow Noun</td>
<td>.75</td>
</tr>
<tr>
<td>Aux \rightarrow Can</td>
<td>.40</td>
</tr>
<tr>
<td>NP \rightarrow Pro</td>
<td>.40</td>
</tr>
<tr>
<td>Pro \rightarrow you</td>
<td>.40</td>
</tr>
<tr>
<td>Verb \rightarrow book</td>
<td>.30</td>
</tr>
<tr>
<td>PNoun \rightarrow TWA</td>
<td>.40</td>
</tr>
<tr>
<td>Noun \rightarrow flights</td>
<td>.50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rules</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>S \rightarrow Aux NP VP</td>
<td>.15</td>
</tr>
<tr>
<td>NP \rightarrow Pro</td>
<td>.40</td>
</tr>
<tr>
<td>VP \rightarrow V NP</td>
<td>.40</td>
</tr>
<tr>
<td>NP \rightarrow Nom</td>
<td>.05</td>
</tr>
<tr>
<td>Nom \rightarrow PNoun</td>
<td>.05</td>
</tr>
<tr>
<td>Nom \rightarrow Noun</td>
<td>.75</td>
</tr>
<tr>
<td>Aux \rightarrow Can</td>
<td>.40</td>
</tr>
<tr>
<td>NP \rightarrow Pro</td>
<td>.40</td>
</tr>
<tr>
<td>Pro \rightarrow you</td>
<td>.40</td>
</tr>
<tr>
<td>Verb \rightarrow book</td>
<td>.30</td>
</tr>
<tr>
<td>PNoun \rightarrow TWA</td>
<td>.40</td>
</tr>
<tr>
<td>Noun \rightarrow flights</td>
<td>.50</td>
</tr>
</tbody>
</table>
PCFGs: problems

- **CFG assumption**
  - Expansions of non-terminal symbols are independent. PCFG retains such assumption (i.e., probabilities can be multiplied ...)

- **Problems related to:**
  - **Structural dependencies** – Francis et al. 99
    - For **subjects NP**:
      - $NP \rightarrow Pronoun$ (91%), $NP \rightarrow Det Noun$ (9%)
    - For **direct objects NP**:
      - $NP \rightarrow Pronoun$ (34%), $NP \rightarrow Det Noun$ (66%)
    - But PCFG is not aware of this!
  - **Lexical dependencies** – example: PP attachment
    - *John saw the moon with the telescope*
    - *John saw the man with the hat*
    - Same structural form, only words change… see next slide...
PCFGs: lexical dependency

- choice 1: $\text{VP} \rightarrow \text{VBD NP PP with } P_1$
- choice 2: $\text{VP} \rightarrow \text{VBD NP with } P_2$
  $\text{NP} \rightarrow \text{NP PP with } P_3$
- Choice depends on a word: *moon vs man*
- But PCFG will always prefer 1 or 2
  - depends on $P_1, P_2, P_3$
- Frequency of the rule is not enough!
Improving PCFSs by parent annotation

- For solving the *structural problem*, one could *split* the non-terminals into many versions
  - For the example: split NP in NP\textsubscript{subjects} and NP\textsubscript{direct-objects}
- A way to implement it: parent annotation on the nodes
  - E.g.: $VP \rightarrow VDB \ NP \ p_1$
    $$VP^S \rightarrow VDB \ NP^VP \ p_2 \neq p_1$$
- Use a grammar that parent-annotate the phrasal non-terminals (except POS)
Improving PCFSs by parent annotation

- Adding parent annotation on the POS could be useful
  - In the example, splitting the POS IN into IN^SBAR allows *if* to prefer a sentential complement, resulting in the correct verbal parse

\[
\text{IN}^\text{SBAR} \rightarrow \text{if} \\
\text{forces the parser to use the expansion} \\
\text{SBAR}^\text{VP} \rightarrow \text{IN}^\text{SBAR} \ S^\text{SBAR} \\
\text{instead of PP}^\text{VP} \rightarrow \text{IN} \ NP^\text{PP}
\]

- Node-splitting increases the size of the grammar, reducing the amount of training data available for each grammar rule
Lexicalized parse trees

For solving the *lexical problem*, each non-terminal symbol in parse tree is annotated with a single word (its *lexical head* or *headword*) and POS

The lexical head of a constituent is its “most important” word

---

**Internal Rules**
- TOP → S(dumped,VBD)
- S(dumped,VBD) → NP(workers,NNS)
- NP(workers,NNS) → NNS(workers,NNS)
- VP(dumped,VBD) → VBD(dumped,VBD)
- NNS(workers,NNS) → workers
- VBD(dumped,VBD) → dumped
- NP(sacks,NNS) → NNS(sacks,NNS)
- PP(into,P) → P(into,P)
- NP(sacks,NNS) → sacks
- P(into,P) → into
- NP(bin,NN) → bin
- NNS(sacks,NNS) → sacks
- DT(a,DT) → a
- NN(bin,NN) → bin

**Lexical Rules**
- NNS(workers,NNS) → workers
- VBD(dumped,VBD) → dumped
- NNS(sacks,NNS) → sacks
- P(into,P) → into
- DT(a,DT) → a
- NN(bin,NN) → bin
Probabilistic Lexicalized CFGs

- The headword for a node is set to the headword of its head daughter, and the head tag to the POS tag of the headword
  - E.g.: \( \text{VP(dumped,VDB)} \rightarrow \text{VDB(dumped,VDB) NP(sacks,NNS) PP(into,P)} \)

\[
p = \frac{C(\text{VP(dumped,VDB)} \rightarrow \text{VDB(dumped,VDB) NP(sacks,NNS) PP(into,P)})}{C(\text{VP(dumped,VDB)})}
\]

- **lexical rules** express expansions of *preterminals* (i.e., POS’s) to words
  - Deterministic (i.e., they have probability 1): a lexicalized preterminal like \( \text{NN(bin,NN)} \) can only expand to the word *bin*:
    \[
    \text{NN(bin, NN)} \rightarrow \text{bin} \quad P=1
    \]

- **internal rules** express the other rule expansions
  - I.e.: \( \text{VP} \rightarrow \text{NP PP} \quad P=? \)
  - For the internal rules we will need to estimate probabilities
The “head daughter” is the “most important” daughter of the node

- Choosing such head daughters is complicated and indeed controversial
- Determined by a set of rules defined in modern linguistic theories of syntax
  - See the Jurafsky’s book for an example

For calculating $p$

- We need a huge corpus! Probably most of those rules would be associated with a zero probability
Probabilistic Lexicalized CFGs

- Make some independence assumptions:
  - break down each rule, so that we would estimate the probability as the product of smaller independent probability estimates...
  - …for which we could acquire reasonable counts

- Collins’ model 1 (simplified)
  - A CFG rule is thought of as: \( LHS \rightarrow L_n L_{n-1} \ldots L_1 H R_1 \ldots R_{n-1} R_n \)
  - Generative story:
    - First, generate the head of the rule (i.e., \( H \))
    - Then, generate the dependents of the head, one by one, from the inside out
    - Each of these generation steps will have its own probability
    - The STOP non-terminal at the left and right edges of the rule…
      - …will allow the model to know when to stop generating dependents on a given side
    - E.g.: \( VP(dumped,VBD) \rightarrow STOP VBD(dumped,VBD) \) \( NP(sacks,NNS) \) \( PP(into,P) \) STOP

D. Jurafsky & J. H. Martin Speech and Language Processing
Collin’s Model 1 (simplified)

1) First generate the head \( VBD(dumped, VBD) \) with probability \( P_h(H|LHS) = P(VBD(dumped, VBD) | VP(dumped, VBD)) \)

2) Then generate the left dependent (which is STOP, since there isn’t one) with probability \( P_L(STOP| VP(dumped, VBD), VBD(dumped, VBD)) \)

3) Then generate right dependent \( NP(sacks, NNS) \) with probability \( P_r(NP(sacks, NNS) | VP(dumped, VBD), VBD(dumped, VBD)) \)

4) Then generate the right dependent \( PP(into, P) \) with probability \( P_r(PP(into, P) | VP(dumped, VBD), VBD(dumped, VBD)) \)

5) Finally generate the right dependent STOP with probability \( P_r(STOP | VP(dumped, VBD), VBD(dumped, VBD)) \)

“generate” == search into the treebank
Collin’s Model 1 (simplified)

Thus:

\[ P(\text{VP}(\text{dumped}, \text{VBD}) \rightarrow \text{VBD}(\text{dumped}, \text{VBD}), \text{NP}(\text{sacks}, \text{NNS}) \, \text{PP}(\text{into}, \text{P})) = \]

\[ = P_H(\text{VBD}(\text{dumped}, \text{VDB}) \mid \text{VP}(\text{dumped}, \text{VBD})) \times \]

\[ P_L(\text{STOP} \mid \text{VP}(\text{dumped}, \text{VDB}), \text{VBD}(\text{dumped}, \text{VDB})) \times \]

\[ P_R(\text{NP}(\text{sacks}, \text{NNS}) \mid \text{VP}(\text{dumped}, \text{VDB}), \text{VBD}(\text{dumped}, \text{VDB})) \times \]

\[ P_R(\text{PP}(\text{into}, \text{P}) \mid \text{VP}(\text{dumped}, \text{VDB}), \text{VDB}(\text{dumped}, \text{VDB})) \times \]

\[ P_R(\text{STOP} \mid \text{VP}(\text{dumped}, \text{VDB}), \text{VBD}(\text{dumped}, \text{VDB})) \]

And, for example:

\[ P_R(\text{NP}(\text{sacks}, \text{NNS}) \mid \text{VP}(\text{dumped}, \text{VDB}), \text{VDB}(\text{dumped}, \text{VDB})) = \]

\[ = \frac{C(\text{VP}(\text{dumped}, \text{VDB}) \rightarrow \ldots \text{<everything>...} \, \text{VDB}(\text{dumped}, \text{VDB}), \text{NP}(\text{sacks}, \text{NNS}) \, \ldots \text{<everything>...})}{C(\text{VP}(\text{dumped}, \text{VDB}))} \]

\[ P_L(\text{STOP} \mid \text{VP}(\text{dumped}, \text{VDB}), \text{VDB}(\text{dumped}, \text{VDB})) = \]

\[ = \frac{C(\text{VP}(\text{dumped}, \text{VDB}) \rightarrow \text{VDB}(\text{dumped}, \text{VDB}), \ldots \text{<everything>...})}{C(\text{VP}(\text{dumped}, \text{VDB}))} \]
Other syntactic models

- Grammatical relationships
  - Subject
    - I booked a flight to New York
    - The flight was booked by my agent.
  - Direct object
    - I booked a flight to New York
  - Complement
    - I said that I wanted to leave
Dependency Parsing

- Links from word to word, instead of constituent units
- Approach based on European tradition (from ancient Greece); not so used in America
- The *Subject* and *Object* concepts are precursors of subcategorization (also known as ‘valence’) and linked to the Dependency theory (Dependency Grammar)
- Dependency parsing is widely used as a computational model
- The analysis of relationships among words is useful at several levels
Dependency Parsing

<ROOT>

main:

GAVE

subj:
dat:

I

HIM

obj:

ADDRESS

attr:
pnct:

MY

.
Dependency Grammars

- Suited for languages having many variations in word ordering
- Jarvinen & Tapanainen 97
- Link Grammar (Sleator & Temperley 93)
- Constraint Grammar (Karlsson et al. 95)
Dependencies

- **Dependency**  **Description**
  - **Subj**: syntactic subject
  - **Obj**: direct object (incl. Sent. Compl.)
  - **Dat**: indirect object
  - **Pcomp**: complement of a preposition
  - **Comp**: predicate nominals (copulas’ compl)
  - **Tmp**: temporal adverbials
  - **Loc**: location adverbials
  - **Attr**: premodifying (attributive) nominals (gen., etc.)
  - **Mod**: nominal postmodifiers (prepos. phrases, etc.)