INF4820

Chart Parsing

Erik Velldal

University of Oslo

Oct. 20, 2009

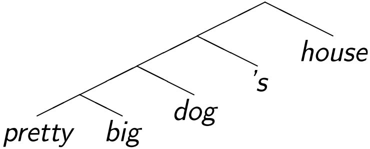
Topics for Today

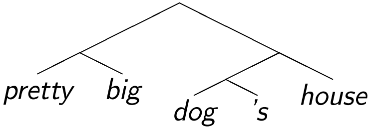
- Continue looking at *parsing*
  - Analysis of sentence structure
  - Natural language understanding
- The ambiguity challenge
  - Last week implicitly assumed that we could either explore all parses in parallel (requires an unrealistic amount of memory), or that we could use a backtracking approach (too inefficient due to the degree of ambiguity in realistic grammars).
  - Today we look at dynamic programming for parsing.
    - Chart Parsing; CKY, Earley, etc.
    - Ambiguity packing

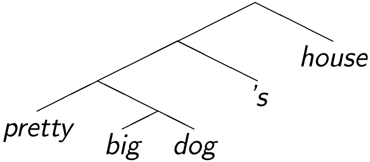
Ambiguity

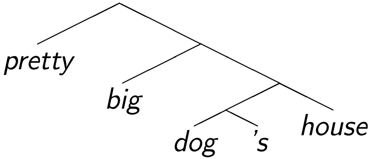
- Consider the possible PP-attachments in a sentence like
  I called the guy with the iPhone from work.
- Global: Several ways to derive a full tree for the sentence.
- Local: Even when there's only one grammatical analysis for the full sentence in the end, there might still be several possible analyses for words and sub-strings.
- Also, we typically want the possibility to access to *all* grammatical complete parses for a given string, and the same sub-trees re-enter in different parses.
- Trees do not provide a good way of representing ambiguity: each possibility requires a separate tree.
- Local ambiguities multiply. . .

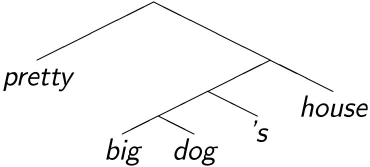
...pretty big dog's house...











Ambiguity (cont'd)

- Recall the efficiency problems with backtracking approaches like recursive descent.
- Consider the famous *garden path* sentence

  The horse raced [PP *past the barn*] *fell.*
- Structural and lexical ambiguities often lead the parser to build trees that it may eventually *discard* because they cannot be used in a complete parse for the whole input.
- The same sub-tree may be built several times: when backtracking the parser forgets about the previous structures and starts all over again.
- Exponential complexity in the worst case. Waste time by repeatedly re-parsing the same sub-string, and waste memory representing the same sub-trees several times.

Dynamic Programming for Parsing

- Dynamic Programming: Simplify a search problem by systematically computing solutions to sub-problems and storing them in a table. The overall problem is solved by re-using the solutions for the sub-problems.

Dynamic Programming for Parsing

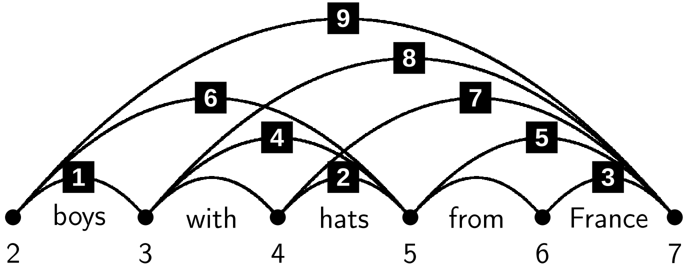
- Dynamic Programming: Simplify a search problem by systematically computing solutions to sub-problems and storing them in a table. The overall problem is solved by re-using the solutions for the sub-problems.
- For parsing, the sub-problems are analyses of sub-strings, and the table represents a chart.
- The chart can be visualized as a graph, recording the sub-trees that have been found, indexed by the string positions they span.
  - Vertices (nodes): Positions in the string w_1^n, starting from before the first word (0), ending after the final word (n):

    $_0$ *Kim* $_1$ *adored* $_2$ *snow* $_3$ *in* $_4$ *Oslo* $_5$

  - Edges (arcs): Span vertices from a start point to an end, representing a rule instantiation over a sub-string.

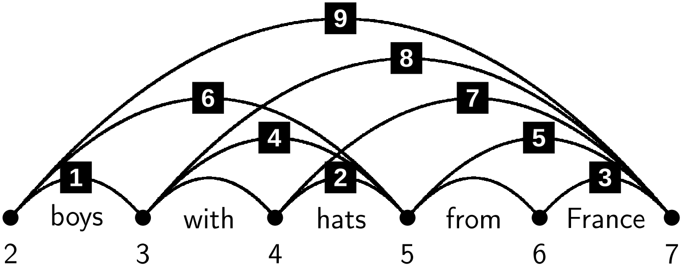
Bounding Ambiguity — The Parse Chart

- For many sub-strings, more than one way of deriving the same category.
- NPs: **1** | **2** | **3** | **6** | **7** | **9**
- PPs: **4** | **5** | **8**



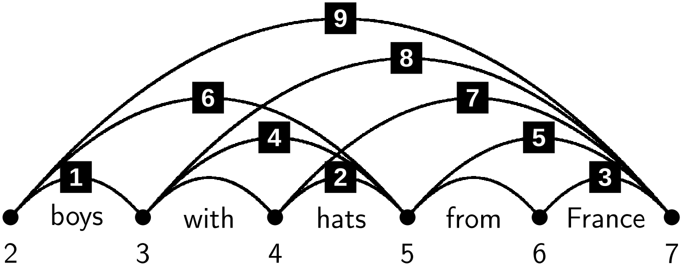
Bounding Ambiguity — The Parse Chart

- For many sub-strings, more than one way of deriving the same category.
- NPs: **1** | **2** | **3** | **6** | **7** | **9**
- PPs: **4** | **5** | **8**
- **9** $\equiv$ **1** + **8** | **6** + **5**



Bounding Ambiguity — The Parse Chart

- For many sub-strings, more than one way of deriving the same category.
- NPs: **1** | **2** | **3** | **6** | **7** | **9**
- PPs: **4** | **5** | **8**
- **9** $\equiv$ **1** + **8** | **6** + **5**
- *Parse forest*: a single item represents multiple trees (Billot & Lang, 89)



CKY (Cocke, Kasami, & Younger)

- The simplest chart algorithm.
- The simplest version of CKY is for a CFG in Chomsky Normal Form:
  - $\alpha \rightarrow \beta_1\beta_2$ or $\alpha \rightarrow w$ (for $\{\alpha, \beta_1, \beta_2\} \subseteq C$ and $w \in \Sigma$)

CKY (Cocke, Kasami, & Younger)

- The simplest chart algorithm.
- The simplest version of CKY is for a CFG in Chomsky Normal Form:
  - $\alpha \rightarrow \beta_1 \beta_2$ or $\alpha \rightarrow w$ (for $\{\alpha, \beta_1, \beta_2\} \subseteq C$ and $w \in \Sigma$)
- Visualize the chart as an n-by-n matrix or table.
- Use chart to record partial analyses, indexing them by string positions.
  - Row indexes start.
  - Column indexes end.
- Processing the input left to right, we incrementally fill the chart table.
- CKY is designed to guarantee that the parser only looks for rules that use a constituent from i to j after it has determined all the constituents that end at i. Otherwise something might be missed.

The CKY Algorithm

input: $w_1, \ldots, w_n$

---

for $j = 1$ **to** n **do**
  chart$_{[j-1,j]} \leftarrow \{\alpha \mid \alpha \rightarrow w_j \in P\}$
  for $i = j - 2$ **down to** 0 **do**
    for $k = i + 1$ **to** $j - 1$ **do**
    chart$_{[i,j]} \leftarrow$ chart$_{[i,j]} \cup$
        $\{\alpha \mid \alpha \rightarrow \beta_1 \, \beta_2 \in P, \beta_1 \in$ chart$_{[i,k]}, \beta_2 \in$ chart$_{[k,j]}\}$

[1,3] ← [1,2] + [2,3]
[0,3] ← [0,1] + [1,3]
...
[3,5] ← [3,4] + [4,5]
[2,5] ← [2,3] + [3,5]
[1,5] ← [1,2] + [2,5]
[0,5] ← [0,1] + [1,5]

	1	2	3	4	5
0	NP		S		S
1		V	VP		VP
2			NP		NP
3				P	PP
4					NP

The CKY Algorithm (cont'd)

- What's missing?
  - So far we just have a chart recognizer: We only determine whether the input is in the language generated by the grammar.
  - To read out a parse tree, each α in the chart need to record pointers to which β_i and β_j it combines.

Chart Parsing

- Rigid control structure of CKY as defined above: Working left to right and bottom-up, fill the upper triangular matrix column by column.
- In the more general formulation of "active" chart parsing as introduced by Martin Kay, the order of computation is more flexible:
  - No assumptions about earlier results.
  - *Active edges* encode partial rule instantiations, "waiting" for additional (adjacent and passive) constituents to complete: $[1, 2, \mathrm{VP} \rightarrow \mathrm{V} \bullet \mathrm{NP}]$.
  - Parser can fill in chart cells in *any* order and guarantee completeness.

Active Chart Parsing

- The items in the parse chart are called edges.
- An edge is a (possibly partial) rule instantiation over a sub-string.
- The chart indexes edges by start and end string position (aka vertices).
- "Dotted rules"; a dot in a rule RHS indicates degree of completion:
  $\alpha \rightarrow \beta_1 ... \beta_{i-1} \bullet \beta_i ... \beta_n$
- Active edges (aka incomplete items) — partial RHS:
  $[1, 2, \text{VP} \rightarrow \text{V} \bullet \text{NP}]$
- Passive edges (aka complete items) — full RHS:
  $[1, 3, \text{VP} \rightarrow \text{V NP} \bullet]$
- The key principle for processing edges is given by what Kay termed The Fundamental Rule:
  $[i,\, j,\, \alpha \rightarrow \beta_1 ... \beta_{l-1} \bullet \beta_l ... \beta_n] + [j,\, k,\, \beta_l \rightarrow \gamma^+ \bullet]$
  $\mapsto [i,\, k,\, \alpha \rightarrow \beta_1 ... \beta_l \bullet \beta_{l+1} ... \beta_n]$

An Example of a (Near-)Complete Chart

	1	2	3	4	5
0	S→NP•VP NP→NP•PP NP→Kim•				S→NP VP•
1		VP→V•NP V→adored•	VP→VP•PP VP→V NP•		VP→VP•PP VP→VP PP• VP→V PP•
2			NP→NP•PP NP→snow•		NP→NP•PP NP→NP PP•
3				PP→P•NP P→in•	PP→P NP•
4					NP→NP•PP NP→Oslo•

$_0$ *Kim* $_1$ *adored* $_2$ *snow* $_3$ *in* $_4$ *Oslo* $_5$

(Even) More Active Edges

	0	1	2	3
0	S→ •NP VP NP→ •NP PP NP→ •Kim	S→NP•VP NP→NP•PP NP→Kim•		S→NP VP•
1		VP→ •VP PP VP→ •V NP V→ •adored	VP→V•NP V→adored•	VP→VP•PP VP→V NP•
2			NP→ •NP PP NP→ •snow	NP→NP•PP NP→snow•
3				

- Processing: *scan*, *predict*, *complete*.
- Edges in each cell $chart_{[i,i]}$ represent "predictions". Can be constructed bottom-up or top-down.
- "Completing"; apply *fundamental rule* until no additional edges can be derived.

The Agenda

- The actual parsing is *chart-driven*; mostly just a question of invoking the fundamental rule (cf. "completing").
- However, we also sometimes consult the grammar rules (cf. "predicting"), and we need some way of deciding in what order to process the new edges.
- Rather than adding new edges to the chart directly, we first add to the agenda.
- The agenda is simply as a set of edges waiting to be added to the chart, and it determines in what order possibilities are tried.
  - Stack agenda: every time an edge is added, it is placed on the front of the agenda. (Depth-first)
  - Queue agenda: every time an edge is added, it is placed on the end of the agenda. (Breadth-first)

Backpointers: Recording the Derivation History

	0	1	1	3
0	2: S→ •NP VP 1: NP→ •NP PP 0: NP→ •Kim	10: S→8•VP 9: NP→8•PP 8: NP→Kim•		17: S→8 15•
1		5: VP→ •VP PP 4: VP→ •V NP 3: V→ •adored	12: VP→11•NP 11: V→adored•	16: VP→15•PP 15: VP→11 13•
2			7: NP→ •NP PP 6: NP→ •snow	14: NP→13•PP 13: NP→snow•
3				

- Use edges to record derivation trees: backpointers to daughters.
- A single edge can represent multiple derivations: backpointer sets.

Ambiguity Packing in the Chart

General Idea

- Maintain only one edge for each α from i to j (the "representative").
- Record alternate sequences of daughters for α in the representative.
- (E.g. only one NP representative for *a pretty big dog's house*)

Implementation

- Group passive edges into *equivalence classes* by identity of α, i, and j.
- Search chart for existing equivalent edge (h, say) for each new edge e.
- When h (the 'host' edge) exists, *pack* e into h to record equivalence.
- e *not* added to the chart, no derivations with or further processing of e.
- *Unpacking*: the process of multiplying out all alternative daughters for all result edges.

Chart Parsing, Summarized

Basic Notions

- Specialized dynamic programming
- Use *chart* to record partial analyses, indexing them by string positions.
- Treat multiple ways of deriving the same category for some sub-string as *equivalent*; pursue only once when combining with other constituents.

Key Benefits

- Avoid redundancy in computation and representation of results.
- Provides a general framework ("algorithm schema") in which alternative parsing strategies can be implemented.
- Efficient indexing of constituents: no search by start or end positions.
- Compute *parse forest* with exponential "extension" in *polynomial* time.

The Hardest Problem Still Remains

- How to make a final choice among all the possible readings?
- Grammatical knowledge vs. world knowledge.
- Identifying the correct reading is an "AI complete" problem.
- Syntactic disambiguation seems to require deeper semantic and pragmatic knowledge: *common sense*.

The Hardest Problem Still Remains

- How to make a final choice among all the possible readings?
- Grammatical knowledge vs. world knowledge.
- Identifying the correct reading is an "AI complete" problem.
- Syntactic disambiguation seems to require deeper semantic and pragmatic knowledge: *common sense*.
- Where to attach the *with-PP*?

$$\text{He scrubbed the dog } \textit{with the} \begin{cases} \textit{collar.} \\ \textit{brush.} \end{cases}$$

The Hardest Problem Still Remains

- How to make a final choice among all the possible readings?
- Grammatical knowledge vs. world knowledge.
- Identifying the correct reading is an "AI complete" problem.
- Syntactic disambiguation seems to require deeper semantic and pragmatic knowledge: *common sense*.
- Where to attach the *with-PP*?

$$\text{He scrubbed the dog } \textit{with the} \begin{cases} \textit{collar.} \\ \textit{brush.} \end{cases}$$

- A good case for empirical methods: Usage statistics as a proxy for common sense.