

A Two-Stage Constraint Based Dependency Parser for Free Word Order Languages

Akshar Bharati, Samar Husain, Dipti Misra Sharma, and Rajeev Sangal

LTRC, International Institute of Information Technology, Hyderabad, India.

{samar, dipti, sangal}@mail.iiit.ac.in

Abstract

The paper proposes a broad coverage two-stage constraint based dependency parser for free word order languages. For evaluating the parser and to ascertain its coverage we show its performance on Hindi which is a free word order language. We compare our results with that of two data-driven parsers which were trained on a subpart of a Hindi Treebank. The final results are good with a maximum attachment and labeled attachment accuracy of 90.1% and 75% respectively. The results point towards the phenomena which the data driven parsers find easier to learn vs. those for which the grammar driven approach can prove more effective.

1 Introduction

Parsing morphologically rich, free word order languages (MoR-FWO) is a challenging task. It has been suggested that free word order languages can be handled better using the dependency based framework than the constituency based one (Hudson, 1984; Shieber, 1985; Mel'cuk, 1988, Bharati et al., 1995). The basic difference between a constituent based representation and a dependency representation is the lack of nonterminal nodes in the latter.

Dependency parsing can be broadly divided into grammar-driven and data-driven parsing (Carroll, 2000). Most of the modern grammar-driven dependency parsers parse by eliminating the parses which do not satisfy the given set of constraints. They view parsing as a constraint-satisfaction problem. Some of the constraint based parsers known in the literature are Karlsson et al. (1995), Maruyama (1990), Bharati et al. (1993, 2002), Tapanainen and Järvinen (1998), Schröder (2002), and more recently, Debusmann et al. (2004) which provides multi-stratal (or multi-dimensional) para-

digms to capture various aspects of a language. Data-driven parsers, on the other hand, use a corpus to induce a probabilistic model for disambiguation (Nivre, 2005; and the references therein).

In this paper, we propose a 2-stage dependency parsing algorithm. We show how a 2-stage modular approach will lead to a *selective resolution of demands*, where complex linguistic phenomena are parsed seamlessly. We describe how the notion of *repair* complements the idea of modularity and makes the overall approach simple, and how the overall system is made robust by producing *partial parse*. The parser takes a set of inequalities in integer programming (IP) using the constraints specified for the language. It then solves them to give possible parses. For evaluating the parser we show its performance on Hindi¹. We later compare this performance with that of two data-driven parsers; Malt (Nivre et al., 2007b), and MSTparser (McDonald et al., 2005). The results show that the proposed parser provides *exhaustive coverage* and is able to handle major linguistic phenomenon with *good performance*. The framework presented in the paper is language independent, however, rules (or demand frames) need to be prepared for the language under consideration. For Hindi, such frames have been prepared (Begum et al., 2008b).

The paper is arranged as follows: Section 2 gives a background of parsing for IL. Section 3 discusses the approach. Section 4 gives the algorithm. Section 5 evaluates the parser and shows its coverage. Section 6 discusses these results.

2 Constraint Parsing

Constraint based parsing using IP has been successfully tried for Indian languages (Bharati et al., 1993, 2002). Under this scheme the parser exploits the syntactic cues present in a sentence and forms

¹ Hindi is a verb final language with free word order and a rich case marking system. It is one of the official languages of India, and is spoken by ~800 million people.

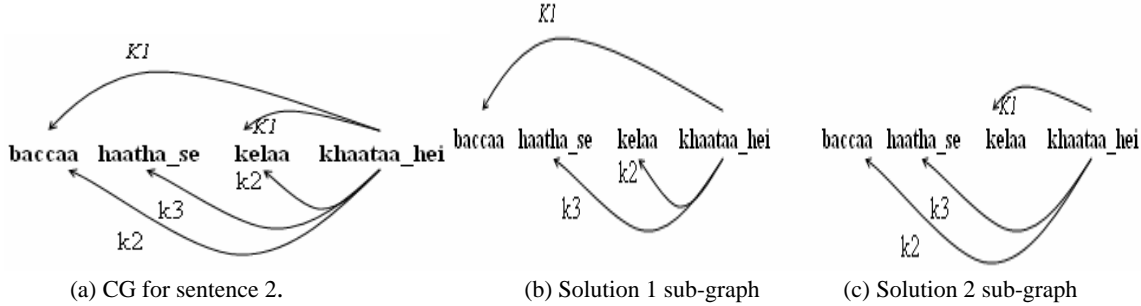


Figure 1. Constraint graph & Parses for sentence (2)

constraint graphs (CG). It then translates the CG into an IP problem. The solutions to the problem provide the possible parses for the sentence.

As part of this framework, a mapping is specified between *karaka*² relations and the postpositions. In the Paninian grammar (Bharati et al., 1995) this mapping is given by a structure called the *basic karaka frame* (earlier called karaka chart). It specifies what karakas are mandatory or optional and what vibhakti (postposition/suffix) they take. Basic karaka frame given in Table 1³ correctly derives sentence (2).

- (2) *baccaa haatha se kelaa khaataa hei*
 child hand INST banana eats is
 ‘The child eats the banana with his hand.’

<i>karaka</i>	<i>vibhakti</i>	Presence
karta (k1)	0	mandatory
karma(k2)	ko or 0	mandatory
Karana(k3)	se or <i>dvaaraa</i>	optional

Table 1³. Basic karaka frame for ‘khaa’ (eat)

This mapping depends on the verb and its tense aspect modality (TAM) label. The mapping is represented by two structures: *basic karaka frame* (or demand frame) and *karaka frame transformations*. The basic karaka frame for a verb or a class of verbs specifies the vibhakti permitted for the applicable karaka relations for a verb when the verb has the basic TAM label. Table 1 gives the frame for the verb ‘khaa’ (eat) when it takes the basic TAM label ‘taa hei’ (which corresponds to ‘present indefinite’). For other TAM labels there are karaka frame transformation rules. Thus, given a verb with some TAM label, appropriate karaka frame can be obtained using its basic karaka frame and the transformation rule depending on its TAM la-

bel (Bharati et al. 1995). The transformations affect not only the vibhakti of karta but also that of other karakas. At times, they also delete karaka roles, that is, the deleted *karaka* roles must not occur in the sentence.

A demand word or word group is an element which makes demands, for example verbs make demands for their karakas through demand frames (Table 1). These demands are satisfied by source word or word groups. A source group becomes a potential candidate for a verb only after it satisfies the vibhakti specification as mentioned in the verb’s demand frame. This can be shown in the form of a CG. Nodes of the graph are the word groups and there is an arc labeled by an appropriate karaka relation from a verb group to a selected source group. Figure 1a shows the CG for (2), where all such source groups are nouns. (In case of sentential arguments, there is an arc from one verb group to another.)

A parse is a sub-graph of the CG thus formed, containing all the nodes of the CG and satisfying the three conditions: (1) For each of the mandatory karakas in a karaka frame for each demand word group, there should be *exactly one outgoing edge labeled by the karaka from the demand group*, (2) For each of the desirable or optional karakas in a karaka frame for each demand word group, there should be *at most one outgoing edge labeled by the karaka from the demand group*, (3) There should be *exactly one incoming arc coming into a source group*.

A CG is converted into an IP problem by introducing a variable x for an arc from node i to j labeled by karaka k in the CG such that for every arc there is a variable. The variables take their values as 0 or 1. A parse is an assignment of 1 to those variables whose corresponding arcs are in the parse sub-graph, and 0 to those that are not. Equality and inequality constraints in integer programming problem can be obtained from the conditions listed

² karaka relations are syntactico-semantic relations, see Bharati et al. (1995) for details.

³ karta (k1), karma (k2) and karana (k3) could be roughly translated as ‘initiator’, ‘theme’ and ‘instrument’ respectively.

earlier. The cost function to be minimized is given as the sum of all the variables. Figure 1b,c shows the solution sub-graphs for sentence (2). Efficient methods based on bipartite graph matching are known for finding solution graphs (Bharati et al., 1993) with or without nesting (Bharati et al., 1995). The parser presented in this paper builds on a modified version of the approach just discussed.

3 Approach

The parser tries to analyze the given input sentence, which has already been tagged and chunked⁴ in two stages; it first tries to parse intra-clausal dependency relations. These relations generally correspond to the argument structure of the verb, noun-noun genitive relation, infinitive-verb relation, infinitive-noun relation, adjective-noun, adverb-verb relations, etc. In the second stage it then tries to handle more complex relations such as conjuncts, relative clause, etc. What this essentially means is a *two-stage resolution of demands*, where the parser *selectively resolves* the demands of specific demand groups at their appropriate stage, for example, verbs in the 1st stage and conjuncts in the 2nd stage.

3.1 Two Stages

We propose a two stage analysis to handle cases like conjuncts, relative clauses, conjunct verb, etc. The two stages indicate a distance between syntax and semantics. And can be thought as a device for modularity. For example, the noun in a conjunct verb (or light verb) phenomenon, normally behaves like a k2 (*karma*⁵) in terms of agreement, disjointness, etc. We illustrate this by a similar English phenomenon,

(3) *Ram took a bath.*

In stage 1, the parser would mark ‘bath’ as k2⁵ of ‘took’. Later, in stage 2, ‘bath’ would be marked as ‘pof’ (‘part of’ relation indicating compounding) instead of k2 indicating that the two together specify the verb. Thus, the first stage treats ‘bath’ like a general argument of the verb. In stage 2, however,

the parser changes it to ‘pof’ after recognizing that it is part of a conjunct verb. Thus, the parser described in section 2 represents stage 1. In the first stage the parser tries to capture all the verb argument relations like karta, karma, etc; in the second stage it marks the operators (representing compounds) like conjunct relation, ‘pof’ relation, relative clause relation, etc. This *selective resolution of demands* gives the parser the power to handle various complex phenomena seamlessly, as it tries to map one level of representation with two-stage parsing. We expect that this approach would also lead to better machine learning, where the machine first learns local dependencies before non-local dependencies.

We introduce a special dummy node named *_ROOT_* which becomes the head of the sentence. This is done so that all the categories including verbs are handled in the same way and that the three basic constraints (cf. section 2) act consistently across all the categories. The only exception now is *_ROOT_* for which we have a single constraint that at the end of stage 2 it should have only one outgoing arc and no incoming arc. As we will see in Section 4, the presence of *_ROOT_* also makes it possible for the parser to give *partial parse*. Such a mechanism is desirable to make the parser robust.

The overall scheme, now, has three types of nodes; (a) Nodes that *look for their children*, for example, verbs; (b) Nodes that *look for their parent*, for example, adverbs, adjectives, participle, relative clause markers, etc., and (c) Nodes which *do not look for either parent or children*. These types follow from their linguistic demands.

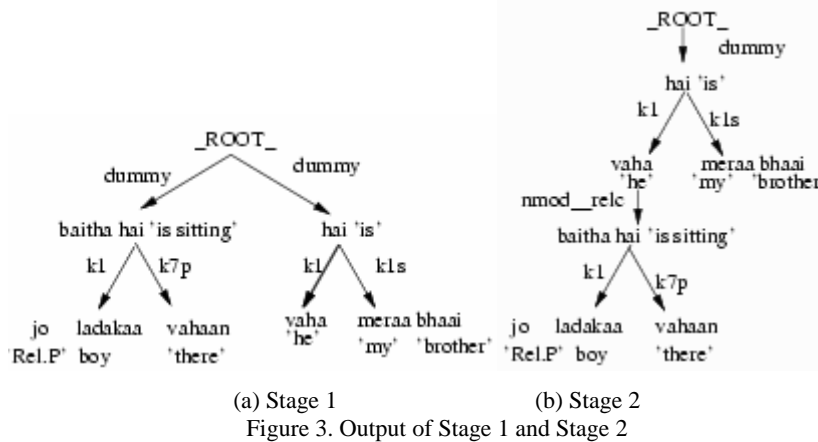
Example 4 shows the two stages of the parser for a relative clause construction, which is shown in brackets.

(4) [*jo ladakaa vahaan baithaa hai*] *vaha meraa bhaai*
 ‘which’ ‘boy’ ‘there’ ‘sitting’ ‘is’ ‘that’ ‘my’ brother
hai
 ‘is’
 ‘The boy who is sitting there is my brother’

Figure 3 shows the output of stage-1 and stage-2 for the example (4). The first stage marks all the dependency relations within both the finite clause ‘*vaha meraa bhaai hai*’ and ‘*jo ladakaa vahaan baithaa hai*’. This can be seen in Figure 3a. Having done that, the second stage finds the elements in the main clause to which the co-referent ‘*jo*’ in the relative clause corefers. In Figure 3b the root of the relative clause (*baitha hai*) gets attached to this

⁴ A chunk is a set of adjacent words which are in dependency relation with each other, and are connected to the rest of the words by a single incoming arc. The parser marks relations between the head of the chunks (inter-chunk relations); this is done to avoid local details and can be thought as a device for modularity. One can, if required, expand these chunks in a post-processing step using few rules, where the intra-chunk relations are made explicit. Experiments have been conducted with high performance in automatically marking intra-chunk relations. Due to lack of space we do not elaborate on this post-processing.

⁵ karma can be roughly translated as ‘object/theme’

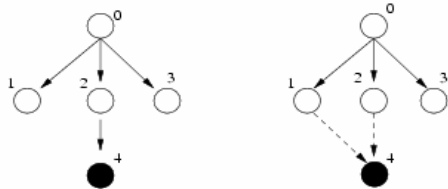


element with a 'relc' (relative clause) relation. Note that at the end of first stage all the finite verbs get attached to `_ROOT_`.

4 Algorithm

The algorithm for stage 1 essentially produces parse for individual clauses. It is called Intermediate parse. Relative clauses are parsed but their parents are not identified. Conjuncts are essentially not processed. Conjuncts and relative clauses are made the children of `_ROOT_` (this is done for any word group which is left hanging).

In stage 2, some of the arcs in the solution graph at the end of stage 1 (intermediate parse tree) get repaired. Therefore, some of the nodes of the intermediate parse tree with the temporarily assigned parent may get a new parent, etc. To accomplish this, the following general principles are followed: (1) For any node which becomes a potential child in stage 2, its arc to its existing parent is open to revision. For example, in Figure 4, after node 4 is identified as a potential child of 1 (shown as a dashed arc between 1 and 4 in (b)) the arc between 4 and 2 is open for revision (shown as a dashed line in (b))



(a) Output of Stage 1 (b) While in Stage 2
Figure 4: General Repair Principle 1

(2) Any node which becomes a potential parent must be re-looked at (this is the complement of 1).

In figure 5, node 2 can now take either node 1 or node 4 as its child.

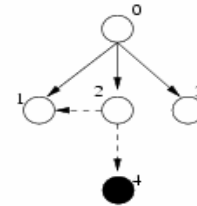


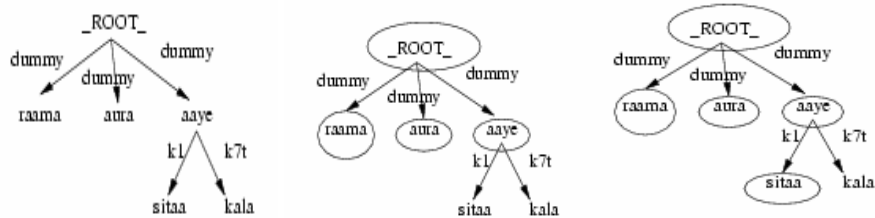
Figure 5: General Repair Principle 2

In Stage-2, a new CG is formed consisting of nodes and possible arcs. Usually, it consists of much smaller number of nodes compared to the overall graph. To compute the list of nodes whose parents or children are to be identified in Stage-2, set 1 and set 2 of nodes are computed. **Set 1** consists of nodes such as: (a) *Conjuncts*, (b) *Nearest verbal ancestor of the 'jo' node*, (c) `_ROOT_` and (d) *Children of `_ROOT_` other than (a) and (b)*. **Set 2** consists of nodes such as; (a) *Possible children and parents of conjuncts*, (b) *Possible heads of the relative clause*. Identification of nodes in Set 2 will generally trigger the repair.

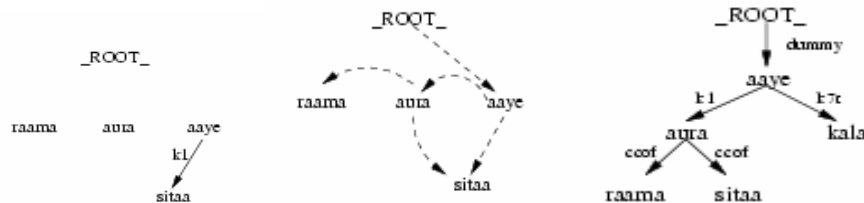
Considering the above point, the algorithm for the second stage is:

- (1) Identify Set 1 nodes for the CG.
- (2) Identify Set 2 nodes
- (3) Remove all outgoing edges from `_ROOT_`.
- (4) Find possible candidates in Set 2 which can satisfy the demands of nodes in Set 1 using their demand frame. Put all such edges in the CG.
- (5) Convert the CG into IP problem.
- (6) Solve the IP equations to get the possible solution parse

.Note that input to the 2nd stage is the intermediate parse from the 1st stage. Figure 6, 7 show the



(a) Intermediate Parse tree (Output of Stage-I) (b) Set 1 nodes (shown circled) (c) Set 1 and Set 2, in circles
Figure 6. Algorithm for Stage 2 (steps 1-2)



(d) Remove arcs (e) New CG (f) Final Solution Graph (Stage-II output)

Figure 7. Algorithm for Stage 2 (steps 3-6)

above steps clearly for sentence (8)

- (8) *raama aura sitaa kala aaye /*
 'Ram' 'and' 'Sita' 'yesterday' 'came'
 Ram and Sita came yesterday.

Even if the second stage fails to give a parse, the parser still manages to give a partial parse (Figure 6 (a)). As noted earlier, this is made possible due to the presence of `_ROOT_`, which can potentially take all the unattached nodes as its children at the end of the 1st stage. This mechanism is part of the constraints system which we saw earlier.

5 Evaluation

Malt Parser (version 0.4) (henceforth, Malt), and MST Parser (version 0.4b) (henceforth, MST) have been tuned for Hindi by Bharati et al. (2008). Parsers were trained on a subset of a Hindi Treebank which contains around 1800 sentences with an average length of 19.85 words and has about 6585 unique tokens. The training set had 1185 sentences, development and test set had 268 and 220 sentences respectively. Elaborate experiments were done to tune the two data driven parsers and get the best results. We use the results obtained by them and compare the results of the proposed constraint based parser (CBP) under same conditions. CBP was tested on the 220 sentences⁶. Table 2 shows

the performance in terms of unlabeled attachments (UA), labeled (L) and labeled attachment (LA) accuracy (Nivre et al, 2007a). In Table 2, CBP shows the performance of the system when only the first parse is considered, while CBP'' shows it for the best parse out of the first 25 parses. MST gives the best results for UA while CBP'' outperforms MST and Malt for L and LA. Table 3 shows the results for some important labels. For most of the labels CBP'' outperforms the other two parsers.

6 Discussion

The performance of the proposed parser lands between MST and Malt. McDonald and Nivre (2007) have shown that MST is good at finding long distance dependencies while Malt on the other hand is better at short dependencies. We are able to do both reasonably well as is clear from the results in Table 3. Our system performs better for most of the short dependencies. For long distance relations like 'ccof' ('conjunct of' relation) we perform better than MST, while for 'main' ('main' identifies the root of the dependency tree) MST is better. The negative results for the system are due to: (a) *Small lexicon (verb frames)*: The total number of verb frames which the parser currently uses is very low. There are a total of around 300 frames, which have been divided into 20 verb classes (Begum et al., 2008). If the parser encounters a verb for which there are no frames, it takes some default action. In

⁶ The test set used to evaluate the parser was acquired from LTRC, IIT, Hyderabad, India. For details on the corpus type, annotation scheme, tagset, etc. see Begum et. al (2008a).

such a case it is possible for the parser to give an incorrect parse. As the coverage of this lexicon increases, the efficiency will automatically increase. (b) *Unhandled constructions*: The parser still doesn't handle some constructions, such as the case when a conjunct takes another conjunct as its dependent. This in fact, is responsible for the low recall for 'ccof'. Getting the head wrong in such cases (or not marking it at all) also affects the recall of 'main'. (c) *Ellipses*: Also not handled are ellipses where the mandatory arguments can be dropped. In such a case the parser may not be able to give the correct parse, and (d) *Correct parse after the first 25 parses*: For the results shown in this paper we do not consider parses beyond the first 25, there were instances when the correct parse was obtained, but was after the 25th parse.

The results show that the proposed parser does consistently well for the arguments of the verb, this is understandable since we are using a rule base. But we also do well for long distance relations too. In particular, the results for 'ccof' and 'main' are very good. In case of 'relc' the performance is relatively low. We found that although the parser was able to find these relations correctly, the parse was not always in the first 25. Table 3 shows that Malt and MST perform badly for relations like k3, k4⁷. The frequency of such relations in the Treebank is much lower than k1 and k2. In some contexts k1 and k2 have features (in the form of postpositions) which are similar to those found for k3, k4. Since k1 and k2 are much frequent than these relations, MST and Malt give selective preference to k1 and k2. They are unable to distinguish the specific context when k1, k2 take the features of k3, k4, etc. and tend to generalize.

There can be various reasons for the low performance of MST and Malt. They could be, (a) the training size, but it has been noted by Hall et al. (2007) that training size alone is not always a very good criterion to judge the low performance, (b) the type of tags being learnt, the tags in the treebank are syntactico-semantic and it has been observed that learning such tags is difficult (Nivre et al., 2007a), (c) Non-projectivity, the treebank has around 10% non-projective trees.

It is clear from the results that the proposed parser with its 2-stage resolution of demands, repair and partial parses is able to provide compre-

hensive coverage to major linguistic phenomenon with good performance. The overall scheme is simple, robust and efficient. Along with working towards improving the present coverage of CBP we plan to explore the possibility of building a hybrid system where we might be able to exploit the advantages of data-driven and grammar-driven parsers. Such an approach becomes essential when we also want to improve the recall values of the relations. The proposed parser performs consistently better in terms of recall as it can exploit the selective feature distribution necessary to identify the relations in a better way. The data driven parsers find it hard to learn such features automatically. A hybrid approach will in effect produce a system which combines the robustness of the data driven parsers with the capability of grammar driven parsers to capture explicit but complex language phenomenon.

7 Conclusion

This paper presents a comprehensive framework for dependency parsing of free word order languages. It has been applied to major phenomenon of Hindi. The parser works in two stages in which the first stage essentially parses clauses. The second stage identifies relations across clauses in particular for relative clause, and also handles conjuncts. The proposed framework is independent of language, however, rules (or demand frames) need to be prepared for the language. When compared with two data-driven languages it outperforms them in correctly identifying many core labels. Prioritizing the parses produced by the system, along with increasing its coverage are the most important tasks for the near future.

References

- R. Begum, S. Husain, A. Dhawaj, D. Sharma, L. Bai, and R. Sangal. 2008a. Dependency annotation scheme for Indian languages. In *Proceedings of IJCNLP-2008*.
- R. Begum, S. Husain, D. Sharma and L. Bai. 2008b. Developing Verb Frames in Hindi. In *the Proc. LREC 2008*.
- A. Bharati, S. Husain, B. Ambati, S. Jain, D. Sharma and R. Sangal. 2008. Two Semantic features make all the difference in Parsing accuracy. In *Proc. of ICON-2008*.

⁷ 'k4' is beneficiary, while 'k3' is instrument

- A. Bharati, R. Sangal and T. P. Reddy. 2002. A Constraint Based Parser Using Integer Programming, In *Proc. of ICON-2002*.
- A. Bharati and R. Sangal. 1993. Parsing Free Word Order Languages in the Paninian Framework. *Proc. of ACL:93*.
- A. Bharati, V. Chaitanya and R. Sangal. 1995. *Natural Language Processing: A Paninian Perspective*, Prentice-Hall of India, New Delhi. <http://ltrc.iiit.ac.in/downloads/nlpbook/nlp-panini.pdf>
- J. Carroll. 2000. Statistical parsing. In R. Dale, H. Moisl, and H. Somers, (eds), *Handbook of Natural Language Processing*, Marcel Dekker, pp. 525–543.
- R. Debusmann, D. Duchier and G. Kruijff. 2004. Extensible dependency grammar: A new methodology. *Proceedings of the Workshop on Recent Advances in Dependency Grammar*, pp. 78–85.
- J. Hall, J. Nilsson, J. Nivre, G. Eryigit, B. Megyesi, M. Nilsson and M. Saers. 2007. Single Malt or Blended? A Study in Multilingual Parser Optimization. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, 933–939
- R. Hudson. 1984. *Word Grammar*, Basil Blackwell, 108 Cowley Rd, Oxford, OX4 1JF, England.
- F. Karlsson, A. Voutilainen, J. Heikkilä and A. Anttila, (eds). 1995. *Constraint Grammar: A language-independent system for parsing unrestricted text*. Mouton de Gruyter.
- H. Maruyama. 1990. Structural disambiguation with constraint propagation. In *Proceedings of ACL:90*.
- R. McDonald and J. Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. In *Proc. of EMNLP-CoNLL*.
- R. McDonald, F. Pereira, K. Ribarov, and J. Hajic. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proc. of HLT/EMNLP*, pp. 523–530.
- I. A. Mel'cuk. 1988. *Dependency Syntax: Theory and Practice*, State University Press of New York.
- J. Nivre, J. Hall, S. Kubler, R. McDonald, J. Nilsson, S. Riedel and D. Yuret. 2007a. The CoNLL 2007 Shared Task on Dependency Parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*.
- J. Nivre, J. Hall, J. Nilsson, A. Chanev, G. Eryigit, S. Kübler, S. Marinov and E Marsi. 2007b. MaltParser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2), 95-135.
- J. Nivre. 2006. *Inductive Dependency Parsing*. Springer.
- J. Nivre, 2005. *Dependency Grammar and Dependency Parsing*. MSI report 05133. Växjö University: School of Mathematics and Systems Engineering.
- S. M. Shieber. 1985. Evidence against the context-freeness of natural language. In *Linguistics and Philosophy*, p. 8, 334–343.
- I. Schröder. 2002. *Natural Language Parsing with Graded Constraints*. PhD thesis, Hamburg University.
- P. Tapanainen, and T. Järvinen. 1997. A non-projective dependency parser. *Proceedings of the 5th Conference on Applied Natural Language Processing*, pp. 64–71.

				k1		k1s		k2		k3		k4			
	UA	L	LA		L	LA	L	LA	L	LA	L	LA	L	LA	
CBP	86.1	65	63	CBP"	P	74.9	74.4	71.5	71.5	54.0	53.7	66.6	66.6	28.5	28.5
				R	71.9	71.4	69.5	69.5	54.0	53.7	66.6	66.6	28.5	28.5	
CBP"	90.1	76.9	75	MST	P	77.2	75.2	54.5	54.5	55.9	52.3	29.6	29.6	0	0
					R	82.3	81.1	38.7	38.7	59.9	53.9	23.1	23.1	0	0
MST	87.8	72.3	70.4	Malt	P	77.6	76.4	66.6	66.6	56.7	51.6	33.3	33.3	0	0
					R	81	80	43.4	43.4	58.2	52.9	16.6	16.6	0	0

Table 2.

Table 3(a) (P: Precision, R: Recall, L: Labeled accuracy, LA: Labeled attachment accuracy)

		k7		r6		ccof		relc		nmod		vmod		main	
		L	LA	L	LA	L	LA	L	LA	L	LA	L	LA	L	LA
CBP"	P	75	71.1	85.5	84.2	98	98	66	66	41.6	33.3	81.1	77.7	91.3	91.3
	R	75	71.1	85.5	84.2	77	77	66	66	41.6	33.3	81.1	77.7	96.8	96.8
MST	P	69.7	64.3	86.6	85.9	85.3	80	90	80	25.0	22.5	82.5	79.7	98.9	98.9
	R	61.6	56.1	73.6	72.7	84.4	79.3	57.1	47.1	45.1	28.5	56.4	54.3	92	92
Malt	P	60.9	54.8	86.3	84.8	84.9	79.3	0	0	0	0	78.6	78.6	92.5	92.5
	R	53.1	47.8	74	72.7	83.5	78.1	0	0	0	0	53.9	53.9	92	92

Table 3(b) (P: Precision, R: Recall, L: Labeled accuracy, LA: Labeled attachment accuracy)