LIN5022 Grammar Models
Dependency Grammars

Thuong-Hai Pham

February 17, 2017

In the scope of this report, we will discuss about the basic concepts and terminologies being used to constitute the Dependency Grammar in section 1.1. Thereafter, we clarify the head-dependant criteria in section 1.2 followed by formal conditions and properties of dependency graph in section 2. To end with the review of Dependency Grammar, four dimensions in variant space is discussed in detail within section 3. Finally, we review different approaches for the problem of dependency parsing.

1 Dependency Grammar

1.1 Basic concepts & terminologies

In fact, Dependency Grammar is not a single consistently established grammar, yet a wide range of variants that share several basic assumptions. The primary underlying idea is the syntactic structure which consists of lexical items, connected by binary asymmetric relations. These relations are called dependencies. Although it is said that this concept had been used early in Panini's work for Sanskrit grammar around 6^{th} to 4^{th} century BCE, we can consider the starting point at the time when it were introduced systematically by [Tesnière, 1959] as quoted in his work (translated to English by [Nivre, 2005])

> The sentence is an organised 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 neighbours, the mind perceives connections, the totality of which forms the structure of the sentence. The structural connections establish dependency relations between the words. 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.

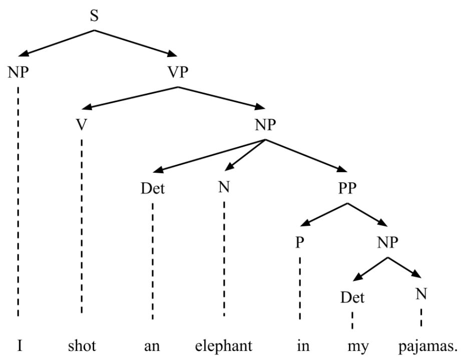
As quoted above, the two parties involved in this type of relation are called governor and subordinate. In literature, the governor can be also called by other names, most popular as head, or regent. While the subordinate is dependant, or modifier.

Figure 1 presents two samples of different grammar trees. While the phrase structure tree (figure 1a) represents phrases (non-terminal nodes) and structural categories, the dependency tree depicts the head-dependent relations (with directed arcs) between its lexical items (terminal nodes). Figure 2 also describes the functional categories of these relations by arc labels.

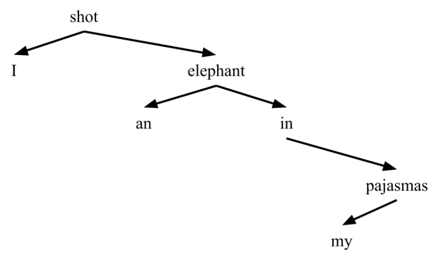
It is important to note that Dependency Grammar is descriptive instead of being a generative system like others. However, some equivalent frameworks are introduced to solve this.

1.2 Heads and dependants conditions

To determine the head H and dependant D roles in the mentioned binary asymmetric relation C, [Zwicky, 1985] proposed the following criteria. To be precise, these criteria are not only important to Dependency Grammar, yet other frameworks which emphasise the role of syntactic heads.



(a) Phrase Structure Grammar (PSD)



(b) Dependency Grammar (DG)

Figure 1: Grammar trees examples

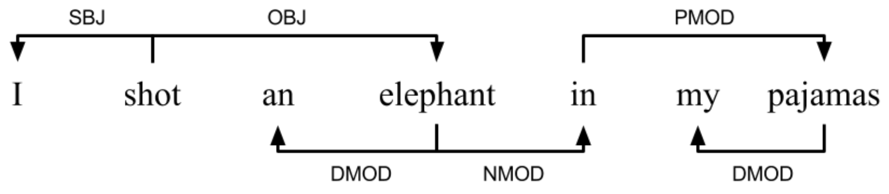


Figure 2: Dependency tree with labels

1. H decides the syntactic category of C and is able to replace the whole construction C

2. H decides the semantic category of C and D specifies H

3. H is obligatory while D can be optional

4. H selects D and specifies whether D is obligatory or not

5. The form of D depends on H (agreement…)

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

It is obvious that the mentioned criteria are mixture of both semantic and syntactic types (cf. criterion 1 and 2...).

By satisfying different combination of these criteria, our head-dependant relationship can be classified into various kinds of dependencies. For examples, endocentric construction refers to a construction that the head can replace the whole construction and does not destroy the structure at the same time. The relation between *elephant* and *in* in Figure 2 is an example of this type. In which, *elephant* can substitute the whole portion of "elephant in my pajamas" (construction between head and dependant which includes the dependant's subordinates). The type of construction may satisfy all the criteria.

On the other hand, also in Figure 2, the exocentric construction between *shot* and *elephant* fulfils all the requirements except the first one. It is obvious that in this type of relation, by replacing the construction by head, we disrupt the whole structure, in syntactic term. The distinction that differs endocentric and exocentric is also related to the distinction between head-argument and head-modifier because, as in our examples, the head-modifier relation is endocentric while head-argument is exocentric.

In addition, there is also the third type of relation, recognised as head-specifier relation, between function words and their arguments (e.g. determiner and noun). In Figure 2, a clear example is *an* and *elephant.* This relation type is exocentric in the same way of head-argument relation. Besides the three kinds mentioned, there are also some situation that is hard to determined. For example, coordination (Tesnère's junction) such as "We would like to have *milk and tea*", in which case determining where the head is may be impossible. The similar problems also occur in complex verb group (auxiliary and main verb), subordinate clauses and prepositional phrases…

2 Dependency graphs

Our dependency tree in Figure 1b can be defined as a directed graph $G = \{V, A, <\}$ in which:

- V is the set of node (or vertices)
- A is the set of arcs (or directed edges)
- $<$ depicts the word order on V

While nodes in V are usually labelled with word forms, in the labelled version of the graph (Figure 2), element of set A is a triple (i, j, l) where $i, j \in V, l \in L$ which L is the set of available arc labels.

2.1 Formal conditions

To capture the dependent relations between lexical items, there are some constraints on the defined graph that needs to be followed. To formally state these constraints, we need to define these following notation of graph G:

- Node i is the head of node j or i governs j or j depends on i

$$i \rightarrow j \equiv \exists l \in L : (i, j, l) \in A$$

- Node i and j have a relationship:

$$i \leftrightarrow j \equiv i \rightarrow j \vee j \rightarrow i$$

- Node i is an ancestor of node j (there exists a path from i to j through arcs in G):

$$i \rightarrow^* j \equiv \exists i' : i \rightarrow i', i' \rightarrow^* j$$

- Node i and j have a long-distance relationship (i and j are weakly/strongly connected)
$$i \leftrightarrow^* j \equiv \exists i' : i \leftrightarrow i', i' \leftrightarrow^* j$$

By having defined those notations, the constraints on graph G can be listed as follow:

- G is weakly connected, i.e. there always exists an undirected path between any nodes in G (connectedness)
$$i \leftrightarrow^* j, \forall i, j \in V$$

- G has no cycle (acyclic)
$$\neg(i \rightarrow^* j \wedge j \rightarrow^* i), \forall i, j \in V$$

- Each node in G has only one head (single-head)
$$\forall i, j \in V, i \rightarrow j : \nexists i', i \rightarrow j$$

- G is projective
$$\forall i, j \in V, i \rightarrow j : i \rightarrow^* i', \forall i' \in V, i < i' < j \vee j < i' < i$$

The final condition, projectivity, can be interpreted as for any node (lexical item) that lies between head i and j (left arc with $j < i' < i$ or right arc with $i < i' < j$) must take i as its ancestor. This is to make sure of there is no intersection between arcs when they are represented as in Figure 2. Please note that most theoretical frameworks do not assume projectivity which needed to denote long-distance dependencies. In addition, projectivity is not taken for granted in free-word-order languages.

2.2 Formal properties

Consequently, graph G as defined and obeyed the constraints above holds its own properties:

- Anti-symmetric: in which $\forall i, j \in V, i \rightarrow j \Rightarrow j \not\rightarrow i$
- Anti-reflexive: a node can be its own head $i \rightarrow j \Rightarrow i \neq j$
- Anti-transitive: $\forall i, j, k \in V, i \rightarrow j, j \rightarrow k \Rightarrow i \not\rightarrow k$ which means there is no direct arc from one (non-parent) ancestor to a node. This is also guaranteed by single-head condition.

3 Variants of Dependency Grammar

As stated in the beginning of this report, Dependency Grammar is not a single well-established framework but a varieties of frameworks that share several standard assumptions. These variants can be classified by four big open questions serve as four dimensions.

3.1 Necessity and sufficiency

The first question to be asked is whether dependency structure is sufficient as well as necessary. The original dependency structure of connection (as quoted from Tesnère) does not rise this question. However, the concept of junction and transfer draw our attention in different kinds of dependency. While junction was introduced in the previous section, transfer depicts the relation between function word and the word that is being changed, in term of syntactic category.

3.2 Mono-stratal vs. multi-stratal

The second dimension categorises mono-stratal and multi-stratal frameworks. To be more precise, the mono-stratal framework relies on one single syntactic representation, which class includes the original theory. While multi-stratal counterparts, as most theories of Dependency Grammar, take into account several layers of representation. For example, Mean Text Theory [Melčuk, 1988] allows different dependency layers which are surface syntactic and deep syntactic representations. The need of the multi-layer dependency is very intuitive. Let us take the two following sentences:

```
I am a student.
We are students.
```

In traditional sense of dependency, subject depends on the main verb, as *I* to *am* or *we* to *are*. However, the form of these verbs also depend on the subjects (in case of agreement). Hence, in this case, dependence of pronouns *I*, *we* on the verb is syntactic, whereas, dependence of the verb (word forms) on the subjects is morphological.

3.3 Nature of lexical elements (nodes)

The third issue that attracts people's concern is what can be defined as the node of our dependency structure. Even though the basic assumption that the dependency relations are established between lexical elements instead of phrases (as in Phrase Structure Grammar), these elements vary between theories within the Dependency Grammar family. The original perspective assigns these nodes plainly the word forms in the sentence. Still, these nodes can be constructed by other entities such as lemmas or morphemes (in morphological dependency [Melčuk, 1988]).

3.4 Nature of dependency types (arc labels)

The fourth dimension of this variation space is the list of dependency types (i.e. arc labels such as SBJ, OBJ, NMOD...) employed by different theories. While some theories make use of common set of grammatical functions (e.g.

subject, object...), the other deeper/sub-classification set (agent, patient, goal...)can also be used. Moreover, it is also possible to use unlabelled dependency structure, albeit not common among linguistic theories.

4 Dependency Parsing

4.1 Deterministic parsing

In deterministic parsing, the parser attempts to derive a syntactic representation after having processed a sequence of parsing actions. One notable incremental algorithm [Covington, 2001] links each new word to each preceding one in one single pass, with the complexity $O(n^2)$. More efficient, the shift-reduce type algorithm by [Nivre, 2003] introduced three actions when parsing: shift, left-arc, right-arc. These actions are applied on a *configuration* which consists of: a stack S of partially processed items, a buffer B initiated with our sentence and an empty list of relations R. Details of the actions are

- Shift (SHIFT): remove a word from buffer B and push it into stack S
- Left-arc (LEFT): assert dependency relation between the top element of stack S and the word beneath. If true, then remove the lower word (word on top of the S is head)
- Right-arc (RIGHT): assert dependency relation between the second top and top element of stack S. If true, then remove the top element (word on top of the S is dependant)

Table 1 illustrates step-by-step actions of the algorithm. It is clearly that the algorithm runs in $O(n)$ time with a greedy strategy. The only thing left is how can our algorithm determines what action needed to be applied in a specific step. To do this, the author trained an *oracle* with tree bank data to classify the required action given a configuration. Features used for this training step can be taken into consideration as word forms, part-of-speech, dependency type, etc. In general, the more features used, the better our oracle performs in term of accuracy. However, to avoid sparsity and achieve better generalisation, it is suggested to focus on most significant features at the moment of predicting (top 2 elements of the stack S...)

Action	**Stack**	**Buffer**	**Relations**
	[root]	[I, shot, an, elephant, in, my, pajamas]	
SHIFT	[root, I]	[shot, an, elephant, in, my, pajamas]	
SHIFT	[root, I, shot]	[an, elephant, in, my, pajamas]	
LEFT	[root, shot]	[an, elephant, in, my, pajamas]	(I←shot)
SHIFT	[root, shot, an]	[elephant, in, my, pajamas]	
SHIFT	[root, shot, an, elephant]	[in, my, pajamas]	
LEFT	[root, shot, elephant]	[in, my, pajamas]	(an←elephant)
SHIFT	[root, shot, elephant, in]	[my, pajamas]	
SHIFT	[root, shot, elephant, in, my]	[pajamas]	
SHIFT	[root, shot, elephant, in, my, pajamas]	[]	
LEFT	[root, shot, elephant, in, pajamas]	[]	(my←pajamas)
RIGHT	[root, shot, elephant, in]	[]	(in→pajamas)
RIGHT	[root, shot, elephant]	[]	(elephant→in)
RIGHT	[root, shot]	[]	(shot→elephant)
RIGHT	[root]	[]	(root→shot)

Table 1: Nivre's algorithm example

4.2 Non-projective dependency parsing

Nevertheless, the algorithms mentioned in previous section can only perform on projective dependency structure. Without projectivity assumption, there are two types of approaches to deal with the problem. First, one can apply specialised algorithm for non-projective dependency parsing such as a set of constraint satisfaction methods ([Duchier and Debusmann, 2001], [Foth et al., 2004]) or graph-based algorithm by finding minimum spanning tree [McDonald et al., 2005]. The second type is to do post-processing. This post-processing techniques generally require two steps:

1. Approximate the best possible projective dependency graph.
2. Improve our approximate solution by replacing projective arcs with non-projective arcs.

The group of approaches include approximate non-projective parsing [McDonald and Pereira, 2006] or pseudo-projective parsing [Nivre et al., 2006] in which the algorithm *projectivise* the training data by providing “projective head” (nearest permissible ancestor) instead of “real head”.

References

[Covington, 2001] Covington, M. A. (2001). A fundamental algorithm for dependency parsing. In *Proceedings of the 39th annual ACM southeast conference*, pages 95–102. Citeseer.

[Duchier and Debusmann, 2001] Duchier, D. and Debusmann, R. (2001). Topological dependency trees: A constraint-based account of linear precedence. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 180–187. Association for Computational Linguistics.

[Foth et al., 2004] Foth, K. A., Daum, M., and Menzel, W. (2004). A broad-coverage parser for german based on defeasible constraints. *Constraint Solving and Language Processing*, page 88.

[McDonald et al., 2005] McDonald, R., Pereira, F., Ribarov, K., and Hajič, J. (2005). Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 523–530. Association for Computational Linguistics.

[McDonald and Pereira, 2006] McDonald, R. T. and Pereira, F. C. (2006). Online learning of approximate dependency parsing algorithms. In *EACL*, pages 81–88.

[Melčuk, 1988] Melčuk, I. A. (1988). *Dependency syntax: theory and practice*. SUNY press.

[Nivre, 2003] Nivre, J. (2003). An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT*. Citeseer.

[Nivre, 2005] Nivre, J. (2005). Dependency grammar and dependency parsing. *MSI report*, 5133(1959):1–32.

[Nivre et al., 2006] Nivre, J., Hall, J., Nilsson, J., Eryiit, G., and Marinov, S. (2006). Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 221–225. Association for Computational Linguistics.

[Tesnière, 1959] Tesnière, L. (1959). *Eléments de syntaxe structurale*. Librairie C. Klincksieck.

[Zwicky, 1985] Zwicky, A. M. (1985). Heads. *Journal of linguistics*, 21(01):1–29.