What’s the Problem?

- Natural Languages are only mildly context sensitive, and most constructions can be handled as context-free.
  “the overwhelming majority of syntactic structures are projective in most languages” Nivre (2010)

- We’ve already seen a number of parsing algorithms for context-free languages (CKY, active charts, Earley).

- Why consider dependency parsing as a distinct topic?

---

Dependency Graphs and Dependency Trees

Given their properties (connected, single-headed, acyclic, and projective, cf. Lecture 15), projective dependency graphs are actually trees.

I heard Cecilia teach the horses to sing
Why Dependency Parsing? - 1

Dependency trees provide direct links from predicates to arguments

- useful in many applications that use syntactic parsing
- CFG-based approaches often extract this same information from trees anyway
- more suitable for languages with free or flexible word order, where indirect encoding of grammatical functions in PS trees can become very complex

Phrase Structure Trees vs. Dependency Trees

Why Dependency Parsing? - 2

Reduces complexity of parsing

- constructing a dependency tree for sentence \( S = w_1, \ldots, w_n \) requires assigning a syntactic head \( h(w_i) \) and a dependency label \( d(w_i) \) to each \( w_i \)
- much more constrained task than building a PS tree, with arbitrarily many non-terminal nodes

Why Dependency Parsing - Summary

- Dependency trees provide a transparent encoding of predicate-argument structure despite their constrained nature
- Dependency trees provide support for efficient and accurate parsing because of their constrained nature Nivre (2010)
- Good balance of expressivity and complexity
Three Types of Dependency Parsers Nivre (2010)

1. Mapping dependency trees to phrase structure trees, then use standard CF parser such as CKY (or a bespoke form, designed for CF dependency grammars);
   - Eisner (1996) gives a clever algorithm that reduces the complexity to $O(n^3)$, by producing parse items with heads at the ends rather than in the middle
2. Graph-based dependency parsing, based on maximum spanning trees (MST parser)
   - scores dependencies independently using a machine learned classifier
3. Transition-based dependency parsing
   - an extension of shift-reduce parsing (MALT parser).
   - greedy choice of attachments guided by machine learning classifiers

Dep Tree to PS Tree: Standard Dep Tree

For each node $w$ with outgoing arcs (i.e., each node that has dependents),
- Map the subtree rooted at $w$, with direct arcs to subtrees $t_1, \ldots, t_n$, to
- a subtree rooted at node $X_w$ dominating a terminal node $w$ and subtrees corresponding to $t_1, \ldots, t_n$ in the order given by the original word order.

Context Free Dependency Parsing

Based on the observation that any projective dependency tree can be mapped to an equivalent phrase structure tree in which non-terminal symbols are indexed by words

Start by looking at mapping dependency trees (Dep Trees) to phrase structure (PS) Trees. They will not be standard PS trees like:
The grammar is context-free, but fully lexicalized, with one or more rules for each "word phrase" (e.g., Xnews) or labelled word phrase (e.g., Xnews:subj)

X\_had \xrightarrow{} X\_news had X\_effect X\_\-.
X\_effect \xrightarrow{} X\_little effect X\_on
X\_on \xrightarrow{} on X\_markets
X\_little \xrightarrow{} little

Eisner (1996) showed that the worst-case complexity of parsing these lexicalized grammars can be reduced by processing left and right dependents independently.
Graph-based Dependency Parsing

**Goal:** Find highest scoring dependency tree $T$ in the space of all possible trees for a sentence $S$.

- If $S$ is unambiguous, $T$ is its correct structure;
- If $S$ is ambiguous, $T$ is its most likely structure.

Search is guided by a function for scoring dependency in terms of their probabilities

Function is induced from syntactically annotated corpora (i.e., treebanks) using machine learning

Purely data-driven method, no grammatical constraints are assumed

Reminder: Global Linear Models (GLMs)

- A generating function $GEN$ maps an input $x$ to candidate outputs $y_1, \ldots, y_n$, so that $GEN(x) = \{y_1, \ldots, y_n\}$
- Each feature function $f_i$ maps a pair $(x, y_j)$ to a feature value $f_i(x, y_j)$
- Features are combined in a linear model:
  $$ \sum_i \lambda_i f_i(x, y) $$
- The goal of learning is to find the feature weights $\lambda_i$ so as to find the best $y_j$.

Graph-based Dependency Parsing as a GLM

- For graph-based dependency parsing, $GEN$ maps an input sentence $x$ to its complete set of dependency parse trees.
- We have a set of feature functions $f_i$ that map $x$ and a dependency parse tree $y$ to a feature value $f_i(x, y)$
- Features are combined using a linear model – usually the perceptron algorithm or a large-margin version of the perceptron.

We will look at this now in more detail.
Maximum Spanning Tree Parsing

- McDonald et al. (2005) propose an extreme version of GLM graph-based dependency parsing that starts with a totally connected dependency graph, with a directed edge between every ordered pair of distinct words and a directed edge from the root to every word.
- A linear model learned from labeled data assigns a weighted score \( s(i,j) \) to each directed edge, where \( s(i,j) = \lambda f(i,j) \)
- In general, \( s(i,j) \neq s(j,i) \)
- Finding the highest scoring dependency tree \( \equiv \) finding the maximum spanning tree (MST) in a graph containing all possible arcs.
- This can be solved in \( O(n^2) \) time using the Chu-Liu-Edmonds algorithm, which can find both projective and non-projective MSTs.

Chu-Liu-Edmonds (CLE) Algorithm

- Each node \( j \) in the graph greedily selects the incoming edge with the highest score \( s(i,j) \).

E.g., Of all the arcs coming in to the node labelled John, the one with weight 30 \( > > 11 \) \( > > 9 \).
- If the resulting graph is a tree, it is the maximum spanning tree.
Computing Scores for $C$

\[
s(Mary, C) = 11 + 20 = 31 \\
s(ROOT, C) = 10 + 30 = 40
\]

Recursion Step for CLE Algorithm

- An MST in the contracted graph can be transformed into an MST for the original graph, so we recursively call CLE algorithm on the contracted graph—
  - Greedily collecting incoming edges to all nodes:

Reconstruction Step for CLE Algorithm

- Reconstruct the uncontracted graph, in which the outward edge from the contracted node to Mary was from the word saw, while the inward edge from ROOT was to saw and from there to John.

Arc-factored Model

- Using CLE algorithm with the arc-factored model
  - finds highest scoring dependency tree very efficiently during parsing, $O(n^2)$
  - can achieve state-of-the-art parsing accuracy (McDonald et al. 2005)
- But parsing accuracy depends on how the scoring function for subgraphs is learned from treebank data
- Standard approach, pioneered by McDonald et al. (2005) uses the perceptron algorithm or a large-margin version, to learn a weight vector that favors globally optimal dependency trees.
• Simple categorical features used in scoring \((w_i, l, w_j)\) include:
  - Identity of \(w_i\), PoS-tag of \(w_i\)
  - Identity of \(w_j\), PoS-tag of \(w_j\)
  - Label of \(l\), Direction of \(l\)
  - Sequence of PoS-tags between \(w_i\) and \(w_j\)
  - PoS-tag of \(w_{i-1}\), PoS-tag of \(w_{j-1}\)
  - PoS-tag of \(w_{j+1}\), PoS-tag of \(w_{i+1}\)
  - Number words between \(w_i\) and \(w_j\)

• Does not require spanning tree to be projective, so this is an efficient parsing algorithm for arbitrary dependency trees

---

**Transition-Based Dependency Parsing**

• Completely data-driven, like graph-based parsing relies on machine learning from treebank data

• Instead of learning to score dependency trees, factored into subgraphs, a transition-based parser learns to score possible next actions in a state machine for deriving dependency trees

• Main advantage is that scoring function learned can take a very rich set of features into account, although the model is only optimized w.r.t. local decisions as opposed to global structures

• See Stanford NLP lecture on “Greedy Transition-Based Parsing”

---

**Summary**

• MST, graph-based dependency parser,
  - finds highest scoring dependency tree from subgraphs, i.e., learns a globally optimal dependency tree, \(O(n^2)\) using CLE algorithm
  - can deal with non-projective dependency structures

• MALT, transition-based dependency parser,
  - can optimize with respect to local decisions based on a very rich set of features
  - provides VERY fast linear time parsing, \(O(n)\)
  - cannot deal with non-projective dependency structures

• McDonald and Nivre compare accuracy for many languages of graph-based dependency parsing (GB) and transition-based dependency parsing (TB)

• Comparison shows that GB has high precision (around 70%) but low recall (falling to around 40%) of long range dependencies, including the non-projective ones

• TB is slightly better on short range dependencies.
Handling Non-projectivity

• Just declare defeat on nonprojective arcs
• Use a dependency formalism that only admits projective representations (like CFG). Works reasonably well for English.
• Use a postprocessor to a projective dependency parsing algorithm to identify and resolve nonprojective links
• Add extra types of transitions that can model at least most non-projective structures
• Move to a parsing mechanism that does not use or require any constraints on projectivity (e.g., the graph-based MST Parser)

References


