Chart Parsing

- General search methods are not best for syntactic parsing because the same syntactic constituent may be rederived many times as a part of larger constituents.
  
  For example, an NP could be part of different VPs or PPs.

- Chart parsing uses a chart to keep track of partial derivations so nothing has to be rederived.

- Chart parsers also use an agenda to prioritize the constituents to be processed. The agenda can be implemented as a stack or a queue to simulate depth-first or breadth-first search, or as a priority queue for best-first search.

Bottom-up Chart Parsing

1. If the agenda is empty, look up the next word (scanning the sentence left to right) in the lexicon and add its possible parts-of-speech to the agenda. Include its position in the input sentence.

   For example, if the first word is an article then it would be added as ART ($p_0$-$p_1$).

2. Select a constituent $C$ from the agenda.

3. Add $C$ to the chart using the arc extension algorithm.

The Arc Extension Algorithm

To add a constituent $C$ that spans positions $p_i$-$p_j$

1. **INSERT** $C$ into the chart in positions $p_i$-$p_j$.

2. **SEARCH** for grammar rules that begin with the constituent $C$.
   
   For each rule, add an active arc to the chart of the form:
   
   $$X \rightarrow C^* X_1 X_2 ... X_n$$
   
   spanning positions $p_i$-$p_j$.

3. **EXTEND** all active arcs of the form $X \rightarrow X_1 ... C^* ... X_n$ spanning positions $p_{m^*}$-$p_{n^*}$ by adding a COPY of the arc to the chart that includes the new constituent:

   $$X \rightarrow X_1 ... C^* ... X_n$$

   spanning positions $p_{m^*}$-$p_{n^*}$.

   Do **not** delete the original active arc, always create a copy!!

4. For each **NEWLY COMPLETED ARC** of the form $X \rightarrow X_1 ... X_n C^*$, spanning positions $p_{m^*}$-$p_{n^*}$, add a new constituent of type $X$ onto the agenda for positions $p_{m^*}$-$p_{n^*}$.

An Example: the grammar and lexicon

Sentence: *The smelly dog drank the water.*

Grammar

- $S \rightarrow NP VP$
- $NP \rightarrow ART N$
- $NP \rightarrow ADJ N$
- $NP \rightarrow ART ADJ N$
- $VP \rightarrow V$
- $VP \rightarrow V NP$

Lexicon

- the: ART
- smelly: ADJ
- dog: V N
- drank: V
- water: V N
Top-down vs. Bottom-up Chart Parsing

- Bottom-up chart parsing checks the input and builds each constituent exactly once. Avoids duplication of effort!

- But bottom-up chart parsing may build constituents that cannot be used legally.
  For example, given “the can”, bottom-up parsing will build a VP for “can” even if no grammar rules allow a VP to follow an article.

- Top-down chart parsing is highly predictive. Only grammar rules that can be legally applied will be put on the chart.

Top-Down Chart Parsing Algorithm (Earley Algorithm)

Initialize: For each grammar rule with $S$ as its left-hand side, add the arc $S \rightarrow \circ X_1...X_k$ to the chart using the top-down arc introduction algorithm.

While there are input words left:
1. If the agenda is empty, add the next word to the agenda.
2. Select a constituent $C$ from the agenda.
3. Combine $C$ with every active arc on the chart using the arc extension algorithm. Add any new constituents to the agenda.
4. For any active arcs created by step 3, add them to the chart using the top-down arc introduction algorithm.

Top-Down Arc Introduction Algorithm

When adding arc $Y \rightarrow C_1... \circ C_i...C_n$ ending at position $j$:

For each rule $C_i \rightarrow X_1...X_k$ in the grammar, recursively add a new arc $C_i \rightarrow \circ X_1...X_k$ from position $j$ to $j$.

Chart Parsing: Pros and Cons

Pros
- Relatively efficient because each constituent is generated exactly once.
- Easy to generate a single parse, $N$ parses, or all possible parses.
- If no complete parse is found, then easy to gather pieces and construct a partial parse.
- Could search for only certain types of constituents, such as NPs and VPs.

Cons
- Parsing is still not cheap, especially if the grammar is large.
- The chart can become quite large in some cases.
Modifying a Chart Parser to Handle Features

Given an arc on the chart of the form:

\[ X \rightarrow X_1 \ldots \circ X_i \ldots X_n \]

and a constituent of type \( C \) that can extend the arc:

1. Find an instantiation of the variables such that all features specified in \( X_i \) are satisfied by \( C \)

2. Create a copy of the arc with the variables instantiated as specified by the previous step

3. Add the copy to the chart in the usual fashion

An Example

Suppose the following arc is on the chart:

\[ \text{NP agr} <w:3s,3p> \rightarrow (\text{ART agr} <w:3s,3p>) \circ (\text{N agr} <w:3s,3p>) \]

And the arc can be extended with the constituent:

\[ \text{N root table agr 3s} \]

The new arc would be:

\[ \text{NP agr} <w:3s> \rightarrow (\text{ART agr} <w:3s>) \circ (\text{N agr} <w:3s>) \]

Syntactic Ambiguities

Syntactic ambiguities can cause the number of possible parse trees to explode.

- PP attachment ambiguity:
  \[ I \text{ saw the man on the hill with a telescope} \]

- Noun phrase bracketing:
  \[ \text{plastic cat food can cover} \]

- Conjunctions and appositives:
  \[ \text{Rover, my fish, and Fluffy} \]

A medium-sized sentence with a moderate grammar can have over 1000 legal parse trees!

Dealing with Syntactic Ambiguity

One approach to minimizing syntactic ambiguity is to modify the grammar. Consider this grammar:

\[
\begin{align*}
NP & \rightarrow NP1 \\
NP1 & \rightarrow \text{noun} \\
NP1 & \rightarrow NP1 \text{ NP1}
\end{align*}
\]

versus this grammar rule (if the Kleene + operator is legal):

\[ NP \rightarrow \text{noun} + \]

or if the Kleene + operator is not legal:

\[
\begin{align*}
NP & \rightarrow \text{noun} \\
NP & \rightarrow \text{noun noun} \\
NP & \rightarrow \text{noun noun noun} \\
NP & \rightarrow \text{noun noun noun noun}
\end{align*}
\]

These grammar rules look more complicated but eliminate all structural ambiguity!
Packing

- Another way to reduce structural ambiguity is to use a packed chart.

- Packing combines constituents of the same type that include the same words, no matter how they were derived.

  For example, \([\text{cat food can}] \text{ and } [\text{cat food can}]\)
  would be combined into the same NP.

- You can keep track of the different derivations for a constituent even when the chart contains only one item representing them all.

Shallow Parsing

- Instead of trying to generate a complete parse tree for a sentence, shallow parsers generate fragments representing local syntactic constituents. (Also called partial parsers or chunkers.)

- Shallow parsers typically try to identify NPs, VPs, and PPs (and occasionally other constituents).

- These local syntactic constituents can be identified (relatively) reliably using simple grammar rules and heuristics.

- Most shallow parsers use finite state machines to recognize a regular grammar.

Shallow Parsing as Classification

Shallow parsers can be built with supervised learning techniques using an annotated training corpus.

The trick is to view shallow parsing as a classification or tagging problem. A common scheme is IOB tagging, where B=Beginning, I=Internal, and O=Outside.

\(\text{John Smith gave Mary a book about NLP.}\)
Benefits of Shallow Parsing

- Deep syntactic structure may not be important for some NLP applications.

- Some ambiguity issues can be ignored if they are not critical for identifying the fragments.

- Some structural issues can be delayed and left for semantic analysis.

- Shallow parsers are more robust with ungrammatical or ill-formed input.

- Shallow parsers are usually much faster than full parsers.

Weaknesses of Shallow Parsing

- Usually does not handle embedded relative clauses well.
  
  Ex: *I gave the boy that was sick some medicine.*

- Often has trouble recognizing reduced relative clauses.
  
  Ex: *The woman killed last night was an important diplomat.*

- Attachments are usually not attempted.