Introduction to Natural Language Syntax and Parsing
Lecture 3: Graph-Based Dependency Parsing

Stephen Clark

October 13, 2015

Untyped Dependency Trees Much of the literature on dependency parsing is concerned with *untyped* dependency trees, where the edges between words are not labelled with grammatical relations. We'll also consider the untyped case, although extending the various parsing algorithms to deal with typed edges is straightforward.

The example on the slide shows a projective dependency tree, with an alternative definition of projectivity. The definition given so far is that a tree is projective iff the tree can be drawn in two dimensions without any edges crossing. An equivalent definition is that a tree is projective iff an edge from word w to word u implies that w is an ancestor of all words between w and u. For example, consider the edge from *hit* to *with*: all the words in between can also be reached from *hit* (i.e. are ancestors of *hit*). Now imagine that there is an edge from *hit* to the second *the*, i.e. a crossing edge in the example, replacing the edge between *bat* and *the*. This ruins the projectivity, since there is an edge from *with* to *bat*, but the word *the* in between *with* and *bat* is no longer an ancestor of *with*.

Edge-Based Linear Model As a reminder, we're considering first-order edge-based models where the score for a tree is the sum of individual scores for each edge; and the score for an edge is a linear sum defined as a dot product between a weight vector and feature vector.

Dependency Parsing Formally This slide provides some notation for the edge-based linear model.

Maximum Spanning Trees The directed graph G_x, for sentence $\mathbf{x}$, is a set of vertices (or nodes) V_x and a set of edges E_x. V_x is the set of words in $\mathbf{x}$ plus an additional dummy root note x_0. E_x is the set of all possible directed edges between words in $\mathbf{x}$, with the following exceptions: there are no reflexive edges (i.e. an edge from a word to itself), and x_0 cannot be the child of an edge.

The reason for considering G_x is that finding the highest-scoring dependency tree for $\mathbf{x}$ is equivalent to a well-known problem in graph theory, namely finding the *maximum spanning tree* (MST) in G_x. Finding the MST is also known as the *maximum arborescence problem*. Restricting the tree to be projective results in finding the MST which is also projective.

Decoding: finding the MST There is a classic algorithm from the 60s — the Chu-Liu-Edmonds algorithm — for finding the MST for non-projective trees, with an $O(n^2)$ implementation. The projective case is computationally harder, because now we have to find trees that satisfy a particular set of constraints (corresponding to the projectivity). We'll consider a straightforward adaption of the chart-based CKY algorithm, which runs in cubic time for CFGs, but in $O(n^5)$ time for dependency grammars. Eisner [1] introduced a variant of the chart-based algorithm which runs in cubic time for dependency grammars, and this is the one that is typically implemented in practice, for example in McDonald's MST parser.

CKY-style Dependency Parsing The CKY algorithm operates bottom-up, using CFG rules of the form $A \rightarrow B\ C$, where A, B and C are non-terminals from the CFG. The complexity of the algorithm is $O(G^2 n^3)$, where G is a grammar constant related to the number of non-terminals, and n is the length of the sentence. An informal analysis is as follows: there are $O(n^2)$ cells in the chart; for each cell we have to consider a number of split points, for which there are $O(n)$; and for each split point we have to consider $O(G^2)$ combinations of non-terminals (B and C on the RHS of the rule above).

A useful perspective on the dependency parsing problem is to consider each edge in a dependency tree as a CFG rule. Consider the edge (*hits* $\rightarrow$ *ball*). We can consider this edge as having arisen from the application of the CFG rule (*hits* $\rightarrow$ *hits ball*). So now the number of combinations of non-terminals — $O(G^2)$ above — is no longer a constant but $O(n^2)$, resulting in an overall complexity of $O(n^5)$.

Why CKY is $O(n^5)$ and not $O(n^3)$ The example on the slide is designed to show that all possible pairs of heads have to be considered when deciding which edges to add to the chart (giving the additional $O(n^2)$ complexity). Consider the phrase *visiting relatives*. If the sentence is ... *advocate visiting relatives*, then the dependency link is between *advocate* and *visiting* (since *visiting* is a verb and is the head of *visiting relatives* in this case). But if the sentence is ... *hug visiting relatives*, then the dependency link is between *hug* and *relatives* (since *visiting* is an adjective and *relatives* is the head of *visiting relatives* in this case).

Dependency Parsing Algorithms The slide summarises the various algorithms available for dependency parsing. We'll be focusing on graph-based algorithms, but there is an alternative, namely shift-reduce parsing. The linear-time complexity of shift-reduce algorithms make them an attractive alternative to graph-based chart parsing.

Shift-Reduce Dependency Parsing The example on the slides demonstrates one method of how to implement a shift-reduce parser, with a set of four possible transition actions: { `shift`, `reduce`, `arcLeft`, `arcRight` }. The key data structures are the stack and the queue. The queue contains a list of words yet to be processed, and the stack contains partial trees as the complete tree is being built.

Greedy Local Search Given a sentence, there are many possible sequences of transitions leading to a dependency tree (each possible tree has a separate transition sequence). One way to handle the ambiguity is to use a statistical classifier to make a single decision at each point in the parsing process, and stick with that decision. This is a *greedy* algorithm which is linear-time in the length of the sentence and potentially results in a very fast parser, substantially faster than the graph-based chart parser.

Beam Search The downside of the greedy approach is that, if the classifier makes a mistake, there is no way for the parser to recover later in the parsing process. One way to mitigate this problem is to use beam search instead, where K possible decisions — the K with the highest scores according to the classifier — are retained at each parsing step. Using beam search in this way typically results in a significant improvement in accuracy, with beam sizes of around 32 leading to a good trade-off between improved accuracy and loss in speed.

Shift-reduce parsing with beam search is still linear in the length of the sentence, but now has a constant associated with the size of the beam. So using a beam size of 64, say, would result in a significantly slower parser than the greedy parser.

Readings for Today's Lecture

- Spanning Tree Methods for Discriminative Training of Dependency Parsers. Ryan McDonald, Koby Crammer and Fernando Pereira. UPenn CIS Technical Report: MS-CIS-05-11.
- Characterizing the Errors of Data-Driven Dependency Parsing Models. R. McDonald and J. Nivre. Empirical Methods in Natural Language Processing and Natural Language Learning Conference (EMNLP-CoNLL), 2007.

References

[1] Jason Eisner. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th COLING Conference*, pages 340–345, Copenhagen, Denmark, 1996.