2
Dependency Parsing

Dependency Parsing presents a number of advantages when compared to constituent syntactic parsing. According to Covington [13], dependency parsing has three main advantages. First, dependency links are close to semantic relationships needed for the next step of interpretation. Second, the dependency tree contains one node per word, instead of mid-level nodes as in constituent trees, making the task of parsing more straightforward, even lending itself to pure *corpus* based approaches. At last, dependency parsing lends itself to word-at-a-time operation, i.e., parsing can be done by accepting and attaching words, not needing complete sentences for it.

Dependency Parsing presents other advantages, as transparency, since the predicate-argument structure is directly encoded and even fragments of a parsing output can be directly interpreted (in the case of *labeled* dependency graphs). Also, when assuming non-projective graphs, dependency structure is independent of word order and more suitable for free word order languages.

Finally, Dependency Parsing proved its usefulness in many applications such as Question Answering [14, 15, 16, 17], Machine Translation [18, 19, 20], Information Extraction [21, 22, 23] and Natural Language Generation [24, 25]. Additionally, Semantic Role Labeling greatly benefits from dependency parsing as shown by Hacioglu [10].

In this chapter, we thoroughly present dependency parsing. First, the characteristics that define its grammar and foundations are described. Then, we describe data-driven approaches to solve dependency parsing, analyze their advantages and limitations and the current state-of-the-art.

2.1
Basic Concepts

The roots of dependency grammar can be traced back to medieval theories of grammar [26] or even to Panini's grammar of Sanskrit [27], several centuries before Christ. Although largely developed in Europe by traditional grammarians

as a form for syntactic representation [28], it is clear that the modern tradition of dependency grammar starts with the work of Lucien Tesnière, a french linguist of the beginning of the 20^{th} century.

Tesnière presents his idea in the following way in his work [29]:

> The sentence is an *organized whole*, the constituent elements of which are *words*. Every word that belongs to a sentence ceases 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 word. Each connection in principle unites a *superior* term and an *inferior* term. The superior term receives the name *governor*. The inferior term receives the name *subordinate*. Thus, in the sentence *Alfred parle* (...), *parle* is the governor and *Alfred* the subordinate.[1]

As stated by Tesniére, the core assumption shared by all theories and formalisms of dependency grammar is that a syntactic structure consists of *lexical* elements linked by binary *asymmetrical* relations called *dependencies* [30].

Dependency grammar, thus, clearly deviates from the traditional phrase-structure grammars by its lack of phrasal nodes. This can be noticed in figure 2.1 [30] that shows an example of a dependency structure in a English sentence, whereas figure 2.2 shows the same sentence with its constituent structure. In this case, the arcs go from each *head* to their *dependents* and are labeled with their dependency relation type [2].

One of the central aspects of dependency grammars are the criteria for establishing dependency relations and for distinguishing the head from the dependent in these relations. These criteria have been greatly discussed in the dependency grammar tradition and below are some of them that have been proposed for identifying a syntactic relation between a head *H* and a dependent *D* in a construction *C*.

1. *H* determines the syntactic category of *C* and can often replace *C*.

[1] In the original: *La phrase est un ensemble organisé dont les éléments constituants sont les mots. Tout mot qui fait partie d'une phrase cesse par luimême d'être isolé comme dans le dictionnaire. Entre lui et ses voisins, l'esprit aperçoit des connexions, dont l'ensemble forme la charpente de la phrase. Les connexions structurales établissent entre les mots des rapports de dépendance. Chaque connexion unit en principe un terme supérieur à un terme inférieur. Le terme supérieur reçoit le nom de régissant. Le terme inférieur reçoit le nom de subordonné. Ainsi dans la phrase Alfred parle (...), parle est le régissant et Alfred le subordonné.*

[2] Although Tesniére uses the terms *governor* and *subordinate*, alternative terms are *head-dependent* and *regent-modifier*. In this work, we use the *head-dependent* forms.

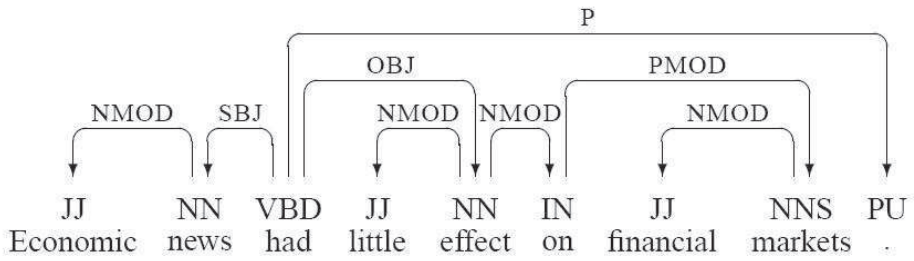

Figure 2.1: An example of a dependency graph.

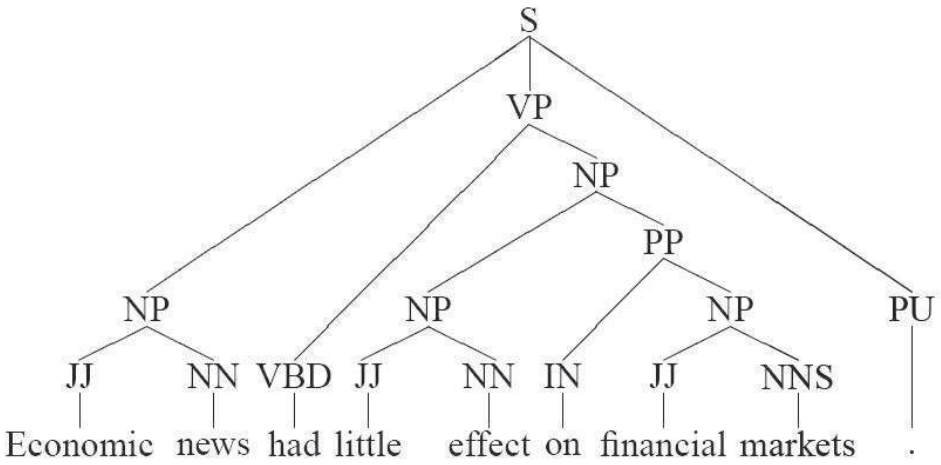

Figure 2.2: An example of a constituent grammar tree.

2. *H* determines the semantic category of *C*; *D* gives semantic specification.

3. *H* is obligatory; *D* may be optional.

4. *H* selects *D* and determines whether *D* is obligatory or optional.

5. The form of *D* depends on *H* (agreement or government).

6. The linear position of *D* is specified with reference to *H*.

This list comprehends syntactic and semantic criteria, leading some theorists to suggest that some instances of dependency relations might not satisfy all of the proposed criteria. Like this one, there are other open questions on which the theorists diverge, hence giving birth to different grammar frameworks.

Some of these questions refer to whether the notion of dependency is assumed as *sufficient* or not – and extra relations would be needed to properly analyze a sentence syntax. In the same way, other theorists question if only one layer of syntactic representation is enough (mono-stratal) or if several layers

are needed to correctly describe syntax (multi-stratal). And even though most theorists agree that the lexical elements on which dependency relations are held are words, some suggest that lemmas or even several words should be used instead.

However, one of the most debated question regards whether dependency relations build a linearly ordered structure, i.e., follow the order in which the words appear in a sentence. This issue is deeply related to the *projectivity* constraint.

2.1.1 Projectivity

According to Nivre and Nilsson [31] "an arc (i, j) is *projective* if and only if all nodes occurring between *i* and *j* are *dominated* by *i* (where dominates is the *transitive closure* of the arc relation)". A simpler and less formal way of presenting this is by describing that *projectivity* does not allow crossing-arcs when representing dependency relation with arcs (considering that all arcs should go over the sentence, but not below it). Figure 2.3 shows a Czech sentence with a non-projective arc coming from *jedna* to *Z* [32].

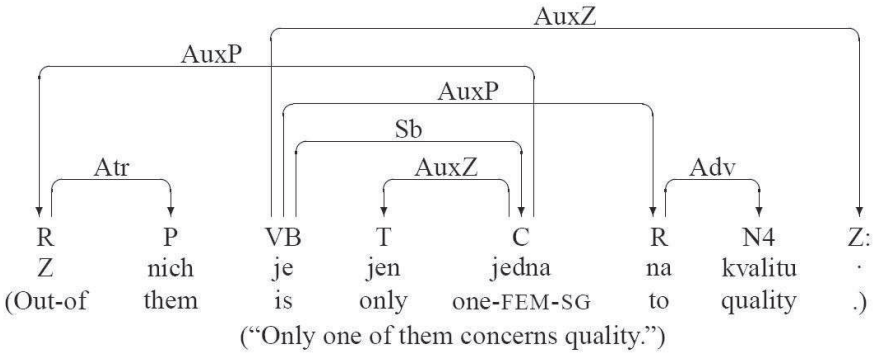

Figure 2.3: An example of a non-projective arc.

Therefore, to assume the projectivity constraint or to define a projective dependency grammar means to only accept relation arcs that are projective.

However, non-projective relations are needed in a dependency grammar to account for long distance relations, as well as to better deal with free word order languages [28]. Additionally, the inclusion of non-projective structures makes the parsing problem more complex and compromises efficiency, accuracy and robustness [31], what leads most transition-based parsers to only build projective dependency graphs [33].

2.2 Data-driven Dependency Parsing

Dependency Parsing can be defined as outputting the dependency graph of an input sentence according to a given dependency grammar. As seen in the previous section, many questions regarding the criteria about dependency relations gave birth to many different dependency grammars. Conversely, in Natural Language Processing a common set of characteristics can be generally assumed about the used grammars: the lexical items that define the nodes are the word forms; the parsing is concerned with only one layer (mono-stratal) of relations; and, finally, the dependency syntax is assumed to be sufficient.

Carrol and Charniak [34] propose the first dependency parsers to use data in ther models, but their parser is, actually, grammar-driven. Their model is based on a formal dependency grammar and uses the *corpus* data only to solve disambiguations left by the grammar based model. In other words, this parsing consists in the derivation of all analysis that are permissible according to the grammar and the selection of the most probable analysis according to the generative model.

Other parsers improve Carroll and Charniak's results while still being based on a formal dependency grammar in combination with a generative probabilistic model, such as the work of Eisner [35] and Collins [36]. Samuelsson's probabilistic model goes on by allowing non-projective dependency graphs and producing labeled dependencies [37]. However, only recently, models that are not based on a formal grammar and are generated purely based on *corpus* data have been proposed.

2.2.1 Transition-based Models

In Transition-based models (or Deterministic Discriminative parsing) a deterministic parser is used to construct dependency structures by having the next action of the parser predicted by a classifier trained in the available data. In this case, no formal grammar is used when inducing the parser model.

In [38], [39] and [40] Support Vector Machines classifiers [41] predict the next action of a parser in order to build an *unlabeled* dependency structure. In these systems the parsing is done according to a shift-reduce model. In shift-reduce parsing, the parser is considered to be initially located at the beginning of the sentence and, at each step, chooses from three different actions. Let the target

words be w_i – the word before the parser – and w_{i+1} – the word after the parser, the possible parser actions are the following [3].

1. **Shift**: The parser simply moves one word along the sentence, adding no dependency relation. The target words change from w_i and w_{i+1} to w_{i+1} and w_{i+2}.

2. **Right**: Builds a dependency relation between words w_i and w_{i+1} with the right word w_{i+1} as head of the left word w_i; reduces the target words into w_{i+1}, making w_{i-1} and w_{i+1} the new target words.

3. **Left**: Builds a dependency relation between words w_i and w_{i+1} with the left word w_i as head of the right word w_{i+1}; reduces the target words into w_i, making w_{i-1} and w_i the new target words.

The processing of a sentence consists in passing it from left to right until no more dependency relations can be added. Since each passing may use up to n steps and up to $n-1$ passes may be required, the worst time complexity is $O(n^2)$. This deterministic discriminative dependency parsing achieves an accuracy near to the state-of-the-art when evaluated in the *Wall Street Journal* section of the *Penn Treebank* [40].

Additionally, the framework of inductive parsing proposed by Nivre et al. [43, 44] enhances this approach with three main differences. First, this framework builds *labeled* dependency graphs, i.e., the dependency arcs have types according to what kind of dependency relation they represent. It also constructs the complete dependency graph in only a single pass over the data. Finally, instead of using Support Vector Machines, Nivre et al. [43] uses Memory-Based Learning in its classifiers.

Pseudo-Projective Parsing

One of the major drawbacks in Transition-based models is its inability of dealing with non-projective arcs. Given the way the parser is structured, only arcs between neighboring words or reduced words can be created, thus limiting it to arcs under a transitive closure, or projective arcs. Nevertheless, a pseudo-projective approach can be applied to overcome this limitation [31].

In pseudo-projective parsing, a preprocess step turns every non-projective arcs into projective ones. Additionally, when doing this, the information regarding the non-projectivity is added in the label of the arc, generating a new label.

[3] Some works present four possible actions, splitting the reduce and arc creation into two different parser actions [42]. Other works use only *Shift* and *Right* actions when applying it to Japanese, since it is a strictly head-final language [38, 39].

This new label allows the new projective arc to be turned back to the original non-projective arc. As a result, when the parsing classifier learns to correctly label the arcs, it will also learn to predict the new pseudo-projective labels. Therefore, after a pseudo-projective parser is applied, a post-process step changes the pseudo-projective arcs back into their corresponding non-projective arcs.

This pseudo-projective approach significantly improves overall parsing accuracy for non-projective *corpus* [42], obtaining the best reported performance for robust non-projective parsing of Czech [31].

Finally, there are several powerful system that use transition-based models, with or without the pseudo-projective approach [45, 46, 47, 48, 49, 50].

2.2.2 Graph-based Models

Another type of models that do not use a formal grammar as basis in their parsing are Graph-based models. However, while transition-based models try to locally find the best dependency relations, Graph-based models learn a model of the globally best dependency graph given an input sentence.

Generally, Graph-based models define a scoring or probability function over a set of all possible parsers. First, during the learning step, the set of parameters of this function is estimated. Later, during the parsing step, the graph that maximizes the score given by this function is built, therefore building the dependency graph.

Most systems that use Graph-based models differ mainly in the type and structure of the scoring function, the method to estimate the function's parameters and the search algorithm that infers the best parse given a score.

Scoring Function

The simplest type of scoring function is referred to as *first-order model* or *edge-factored model*. This type of function sums a set of local attachment scores. Each local score is calculated based on the *dot product of a weight vector* and a *feature representation of the attachment*. Several systems use first-order models [51] [52] [53] [54] [55].

A second-order model extends the first model to incorporate a sum over scores for pairs of adjacent arcs in the tree [56] and can even take head-grandchild relations into account [57].

Usually, the scoring function is decomposed into functions that score local properties, like arcs or pairs of adjacent arcs, but global properties of the graph can also be taken into account, like children of nodes and its siblings [58].

Estimation of Parameters

The estimation of function's parameters is actually the learning step of Graph-based models. In this case, inference-based methods are commonly used to set those parameters so the scoring function can output the correct dependency graph. Inference-based methods include *passive-aggressive learning* [52] [51], *averaged perceptrons* [56] and *Margin Infused Relaxed Algorithm* [53]. Another common method to estimate parameters of the scoring function is the *maximum conditional likelihood* [58] [54].

Parsing Inference

The inference of the correct dependency parsing is done by searching for the highest scoring graph given the scoring of the functions in the first step. Therefore, it depends on the chosen factorization and whether the treebank allows non-projective or only projective dependency graphs.

First-order models that allow non-projective relations simply use *Maximum Spanning Tree* algorithms like the works of [52] [53] [54], while projective cases use a *dynamic programming* algorithm proposed by Eisner [59], as the works of [51].

Finally, Carreras [56] extends Eisner's algorithm to infer the best graph in his second-order model.

2.2.3 Conference on Computational Natural Language Learning Shared Task

Each year the Conference on Computational Natural Language Learning features a shared task, in which participants train and test their systems on exactly the same data set, in order to better compare systems. In 2006 and 2007, the CoNLL shared tasks were strictly on Multilingual Dependency Parsing, aiming to define and extend the state-of-the-art in this task.

Ideally, a parser should be trainable for any language, possibly by adjusting a small number of hyper-parameters, therefore the use of different languages on the evaluation of the proposed systems.

In this work we evaluate our tagging style on three *corpora* of the CoNLL 2006 shared task that are publicly available, namely Danish, Dutch and Portuguese. Table 2.1 presents the results of every participating team by the occasion of the conference. The metric presented is the Unlabeled Attachment Score, i.e., the accuracy of the systems in predicting the correct head of the tokens.

Team	Danish	Dutch	Portuguese
Canisius et al.[60]	82.93	77.79	85.61
Attardi[61]	78.84	68.93	85.03
Wu[62]	83.39	71.75	85.57
Carreras et al.[63]	85.67	71.39	87.76
Yuret[64]	78.16	66.17	79.46
Bick[65]	80.54	74.47	84.29
Nivre et al.[66]	89.80	81.35	91.22
Schiehlen[67]	81.94	75.59	81.27
Ma[68]	79.90	64.07	77.10
Dreyer et al.[69]	77.45	68.33	82.41
O'Neil	88.78	81.73	89.70
Xuan Do	86.85	76.25	88.60
Johansson[70]	86.59	76.01	88.40
McDonald et al.[71]	**90.58**	**83.57**	**91.36**
Riedel et al.[72]	89.66	82.91	89.42
Sagae	86.53	80.71	89.78
Shimizu[73]	81.72	–	–
Corston-Oliver[74]	87.94	74.83	88.96
Cheng[75]	88.64	75.49	90.30
Average of All Teams	84.52	75.07	86.46
Standard Deviation	4.29	5.78	4.17

Table 2.1: Unlabeled Attachment Score of CoNLL 2006 Systems.