Dependency Grammars

CL1: Jordan Boyd-Graber

University of Maryland

October 7, 2013

Adapted from slides by Neelamadhav Gantayat and Ryan MacDonald
Outline

1. Motivation

2. Algorithms for Dependency Parsing

3. Evaluation and Estimation
- Turns sentence into syntactic structure
- Essential for information extraction and other NLP tasks

Lucien Tesnière, 1959

The sentence is an organized whole, the constituent elements of which are words. Every word that belongs to a sentence ceases by itself to be isolated as in the dictionary. Between the word and its neighbors, the mind perceives connections, the totality of which forms the structure of the sentence. The structural connections establish dependency relations between the words.
Basic Assumption: Syntactic structure essentially consists of lexical items linked by binary asymmetrical relations called dependencies.
Figure: Output of Stanford dependency parser
Verb has an artificial root

Notion of phrases: “by” and its children (more next week)

So how do we choose these edges?
$D$ is likely a dependent of head $H$ in construction $C$:

- $H$ determines syntactic category of $C$ and can often replace $C$
- $H$ gives semantic specification of $C$; $D$ specifies $H$
- $H$ is obligatory; $D$ may be optional
- $H$ selects $D$ and determines whether $D$ is obligatory
- The form of $D$ depends on $H$ (agreement or government)
- The linear position of $D$ is specified with reference to $H$
Some clear cases . . .

- Modifiers: “nmod” and “vmod”
- Verb slots: “subject” and “object”
Which direction?

Some tricky cases . . .

- Complex verb groups
- Subordinate clauses
- Coordination
- Prepositions
- Punctuation

I can see that they rely on this and that.
Some tricky cases . . .

- Complex verb groups
- Subordinate clauses
- Coordination
- Prepositions
- Punctuation

I can see that they rely on this and that.
Which direction?

Some tricky cases . . .

- Complex verb groups
- Subordinate clauses
- Coordination
- Prepositions
- Punctuation

I can see that they rely on this and that.
Which direction?

Some tricky cases . . .

- Complex verb groups
- Subordinate clauses
- Coordination
- Prepositions
- Punctuation

I can see that they rely on this and that.
Which direction?

Some tricky cases . . .

- Complex verb groups
- Subordinate clauses
- Coordination
- Prepositions
- Punctuation

I can see that they rely on this and that.
Some tricky cases . . .

- Complex verb groups
- Subordinate clauses
- Coordination
- Prepositions
- Punctuation

I can see that they rely on this and that.
Which direction?

Some tricky cases . . .

- Complex verb groups
- Subordinate clauses
- Coordination
- Prepositions
- Punctuation

I can see that they rely on this and that.
Which direction?

Some tricky cases . . .
- Complex verb groups
- Subordinate clauses
- Coordination
- Prepositions
- Punctuation

I can see that they rely on this and that.
Some tricky cases . . .

- Complex verb groups
- Subordinate clauses
- Coordination
- Prepositions
- Punctuation

I can see that they rely on this and that.
Which direction?

Some tricky cases . . .
- Complex verb groups
- Subordinate clauses
- Coordination
- Prepositions
- Punctuation

I can see that they rely on this and that.
Dependency Parsing

- **Input:** Sentence $x = w_0, w_1, \ldots, w_n$
- **Output:** Dependency graph $G = (V, A)$ for $x$ where:
  - $V = 0, 1, \ldots, n$ is the vertex set,
  - $A$ is the arc set, i.e., $(i, j, k) \in A$ represents a dependency from $w_i$ to $w_j$ with label $l_k \in L$
- **Notational Conventions**
  - $i \rightarrow j \equiv \exists k : (i, j, k) \in A$ (unlabeled dependency)
  - $i \leftrightarrow j \equiv i \rightarrow \forall j \rightarrow i$ (undirected dependency)
  - $i \rightarrow *j \equiv i = j \lor \exists i' : i \rightarrow i', i' \rightarrow *j$ (unlabeled closure)
  - $i \leftrightarrow *j \equiv i \lor \exists i' : i \leftrightarrow i', i \leftrightarrow *j$ (undirected closure)
Conditions

- Intuitions
  - Syntactic structure is complete (Connectedness)
  - Syntactic structure is hierarchical (Acyclic)
  - Every word has at most one syntactic head (Single-Head)
- Connectedness is enforced by adding special root node
Conditions

- **Connected**: \( \forall i, j \in V, i \leftrightarrow *j \)
- **Acyclic**: If \( i \to j \), then not \( j \to *i \)
- **Single-head**: If \( i \to j \), then not \( i' \to j \forall i' \neq i \)
- **Projective**: If \( i \to j \), then \( i \to *i' \) for any \( i' \) such that \( i < i' < j \) or \( j < i' < i \).
Projectivity

- Equivalent to planar embedding
- Most theoretical frameworks do not assume projectivity
- Non-projective structures needed for free word order and long-distance dependencies

Non-projective example

- The algorithm later we’ll discuss is projective
Outline

1 Motivation

2 Algorithms for Dependency Parsing

3 Evaluation and Estimation
Many algorithms exist (good overview in Kübler et al)

We will focus on a arc-factored projective model

- arc-factored: Score factorizes over edges
- projective: no crossing lines (planar embedding)

This is a common, but not universal assumption
How good is a given tree?

1. \( \text{score}(G) = \text{score}(V, A) \in \mathbb{R} \)

2. Arc-factored assumption:
   \[
   \text{score}(G) = \sum_{(w_i, r, w_j) \in A} \psi_{w_i, r, w_j}
   \]  

3. Further simplification for class:
   \[
   \text{score}(G) = \sum_{(w_i, w_j) \in A} \psi_{w_i, w_j}
   \]  

4. You can think about this probabilistically when
   \[
   \psi_{w_i, w_j} \equiv \log(p((w_i, w_j) \in A))
   \]
Dynamic Programming

- A parser should avoid re-analyzing sub-strings because the analysis of a substring is independent of the rest of the parse.
- The parser’s exploration of its search space can exploit this independence: dynamic programming (CS) / chart parsing (ling)
- Once solutions to sub-problems have been accumulated, solve the overall problem by composing them
- Sub-trees are stored in a chart, which records all substructures:
  - re-parsing: sub-trees are looked up, not reparsed
  - ambiguity: chart implicitly stores all parses
Central Idea: Spans

- Like Viterbi algorithm, we’ll solve sub problems to find the overall optimum
- Our overall goal is to find the best parse for the entire sentence
Central Idea: Spans

- Like Viterbi algorithm, we’ll solve sub problems to find the overall optimum
- Our overall goal is to find the best parse for the entire sentence
Central Idea: Spans

- Like Viterbi algorithm, we’ll solve sub problems to find the overall optimum
- Our overall goal is to find the **best parse** for the **entire sentence**
Central Idea: Spans

- Like Viterbi algorithm, we’ll solve sub problems to find the overall optimum
- Our overall goal is to find the **best parse** for the **entire sentence**
Central Idea: Spans

- Like Viterbi algorithm, we’ll solve sub problems to find the overall optimum
- Our overall goal is to find the **best parse** for the **entire sentence**
Central Idea: Spans

- Like Viterbi algorithm, we’ll solve sub problems to find the overall optimum
- Our overall goal is to find the **best parse** for the **entire sentence**
To do this, we’ll find the **best parse** for contiguous **spans** of the sentence, characterized by

- start 0...n
- stop 0...n
- direction ←, →
- completeness ○, ·

Each span gets an entry in a 4D chart (same as 2D chart for POS tagging)

Find the overall tree that gives highest score
We write the total score of these spans $C_{[s][t][\rightarrow][\cdot]}$

“Root” of this subtree is at word $s$

Can have arbitrary substructure until word $t$, but cannot take additional right children
We write the total score of these spans $C[s][t][\leftarrow][\cdot]$.

“Root” of this subtree is at word $t$.

Can have arbitrary substructure until word $s$, but cannot take additional left children.
We write the total score of these spans \( C[s][t][\rightarrow][\circ] \)

“Root” of this subtree is at word \( s \)

Can have arbitrary substructure until word \( t \), but **can** take additional right children
Right Incomplete Spans

- We write the total score of these spans $C[s][t][\rightarrow][\circ]$
- “Root” of this subtree is at word $s$
- Can have arbitrary substructure until word $t$, but can take additional right children
We write the total score of these spans $C[s][t][\leftarrow][\circ]$

“Root” of this subtree is at word $t$

Can have arbitrary substructure until word $s$, but **can** take additional left children
Can accept additional left children

- We write the total score of these spans $C[s][t][\leftarrow][\circ]$
- “Root” of this subtree is at word $t$
- Can have arbitrary substructure until word $s$, but can take additional left children
$C[0][L][\rightarrow][\cdot]$ contains the best score for the overall tree.
$C[0][L][\rightarrow][\cdot]$ contains the best score for the overall tree.
$C[0][L][\rightarrow][\cdot]$ contains the best score for the overall tree.

Where's the main verb?
$C[0][L][→][·]$ contains the best score for the overall tree.
$C[0][\downarrow[\rightarrow]][\cdot]$ contains the best score for the overall tree.
Building Incomplete Spans

Left incomplete spans are built by joining left complete to right complete

\[ C[s][t][\leftarrow][\circ] = \max_{s \leq q < t} C[s][q][\rightarrow][\cdot] + C[q + 1][t][\leftarrow][\cdot] + \lambda(w_t, w_s) \] (4)
Right incomplete spans are built by joining right complete to left complete

\[
C[s][t][\rightarrow][\circ] = \max_{s \leq q < t} C[s][q][\rightarrow][\cdot] + C[q + 1][t][\leftarrow][\cdot] + \lambda(w_s, w_t) \tag{4}
\]
Right incomplete spans are built by joining right complete to left complete

\[ C[s][t][→][\circ] = \max_{s \leq q < t} C[s][q][→][\cdot] + C[q + 1][t][←][\cdot] + \lambda(w_s, w_t) \quad (4) \]
Right incomplete spans are built by joining right complete to left complete

\[ C[s][t][\rightarrow][\circ] = \max_{s \leq q < t} C[s][q][\rightarrow][\cdot] + C[q+1][t][\leftarrow][\cdot] + \lambda(w_s, w_t) \] (4)
Completing Spans

Right complete spans are built by taking an incomplete right span and then “completing” it with a right complete span

\[ C[s][t][\rightarrow][\cdot] = \max_{s < q \leq t} C[s][q][\rightarrow][\circ] + C[q][t][\rightarrow][\cdot] \]
Completing Spans

Left complete spans are built by taking an incomplete left span and then “completing” it with a left complete span

\[ C[s][t][\leftarrow][\cdot] = \max_{s \leq q < t} C[s][q][\leftarrow][\cdot] + C[q][t][\leftarrow][\circ] \]
Example Sentence
Example Sentence
Example Sentence
Example Sentence
Example Sentence
Example Sentence
Example Sentence
Final step

Look at cell at corresponding to 0 to the length of the sentence, complete, and directed to the right. That is the best parse.
As you build the chart, you must keep track of what the best subtrees were to construct each cell; call this $b$

Then look at $b[0][L][→][.]$, and recursively build the tree

Complexity is $O(n^3)$:
- Table is size $O(n)$
- Each cell requires at most $n$ possible subtrees
Extensions to Dependency Parsing

- Horizontal and vertical Markovization (node depends on siblings and grandparents in tree—logical!)
  - “saw with telescope” more likely than “bridge with telescope” (grandparent)
  - “fast sports car” more likely than “fast slow car” (sibling)
- Graph algorithms: allow non-projectivity
- Sequential processing (transition-based models) that are more psycholinguistically plausible (and are now performing quite well)
Extensions to Dependency Parsing

- Horizontal and vertical Markovization (node depends on siblings and grandparents in tree—logical!)
  - “saw with telescope” more likely than “bridge with telescope” (grandparent)
  - “fast sports car” more likely than “fast slow car” (sibling)
- Graph algorithms: allow non-projectivity
- Sequential processing (transition-based models) that are more psycholinguistically plausible (and are now performing quite well)
Motivation

Algorithms for Dependency Parsing

Evaluation and Estimation
Where does the attachment score come from?

- Language model: vertical rather than horizontal
  - How likely is the noun “bagel” the child of the “verb” eat?
  - Back off to noun being the child of the verb “eat” . . .
  - Back off to a noun being the child of a verb
Where does the attachment score come from?

- Language model: vertical rather than horizontal
  - How likely is the noun “bagel” the child of the “verb” eat?
  - Back off to noun being the child of the verb “eat” . . .
  - Back off to a noun being the child of a verb

- Discriminative models: minimize errors
Evaluation Methodology

- How many sentences are exactly correct

- Edge accuracy

- Labeled attachment score (LAS): i.e. Tokens with correct head and label

- Unlabeled attachment score (UAS): i.e. Tokens with correct head

- Label accuracy (LA): i.e. Tokens with correct label

Performance on downstream task (e.g., information extraction)
Evaluation Methodology

- How many sentences are exactly correct
- Edge accuracy
  1. Labeled attachment score (LAS):
     i.e. Tokens with correct head and label
  2. Unlabeled attachment score (UAS):
     i.e. Tokens with correct head
  3. Label accuracy (LA):
     i.e. Tokens with correct label
- Performance on downstream task (e.g., information extraction)
In Class

- Work-through examples
- Talk through dependency homework
- Discuss HW1 results