Natural Language Processing

DSECL ZG565

Session 7: Dependency Parsing

Dr. Ashish Kulkarni

BITS Pilani

May 15, 2021

Session Content

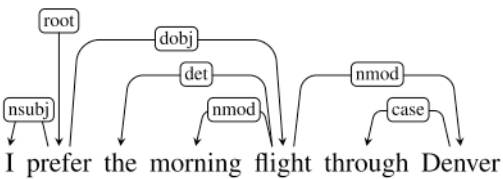
1. Introduction
2. Dependency Relations
3. Dependency Formalism
4. Dependency Treebanks
5. Transition-Based Dependency Parsing
6. Graph-Based Dependency Parsing

Dependency Grammars

So far we have studied context-free grammars leading to generation of constituent-based representation of sentences

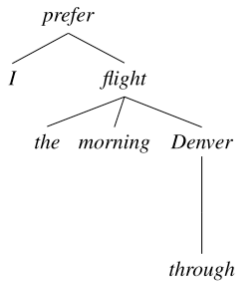
There is another family of grammars called dependency grammars where the syntactic structure of a sentence is described solely in terms of words (or lemmas) in a sentence and an associated set of directed binary grammatical relations between the words.

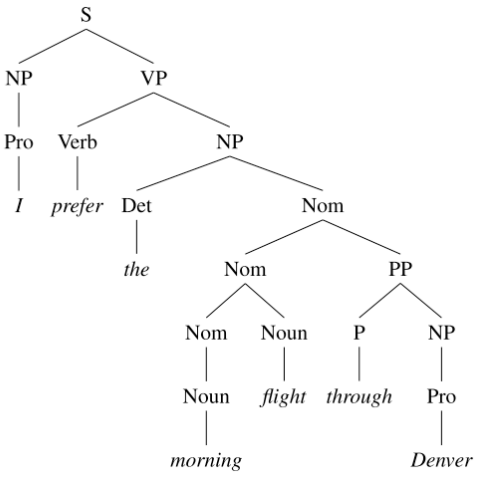
The resulting structure is called a typed dependency structure



Dependency Parse Tree

The parse tree resulting from dependency parsing is called a dependency parse tree





Why Dependency Parsing?

The directed relations directly encode important information that is often buried in the more complex phrase structure (constituency) parses.

- arguments to the verb prefer are directly linked to it in the dependency structure;
- *morning* and *Denver*, modifiers of flight, are directly linked to it.

The head-dependent relations provide an approximation to the semantic relationship between predicates and their arguments that makes them directly useful for many applications such as co-reference resolution, question answering and information extraction.

Why Dependency Parsing?

Ability to deal with languages that are morphologically rich and have a relatively free word order

A morphologically rich language has a wider variety of morphemes for different functions. *E.g.* Hindi has multiple case markers - ergative ने , accusative को , dative को, instrumental से *etc.*

Thus, in राम ने मोहन को खिलौना दिया and मोहन को राम ने खिलौना दिया the word order doesn't change the meaning.

While a phrase-structure grammar would need multiple rules to capture different word positions, a **dependency grammar abstracts away from word-order information**.

Dependency Relations

The binary relations in dependency structures intuitively based on the notion of grammatical relation.

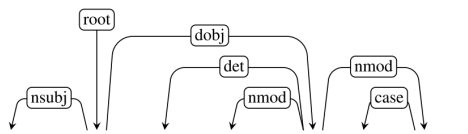
The arguments to these relations consist of head and dependent

The dependency relations are classified based on the role that the dependent plays with respect to its head.

Universal Dependencies

Provides an inventory of dependency relations, broadly broken into two sets:

- **clausal relations** describe syntactic roles with respect to a predicate;
- **modifier relations** categorize the ways that words can modify their heads.



The clausal relations NSUBJ and DOBJ identify the subject and direct object of the predicate *cancel*, while the NMOD, DET, and CASE relations denote modifiers of the nouns *flights* and *Houston*.

Clausal Argument Relations	**Description**
NSUBJ	Nominal subject
DOBJ	Direct object
IOBJ	Indirect object
CCOMP	Clausal complement
XCOMP	Open clausal complement
Nominal Modifier Relations	**Description**
NMOD	Nominal modifier
AMOD	Adjectival modifier
NUMMOD	Numeric modifier
APPOS	Appositional modifier
DET	Determiner
CASE	Prepositions, postpositions and other case markers

Table: Select dependency relations from the Universal Dependency set

Formal Definition

A dependency structure is a directed graph $G = (V, A)$ of vertices V and arcs (an ordered pair of vertices), A.

- Typically, V corresponds to the set of words, punctuation, stems and affixes;
- A captures the dependency relations.

Additional constraints might apply for computational purpose:

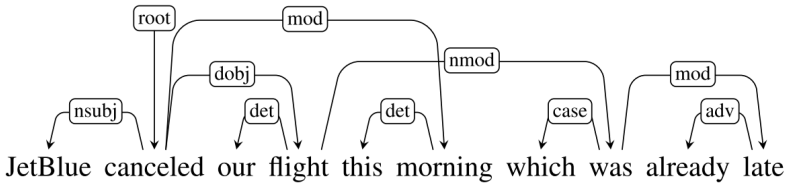
1. single designated root node that has no incoming arcs
2. except the root node, each vertex has exactly one incoming arc
3. there is a unique path from the root node to each vertex in V
4. they are acyclic

Projectivity

Another computationally motivated constraint that is typically imposed is that of projectivity.

An arc from a head to a dependent is said to be projective if there is a path from the head to every word that lies between the head and the dependent in the sentence.

A dependency tree is projective if all its arcs are projective.



The above structure is not projective. Why?

Sentences with non projective structures require more flexible parsing approaches.

Dependency Treebanks

Treebanks play a critical role in the development and evaluation of dependency parsers. There are different approaches to generate them:

Have human annotators directly generate dependency structures for a given corpus

Use automatic parsers to provide an initial parse followed by manual correction

Translate existing constituent-based treebanks into dependency trees using head rules.

- limited by the information present in the original constituent trees;
- fail to integrate morphological information with the phrase structure trees;
- cannot represent non-projective structures.

Transition-Based Dependency Parsing - The Problem

Key element is the notion of a configuration comprising a stack, an input buffer of words/tokens, and a set of dependency relations.

Start state: An initial configuration in which the stack contains the ROOT node, the input buffer has the words or lemmatized tokens in the sentence, and the set of dependency relations is empty.

Goal state: The stack and the word list is empty and the set of dependency relations represent the dependency parse.

The parsing problem can then be posed as a **search through the space of possible configurations** to move from the start state to the goal state.

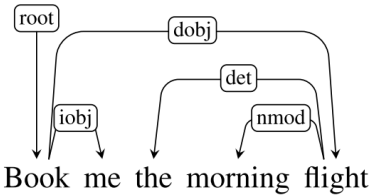
Transition-Based Dependency Parsing - The Approach

In transition-based parsing, we progressively produce new configurations using three transition operators that operate on the top two elements of the stack:

- LEFTARC: Assert a head-dependent relation between the word at the top of the stack and the word directly beneath it; remove the lower word from the stack; since the ROOT node cannot have any incoming arcs, LEFTARC operator cannot be applied when ROOT is the second element of the stack
- RIGHTARC: Assert a head-dependent relation between the second word on the stack and the word at the top; remove the word at the top of the stack
- SHIFT: Remove the word from the front of the input buffer and push it onto the stack.

At each step, the parser consults an oracle that provides the correct transition operator to use given the current configuration.

Illustration



Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book→me)
3	[root, book]	[the, morning, flight]	SHIFT	
4	[root, book, the]	[morning, flight]	SHIFT	
5	[root, book, the, morning]	[flight]	SHIFT	
6	[root, book, the, morning, flight]	[]	LEFTARC	(morning←flight)
7	[root, book, the, flight]	[]	LEFTARC	(the←flight)
8	[root, book, flight]	[]	RIGHTARC	(book→flight)
9	[root, book]	[]	RIGHTARC	(root→book)
10	[root]	[]	Done	

Comments About Transition-Based Dependency Parsing

- complexity is linear in the length of the sentence;
- a single parse is returned in the end;
- there may be more than one paths that lead to the same result;
- there may be other transition sequences that lead to different equally valid parses;
- a greedy algorithm - the oracle provides a single choice at each step and the parser proceeds with that choice, no other options are explored, no backtracking is employed;
- we are assuming that the oracle always provides the correct operator at each point in the parse; incorrect choices will lead to incorrect parses;
- to produce labeled trees, we can parameterize the LEFTARC and RIGHTARC operators with dependency labels, as in LEFTARC(NSUBJ) or RIGHTARC(DOBJ).

Creating an Oracle

Typically, supervised machine learning models play the role of an oracle

Training data: derived out of dependency treebanks by simulating the operation of a parser in the context of a reference dependency tree.

Given a reference parse and a configuration, the training oracle proceeds as follows:

- Choose LEFTARC if it produces a correct head-dependent relation given the reference parse and the current configuration;
- Otherwise, choose RIGHTARC if (1) it produces a correct head-dependent relation given the reference parse and (2) all of the dependents of the word at the top of the stack have already been assigned;
- Otherwise, choose SHIFT.

Oracle training

Features:

Source	Feature templates		
One word	$s_1.w$	$s_1.t$	$s_1.wt$
	$s_2.w$	$s_2.t$	$s_2.wt$
	$b_1.w$	$b_1.w$	$b_0.wt$
Two word	$s_1.w \circ s_2.w$	$s_1.t \circ s_2.t$	$s_1.t \circ b_1.w$
	$s_1.t \circ s_2.wt$	$s_1.w \circ s_2.w \circ s_2.t$	$s_1.w \circ s_1.t \circ s_2.t$
	$s_1.w \circ s_1.t \circ s_2.t$	$s_1.w \circ s_1.t$	

Learning:

- typically, multinomial logistic regression and support vector machines;
- neural networks and deep learning approaches have also been applied successfully to transition-based parsing.

Graph-Based Dependency Parsing

Graph-based approaches to dependency parsing search through the space of possible trees for a given sentence for a tree (or trees) that maximize some score.

Formally, given a sentence S, we are looking for the best dependency parse tree in $\mathcal{G}_S$, the space of all possible trees for that sentence, that maximises some score:

$$\hat{T}(S) = \underset{t \in \mathcal{G}_S}{\arg\max}\, score(t, S)$$

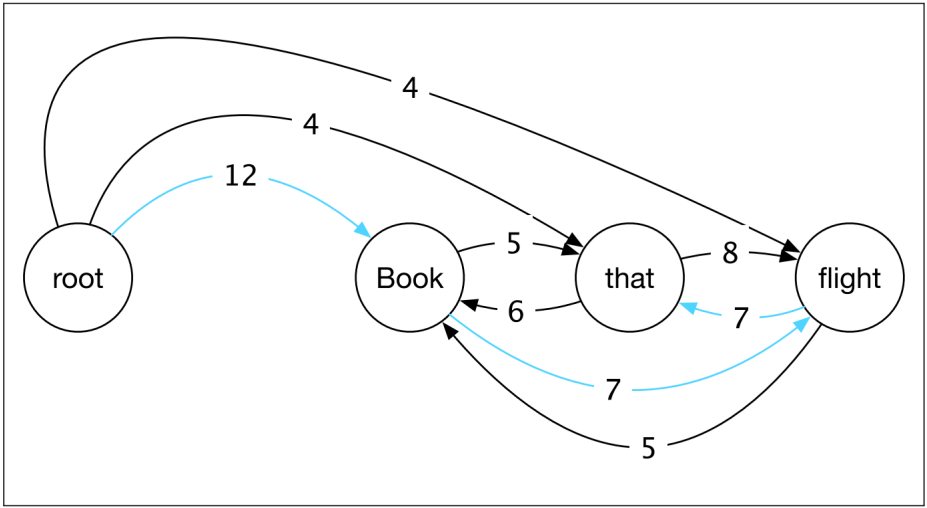
The score of a tree is computed based on an edge factored approach where the score for a tree is based on the scores of the edges that comprise the tree:

$$score(t, S) = \sum_{e \in t} score(e)$$

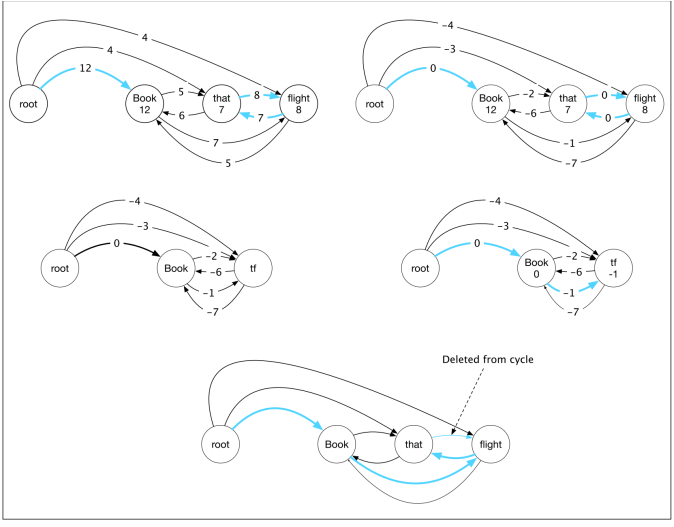
Why Graph-Based Dependency Parsing?

- Capable of producing non-projective trees;
- Transition-based methods have high accuracy on shorter dependency relations but accuracy declines significantly as the distance between the head and dependent increases. Graph-based methods avoid this difficulty by scoring entire trees, rather than relying on greedy local decisions.

Greedy Algorithm for Parsing



Parsing with Correction



Features and Training

Given a sentence S and a candidate tree T, edge-factored parsing models reduce the score for the tree to a sum of the scores of the edges that comprise the tree.

$$score(S, T) = \sum_{e \in T} score(S, e)$$

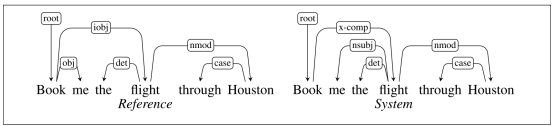
$$score(S, e) = \sum_{i=1}^{N} w_i f_i(S, e)$$

where, f_i are the features and w_i are the corresponding feature weights

Evaluation

Exact Match (EM):

- number of sentences are parsed correctly
- pessimistic metric with most sentences marked wrong



Labeled Attachment Score (LAS): percentage of words in an input that are assigned the correct head with the correct dependency relation. LAS: $4/6$ in the above example.

Unlabeled Attachment Score (UAS): Similar to LAS but only looks at the correctness of the assigned head, ignoring the dependency relation. UAS: $5/6$ in the above example.

Precision and **Recall** per dependency relation