University of Oslo : Department of Informatics

INF4820: Algorithms for Artificial Intelligence and Natural Language Processing

Generalized Chart Parsing

Stephan Oepen & Erik Velldal

Language Technology Group (LTG)

November 4, 2015

Overview

Last Time

- Context-Free Grammar
- Treebanks
- Probabilistic CFGs
- Syntactic Parsing
  - Naïve: Recursive-Descent
  - Dynamic Programming: CKY

Overview

Last Time

- Context-Free Grammar
- Treebanks
- Probabilistic CFGs
- Syntactic Parsing
  - Naïve: Recursive-Descent
  - Dynamic Programming: CKY

Today

- Generalized Chart Parsing
- Inside the Parse Forest
- Viterbi Tree Decoding
- Parser Evaluation

CFGs (Formally, this Time)

Formally, a CFG is a quadruple: $G = \langle C, \Sigma, P, S \rangle$

CFGs (Formally, this Time)

Formally, a CFG is a quadruple: $G = \langle C, \Sigma, P, S \rangle$

- C is the set of categories (aka *non-terminals*),
  - $\{\mathrm{S}, \mathrm{NP}, \mathrm{VP}, \mathrm{V}\}$

CFGs (Formally, this Time)

Formally, a CFG is a quadruple: $G = \langle C, \Sigma, P, S \rangle$

- C is the set of categories (aka *non-terminals*),
  - $\{\text{S}, \text{NP}, \text{VP}, \text{V}\}$
- Σ is the vocabulary (aka *terminals*),
  - $\{\text{Kim}, \text{snow}, \text{adores}, \text{in}\}$

CFGs (Formally, this Time)

Formally, a CFG is a quadruple: $G = \langle C, \Sigma, P, S \rangle$

- C is the set of categories (aka *non-terminals*),
  - $\{\text{S}, \text{NP}, \text{VP}, \text{V}\}$
- Σ is the vocabulary (aka *terminals*),
  - $\{\text{Kim}, \text{snow}, \text{adores}, \text{in}\}$
- P is a set of category rewrite rules (aka *productions*)

$$\text{S} \rightarrow \text{NP VP} \qquad \text{NP} \rightarrow \text{Kim}$$
$$\text{VP} \rightarrow \text{V NP} \qquad \text{NP} \rightarrow \text{snow}$$
$$\text{V} \rightarrow \text{adores}$$

CFGs (Formally, this Time)

Formally, a CFG is a quadruple: $G = \langle C, \Sigma, P, S \rangle$

- C is the set of categories (aka *non-terminals*),
  - {S, NP, VP, V}
- Σ is the vocabulary (aka *terminals*),
  - {Kim, snow, adores, in}
- P is a set of category rewrite rules (aka *productions*)

S → NP VP	NP → Kim
VP → V NP	NP → snow
	V → adores

- $S \in C$ is the *start symbol*, a filter on complete results;

CFGs (Formally, this Time)

Formally, a CFG is a quadruple: $G = \langle C, \Sigma, P, S \rangle$

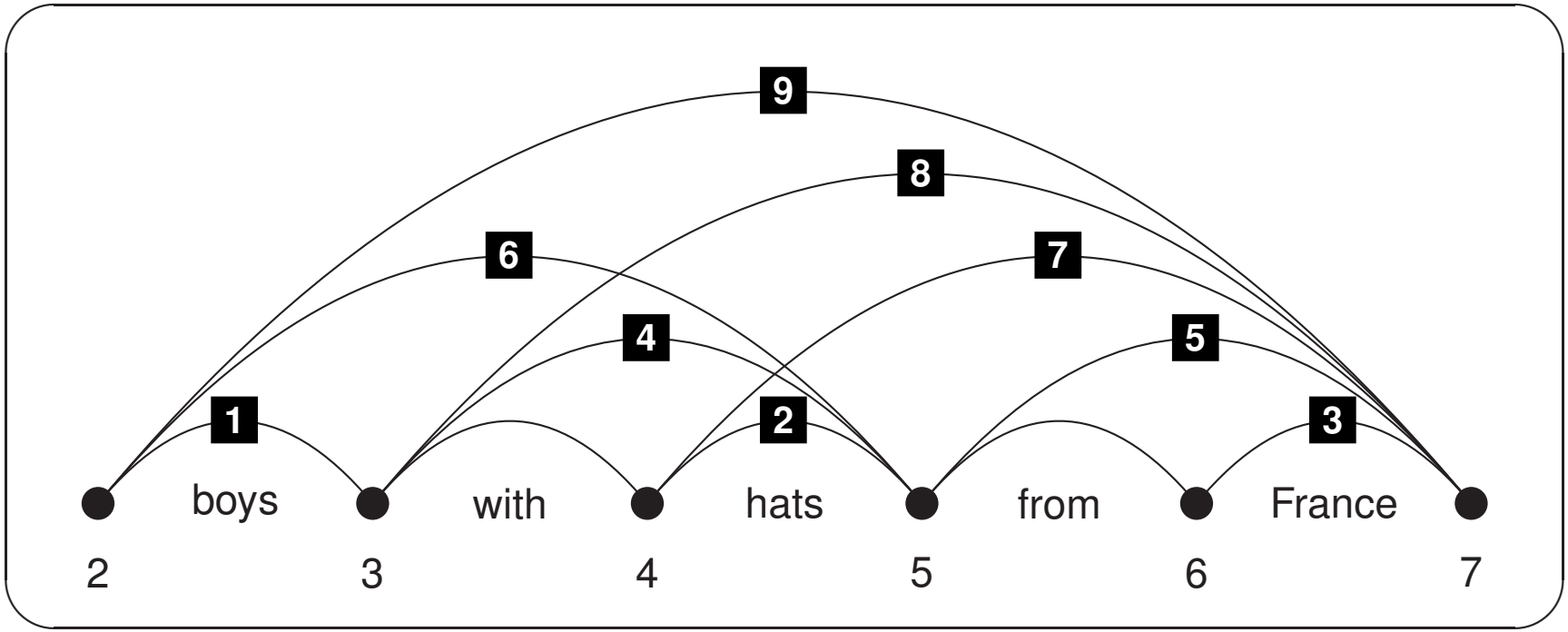
- C is the set of categories (aka *non-terminals*),
  - {S, NP, VP, V}
- Σ is the vocabulary (aka *terminals*),
  - {Kim, snow, adores, in}
- P is a set of category rewrite rules (aka *productions*)

$$\begin{array}{ll} \text{S} \rightarrow \text{NP VP} & \text{NP} \rightarrow \text{Kim} \\ \text{VP} \rightarrow \text{V NP} & \text{NP} \rightarrow \text{snow} \\ & \text{V} \rightarrow \text{adores} \end{array}$$

- $S \in C$ is the *start symbol*, a filter on complete results;
- for each rule $\alpha \rightarrow \beta_1, \beta_2, ..., \beta_n \in P$: $\alpha \in C$ and $\beta_i \in C \cup \Sigma$

A Key Insight: Local Ambiguity

- For many substrings, more than one way of deriving the same category;
- NPs: **1** | **2** | **3** | **6** | **7** | **9**; PPs: **4** | **5** | **8**; **9** ≡ **1** + **8** | **6** + **5**;
- *parse forest* — a single item represents multiple trees [Billot & Lang, 89].



The CKY (Cocke, Kasami, & Younger) Algorithm

for ($0 \leq i < |\mathit{input}|$) do
  $\mathit{chart}_{[i,i+1]} \leftarrow \{\alpha \,|\, \alpha \rightarrow \mathit{input}_i \in P\}$;
for ($1 \leq l < |\mathit{input}|$) do
  for ($0 \leq i < |\mathit{input}| - l$) do
    for ($1 \leq j \leq l$) do
      if ($\alpha \rightarrow \beta_1\,\beta_2 \in P \wedge \beta_1 \in \mathit{chart}_{[i,i+j]} \wedge \beta_2 \in \mathit{chart}_{[i+j,i+l+1]}$) then
        $\mathit{chart}_{[i,i+l+1]} \leftarrow \mathit{chart}_{[i,i+l+1]} \cup \{\alpha\}$;

Kim adored snow in Oslo

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

Limitations of the CKY Algorithm

Built-In Assumptions

- *Chomsky Normal Form* grammars: $\alpha \rightarrow \beta_1\beta_2$ or $\alpha \rightarrow \gamma$ ($\beta_i \in C$, $\gamma \in \Sigma$);
- breadth-first (aka exhaustive): always compute all values for each cell;
- rigid control structure: bottom-up, left-to-right (one diagonal at a time).

Limitations of the CKY Algorithm

Built-In Assumptions

- *Chomsky Normal Form* grammars: $\alpha \rightarrow \beta_1 \beta_2$ or $\alpha \rightarrow \gamma$ ($\beta_i \in C$, $\gamma \in \Sigma$);
- breadth-first (aka exhaustive): always compute all values for each cell;
- rigid control structure: bottom-up, left-to-right (one diagonal at a time).

Generalized Chart Parsing

- Liberate order of computation: no assumptions about earlier results;
- *active edges* encode partial rule instantiations, 'waiting' for additional (adjacent and passive) constituents to complete: $[1, 2, \text{VP} \rightarrow \text{V} \bullet \text{NP}]$;
- parser can fill in chart cells in *any* order and guarantee completeness.

Chart Parsing — Specialized Dynamic Programming

Basic Notions

- Use *chart* to record partial analyses, indexing them by string positions;
- count inter-word vertices; CKY: chart row is *start*, column *end* vertex;
- treat multiple ways of deriving the same category for some substring as *equivalent*; pursue only once when combining with other constituents.

Chart Parsing — Specialized Dynamic Programming

Basic Notions

- Use *chart* to record partial analyses, indexing them by string positions;
- count inter-word vertices; CKY: chart row is *start*, column *end* vertex;
- treat multiple ways of deriving the same category for some substring as *equivalent*; pursue only once when combining with other constituents.

Key Benefits

- Dynamic programming (memoization): avoid recomputation of results;
- efficient indexing of constituents: no search by start or end positions;
- compute *parse forest* with exponential 'extension' in *polynomial* time.

Chart Parsing: Key Ideas

- The parse *chart* is a two-dimensional matrix of *edges* (aka chart items);
- an edge is a (possibly partial) rule instantiation over a substring of input;
- the chart indexes edges by start and end string position (aka vertices);
- dot in rule RHS indicates degree of completion: $\alpha \rightarrow \beta_1 \ldots \beta_{i-1} \bullet \beta_i \ldots \beta_n$;
- *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 Fundamental Rule

$$[i,\ j,\ \alpha \rightarrow \beta_1 \ldots \beta_{i-1} \bullet \beta_i \ldots \beta_n] + [j,\ k,\ \beta_i \rightarrow \gamma^+ \bullet]$$
$$\mapsto [i,\ k,\ \alpha \rightarrow \beta_1 \ldots \beta_i \bullet \beta_{i+1} \ldots \beta_n]$$

An Example of a (Near- and Over-)Complete Chart

	1	2	3	4	5
0	NP → NP • PP S → NP • VP NP → kim •				S → NP VP •
1		VP → V • NP V → adores •	VP → VP • PP VP → V NP •		VP → VP • PP VP → VP PP • VP → V NP •
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* $_1$ *adores* $_2$ *snow* $_3$ *in* $_4$ *Oslo* $_5$

(Even) More Active (and Passive) Edges

	0	1	2	3
0	$S \rightarrow \bullet\ NP\ VP$ $NP \rightarrow \bullet\ NP\ PP$ $NP \rightarrow \bullet$ kim	$S \rightarrow NP \bullet VP$ $NP \rightarrow NP \bullet PP$ $NP \rightarrow$ kim $\bullet$ kim		$S \rightarrow NP\ VP \bullet$
1		$VP \rightarrow \bullet\ VP\ PP$ $VP \rightarrow \bullet\ V\ NP$ $V \rightarrow \bullet$ adores	$VP \rightarrow V \bullet NP$ $V \rightarrow$ adores $\bullet$ adores	$VP \rightarrow VP \bullet PP$ $VP \rightarrow V\ NP \bullet$
2			$NP \rightarrow \bullet\ NP\ PP$ $NP \rightarrow \bullet$ snow	$NP \rightarrow NP \bullet PP$ $NP \rightarrow$ snow $\bullet$ snow
3				

- Include all grammar rules as *epsilon* edges in each $chart_{[i,i]}$ cell.
- after initialization, apply *fundamental rule* until fixpoint is reached.

Combinatorics: Keeping Track of Remaining Work

The Abstract Goal

- Any chart parsing algorithm needs to check all pairs of adjacent edges.

A Naïve Strategy

- Keep iterating through the complete chart, combining all possible pairs, until no additional edges can be derived (i.e. the fixpoint is reached);
- frequent attempts to combine pairs multiple times: deriving 'duplicates'.

Combinatorics: Keeping Track of Remaining Work

The Abstract Goal

- Any chart parsing algorithm needs to check all pairs of adjacent edges.

A Naïve Strategy

- Keep iterating through the complete chart, combining all possible pairs, until no additional edges can be derived (i.e. the fixpoint is reached);
- frequent attempts to combine pairs multiple times: deriving 'duplicates'.

An Agenda-Driven Strategy

- Combine each pair exactly once, viz. when both elements are available;
- maintain *agenda* of new edges, yet to be checked against chart edges;
- new edges go into agenda first, add to chart upon retrieval from agenda.

Backpointers: Recording the Derivation History

	0	1	2	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 → • adores	12: VP → 11 • NP 11: V → adores •	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

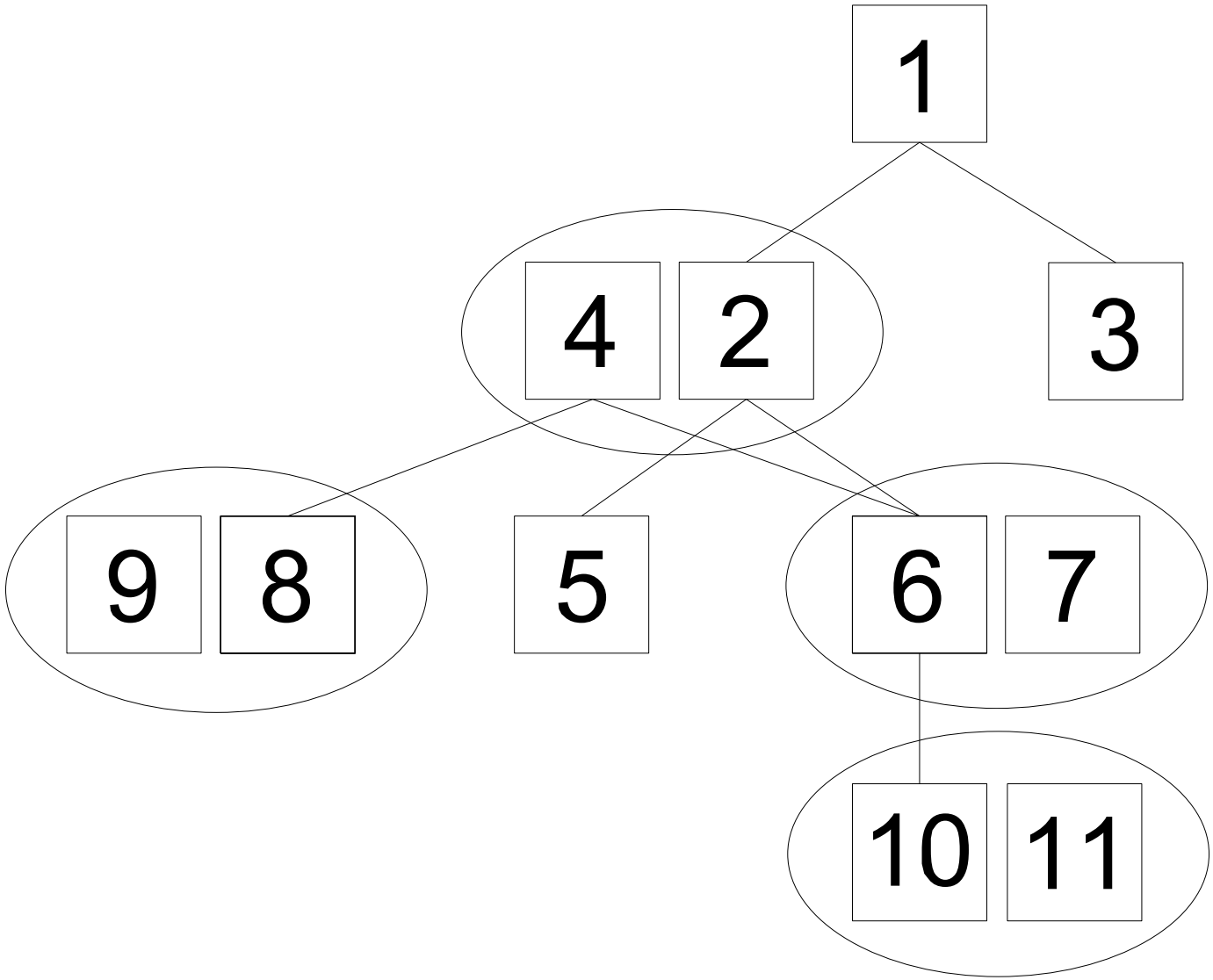
- Maintain only one edge for each α from i to j (the 'representative');
- record alternate sequences of daughters for α in the representative.

Implementation

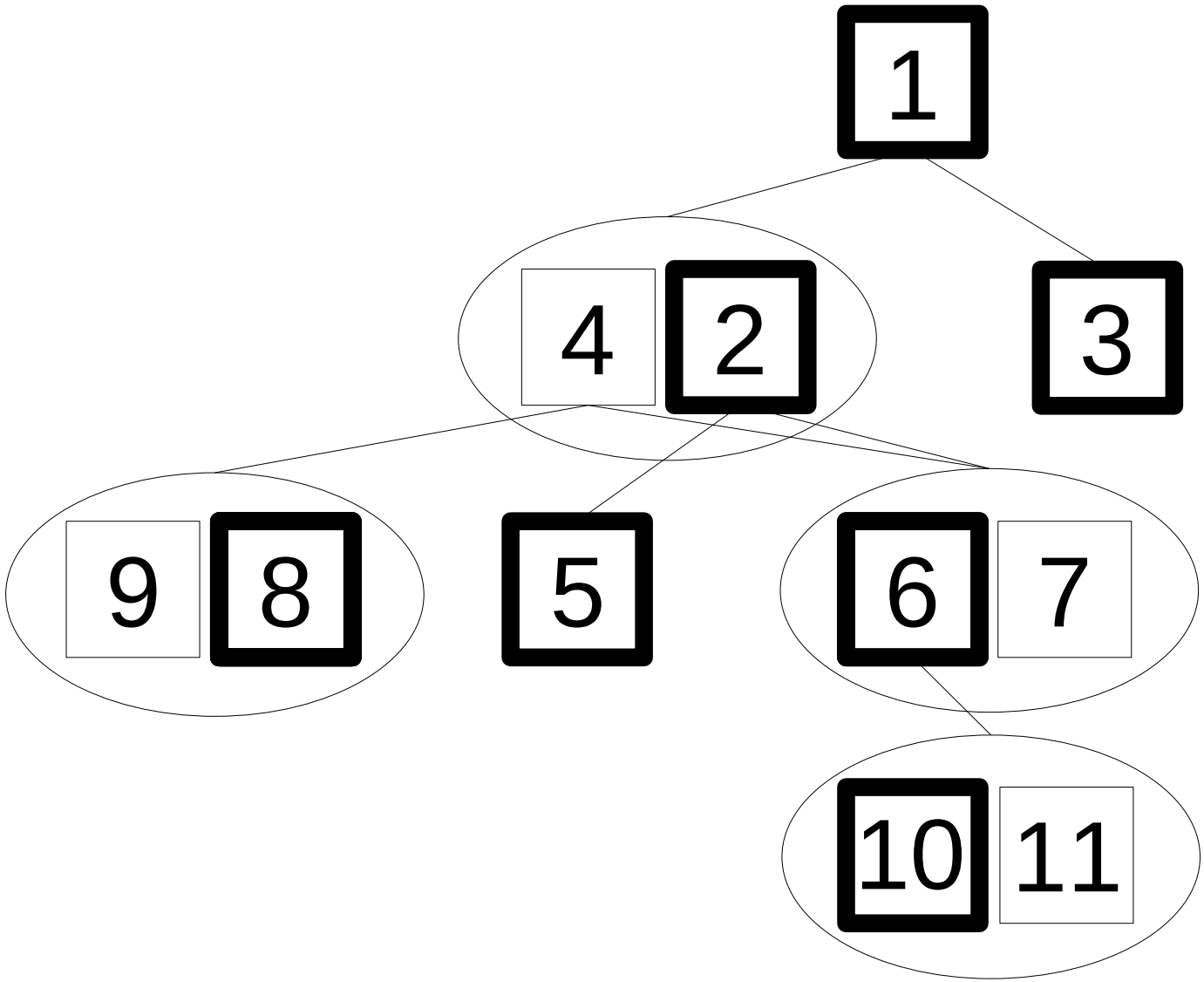
- Group passive edges into *equivalence classes* by identity of α, i, and j;
- search chart for existing equivalent edge (h, say) for each new edge e;
- when h (the 'host' edge) exists, *pack* e into h to record equivalence;
- e *not* added to the chart, no derivations with or further processing of e;

$\rightarrow$ *unpacking* multiply out all alternative daughters for all result edges.

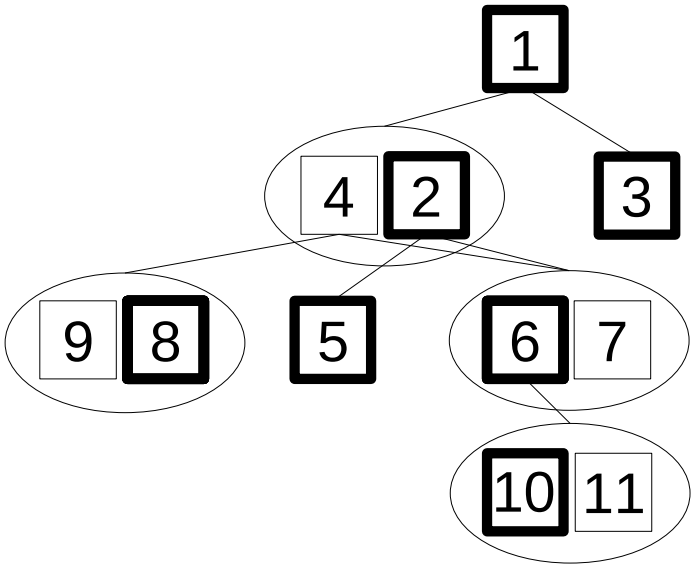
An Example (Hypothetical) Parse Forest



An Example (Hypothetical) Parse Forest



Unpacking: Cross-Multiplying Local Ambiguity



$$
\begin{aligned}
1 &\rightarrow \langle 2\ 3 \rangle \mid \langle 4\ 3 \rangle \\
2 &\rightarrow \langle 5\ 6 \rangle \mid \langle 5\ 7 \rangle \\
4 &\rightarrow \langle 8\ 6 \rangle \mid \langle 8\ 7 \rangle \mid \langle 9\ 6 \rangle \mid \langle 9\ 7 \rangle \\
6 &\rightarrow \langle 10 \rangle \mid \langle 11 \rangle
\end{aligned}
$$

How many complete trees in total?

In Conclusion—What Happened this Far

Syntactic Structure

- Languages (formal or natural) exhibit complex, hierarchical structures;
- grammars encode rules of the language: dominance and sequencing;
- context-free grammar ‘generates’ a language: strings and derivations;
- ambiguity in natural language grows exponentially: a search problem;
- bounding (or ‘packing’) of local ambiguity madantory for tractability;
- chart parsing uses dynamic programming: free order of computation.

In Conclusion—What Happened this Far

Syntactic Structure

- Languages (formal or natural) exhibit complex, hierarchical structures;
- grammars encode rules of the language: dominance and sequencing;
- context-free grammar ‘generates’ a language: strings and derivations;
- ambiguity in natural language grows exponentially: a search problem;
- bounding (or ‘packing’) of local ambiguity madantory for tractability;
- chart parsing uses dynamic programming: free order of computation.

Coming up Next

- Viterbi adaptation over parse forest; PTB parsing; parser evaluation.

Ambiguity Resolution Remains a (Major) Challenge

The Problem

- With broad-coverage grammars, even moderately complex sentences typically have multiple analyses (tens or hundreds, rarely thousands);
- unlike in grammar writing, exhaustive parsing is useless for applications;
- identifying the 'right' (intended) analysis is an 'AI-complete' problem;
- inclusion of (non-grammatical) sortal constraints nowadays undesirable.

Once Again: Probabilities to the Rescue

- Design and use statistical models to select among competing analyses;
- for string S, some analyses T_i are more or less likely: maximize $P(T_i|S)$;

$\rightarrow$ Probabilistic Context Free Grammar (PCFG) is a CFG plus probabilities.

Probability Theory and Natural Language?

The most important questions of life are, for the most part, really only questions of probability. (Pierre-Simon Laplace, 1812)

Probability Theory and Natural Language?

The most important questions of life are, for the most part, really only questions of probability. (Pierre-Simon Laplace, 1812)

Special wards in lunatic asylums could well be populated with mathematicians who have attempted to predict random events from finite data samples. (Richard A. Epstein, 1977)

Probability Theory and Natural Language?

The most important questions of life are, for the most part, really only questions of probability. (Pierre-Simon Laplace, 1812)

Special wards in lunatic asylums could well be populated with mathematicians who have attempted to predict random events from finite data samples. (Richard A. Epstein, 1977)

But it must be recognized that the notion 'probability' of a sentence is an entirely useless one, under any known interpretation of this term. (Noam Chomsky, 1969)

Probability Theory and Natural Language?

The most important questions of life are, for the most part, really only questions of probability. (Pierre-Simon Laplace, 1812)

Special wards in lunatic asylums could well be populated with mathematicians who have attempted to predict random events from finite data samples. (Richard A. Epstein, 1977)

But it must be recognized that the notion 'probability' of a sentence is an entirely useless one, under any known interpretation of this term. (Noam Chomsky, 1969)

Every time I fire a linguist, system performance improves. (Fredrick Jelinek, 1980s)

Generalized Chart Parsing

Initialization

- for each *word* in input string
  - add passive lexical edge $\langle word \bullet \rangle$ to chart
  - for each $\alpha \rightarrow word \in P$
    - add passive $\langle \alpha \rightarrow word \bullet \rangle$ edge to agenda

Main Loop

- while *edge* $\leftarrow$ pop-agenda()
  - if equivalent edge in chart, pack; otherwise insert *edge*
  - if *edge* is passive
    - for each active edge *a* to the left, fundamental-rule(*a*, *edge*)
    - predict new edges from P, and add to the agenda
  - else
    - for each passive edge *p* to the right, fundamental-rule(*edge*, *p*)

Termination

- return all edges with category S that span the full input

Viterbi Decoding over the Parse Forest

- Recall the Viterbi algorithm for HMMs

$$v_i(x) = \max_{k=1}^{L} [v_{i-1}(k) \cdot P(x|k) \cdot P(o_i|x)]$$

Viterbi Decoding over the Parse Forest

- Recall the Viterbi algorithm for HMMs

$$v_i(x) = \max_{k=1}^{L} [v_{i-1}(k) \cdot P(x|k) \cdot P(o_i|x)]$$

- In our parse forest, we no longer have a linear order, but we can still build up cached Viterbi values successively:

$$v(e) = \max \left[P(\beta_1, \ldots \beta_n|\alpha) \times \prod_i v(\beta_i) \right]$$

- Similar to HMM decoding, we also need to keep track of the set of daughters that led to the maximum probability.

Viterbi Decoding over the Parse Forest

- Recall the Viterbi algorithm for HMMs

$$v_i(x) = \max_{k=1}^{L} \left[v_{i-1}(k) \cdot P(x|k) \cdot P(o_i|x) \right]$$

- In our parse forest, we no longer have a linear order, but we can still build up cached Viterbi values successively:

$$v(e) = \max \left[P(\beta_1, \ldots \beta_n | \alpha) \times \prod_i v(\beta_i) \right]$$

- Similar to HMM decoding, we also need to keep track of the set of daughters that led to the maximum probability.
- Implementation: Cache the highest-scoring edge within e, recording the maximum probability of its sub-tree and the daughter sequence that led to it.