Dependency Parsing

Prashanth Mannem
mannemp@eecs.oregonstate.edu
Syntax

- Study of the way sentences are constructed from smaller units
- Formal systems that enable this
  - Phrase Structure Grammar
  - Dependency Grammar
- More
  - Tree Adjoining Grammar (TAG),
  - Categorical Grammar
Phrase Structure Grammar

- Constituents as building blocks
- Phrase structure rules to form constituents
  - Recursive
  - Lexicalized

[S [NP Sue/NNP] [VP walked/VBD [PP into/P [NP the/DT store/NN ]]]]
Dependency Grammar

• The idea of dependency structure goes back a long way
  • To Pāṇini’s grammar (c. 5th century BCE)

• Constituency is a new invention
  • 20th century

• Modern work often linked to work of L. Tesniere (1959)
  • Dominant approach in “East” (Eastern bloc/East Asia)

• Among the earliest kinds of parsers in NLP, even in US:
  • David Hays, one of the founders of computational linguistics, built early (first?) dependency parser (Hays 1962)
Dependency Grammar

- Dependency tree of a sentence is a set of modifier-modified relations/dependencies
- Represented by a directed arc modifier ← modified
- NP phrases

The dog

The huge dog

The huge lovable dog

dog

dog

dog

the huge

the huge lovable
Dependency Grammar

dog with a very loud bark

the huge lovable
very loud bark with a very lovable huge loud bark
very loud bark with a dog

the huge lovable

loud very bark

Dependency Grammar
Dependency Grammar

dog --with-- bark

the huge lovable

a loud

very
very loud bark with

the huge lovable

a loud very
very loud bark with the huge lovable dog

a loud very
the huge lovable dog with a very loud bark
Dependency Grammar

- Syntactic structure consists of lexical items, linked by binary asymmetric relations called dependencies.

- Interested in grammatical relations between individual words (governing & dependent words).

- Does not propose a recursive structure
  - Rather a network of relations.

- These relations can also have labels.
Red figures on the screen indicated falling stocks

John booked me a flight from Houston to Portland to attend the seminar
John booked me a flight from Houston to Portland to attend the seminar.
Red figures on the screen indicated falling stocks.

John booked me a flight from Houston to Portland to attend the seminar.
Phrasal nodes are missing in the dependency structure when compared to constituency structure.
Comparison

- **Dependency structures explicitly represent**
  - Head-dependent relations (**directed arcs**)
  - Functional categories (**arc labels**)
  - Possibly some structural categories (**parts-of-speech**)

- **Phrase structure explicitly represent**
  - Phrases (**non-terminal nodes**)
  - Structural categories (**non-terminal labels**)
  - Possibly some functional categories (**grammatical functions**)
Parsing DG over PSG

- **Dependency Parsing** is more straightforward
  - Parsing can be reduced to labeling each token $w_i$ with $w_j$

- Direct encoding of predicate-argument structure
  - Fragments are directly interpretable

- Dependency structure independent of word order
  - Suitable for free word order languages (like Indian languages)
Outline

- Introduction
- **Dependency Parsing**
  - Formal definition
- **Parsing Algorithms**
  - Introduction
  - Dynamic programming
  - Deterministic search
Dependency Tree

- **Formal definition**
  - An input word sequence $w_1 \ldots w_n$
  - Dependency graph $D = (W, E)$ where
    - $W$ is the set of nodes i.e. word tokens in the input seq.
    - $E$ is the set of unlabeled tree edges $(w_i, w_j)$ ($w_i, w_j \in W$).
    - $(w_i, w_j)$ indicates an edge from $w_i$ (parent) to $w_j$ (child).

- Task of mapping an input string to a dependency graph satisfying certain conditions is dependency parsing
Well-formedness

- A dependency graph is well-formed iff

  - **Single head**: Each word has only one head.

  - **Acyclic**: The graph should be acyclic.

  - **Connected**: The graph should be a single tree with all the words in the sentence.

  - **Projective**: If word A depends on word B, then all words between A and B are also subordinate to B (i.e. dominated by B).
Non-projective dependency tree

* Crossing lines

English has very few non-projective cases.
Outline

- Introduction
  - Phrase Structure Grammar
  - Dependency Grammar
  - Comparison and Conversion

- Dependency Parsing
  - Formal definition

- Parsing Algorithms
  - Introduction
  - Dynamic programming
  - Deterministic search
Dependency Parsing

- Dependency based parsers can be broadly categorized into
  - **Grammar driven** approaches
    - Parsing done using grammars.
  - **Data driven** approaches
    - Parsing by training on annotated/un-annotated data.
Dependency Parsing

- Dependency based parsers can be broadly categorized into
  - **Grammar driven** approaches
    - Parsing done using grammars.
  - **Data driven** approaches
    - Parsing by training on annotated/un-annotated data.

- These approaches are **not** mutually exclusive.
Covington’s Incremental Algorithm

- Incremental parsing in $O(n^2)$ time by trying to link each new word to each preceding one [Covington 2001]:

\[
\text{PARSE}(x = (w_1, \ldots, w_n))
\]

1. \text{for } i = 1 \text{ up to } n
2. \text{for } j = i - 1 \text{ down to } 1
3. \text{LINK}(w_i, w_j)
Covington’s Incremental Algorithm

- Incremental parsing in $O(n^2)$ time by trying to link each new word to each preceding one [Covington 2001]:

\[
\text{PARSE}(x = (w_1, \ldots, w_n))
\]

1. for $i = 1$ up to $n$
2. for $j = i - 1$ down to $1$
3. \text{LINK}(w_i, w_j)

- Constraints such as \text{Single-Head} and Projectivity can be incorporated into the \text{LINK} operation.
Parsing Methods

- **Main traditions**
  - Dynamic programming
    - CYK, Eisner, McDonald MST
  - Deterministic search
    - Covington, Yamada and Matsumuto, Nivre
Dynamic Programming

- Basic Idea: Treat dependencies as constituents.
- Use, e.g., CYK parser (with minor modifications)
Dependency Chart Parsing

- Grammar is regarded as context-free, in which each node is lexicalized.
- Chart entries are subtrees, i.e., words with all their left and right dependents.
- **Problem:** Different entries for different subtrees spanning a sequence of words with different heads.
- $O(n^5)$
Generic Chart Parsing

for each of the $O(n^2)$ substrings, 
for each of $O(n)$ ways of splitting it, 
for each of $\leq S$ analyses of first half 
for each of $\leq S$ analyses of second half, 
for each of $\leq c$ ways of combining them: 
combine, & add result to chart if best 

$O(n^3S^2c)$

[cap spending] + [at $300$ million] = [[cap spending] [at $300$ million]] 

$\leq S$ analyses $\leq S$ analyses $\leq cS^2$ analyses 
of which we keep $\leq S$
Headed constituents ...

... have too many signatures.

**How bad is \( \Theta(n^3 S^2 c) \)?**

For **unheaded** constituents, \( S \) is constant: \( \text{NP, VP} \ldots \)
(similarly for dotted trees). So \( \Theta(n^3) \).

But when **different heads** \( \Rightarrow \) **different signatures**, the average
substring has \( \Theta(n) \) possible heads and \( S=\Theta(n) \) possible
signatures. So \( \Theta(n^5) \).
Dynamic Programming Approaches

- Original version \([\text{Hays 1964}]\) (grammar driven)
- Link grammar \([\text{Sleator and Temperley 1991}]\) (grammar driven)
- Bilexical grammar \([\text{Eisner 1996}]\) (data driven)
- Maximum spanning tree \([\text{McDonald 2006}]\) (data driven)
Eisner 1996

- Two novel aspects:
  - Modified parsing algorithm
  - Probabilistic dependency parsing

- Complexity: $O(n^3)$

- Modification: Instead of storing subtrees, store spans

- Span: Substring such that no interior word links to any word outside the span.

- Idea: In a span, only the boundary words are active, i.e. still need a head or a child

- One or both of the boundary words can be active
Example

Red figures on the screen indicated falling stocks.
Example

Spans:

\{ Red, figures \} \{ indicated, falling, stocks \}
Assembly of correct parse

Start by combining adjacent words to minimal spans

\{Red figures\} \{figures on\} \{on the\}
Assembly of correct parse

Combine spans which overlap in one word; this word must be governed by a word in the left or right span.
Assembly of correct parse

Combine spans which overlap in one word; this word must be governed by a word in the left or right span.
Assembly of correct parse

Combine spans which overlap in one word; this word must be governed by a word in the left or right span.

Invalid span
Assembly of correct parse

Combine spans which overlap in one word; this word must be governed by a word in the left or right span.

\{indicated, falling\} + \{falling, stocks\} → \{indicated, falling, stocks\}
Eisner 1996

- Two novel aspects:
  - Modified parsing algorithm
  - Probabilistic dependency parsing
- Complexity: $O(n^3)$
McDonald’s Maximum Spanning Trees

- Score of a dependency tree = sum of scores of dependencies
- Scores are independent of other dependencies
- If scores are available, parsing can be formulated as maximum spanning tree problem
- Two cases:
  - Projective: Use Eisner’s parsing algorithm.
- Uses online structured perceptron for determining weight vector $\mathbf{w}$
Parsing Methods

- **Main traditions**
  - Dynamic programming
    - CYK, Eisner, McDonald
  - Deterministic parsing
    - Covington, Yamada and Matsumuto, Nivre
Deterministic Parsing

- **Basic idea:**
  - Derive a single syntactic representation (dependency graph) through a **deterministic** sequence of **elementary** parsing actions
  - Sometimes combined with backtracking or repair

- **Motivation:**
  - Psycholinguistic modeling
  - Efficiency
  - Simplicity
Yamada and Matsumoto

- Parsing in several rounds: deterministic bottom-up $O(n^2)$
- Looks at pairs of words
- 3 actions: shift, left, right

- **Shift**: shifts focus to next word pair
Yamada and Matsumoto

**Left:** decides that the left word depends on the right one

```
I saw a girl with
PRP VBD DT NN IN
⇒
```

**Right:** decides that the right word depends on the left word

```
I saw girl with
PRP VBD NN IN
⇒
```

```
↑ a
DT
```

```
↑ a
DT
```

```
↑ girl
NN
```

```
↑ a
DT
```
Parsing Algorithm

- Go through each *pair* of words
  - Decide which *action* to take

- If a relation was detected in a pass, do another pass

- E.g. *the little girl*
  - First pass: relation between *little* and *girl*
  - Second pass: relation between *the* and *girl*

- Decision on action depends on word pair and context
Parsing

- Data-driven deterministic parsing:
  - Deterministic parsing requires an oracle.
  - An oracle can be approximated by a classifier.
  - A classifier can be trained using treebank data.

- Learning algorithms:
  - Maximum entropy modeling (MaxEnt) [Cheng et al. 2005]
  - Structured Perceptron [McDonald et al. 2006]
Evaluation of Dependency Parsing: Simply use (labeled) dependency accuracy

Accuracy = \frac{\text{number of correct dependencies}}{\text{total number of dependencies}}

= \frac{2}{5} = 0.40 = 40\%
Feature Models

Learning problem:

- Approximate a function from parser states, represented by feature vectors to parser actions,
  - Given a training set of gold standard trees.

Typical features:

- Tokens and POS tags of:
  - Target words
  - Linear context (neighbors in S and Q)
  - Structural context (parents, children, siblings in G)
  - Can not be used in dynamic programming algorithms.
Summary

- Provided an intro to dependency parsing and various dependency parsing algorithms
- Read up Nivre’s and McDonald’s tutorial on dependency parsing at ESSLLI’ 07
References

- Nivre’s and McDonald’s tutorial on dependency parsing at ESSLLI' 07

- Dependency Grammar and Dependency Parsing
  http://stp.lingfil.uu.se/~nivre/docs/05133.pdf

- Online Large-Margin Training of Dependency Parsers
  R. McDonald, K. Crammer and F. Pereira
  ACL, 2005

- Pseudo-Projective Dependency Parsing.
  Nivre, J. and J. Nilsson
  ACL, 2005
Phrase Structure Grammar

• Phrases (non-terminal nodes)
• Structural categories (non-terminal labels)
• CFG Rules
  o Recursive
  o Lexicalized

[Sue walked into the store]

[S [NP Sue] [VP walked into the store]]

[S [NP Sue] [VP [VBD walked] [PP into the store]]]

[S [NP Sue] [VP [VBD walked] [PP [P into] [NP the store]]]]

[S [NP Sue] [VP [VBD walked] [PP [P into] [NP [DT the] [NN store]]]]]

[S → NP VP]
[VP → VBD PP]
[PP → P NP]
[NP → DT NN]
Phrase Structure Grammar

[S Sue walked into the store]  
[S [NP Sue] [VP walked into the store]]  
[S [NP Sue] [VP [VBD walked] [PP into the store]]]  
[S [NP Sue] [VP [VBD walked] [PP [P into] [NP the store]]]]  
[S [NP Sue] [VP [VBD walked] [PP [P into] [NP [DT the] [NN store]]]]]

Phrases
(non-terminal nodes)

Structural categories
(non-terminal labels)
Eisner’s Model

- **Recursive Generation**
  - Each word generates its actual dependents
  - **Two Markov chains:**
    - Left dependents
    - Right dependents
Eisner’s Model

\[ P(tw(1), \ldots, tw(n), links) = \prod_{i=1}^{n} P(lc(i)|tw(i))P(rc(i)|tw(i)) \]

where

\[ tw(i) \text{ is } i^{\text{th}} \text{ tagged word} \]

\[ lc(i) \text{ & } rc(i) \text{ are the left and right children of } i^{\text{th}} \text{ word} \]

\[ P(lc(i)|tw(i)) = \prod_{j=1}^{m} P(tw(lc_{j}(i))|t(lc_{j-1}(i)), tw(i)) \]

\[ P(rc(i)|tw(i)) = \prod_{j=1}^{m} P(tw(rc_{j}(i))|t(rc_{j-1}(i)), tw(i)) \]

where

\[ lc_{j}(i) \text{ is the } j^{\text{th}} \text{ left child of the } i^{\text{th}} \text{ word} \]

\[ t(lc_{j-1}(i)) \text{ is the tag of the preceding left child} \]
Nivre’s Algorithm

- Four parsing actions:
  - **Shift**:
    
    \[
    \text{Shift} \; [\ldots]S \; [w_i, \ldots]Q \rightarrow [\ldots, w_i]S \; [\ldots]Q
    \]

  - **Reduce**:
    
    \[
    \text{Reduce} \; [\ldots, w_i]S \; [\ldots]Q \; \exists w_k : w_k \rightarrow w_i \rightarrow [\ldots]S \; [\ldots]Q
    \]

  - **Left-Arc**:
    
    \[
    \text{Left-Arc} \; [\ldots, w_i]S \; [w_j, \ldots]Q \; \neg \exists w_k : w_k \rightarrow w_i \rightarrow [\ldots]S \; [w_j, \ldots]Q \; w_i \leftarrow w_j
    \]

  - **Right-Arc**:
    
    \[
    \text{Right-Arc} \; [\ldots, w_i]S \; [w_j, \ldots]Q \; \neg \exists w_k : w_k \rightarrow w_j \rightarrow [\ldots, w_i, w_j]S \; [\ldots]Q \; w_i \rightarrow w_j
    \]
Nivre’s Algorithm

- Characteristics:
  - **Arc-eager** processing of right-dependents
  - Single pass over the input gives time worst case complexity $O(2n)$
Red figures on the screen indicated falling stocks.
Example

\[
\text{Shift}
\]
Example

\[
\left\{ \begin{array}{c}
\text{Red} \\
\text{figures on the screen indicated falling stocks}
\end{array} \right\}_Q
\]

Left-arc
Example

\[
\text{[ROOT] \ Red \ figures \ } \left( \text{on the screen indicated falling stocks} \right)
\]

Shift
Example

\[
\begin{array}{c}
\text{\_ROOT\_ } \text{Red} \quad \text{figures on } S \\
\end{array}
\begin{array}{c}
\text{the screen indicated falling stocks } Q
\end{array}
\]

Right-arc
Example

\[
\begin{array}{c}
\text{\_ROOT\_} \quad \text{Red} \\
\downarrow \quad \downarrow \\
\text{figures} \quad \text{on} \quad \text{the} \\
\end{array}
\]

\[
\begin{array}{c}
\text{\textit{screen indicated falling stocks}} \\
\end{array}
\]

Shift
Example

Left-arc

\[
\begin{array}{c}
\text{Red figures on the screen indicated falling stocks} \\
\text{S} \\
\text{Q}
\end{array}
\]
Example

Red figures on the screen indicated falling stocks

Right-arc
Example

Red figures on the screen indicated falling stocks.

Reduce
Example

Reduce
Example

```
\begin{array}{c}
\text{Red figures on the screen indicated falling stocks}
\end{array}
```

**Left-arc**
Example

Red figures on the screen indicated falling stocks

Right-arc
Example

Red figures on the screen indicated falling stocks

Shift
Example

Left-arc

Red figures on the screen indicated falling stocks
Example

Red figures on the screen indicated falling stocks

Right-arc
Example

Red figures on the screen indicated falling stocks

Reduce
Example

Red figures on the screen indicated falling stocks

Reduce