Observation: Phrases Are Related to Context-Free Grammars

- Phrase structure of a sentence corresponds to the derivation tree under the grammar that generates / recognizes the sentence.
- Example:
  - $S \rightarrow NP \ VP$ (a sentence has a subject and a predicate)
  - $NP \rightarrow N$ (a noun is a noun phrase)
  - $VP \rightarrow V \ NP$ (a verb phrase consists of a verb and its object)
- Lexicon part of the grammar:
  - $N \rightarrow$ dog | cat | man | car | John …
  - $V \rightarrow$ see | sees | saw | bring | brings | brought | …
Lexicon

• In practice the lexical part can (and should) be implemented separately from the grammar.
• The nonterminals of the lowest level (immediately above the terminals) might be POS tags.
  – Then morphological analysis and tagging (disambiguation of MA) solves the lowest level of the phrase tree.
    • In fact, disambiguation is not necessary. There will be other ambiguities in the tree anyway. The parser can take care of them.
  – The grammar works only with POS tags.
  – This is why we sometimes talk about preterminals (the nonterminals immediately above the leaf nodes).
An Extended Grammar Example for Czech (7 Cases!)

- NP → N | AP N
- AP → A | AdvP A
- AdvP → Adv | AdvP Adv
- NP_nom → N_nom
- NP_nom → AP_nom N_nom
- NP_gen → N_gen NP_gen
- N → pán | hrad | muž | stroj …
- A → mladý | velký | zelený …
- Adv → velmi | včera | zeleně …
- N_nom → pán | hrad | muž …
- N_gen → pána | hradu | muže …
- N_dat → pánovi | hradu | muži …
- N_acc → pána | hrad | muže …
- N_voc → pane | hrade | muži …
- N_loc → pánovi | hradu | muži …
- N_ins → pánem | hradem …
An Extended Grammar Example for Czech (Verbs)

- VP → VP\textsubscript{obligatory}
- VP → VP\textsubscript{obligatory} OptMod
- VP\textsubscript{obligatory} → V\textsubscript{intr}
- VP\textsubscript{obligatory} → V\textsubscript{trans} NP\textsubscript{acc}
- VP\textsubscript{obligatory} → V\textsubscript{ditr} NP\textsubscript{dat} NP\textsubscript{acc}
- VP\textsubscript{obligatory} → V\textsubscript{mod} VINF
- OptMod → AdvP\textsubscript{location} | AdvP\textsubscript{time} | AdvP\textsubscript{manner} ...
- V\textsubscript{intr} → šedivět | brzdit ...
- V\textsubscript{trans} → koupit | ukrášt ...
- V\textsubscript{ditr} → dát | půjčit | poslat ...
- V\textsubscript{mod} → moci | smět | muset ...
- … (tens to hundreds of frames)
Unification Grammar

• An alternative to nonterminal splitting

• Instead of seven context-free rules:
  
  - $\text{NP}_{nom} \rightarrow \text{AP}_{nom} \text{N}_{nom}$
  - $\text{NP}_{gen} \rightarrow \text{AP}_{gen} \text{N}_{gen}$
  - $\text{NP}_{dat} \rightarrow \text{AP}_{dat} \text{N}_{dat}$
  - $\text{NP}_{acc} \rightarrow \text{AP}_{acc} \text{N}_{acc}$
  - $\text{NP}_{voc} \rightarrow \text{AP}_{voc} \text{N}_{voc}$
  - $\text{NP}_{loc} \rightarrow \text{AP}_{loc} \text{N}_{loc}$
  - $\text{NP}_{ins} \rightarrow \text{AP}_{ins} \text{N}_{ins}$

• One unification rule:
  
  - $\text{NP} \rightarrow \text{AP} \text{N} := \text{[case} = \text{AP}^\text{case} \# \text{N}^\text{case]}$
Parsing with a Context-Free Grammar

• Hierarchy of grammars:
  – Noam Chomsky (1957): *Syntactic Structures*

• Couple of classical algorithms.
  – CYK (Cocke-Younger-Kasami) … complexity $O(n^3)$
    • John Cocke (“inventor”)
    • Tadao Kasami (1965), Bedford, MA, USA (another independent “inventor”)
    • Daniel H. Younger (1967) (computational complexity analysis)

• Constraint of CYK: grammar is in CNF (Chomsky Normal Form), i.e. the right-hand side of every rule consists of either two nonterminals or one terminal. (CFGs can be easily transformed to CNF.)
 Parsing with a Context-Free Grammar

- **Chart parser**: CYK requires a data structure to hold information about partially processed possibilities. Turn of 1960s and 1970s: the chart structure proposed for this purpose.
- Jay Earley (1968), PhD thesis, Pittsburgh, PA, USA
  - A somewhat different version of chart parsing.
- For details on chart parser, see the earlier lecture about morphology and context-free grammars.
Syntactic Analysis (Parsing)

- Automatic methods of finding the syntactic structure for a sentence
  - Symbolic methods: a phrase grammar or another description of the structure of language is required. Then: the chart parser.
  - Statistical methods: a text corpus with syntactic structures is needed (a treebank).
  - Hybrid methods: a simple grammar, ambiguities solved statistically with a corpus.
    - Chunking / shallow parsing
Probabilistic Context-Free Grammars

- PCFG (probabilistic context-free grammars)
- If there are several possible parses we want to weigh them.
- Competing parses are caused by competing rules with the same left-hand side.
- The idea: probabilistic distribution for rules with the same left-hand side.
  - Example: grammar has $VP \rightarrow V \ NP$ and $VP \rightarrow V \ NP \ PP$.
  - The input sentence allows both these readings, too.
  - But we know (e.g.) that the second way of building a $VP$ is more frequent:
    - $p(V \ NP \mid VP) = 0.3$
    - $p(V \ NP \ PP \mid VP) = 0.7$
Ambiguous Parse

- S → NP VP
- VP → V NP PP
- VP → V NP
- NP → N
- NP → N PP
- PP → PREP N
- N → man
- N → woman
- N → car
- V → saw
- PREP → in

man saw woman in car
Probability of Parse Tree

- Both phrases / parses are “grammatical”.
- Different readings. Which one is better in this context?
- Probabilistic context-free grammar:
  - Relations between parent and child nodes.
  - Probability of derivation, use of a rule.
  - Probability of the whole parse tree ($r_i$ are grammar rules used to generate the sentence $S$ whose parse is $T$):

$$p(T) = \prod_{i=1}^{n} p(r_i)$$
Assumptions

- Application of a rule is independent of application of other rules in the sentence (very strong and improbable assumption).
- Independence of context of other subtrees.
- Independence of context of ancestors (higher levels).
- Independence of location in the sentence (word order) or in the tree.
Rule Probability

- Rule $r_i$: $A \rightarrow \alpha$.
- Let’s denote $R_A$ the set of all rules $r_j$ whose left-hand side is the nonterminal $A$.
- Let’s define a probability distribution on $R_A$:

$$\sum_{r \in R_A} p(r) = 1 \quad 0 \leq p(r) \leq 1$$

- In other words:

$$p(r) = p(\alpha | A) \quad r = A \rightarrow \alpha \quad \alpha \in (N \cup T)^+$$
Estimation of Rule Probability

- A treebank based on a context-free grammar (i.e. not a dependency treebank).

\[ r = A \rightarrow \alpha_1 \alpha_2 \ldots \alpha_k \]

\[ p(r) = \frac{c(r)}{c(A)} \]

- Frequency of rule application: how often is there this subtree in the treebank

\[ A \]

\[ \alpha_1 \alpha_2 \ldots \alpha_k \]
Practical Phrase-Based Parsing

- Rule-based parsers, e.g. Fidditch (Donald Hindle, 1983)
- **Collins** parser (Michael Collins, 1996–1999)
  - Probabilistic context-free grammars, lexical heads
  - Labeled precision & recall on Penn Treebank / Wall Street Journal data / Section 23 = 85%
  - Reimplemented in Java by Dan Bikel ("Bikel parser"), freely available
- **Charniak** parser (Eugene Charniak, NAACL 2000)
  - Maximum entropy inspired parser
  - P ~ R ~ 89.5%
  - Mark Johnson: reranker => over 90%
- **Stanford** parser (Chris Manning et al., 2002–2010)
  - Produces dependencies, too. Initial P ~ R ~ 86.4%
Dependency Parsing

Daniel Zeman

http://ufal.mff.cuni.cz/course/npfl094
Dependency Parsing with a Statistical Model

Manually annotated corpus (treebank)

\[ p(\text{edge}([\text{ve}], [\text{dveřích}]))) = p_1 \]
\[ p(\text{edge}([v], [\text{dveřích}]))) = p_2 \]
\[ p(\text{edge}([\text{ve}], [\text{dveře}]))) = p_3 \]
\[ p(\text{edge}([\text{ve}], [\text{dveřím}]))) = p_4 \]

where perhaps:

\[ p_1 > p_2 > p_3 \quad \text{and} \quad p_4 \neq 0 \]
Base Idea: Looking for the Most Probable Tree

- Searching for tree $M$ that represents the given sentence $S$ with the highest possible probability.
- Formally:
  - Bayes rule: $p(M|S)$ is the probability that the sentence whose description is the tree $M$ is $S$.
  - $p(S|M)$ is the probability of the occurrence (existence) of the tree $M$.
  - $p(M)$ is the probability of the occurrence of the sentence $S$. From the point of view of searching the most probable tree this is a mere constant that does not influence the result:

$$p(M|S) = \frac{p(S|M) \cdot p(M)}{p(S)}$$
Base Idea: Looking for the Most Probable Tree

• $p(S)$ is the probability of the occurrence of the sentence $S$. From the point of view of searching the most probable tree this is a mere constant that does not influence the result:

$$\text{arg max}_M \frac{p(S|M) \cdot p(M)}{p(S)} = \text{arg max}_M (p(S|M) \cdot p(M))$$

• So the task is to estimate the probabilities $p(S|M)$ and $p(M)$.
• $p(S|M)$ can be maximized so that the tree $M$ is constructed directly of the words of the sentence $S$. 
Tree Probability

- Assumption: dependencies do not depend on each other (strong and wrong—a better model would be more careful)
- Product of probabilities of dependencies.

\[ p(M) = \prod_{i=1}^{n} p(h_i) \]

- How to find the most probable tree? Similarly to tagging: by Viterbi algorithm! (The middle course between the greedy algorithm and backtracking.)
Graph-Based Parsers: MST

- McDonald et al., HLT-EMNLP 2005
- [http://sourceforge.net/projects/mstparser/](http://sourceforge.net/projects/mstparser/)
- MST = maximum spanning tree =
  \[
  \text{CS: nejlépe ohodnocená kostra (orientovaného) grafu}
  \]
- Start with a total graph.
  - We assume that there can be a dependency between any two words of the sentence.
- Gradually remove poorly valued edges.
- A statistical algorithm will take care of the valuation.
  - It is trained on edge features.
  - Example features: lemma, POS, case… of governing / dependent node.
MST Parser

- Feature engineering (tell the parser what features to track) by modifying the source code (Java).
- Not easy to incorporate 2\textsuperscript{nd} order features
  - I.e. edge weight depends e.g. on POS tag of its grandparent.
- Parser can be run in nonprojective mode.
- Training on the whole PDT reportedly takes about 30 hours.
  - It is necessary to iterate over all feature combinations and look for the most useful ones.
- In comparison to that, the parsing proper is quite fast.
Transition-Based Parsers: Malt

- Nivre et al., *Natural Language Engineering*, 2007
- [http://maltparser.org/](http://maltparser.org/)
- Based on *transitions* from one configuration to another.
- Configuration:
  - Input buffer (words of the sentence, left-to-right)
  - Stack
  - Output tree (words, dependencies and dependency labels)
- Transitions:
  - Shift: move word from buffer to stack
  - Larc: left dependency between two topmost words on stack
  - Rarc: right dependency between two topmost words on stack
Malt Parser

• Parser driven by oracle that selects the transition operation based on the current configuration.

• Training: decompose the tree from training data to a sequence of configurations and transitions
  – Sometimes there are more than one possibility
    • Various learning strategies: e.g. create dependencies eagerly, as soon as possible.

• The oracle learns based on the features of the configuration.
  – E.g. word, lemma, POS, case, number…
    • \(n^{\text{th}}\) word from the top of the stack
    • \(k^{\text{th}}\) word remaining in the buffer
    • particular node in output tree part created so far
Malt Parser

• Again, a machine learning algorithm is responsible for training, here the *Support Vector Machines (SVM)*.
  – Classifier. Input vectors: values of all features of the current configuration.
  – In addition, during training there is the output value, i.e. action identifier (Shift / Larc / Rarc).
  – The trained oracle (SVM) tells the output value during parsing.

• Training on the whole PDT may take weeks!
  – Complexity $O(n^2)$ where $n$ is number of training examples.
  – Over 3 million training examples can be extracted from PDT.

• Parsing is relatively faster (~ 1 sentence / second) and can be parallelized.
Example of Malt Parsing

• stack = #
• buffer = Pavel dal Petrovi dvě hrušky .
• English = Paul gave to-Peter two pears .
Example of Malt Parsing

- stack = #
- buffer = Pavel dal Petrovi dvě hrušky .
- tree =

SHIFT

- stack = # Pavel
- buffer = dal Petrovi dvě hrušky .
- tree =
Example of Malt Parsing

- stack = # Pavel
- buffer = dal Petrovi dvě hrušky .
- tree =

**SHIFT**

- stack = # Pavel dal
- buffer = Petrovi dvě hrušky .
- tree =
Example of Malt Parsing

- stack = # Pavel dal
- buffer = Petrovi dvě hrušky.
- tree =

LARC

- stack = # dal
- buffer = Petrovi dvě hrušky.
- tree = dal(Pavel)
Example of Malt Parsing

- stack = # dal
- buffer = Petrovi dvě hrušky.
- tree = dal(Pavel)

**SHIFT**

- stack = # dal Petrovi
- buffer = dvě hrušky.
- tree = dal(Pavel)
Example of Malt Parsing

- stack = # dal Petrovi
- buffer = dvě hrušky .
- tree = dal(Pavel)

**RARC**

- stack = # dal
- buffer = dvě hrušky .
- tree = dal(Pavel,Petrovi)
Example of Malt Parsing

- stack = # dal
- buffer = dvě hrušky .
- tree = dal(Pavel,Petrovi)

SHIFT

- stack = # dal dvě
- buffer = hrušky .
- tree = dal(Pavel,Petrovi)
Example of Malt Parsing

• stack = # dal dvě
• buffer = hrušky .
• tree = dal(Pavel,Petrovi)

**SHIFT**

• stack = # dal dvě hrušky
• buffer = .
• tree = dal(Pavel,Petrovi)
Example of Malt Parsing

- stack = # dal dvě hrušky
- buffer = .
- tree = dal(Pavel,Petrovi)

LARC

- stack = # dal hrušky
- buffer = .
- tree = dal(Pavel,Petrovi),hrušky(dvě)
Example of Malt Parsing

- stack = # dal hrušky
- buffer = .
- tree = dal(Pavel,Petrovi),hrušky(dvě)

RARC

- stack = # dal
- buffer = .
- tree = dal(Pavel,Petrovi,hrušky(dvě))
Example of Malt Parsing

- stack = # dal
- buffer = .
- tree = dal(Pavel,Petrovi,hrušky(dvě))

RARC

- stack = #
- buffer = .
- tree = #(dal(Pavel,Petrovi,hrušky(dvě)))
Example of Malt Parsing

- stack = #
- buffer = .
- tree = #(dal(Pavel, Petrovi, hrušky(dvě)))

SHIFT

- stack = # .
- buffer =
- tree = #(dal(Pavel, Petrovi, hrušky(dvě)))
Example of Malt Parsing

- stack = #.
- buffer =
- tree = #(dal(Pavel,Petrovi,hrušky(dvě))))

RARC

- stack = #
- buffer =
- tree = #(dal(Pavel,Petrovi,hrušky(dvě)),.)
Nonprojective Mode of Malt

- It can be proved that the above transition system is
  - correct
    - resulting graph is always a tree (continuous, cycle-free)
  - complete for the set of **projective trees**
    - every projective tree can be expressed as a sequence of transitions
- **How to add nonprojective dependencies?**
  - New transition operation **SWAP**:
  - Take second topmost word from stack and return it to buffer. That will swap the order of the input words.
  - This action is permitted only for words that have not been swapped before (their order on the stack corresponds to their original order in the sentence).
Nonprojective Parsing Example

- stack = #
- buffer = Soubor se nepodařilo otevřít.
- English = File itself it-did-not-succeed to-open.
Nonprojective Parsing Example

- stack = #
- buffer = Soubor se nepodařilo otevřít.
- tree =

**SHIFT**

- stack = # Soubor
- buffer = se nepodařilo otevřít.
- tree =
Nonprojective Parsing Example

- stack = # Soubor
- buffer = se nepodařilo otevřít .
- tree =

SHIFT

- stack = # Soubor se
- buffer = nepodařilo otevřít .
- tree =
Nonprojective Parsing Example

- stack = # Soubor se
- buffer = nepodařilo otevřít .
- tree =

**SHIFT**

- stack = # Soubor se nepodařilo
- buffer = otevřít .
- tree =

Nonprojective Parsing Example

- stack = # Soubor se nepodařilo
- buffer = otevřít.
- tree =

LARC

- stack = # Soubor nepodařilo
- buffer = otevřít.
- tree = nepodařilo(se)
Nonprojective Parsing Example

- stack = # Soubor nepodařilo
- buffer = otevřít .
- tree = nepodařilo(se)

SHIFT

- stack = # Soubor nepodařilo otevřít
- buffer = .
- tree = nepodařilo(se)
Nonprojective Parsing Example

- stack = # Soubor nepodařilo otevřít
- buffer = .
- tree = nepodařilo(se)

**SWAP**

- stack = # Soubor otevřít
- buffer = nepodařilo .
- tree = nepodařilo(se)
Nonprojective Parsing Example

- stack = # Soubor otevřít
- buffer = nepodařilo .
- tree = nepodařilo(se)

LARC

- stack = # otevřít
- buffer = nepodařilo .
- tree = nepodařilo(se), otevřít(Soubor)
Nonprojective Parsing Example

- stack = # otevřít
- buffer = nepodařilo .
- tree = nepodařilo(se), otevřít(Soubor)

**SHIFT**

- stack = # otevřít nepodařilo
- buffer = .
- tree = nepodařilo(se), otevřít(Soubor)
Nonprojective Parsing Example

• stack = # otevřít nepodařilo
• buffer = .
• tree = nepodařilo(se), otevřít(Soubor)

LARC

• stack = # nepodařilo
• buffer = .
• tree = nepodařilo(se, otevřít(Soubor))
Nonprojective Parsing Example

- stack = # nepodařilo
- buffer = .
- tree = nepodařilo(se, otevřít(Soubor))

RARC

- stack = #
- buffer = .
- tree = #(nepodařilo(se, otevřít(Soubor)))
Nonprojective Parsing Example

- stack = #
- buffer = .
- tree = #(nepodařilo(se, otevřít(Soubor)))

**SHIFT**

- stack = # .
- buffer =
- tree = #(nepodařilo(se, otevřít(Soubor)))
Nonprojective Parsing Example

- stack = #.
- buffer =
- tree = #(nepodařilo(se, otevřít(Soubor)))

RARC

- stack = #
- buffer =
- tree = #(nepodařilo(se, otevřít(Soubor)),.)
Malt and MST Accuracy

• Czech (PDT):
  – MST Parser over 85%
  – Malt Parser over 86%
    • Sentence accuracy (“complete match”) 35%, that is high!
  – The two parsers use different strategies and can be combined (either by voting (third parser needed) or one preparing features for the other)

• Other languages (CoNLL shared tasks)
  – MST was slightly better on most languages.
  – Accuracies not comparable cross-linguistically, figures are very dependent on particular corpora.
Features Are the Key to Success

- Common feature of MST and Malt:
  - Both can use large number of input text features.
  - Nontrivial machine learning algorithm makes sure that the important features will be given higher weight.
  - Machine learning algorithms are general classifiers.
    - Typically there is a library ready to download.
    - The concrete problem (here tree building) must be converted to a sequence of classification decisions, e.g. vectors (feature values + answer).
Newer Parsers

- Parsito / UDPipe (Milan Straka)
- Stanford (Tim Dozat et al.)
- Syntaxnet (Google)
- Mate (Bernd Bohnet)
- Turboparser (André Martins)
- …