Introduction to Data-Driven Dependency Parsing

Introductory Course, ESSLLI 2007

Ryan McDonald\textsuperscript{1} Joakim Nivre\textsuperscript{2}

\textsuperscript{1}Google Inc., New York, USA
E-mail: ryanmcd@google.com

\textsuperscript{2}Uppsala University and Växjö University, Sweden
E-mail: nivre@msi.vxu.se
Overview of the Course

- Dependency parsing (Joakim)
- Machine learning methods (Ryan)
- Transition-based models (Joakim)
- **Graph-based models** (Ryan)
- Loose ends (Joakim, Ryan):
  - Other approaches
  - Empirical results
  - Available software
Notation Reminder

- Sentence $x = w_0, w_1, \ldots, w_n$, with $w_0 = root$
- $L = \{l_1, \ldots, l_{|L|}\}$ set of permissible arc labels
- Let $G = (V, A)$ be a dependency graph for sentence $x$ 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 $l_k \in L$
- By the usual definition, $G$ is a tree
Data-Driven Parsing

- Goal: Learn a good predictor of dependency graphs
- Input: $x$
- Output: dependency graph/tree $G$
- Last lecture:
  - Parameterize parsing by transitions
  - Learn to predict transitions given the input and a history
  - Predict new graphs using deterministic parsing algorithm
- This lecture:
  - Parameterize parsing by dependency arcs
  - Learn to predict entire graphs given the input
  - Predict new graphs using spanning tree algorithms
Lecture 4: Outline

- Graph theory refresher
- Arc-factored models (a.k.a. Edge-factored models)
  - Maximum spanning tree formulation
  - Projective and non-projective inference algorithms
  - Partition function and marginal algorithms – Matrix Tree
    Theorem
- Beyond Arc-factored Models
  - Vertical and horizontal markovization
  - Approximations
Some Graph Theory Reminders

- A graph $G = (V, A)$ is a set of vertices $V$ and arcs $(i, j) \in A$, where $i, j \in V$
- Undirected graphs: $(i, j) \in A \iff (j, i) \in A$
- Directed graphs (digraphs): $(i, j) \in A \not\iff (j, i) \in A$
Multi-Digraphs

- A multi-digraph is a digraph where there can be multiple arcs between vertices
- \( G = (V, A) \)
- \((i, j, k) \in A\) represents the \( k^{th} \) arc from vertex \( i \) to vertex \( j \)
Directed Spanning Trees (a.k.a. Arborescence)

▶ A directed spanning tree of a (multi-)digraph $G = (V, A)$, is a subgraph $G' = (V', A')$ such that:
  ▶ $V' = V$
  ▶ $A' \subseteq A$, and $|A'| = |V'| - 1$
  ▶ $G'$ is a tree (acyclic)

▶ A spanning tree of the following (multi-)digraphs
Weighted Directed Spanning Trees

Assume we have a weight function for each arc in a multi-digraph $G = (V, A)$

Define $w_{ij}^k \geq 0$ to be the weight of $(i, j, k) \in A$ for a multi-digraph

Define the weight of directed spanning tree $G'$ of graph $G$ as

$$w(G') = \prod_{(i,j,k) \in G'} w_{ij}^k$$

Notation: $(i, j, k) \in G = (V, A) \iff$ the arc $(i, j, k) \in A$
Maximum Spanning Trees (MST) of (Multi-)Digraphs

Let $T(G)$ be the set of all spanning trees for graph $G$.

The MST Problem: Find the spanning tree $G'$ of the graph $G$ that has highest weight

$$G' = \arg \max_{G' \in T(G)} w(G') = \arg \max_{G' \in T(G)} \prod_{(i,j,k) \in G'} w_{ij}^k$$

Solutions ... to come.
Arc-Factored Dependency Models

- Remember: Data-driven parsing parameterizes model and then learns parameters from data

- **Arc-factored model**
  - Assumes that the score / probability / **weight** of a dependency graph factors by its arcs

\[
  w(G) = \prod_{(i,j,k) \in G} w_{ij}^k
\]

- \(w_{ij}^k\) is the weight of creating a dependency from word \(w_i\) to \(w_j\) with label \(l_k\)

- Thus there is an assumption that each dependency decision is independent
  - Strong assumption! Will address this later.
Arc-Factored Dependency Models Example

- Weight of dependency graph is $10 \times 30 \times 30 = 9000$

- In practice arc weights are much smaller
Important Concept $G_x$

- For input sentence $x = w_0, \ldots, w_n$, define $G_x = (V_x, A_x)$ as:
  - $V_x = \{0, 1, \ldots, n\}$
  - $A_x = \{(i, j, k) \mid \forall i, j \in V_x \text{ and } l_k \in L\}$
- Thus, $G_x$ is complete multi-digraph over vertex set representing words

**Theorem**

Every valid dependency graph for sentence $x$ is equivalent to a directed spanning tree for $G_x$ that originates out of vertex 0.

- Falls out of definitions of tree constrained dependency graphs and spanning trees
  - Both are spanning/connected (contain all words)
  - Both are trees
Three Important Problems

**Theorem**

Every valid dependency graph for sentence $x$ is equivalent to a directed spanning tree for $G_x$ that originates out of vertex $0$

1. **Inference** $\equiv$ finding the MST of $G_x$

   \[
   G = \arg\max_{G \in T(G_x)} w(G) = \arg\max_{G \in T(G_x)} \prod_{(i,j,k) \in G} w^k_{ij}
   \]

2. Defining $w^k_{ij}$ and its **feature space**

3. **Learning** $w^k_{ij}$
   - Can use perceptron-based learning if we solve (1)
Inference - Getting Rid of Arc Labels

\[ G = \arg \max_{G \in T(G_x)} w(G) = \arg \max_{G \in T(G_x)} \prod_{(i,j,k) \in G} w_{ij}^k \]

- Consider all the arcs between vertexes \( i \) and \( j \)
- Now, consider the arc \((i,j,k)\) such that,

\[ (i,j,k) = \arg \max_k w_{ij}^k \]

**Theorem**

The highest weighted dependency tree for sentence \( x \) must contain the arc \((i,j,k)\) – (assuming no ties)

- Easy proof: if not, sub in \((i,j,k)\) and get higher weighted tree
Inference - Getting Rid of Arc Labels

\[
G = \arg \max_{G \in T(G_x)} w(G) = \arg \max_{G \in T(G_x)} \prod_{(i,j,k) \in G} w_{ij}^k
\]

Thus, we can reduce \( G_x \) from a multi-digraph to a simple digraph

Just remove all arcs that do not satisfy

\[
(i, j, k) = \arg \max_{k} w_{ij}^k
\]

Problem is now equal to the MST problem for digraphs

We will use the **Chu-Liu-Edmonds Algorithm**

[Chu and Liu 1965, Edmonds 1967]
Chu-Liu-Edmonds Algorithm

- Finds the MST originating out of a vertex of choice
- Assumes weight of tree is sum of arc weights
- No problem, we can use logarithms

\[
G = \arg \max_{G \in T(G_x)} \prod_{(i,j,k) \in G} w_{ij}^k
\]

\[
= \arg \max_{G \in T(G_x)} \log \prod_{(i,j,k) \in G} w_{ij}^k
\]

\[
= \arg \max_{G \in T(G_x)} \sum_{(i,j,k) \in G} \log w_{ij}^k
\]

So if we let \( w_{ij}^k = \log w_{ij}^k \), then we get

\[
G = \arg \max_{G \in T(G_x)} \sum_{(i,j,k) \in G} w_{ij}^k
\]
Chu-Liu-Edmonds

$x = \text{root John saw Mary}$
Chu-Liu-Edmonds

- Find highest scoring incoming arc for each vertex

- If this is a tree, then we have found MST!!
Chu-Liu-Edmonds

- If not a tree, identify cycle and contract
- Recalculate arc weights into and out-of cycle
Chu-Liu-Edmonds

- Outgoing arc weights
  - Equal to the max of outgoing arc over all vertexes in cycle
  - e.g., John → Mary is 3 and saw → Mary is 30
Chu-Liu-Edmonds

- Incoming arc weights
  - Equal to the weight of best spanning tree that includes head of incoming arc, and all nodes in cycle
  - root → saw → John is 40 (**)
  - root → John → saw is 29
The weight of the MST of this contracted graph is equal to the weight of the MST for the original graph.

Therefore, recursively call algorithm on new graph.
Chu-Liu-Edmonds

- This is a tree and the MST for the contracted graph!!

- Go back up recursive call and reconstruct final graph
Chu-Liu-Edmonds

- This is the MST!!
Chu-Liu-Edmonds Code

**Chu-Liu-Edmonds**($G_x, w$)
1. Let $M = \{(i^*, j) : j \in V_x, i^* = \arg \max_{i'} w_{ij}\}$
2. Let $G_M = (V_x, M)$
3. If $G_M$ has no cycles, then it is an MST: return $G_M$
4. Otherwise, find a cycle $C$ in $G_M$
5. Let $< G_C, c, ma > = \text{contract}(G, C, w)$
7. Find vertex $i \in C$ such that $(i', c) \in G$ and $ma(i', c) = i$
8. Find arc $(i'', i) \in C$
9. Find all arc $(c, i''') \in G$
10. $G = G \cup \{(ma(c, i'''), i''')\}_{\forall (c, i''') \in G \cup C \cup \{(i', i)\} - \{(i'', i)\}}$
11. Remove all vertices and arcs in $G$ containing $c$
12. return $G$

▶ Reminder: $w_{ij} = \arg \max_k w_{ij}^k$
Chu-Liu-Edmonds Code (II)

\texttt{contract} (G = (V, A), C, w)
1. Let \( G_C \) be the subgraph of \( G \) excluding nodes in \( C \)
2. Add a node \( c \) to \( G_C \) representing cycle \( C \)
3. For \( i \in V - C : \exists i' \in C(i', i) \in A \)
   Add arc \((c, i)\) to \( G_C \) with
   \[ ma(c, i) = \arg \max_{i' \in C} \text{score}(i', i) \]
   \[ i' = ma(c, i) \]
   \[ \text{score}(c, i) = \text{score}(i', i) \]
4. For \( i \in V - C : \exists i' \in C(i, i') \in A \)
   Add edge \((i, c)\) to \( G_C \) with
   \[ ma(i, c) = \arg \max_{i' \in C} [\text{score}(i, i') - \text{score}(a(i'), i')] \]
   \[ i' = ma(i, c) \]
   \[ \text{score}(i, c) = [\text{score}(i, i') - \text{score}(a(i'), i') + \text{score}(C)] \]
   where \( a(v) \) is the predecessor of \( v \) in \( C \)
   and \( \text{score}(C) = \sum_{v \in C} \text{score}(a(v), v) \)
5. return \(< G_C, c, ma >\)
Chu-Liu-Edmonds

- Naive implementation $O(n^3 + |L|n^2)$
  - Converting $G_x$ to a digraph – $O(|L|n^2)$
  - Finding best arc – $O(n^2)$
  - Contracting cycles – $O(n^2)$
  - At most $n$ recursive calls

- Better algorithms run in $O(|L|n^2)$ [Tarjan 1977]

- Chu-Liu-Edmonds searches all dependency graphs
  - Both projective and non-projective
  - Thus, it is an exact non-projective search algorithm!!!

- What about the projective case?
Arc-factored Projective Parsing

- Projective dependency structures are nested
- Can use CFG like parsing algorithms – chart parsing
- Each chart item (triangle) represents the weight of the best tree rooted at word $h$ spanning all the words from $i$ to $j$
  - Analog in CFG parsing: items represent best tree rooted at non-terminal $NT$ spanning words $i$ to $j$
- **Goal**: Find chart item rooted at 0 spanning 0 to $n$

```
  h
 /\  \
 i  j
```

**Base case**
Length 1, $h = i = j$, has weight 1
**Arc-factored Projective Parsing**

- All projective graphs can be written as the combination of two smaller adjacent graphs.

- Inductive hypothesis – algorithm has calculated score of smaller items correctly (just like CKY).
Arc-factored Projective Parsing

- Chart item filled in a bottom-up manner
  - First do all strings of length 1, then 2, etc. just like CKY

Weight of new item: \( \max_{l,j,k} w(A) \times w(B) \times w_{hh'}^k \)

Algorithm runs in \( O(|L|n^5) \)

Use back-pointers to extract best parse (like CKY)
Arc-factored Projective Parsing

- $O(|L|n^5)$ is not that good
- [Eisner 1996] showed how this can be reduced to $O(|L|n^3)$
  - Key: split items so that sub-roots are always on periphery
Inference in Arc-Factored Models

- Non-projective case
  - $O(|L|n^2)$ with the Chu-Liu-Edmonds MST algorithm
- Projective case
  - $O(|L|n^3)$ with the Eisner algorithm
- But we still haven’t defined the form of $w^k_{ij}$
- Or how to learn these parameters
Arc weights as linear classifiers

\[ w_{ij}^k = e^{w \cdot f(i,j,k)} \]

- Arc weights are a linear combination of features of the arc, \( f \), and a corresponding weight vector \( w \)
- Raised to an exponent (simplifies some math ...)
- What arc features?
- [McDonald et al. 2005] discuss a number of binary features
Arc Features: $f(i, j, k)$

Features from [McDonald et al. 2005]:
- Identities of the words $w_i$ and $w_j$ and the label $l_k$

  \[
  \text{head} = \text{saw} \quad \& \quad \text{dependent} = \text{with}
  \]
Arc Features: $f(i, j, k)$

Features from [McDonald et al. 2005]:
- Part-of-speech tags of the words $w_i$ and $w_j$ and the label $l_k$

head-pos=Verb & dependent-pos=Preposition

John saw Mary McGuire yesterday with his telescope

N V N N R P PR N
Arc Features: \( f(i, j, k) \)

Features from [McDonald et al. 2005]:
- Part-of-speech of words surrounding and between \( w_i \) and \( w_j \)

\[
\begin{align*}
\text{inbetween-pos} &= \text{Noun} \\
\text{inbetween-pos} &= \text{Adverb} \\
\text{dependent-pos-right} &= \text{Pronoun} \\
\text{head-pos-left} &= \text{Noun} \\
\ldots
\end{align*}
\]
Arc Features: \( f(i, j, k) \)

Features from [McDonald et al. 2005]:
- Number of words between \( w_i \) and \( w_j \), and their orientation

arc-distance=3
arc-direction=right
Arc Features: $f(i, j, k)$

Label features

arc-label=PP
Arc Features: \( f(i, j, k) \)

- Combos of the above
  - head-pos=Verb & dependent-pos=Preposition & arc-label=PP
  - head-pos=Verb & dependent=with & arc-distance=3
    
    No limit: any feature over arc \((i, j, k)\) or input \(x\)
Learning the parameters

- We can then re-write the inference problem

\[
G = \arg \max_{G \in T(G_x)} \prod_{(i,j,k) \in G} w_{ij}^k = \arg \max_{G \in T(G_x)} \prod_{(i,j,k) \in G} e^{w \cdot f(i,j,k)}
\]

\[
= \arg \max_{G \in T(G_x)} \log \prod_{(i,j,k) \in G} e^{w \cdot f(i,j,k)}
\]

\[
= \arg \max_{G \in T(G_x)} \sum_{(i,j,k) \in G} w \cdot f(i,j,k)
\]

\[
= \arg \max_{G \in T(G_x)} w \cdot \sum_{(i,j,k) \in G} f(i,j,k) = \arg \max_{G \in T(G_x)} w \cdot f(G)
\]

- Which we can plug into online learning algorithms
Inference-based Learning

e.g., The Perceptron

Training data: $T = \{(x_t, G_t)\}_{t=1}^{\vert T \vert}$

1. $w^{(0)} = 0; \ i = 0$
2. for $n: 1..N$
3. for $t: 1..T$
4. Let $G' = \arg \max_{G'} w^{(i)} \cdot f(G')$ (**)
5. if $G' \neq G_t$
6. $w^{(i+1)} = w^{(i)} + f(G_t) - f(G')$
7. $i = i + 1$
8. return $w^i$
Other Important Problems

- $K$-best inference – $O(K \times |L| n^2)$ [Camerini et al. 1980]
- Partition function

$$Z_x = \sum_{G \in T(G_x)} w(G)$$

- Arc expectations

$$\langle i, j, k \rangle_x = \sum_{G \in T(G_x)} w(G) \times 1[(i, j, l) \in G]$$

- Important for some learning & inference frameworks
- Important for some applications
Partition Function: \( Z_x = \sum_{G \in T(G_x)} w(G) \)

- Laplacian Matrix \( Q \) for graph \( G_x = (V_x, A_x) \)
  \[
  Q_{jj} = \sum_{i \neq j, (i,j,k) \in A_x} w_{ij}^k \quad \text{and} \quad Q_{ij} = \sum_{i \neq j, (i,j,k) \in A_x} -w_{ij}^k
  \]

- Cofactor \( Q^i \) is the matrix \( Q \) with the \( i^{th} \) row and column removed

**The Matrix Tree Theorem** [Tutte 1984]

The determinant of the cofactor \( Q^0 \) is equal to \( Z_x \)

- Thus \( Z_x = |Q^0| \) – determinants can be calculated in \( O(n^3) \)
- Constructing \( Q \) takes \( O(|L|n^2) \)
- Therefore the whole process takes \( O(n^3 + |L|n^2) \)
Arc Expectations

\[ \langle i, j, k \rangle_x = \sum_{G \in T(G_x)} w(G) \times 1[(i, j, k) \in A] \]

- Can easily be calculated, first reset some weights

\[ w_{i'i'j}^k = 0 \ \forall i' \neq i \text{ and } k' \neq k \]

- Now, \( \langle i, j, k \rangle_x = Z_x \)

- Why? All competing arc weights to zero, therefore every non-zero weighted graph must contain \((i, j, k)\)

- Naively takes \( O(n^5 + |L|n^2) \) to compute all expectations

- But can be calculated in \( O(n^3 + |L|n^2) \) (see [McDonald and Satta 2007, Smith and Smith 2007, Koo et al. 2007])
$Z_x$ for the Projective Case

- Just augment chart-parsing algorithm

Weight of new item: $\sum_{i,j,k} w(A) \times w(B) \times w_{hh'}^k$

Weight of item rooted at 0 spanning 0 to $n$ is equal to $Z_x$

Also works for Eisner’s algorithm – runtime $O(n^3 + |L|n^2)$
\langle i, j, k \rangle_x \text{ for the Projective Case}

- Can be calculated through $Z_x$, just like the non-projective case
- Can also be calculated using the inside-outside algorithm
- See [Paskin 2001] for more details
Why calculate $Z_x$ and $\langle i, j, k \rangle_x$?

- Useful for many learning and inference problems
  - Min risk-decoding ($\langle i, j, k \rangle_x$)
  - Log-linear parsing models ($Z_x$ and $\langle i, j, k \rangle_x$)
  - Syntactic language modeling ($Z_x$)
  - Unsupervised dependency parsing ($Z_x$ and $\langle i, j, k \rangle_x$)
  - ...

- See [McDonald and Satta 2007] for more
Beyond Arc-factored Models

- Arc-factored models make strong independence assumptions
- Can we do better?
- Rest of lecture
  - NP-hardness of Markovization for non-projective parsing
  - But ... projective case has polynomial solutions!!
  - Approximate non-projective algorithms
Vertical and Horizontal Markovization

- Dependency graphs weight factors over neighbouring arcs
- Vertical versus Horizontal neighbourhoods
Beyond Arc-factored Models

\( N^{th} \) Order Horizontal Markov Factorization

- Assume the **unlabeled parsing** case (adding labels is easy)
- Weights factor over neighbourhoods of size \( N \)

\[ x_{i_0} \]
\[ x_{i_1} \quad \ldots \quad x_{i_j} \quad x_{i_{j+1}} \quad \ldots \quad x_{i_m} \]

- Normal (arc-factored = first-order)

\[ \prod_{k=1}^{m} w_{i_0 i_k} \]

- Second-order – weights over pairs of adjacent (same side) arcs

\[ \prod_{k=1}^{j-1} w_{i_0 i_k i_{k+1}} \times w_{i_0 \cdot i_j} \times w_{i_0 \cdot i_{j+1}} \times \prod_{k=j+1}^{m-1} w_{i_0 i_k i_{k+1}} \]
Non-projective Horizontal Markovization

- Non-projective second-order parsing is NP-hard
  - Thus any order non-projective parsing is NP-hard

- **3-dimensional matching (3DM):** Disjoint sets $X$, $Y$, $Z$ each with $m$ elements. A set $T \subseteq X \times Y \times Z$. Question – is there a subset $S \subseteq T$ such that $|S| = m$ and each $v \in X \cup Y \cup Z$ occurs in exactly one element of $S$

- **Reduction:** Define $G_x = (V_x, A_x)$ as a dense graph, where
  - $V_x = \{v \mid \forall, v \in X \cup Y \cup Z\} \cup \{0\}$
  - $w_{0x_ix_j} = 1, \forall x_i, x_j \in X$
  - $w_{x.y} = 1, \forall x \in X, y \in Y$
  - $w_{x_i,y_j,z_k} = 1, \forall (x, y, z) \in T$
  - All other weights are 0
Non-projective Horizontal Markovization

- Non-projective second-order parsing is NP-hard
- Generate sentence from all $x \in X$, $y \in Y$ and $z \in Z$

weight = 1

root $x_0 \ldots x_i \ldots x_j \ldots x_m$ $y_0 \ldots y_i \ldots y_j \ldots y_m$ $z_0 \ldots z_i \ldots z_j \ldots z_m$
Non-projective Horizontal Markovization

- Non-projective second-order parsing is NP-hard

weight = 1

root  x0 ... xi ... xj ... xm  y0 ... yi ... yj ... ym  z0 ... zi ... zj ... zm
Non-projective Horizontal Markovization

- Non-projective second-order parsing is NP-hard

![Diagram showing a root node with edges to nodes labeled x0, xi, xj, xm, y0, yi, yj, ym, z0, zi, zj, zm. The weight of the arc is 1.]

weight = 1
Non-projective Horizontal Markovization

- Non-projective second-order parsing is NP-hard

\[
\text{root } x_0 \ldots x_i \ldots x_j \ldots x_m \ y_0 \ldots y_i \ldots y_j \ldots y_m \ z_0 \ldots z_i \ldots z_j \ldots z_m
\]

- All other arc weights are set to 0
Non-projective Horizontal Markovization

**Theorem:** There is a 3DM iff there is a dependency graph of weight 1

**Proof:**
- All non-zero weight dependency graphs correspond to a 3DM
- Every 3DM corresponds to a non-zero weight dependency graph
- Therefore, there is a non-zero weight dependency graph iff then there is a 3DM
- See [McDonald and Pereira 2006] for more
Projective Horizontal Markovization

- Can simply augment chart parsing algorithm
- Same for the Eisner algorithm – runtime still $O(|L|n^3)$
Approx Non-proj Horizontal Markovization

- Two properties:
  - Projective parsing is polynomial w/ horizontal Markovization
  - Most non-projective graphs are still primarily projective

- Use these facts to get an approximate non-projective algorithm
  - Find a high scoring projective parse
  - Iteratively modify to create a higher scoring non-projective parse
  - Post-process non-projectivity, which is related to pseudo-projective parsing
Approx Non-proj Horizontal Markovization

**Algorithm**

1. Let $G$ be the highest weighted projective graph
2. Find the arc $(i, j, k) \in G$, a node $i'$ and label $l_{k'}$ such that
   - $G' = G \cup \{(i', j, k')\} - \{(i, j, k)\}$ is a valid graph (tree)
   - $G'$ has highest weight of all possibly changes
3. if $w(G') > w(G)$ then $G = G'$ and return to step 2
4. Otherwise return $G$

**Intuition**: Start with a high weighted graph and make local changes that increase the graphs weight until convergence

**Works well in practice** [McDonald and Pereira 2006]
Vertical Markovization

- Also NP-hard for non-projective case [McDonald and Satta 2007]
  - Reduction again from 3DM
  - A little more complicated – relies on arc labels
- Projective case is again polynomial
  - Same method of augmenting the chart-parsing algorithm
Beyond Arc-Factorization

- For the non-projective case, increasing scope of weights (and as a result features) makes parsing intractable
- However, chart parsing nature of projective algorithms allows for simple augmentations
- Can approximate the non-projective case using the exact projective algorithms plus a post-process optimization
- Further reading:
  [McDonald and Pereira 2006, McDonald and Satta 2007]
Summary – Graph-based Methods

👊 Arc-factored models
  👍 Maximum spanning tree formulation
  👍 Projective and non-projective inference algorithms
  👍 Partition function and arc expectation algorithms – Matrix Tree Theorem

👊 Beyond Arc-factored Models
  👍 Vertical and horizontal markovization
  👍 Approximations
References and Further Reading


Online large-margin training of dependency parsers. In Proc. ACL.

Cubic-time parsing and learning algorithms for grammatical bigram models. Technical Report UCB/CSD-01-1148, Computer Science Division, University of California Berkeley.

▶ D.A. Smith and N.A. Smith. 2007.
Probabilistic models of nonprojective dependency trees. In Proc. EMNLP.


Graph Theory. Cambridge University Press.