DEPENDENCY PARSING USING DEPENDENCY GRAPH

Tomasz Obrębski
Poznań University of Technology*
Adam Mickiewicz University
Poland
e-mail: Tomasz.Obrebski@put.poznan.pl

Abstract

In this paper an efficient algorithm for dependency parsing is described in which ambiguous dependency structure of a sentence is represented in the form of a graph. The idea of the algorithm is shortly outlined and some issues as to its time complexity are discussed.

1 Introduction

The paper describes a computationally efficient dependency parsing algorithm. It has been developed for language engineering applications to precess raw text corpora on the syntactic level. Since our primary concern was the efficiency, we have considered a limited coverage of the syntactic constructions; hence discontinuity, ellipsis, and several other types of complicated, textually infrequent, constructions have not been handled. The algorithm were implemented in the dependency paser **dgp** (**d**ependency **g**raph **p**arser).

2 Syntactic description and parsing algorithm

The syntactic structure of a sentence is represented as a projective dependency tree, whose nodes are labeled with words while arcs are labeled with dependency types (cf. [2]). In addition, syntactic attributes, eg. expressing the property of being the head of a relative clause, may be assigned to nodes during the parsing process. The grammatical description is formulated in the form of constraints determining the possibility of establishing a dependency between a given pair of words and rules of syntactic attributes assignment.

In order to describe the algorithm we introduce the following relations among the nodes of a dependency tree and, also, dependency graph: w_i is a transitive left dependent of w_j, if w_i is a transitive dependent of w_j and all intervening dependencies are directed leftwards (considering linear order of words). The node w_i is a transitive left head of w_j, if w_i is a transitive head of w_j and all intervening dependencies are directed rightwards. The node w_i is visible to w_j (on the

*This work was partially financed by Poznań University of Technology, Research Project No 45-083/03/DS.

left), if w_i is a transitive left head of the left surface neighbour of a transitive left dependent of w_j. The visibility relation expresses the structural condition for two nodes to be connectable.

The basic idea is as follows: dependency tree construction algorithms usually operate by adding, for as long as possible, a grammatically licenced arc between two nodes mutually visible of which none has a head (e.g. [1]). If we drop the letter condition, the algorithm will produce a graph, which will contain all arcs that would be added in all possible (successful and unsuccessful) passes of the tree construction algorithm and those arcs only. All projective dependency trees possible to construct are sub-graphs of this graph. We call this structure a dependency graph. The complexity of the deterministic version of the algorithm following this idea is $O(n^3)$, which is the same as for the tree construction algorithm. In the implementation it is reduced[1] to $O(n^2)$, and the test-measured complexity is still lower. With the core algorithm only context-independent grammatical constraints, i.e. those which refer only to the categories and the relative position of the words being connected, can be taken into account during graph construction. The core algorithm can be extended to handle ambiguous input, obligatory dependencies, and additional order constraints with time complexity increased by a constant factor. Incorporation of mechanisms needed to handle propagation of syntactic attributes, which involve node duplication, significantly affects the theoretical complexity, only slightly affecting, however, the test-measured complexity.

After constructing the graph, we have the information of each node's all (possible) transitive left heads and dependents, as well as nodes visible on the left. Using this information, each possible tree can be generated in time $O(n^2)$ ($O(n)$ in the implementation).

In the **dgp** parser the grammar is represented as a set of two-dimensional tables indexed with word categories, what makes grammatical information instantly accessible. For a fourty-word sentence the graph construction time is about 10ms (PC 1,2 GHz). One tree is generated in approximately 0,1ms. More details about the algorithm can be found in [3].

References

[1] R. Hudson. Towards a computer-testable word grammar of english. In *UCL Working Papers in Linguistics*, volume 1. University College London, 1989.

[2] I. A. Mel'čuk. *Dependency Syntax: Theory and Practice.* State University of New York Press, 1988.

[3] T. Obrębski. Dependency parsing using dependency graph for storing ambiguous structure. ICII Technical report, Poznań University of Technology, Jan 2003.

[1] The algorithm performs computations on sets of transitive left heads and visible nodes. The operations of union and intersection of ordered sets, which theoretically are linear, in the calculation of the implementational complexity are considered as unary operations, as they are realized as operations on bit vectors of constant size.