11-712: NLP Lab Report - Hindi Dependency Parsing

Swabha Swayamdipta

May 2nd, 2014

Abstract

This paper documents the design and implementation of a dependency parser for Hindi, and discusses its evaluation. We review the basics of the Hindi language, a dependency grammar model for the language and briefly outline some annotation guidelines. We create a small treebank following these guidelines and show that our parser obtains up to 78% parsing accuracy on the treebank thus created. We discuss some basic features that help the parser improve its performance.

1 Basic Information about Hindi

Hindi is an Indo-Aryan language, spoken widely in northern India. The native speakers of the language (and several of its dialects) are residents of the Indian states of Delhi, Uttar Pradesh, Haryana, Bihar, Rajasthan and Madhya Pradesh. In 2001, the number of official speakers of the language was estimated to be around 422 million [1], making it the fourth-most-widely spoken language in the world. It is one of the official languages of India, along with English.

The common vernacular of the North Indian and the Pakistani people is Hindustani - which is a mixture of Hindi and Urdu. The Hindi in Hindustani is derived from Sanskrit and the Urdu is derived from Persian, which became the dialect spoken by the Mughals during the Mughal reign in India in the 17th century.

In the late 19th century, there was a movement for standardizing the Hindi language, in an effort to make it closer to the Khariboli dialect spoken in Delhi and surrounding regions, than to Urdu. This involved standardization of the grammar and the orthography, resulting in the Modern Standard Hindi[2].

2 Past Work on the Syntax of Hindi

Syntactic parsing in Hindi had been in the shadows till recently, due to the lack of publicly available corpora. A great amount of effort towards Hindi dependency treebanking was made at the Machine Translation Research Lab of the International Institute of Information Technology, (IIIT) Hyderabad (Bharati et al. (2002)). However, most of the related publications still remain unavailable to free public access, and this was a major limiting factor in our investigation of the past work on the syntax of Hindi.

However, we were able to find some literature on dependency parsing tasks performed using the treebank developed above. According to Venkata et al. (2012), recently two CoNLL shared tasks (Buchholz and Marsi (2006); Nivre (2009)) and two ICON Tools Contests were organized by (Husain

[1] http://en.wikipedia.org/wiki/List_of_languages_by_number_of_native_speakers_in_India
[2] http://en.wikipedia.org/wiki/Hindi#History

(2009)) [3] and (Samar Husain and Gadde (2010)) [4] respectively, which aimed at exploring different approaches (rule-based, statistical and hybrid) for building dependency parsers for various Indian languages, Hindi being one of them. These tasks encouraged multiple groups to participate in these tasks and resulted in state-of-the-art parsers. Malt (Nivre et al. (2007)) and MST (McDonald et al. (2006)) parsers obtained the best performances in all these efforts. Venkata et al. (2012) go on to show that they could improve the results of systems that participated in these tasks by combining the results of MaltParser and MST.

In the latest efforts, Goutam (2012) explore the effect self-training and co-training on Hindi dependency parsing and report limited improvement. Ambati et al. (2013) use Combinatory Categorial Grammar (CCG) to improve Hindi dependency parsing using informative lexical categories.

Our approach differs from all the previous approaches in that we explicitly build a treebank from scratch. In our knowledge, we are the first time users of TurboParser, a higher-order dependency parser, developed at Carnegie Mellon University by Martins et al. (2010), for a dependency parsing task in Hindi.

3 Available Resources

In a recent effort on dependency annotation, a set of treebanks AnnCorra(TreeBanks for Indian Languages) have been developed by Bharati et al. (2002) at IIIT, Hyderabad. AnnCorra contains a fully labeled dependency corpus with 3300 sentences and 71389 tokens, complete with POS tags(simple and coarse), NP chunks and morphological information. Unfortunately, AnnCorra is not freely available for download. However, we were able to gain access to this corpora through researchers at IIIT Hyderabad. We used it as a reference to understand dependency grammar in Hindi and to extract features like POS tags. However, we did not use this data to either train or test our dependency parser.

The ninth workshop on statistical machine translation(WMT 2014) [5] had a shared translation task of Hindi-English, as a low-resource translation task. The training corpus containing about a million sentences was mostly crawled from the web and contains a lot of noise. However, the test corpus[6], which came from news corpora contained about 1130 sentences and 18500 tokens, is much cleaner. We have used this unannotated corpora for developing training as well as test data.

Other publicly available Hindi resources include the Hindi Wordnet[7], which could be exploited for feature extraction for dependency parsing.

4 Survey of Phenomena in Hindi

Hindi is written in the Devanagiri script. There are 10 vowels and 40 consonants in the Devanagiri script. Bars are used on top of the symbols. Hindi is written exactly as it is spoken - there are no silent alphabets, making it very phonetic(Shoebottom (2014)).

The word order in Hindi is is Subject-Object-Verb as opposed to Subject-Verb-Object in English. Hindi follows a relatively free word order, which makes dependency grammars more suitable for its representation than context-free grammars. The second important distinction between Hindi and English is the placement of the adposition. According to Begum et al. (2008), in Hindi the

[3] http://ltrc.iiit.ac.in/icon/2009/nlptools/
[4] http://ltrc.iiit.ac.in/icon/2010/nlptools/
[5] http://www.statmt.org/wmt14/translation-task.html
[6] http://ufallab.ms.mff.cuni.cz/ bojar/hindencorp/
[7] http://catalog.ldc.upenn.edu/LDC2008L02

preposition comes after the noun or pronoun it qualifies i.e., it is more correctly called a *post-position*(PSP). The noun or verb takes the oblique case based on the use of the postposition [8]. There are seven such one-word primary postpositions: *ka*(of), *ko*(to), *ne*, *se*(from, since, by, than), *mein*(in), *par*(on), *tak*(until). In addition to the simple postpositions, there could be compound postpositions, generated by combining the simple postpositions with adverbs and so on.

Nouns in Hindi are classified by two genders(male and female), two numbers(singular and plural) and three cases(direct, oblique and vocative)[9]. Pronouns in the second person are of three types (*tu, tum, aap*) - based on social formality - intimate, familiar, and polite, respectively. Intimate follows the rules of singularity whereas the other two follow plurality[10]. Questions in Hindi are tonal because it does not have the auxiliary "do" (Shoebottom (2014)).

Interestingly, English is highly prevalent in India and as a result, the Hindi vocabulary has imbibed a lot of English words. Often these English words are written in the Devanagiri script. In case of numerals, Hindi has a decimal numeral system. However most numbers from $1 - 99$ follow an irregular pattern, and need to be memorized separately [11]. The numbers, themselves, however are mostly written in English, instead of their Devanagiri counterparts.

4.1 Paninian Dependency Model

Hindi is a moderately morphologically rich language and follows a relatively free word order. In order to model the grammar of Hindi(and other Indian languages), a model called the Paninian model has been suggested by Begum et al. (2008). This model is a dependency grammar model based on "syntactico-semantic relations" called *Karaka*(meaning worker) relations. The Paninian model has previously been successfully applied to language by Pedersen et al. (2004) and Bharati and Sangal (1993).

The main task for annotating with the Paninian grammar model is to find *karaka* relations in the sentence. Sentences have primary modifiers(root nodes which connect to the wall), which are generally the main verbs in the sentence, or the conjuncts that connect the main verbs. The main verb in turn modifies other elements of the sentence which participate in the actions specificied by it. The participants in the actions are called *karaka*. These come from the inflectionally rich Sanskrit, emphasizing the role of post-positions and verbal inflections.

According to Begum et al. (2008), Karakas are of six basic types - *adhikarana* (location), *apaadaan* (source), *sampradaan*(recipient), *karana*(instrument), *karma*(theme) and *karta*(agent). Additional karakas include those for time, direction, similarity, purpose, etc. Noun phrase chunks can be used as units to represent the karakas.

Within each karaka chunk, the noun is the parent of postpositions(*vibhaktis*). The *vibhaktis* serve as connectors of the karakas with the main modifier in the sentence.

For example, in Fig 1, the sentence contains four karaka chunks. The first chunk, stands for "relatives and neighbors" and is an agent karaka. The second chunk denoting the time "after the incident", is the time karaka. The third chunk, denoting the location "outside the school" is a location karaka. The last chunk denoting the theme of the verb, i.e. "the protest" is a theme karaka. Each of these chunks are composed of nouns and postpositions(simple and compound). The head of each chunk connects to the main verb of the sentence.

[8] http://en.wikipedia.org/wiki/Hindustani_grammar#Postpositions
[9] http://en.wikipedia.org/wiki/Hindustani_grammar#Nouns
[10] http://en.wikipedia.org/wiki/Hindustani_grammar#Pronouns
[11] http://en.wikipedia.org/wiki/Hindustani_numerals

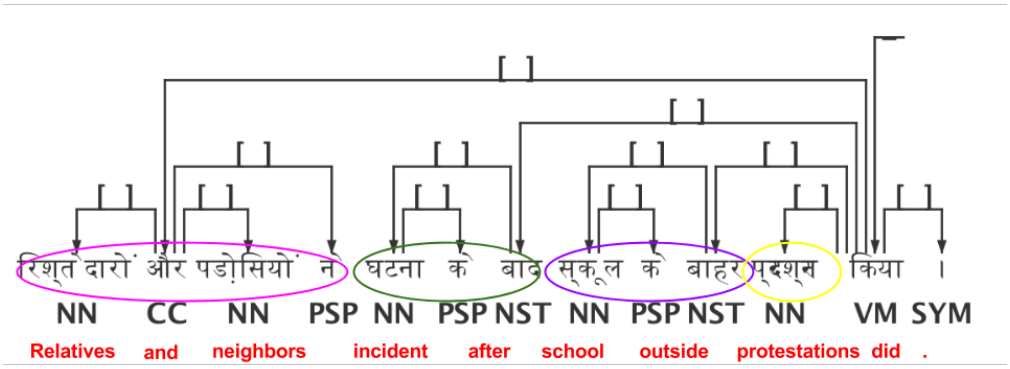

Figure 1: Karakas: Pink = Agent, Green = Time, Violet = Location, Yellow = Theme

5 Initial Design

We have annotated two test corpora A and B, each consisting of about 1000 tokens. Corpus A will be used like a development set, to test the performance of increasing amounts of training data, and to perform error analysis on the produced parses. Its performance will be continously monitored as we develop our system. Corpus B, also containing about 1000 tokens will be used as a test set. We use Corpus B to test performance when we have developed the semi-supervised dependency parser.

We propose to follow a semi-supervised approach for the dependency parsing task for Hindi. A completely supervised approach requires a considerably large amount of training data to perform reasonably well. On the other hand, unlabeled data (raw text) is often available for free in large quantities. Unlabeled data has shown promise in improving the performance of a number of tasks. However, a completely unsupervised dependency parser like Klein (2005) is complex to develop and does not perform as well as a supervised parser, no matter how much data it is trained on. A sweet trade-off between the two is a semi-supervised parser, which requires some, but not a great amount of training data.

We will be using TurboParser to perform our experiments. TurboParser is a state-of-the-art third-order non-projective dependency parser. TurboParser can be used under different settings - edge-factored only(Basic), second-order(Standard) and third-order(Full) modes.

As a baseline, we build a completely supervised dependency parser for Hindi. The only features used by this baseline are Part of Speech(POS) tags. To obtain the Part of Speech tags for our data, we built a POS tagger for Hindi (TurboTagger), trained on the AnnCorra treebank. On an informal examination, these were not very high quality. This is expected because there is little overlap between the AnnCorra and WMT corpora.

5.1 Annotation

Most of the annotation decisions were made in accordance with Begum et al. (2008). However, we made some key changes to the annotation decisions. These changes have been tabulated in Table 1.

A first important decision was to make all the annotations projective. We decided to do this because we were not sure if we shall be using a parser that allows non-projectivity. By the time we decided to use TurboParser, we had already annotated Corpora A and B, and hence continued with this decision.

5.1.1 NST within a *karaka* chunk

NSTs are nouns denoting spatial and temporal expressions. A common convention is to treat the NST as a postposition and treat it as a modifier of the main noun in the chunk. However, NSTs are semantically richer than postpositions. Hence, we made the decision to annotate these as the head of the chunk. An example can be seen in Fig 2.

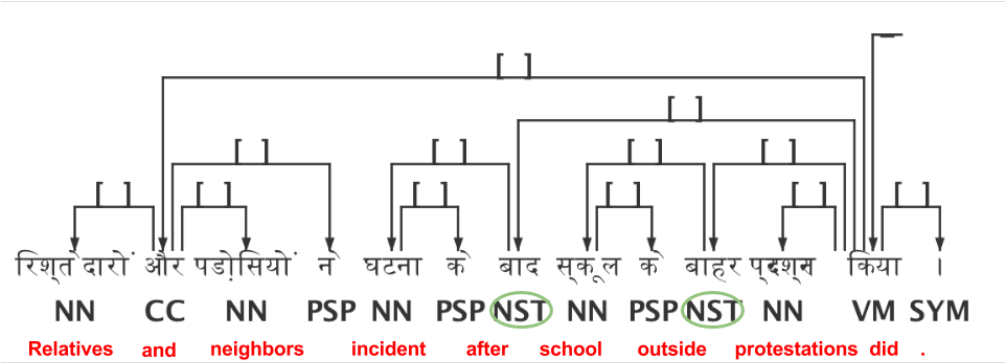

Figure 2: NST is the head of the chunk

5.1.2 Attachment of noun to main verb

Theme karakas are very often nouns (think of them as eventive nouns in English). Since these nouns are very closely related to the verbs immediately following them, one convention is to attach other karakas to this noun and then attach the noun to the main verb of the sentence. However, we have adopted the scheme of attaching all the karakas to the main verb, including the theme karaka. For example, in the sentence in Fig. 3, we have circled the POS tags of such an attachment. These attachments lead to flatter tree structures.

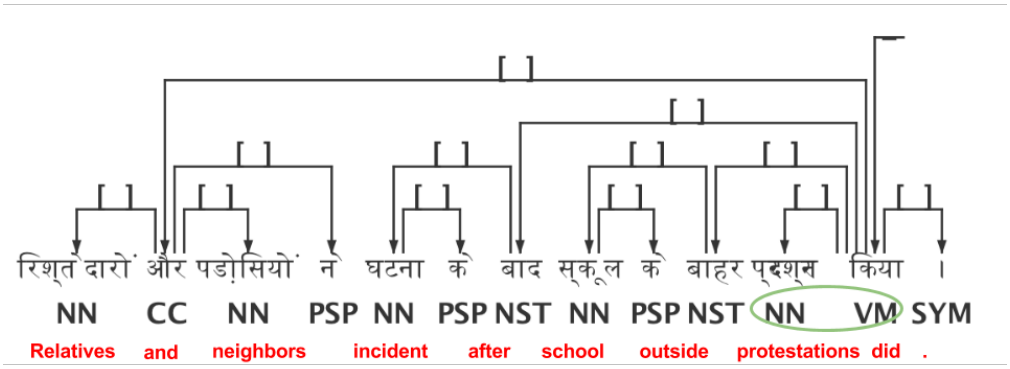

Figure 3: Noun at leaf level

5.1.3 Commas

Three different kinds of commas were encountered in the corpus. The first kind are those in coordinating and subordinating expressions. These have always been attached to the conjuncts(CC). The second kind are aposition commas, which immediately follow a noun, these are always attached to the noun they follow. The last kind of commas are sequence markers, which commonly follow verbs, these have been attached to the verb they follow.

5.1.4 Other Conventions

Two other conventions we followed that are worth mentioning are names and compound quantities. In the case of proper names, the first and middle names are the modifiers of the last name. For compound quantities, like "7 thousand 3 hundred", 7 is modifier of thousand and 3 modifies hundred, but thousand also modifies hundred. In other words, the head final quantity is the head of the expression.

	Modifier	**Head**
1	closest noun to the left of NST	NST
2	VM	NN
3	left and right parantheses	root of expression inside parantheses
4	comma in coordinating/subordinating expressions	conjucts (CC)
5	comma in aposition	noun to the left
6	comma in sequence markers	verb on the left
7	token to the left of a hyphen; hyphen	token to the right of hyphen
8	SYM at the end of sentence	head of sentence
9	first name (XC); middle name	last name
10	tokens indicating quantities	head final quantity

Table 1: Annotation Decisions

The tools used for annotation are: *Sanchay*[12] and Dependency Grammar Annotator(DGA) [13].

6 System Analysis on Corpus A

In this section, we discuss the performance of TurboParser on the annotated test set A, in a completely supervised setting, which serves as the baseline of our task. We follow a progressive system of evaluation as we increase the amount of annotated training data. The results are shown in Table 2. As expected, we get better results as the amount of training data increases. It is interesting that the simple arc-factored mode of TurboParser(Basic) does better when we have very little training data. This is probably a result of overfitting on the higher-order features in the other modes.

7 Lessons Learned and Revised Design

We saw that the parser did well on shorter sentences and made a lot of mistakes on longer sentences. To solve this problem, we tried to include more longer sentences in the training corpus. However, we also discarded unusually long sentences for simplicity of annotation.

[12] http://sanchay.co.in/

[13] http://phobos.ro/roric/DGA/dga.html

Training data size	Basic	Standard	Full
1000 tokens	*65.3668*%	64.9954%	64.4383%
2000 tokens	*67.5952*%	65.5525%	65.6453%
3000 tokens	68.1523%	69.1736%	*69.5450*%
4000 tokens	68.8951%	70.9378%	***71.1235***%

Table 2: Baseline Scores on Corpus A

Other possible problems are the presence of a large number of out of vocabulary words in the test set. Most of these words are English words(written in the Devanagiri script), and are commonly found in news corpora. There is no clear way to avoid this problem.

Next, we proceed to building a semi-supervised dependency parser. Koo et al. (2008) have suggested a simple and effective semisupervised method for training dependency parsers. Their key focus was on the problem of lexical representation, they made use of features that represent word clusters derived from a large unannotated corpus. They used the clusters produced by the Brown algorithm Brown et al. (1992). To obtain these clusters, we have used implementation of the Brown clustering algorithm by Liang (2005). This approach was shown to reduce the amount of training data by nearly half for the same performance on a given test set.

8 System Analysis on Corpus B

We proceeded with more annotation. After annotating about 2000 more tokens, we tested the parser on Corpus B, as well as the full test corpus(A+B), under both the supervised(-Brown) and semi-supervised(+Brown) setting. The results are shown in Table 3. We see that we could get the same accuracy using lesser data in the semi-supervised setting, which illustrates the usefulness of semi-supervised parsing.

Features	Test Corpus	Basic	Standard	Full
-Brown	Corpus B	73.8255%	74.1611%	75.5034%
+Brown	Corpus B	76.0906%	78.0201%	**78.1879**%
-Brown	Corpus A+B	71.7056%	72.8955%	74.394%
+Brown	Corpus A+B	75.6721%	76.3332%	**76.6858**%

Table 3: Semi-supervised learning scores

9 Final Revisions

In a final revision to our model, we added two extra features - coarse POS tags and lemmas, derived from a simple mapping from AnnCorra. The performance is shown in Table 4.

We observed that the performance improved with the coarse POS tags, which was as expected, but was hurt when we added lemmas. One possible reason could be that most words - with the exception of the most frequently occuring words - in our train and test corpora did not have lemma mappings, since they were not present in AnnCorra. It is also possible that the lemma features do not help for the most frequently occuring words. Another reason could be that the hyperparameters used in the learning of TurboParser need to be tuned after adding these kinds of features.

Since the annotations were done at different times through the span of a few months, we suspected that the quality of our annotations might get better with time. Note that corpora A and B

	Basic	Standard	Full
+Brown	75.6721%	76.3332%	76.6858%
+Coarse POS	76.5095%	**77.2146%**	**77.2146%**
+Lemma	76.5095%	77.0824%	76.9943%

Table 4: Coarse POS tag and Lemma features

were created in the very beginning and could have worse annotation decisions. To investigate, we performed a 5-fold cross validation experiment on the entire annotated corpus. The results, reported in Table 5, were not significantly higher than the best scores on the test corpora, indicating that the quality of annotations might not have a lot of variation since the beginning of the annotation phase. We dropped the lemma features here, as they slightly hurt performance.

Setting	Parser Accuracy
Basic	77.2178 %
Standard	77.9119%
Full	**77.9371**%

Table 5: Cross validation using all features

10 Future Work

In conclusion, we annotated a small-sized dataset of about 500 sentences, containing nearly 8300 tokens. We summarized grammatical phenomena in Hindi and developed guidelines for annotating the sentences for unlabeled dependencies. We developed a Turbo parser using simple and coarse part-of-speech features and Brown cluster representations, using our annotated data. We found that coarser representations, like coarse POS tags and Brown clusters help performance. We documented the performance of our parser under various settings in this report. Our data and our parser have been released on github[14] are now publicly available.

In future work, we aim to add more informative features, like morphology, to help the parser learn more subtle nuances of the language. An obvious, but time-consuming direction of improvement would be annotating even more data, maintaining consistency with the annotation decisions. Incorporating more finer-grained linguistic insight into annotation efforts might also pay off. Another approach would be to revisit all annotations to check for possible inconsistencies.

As discussed earlier, one of the major problems in annotation was the placement of modifier-modified relationships for the punctuation. Ongoing efforts on another task indicate that detecting tokens that should not be annotated in the first place actually improves parsing accuracy. These results encourage us to explore such possibilities in our parser.

References

Bharat Ram Ambati, Tejaswini Deoskar, and Mark Steedman. Using ccg categories to improve hindi dependency parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 604–609, Sofia, Bulgaria, August 2013. Association for Computational Linguistics.

[14]https://github.com/swabhs/nlplab

Rafiya Begum, Samar Husain, Arun Dhwaj, Dipti Misra Sharma, Lakshmi Bai, and Rajeev Sangal. Dependency annotation scheme for indian languages. In *IJCNLP*, pages 721–726, 2008.

Akshar Bharati and Rajeev Sangal. Parsing free word order languages in the paninian framework. In *Proceedings of the 31st Annual Meeting on Association for Computational Linguistics*, ACL '93, pages 105–111. Association for Computational Linguistics, 1993. doi: 10.3115/981574.981589.

Akshar Bharati, Rajeev Sangal, Vineet Chaitanya, Amba Kulkarni, Dipti Misra Sharma, and K. V. Ramakrishnamacharyulu. Anncorra: Building tree-banks in indian languages. In *Proceedings of the 3rd Workshop on Asian Language Resources and International Standardization - Volume 12*, COLING '02, pages 1–8, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

Peter F. Brown, Peter V. deSouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. Class-based n-gram models of natural language. *Comput. Linguist.*, 18(4):467–479, December 1992. ISSN 0891-2017.

Sabine Buchholz and Erwin Marsi. Conll-x shared task on multilingual dependency parsing. In *In Proc. of CoNLL*, pages 149–164, 2006.

Rahul Goutam. Exploring self-training and co-training for hindi dependency parsing using partial parses. In *IALP*, pages 37–40. IEEE Computer Society, 2012. ISBN 978-1-4673-6113-2.

Samar Husain. Dependency parsers for indian languages. In *In Proceedings of ICON09 NLP Tools Contest: Indian Language Dependency Parsing*.

Dan Klein. *The Unsupervised Learning of Natural Language Structure*. PhD thesis, Stanford, CA, USA, 2005. AAI3162386.

Terry Koo, Xavier Carreras, and Michael Collins. Simple semi-supervised dependency parsing. In *In Proc. ACL/HLT*, 2008.

Percy Liang. Semi-supervised learning for natural language. In *MASTERS THESIS, MIT*, 2005.

André F. T. Martins, Noah A. Smith, Eric P. Xing, Pedro M. Q. Aguiar, and Mário A. T. Figueiredo. Turbo parsers: Dependency parsing by approximate variational inference. In *EMNLP*, pages 34–44, 2010.

Ryan McDonald, Kevin Lerman, and Fernando Pereira. Multilingual dependency analysis with a two-stage discriminative parser. In *In Proceedings of the Conference on Computational Natural Language Learning (CONLL*, pages 216–220, 2006.

Joakim Nivre. Parsing indian languages with maltparser, 2009.

Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Glsen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95–135, 2007.

Mark Pedersen, Domenyk Eades, Samir K. Amin, and Lakshmi Prakash. Relative clauses in hindi and arabic: A paninian dependency grammar analysis. In Geert-Jan M. Kruijff and Denys Duchier, editors, *COLING 2004 Recent Advances in Dependency Grammar*, pages 9–16, Geneva, Switzerland, August 28 2004. COLING.

Bharat Ram Ambati Samar Husain, Prashanth Mannem and Phani Gadde. The icon-2010 tools contest on indian language dependency parsing. In *In Proceedings of ICON-2010 Tools Contest on Indian Language Dependency Parsing*.

Paul Shoebottom. Language differences: English - hindi, May 2014. URL `http://esl.fis.edu/grammar/langdiff/hindi.htm`.

B. Venkata, Seshu Kumari, and Ramisetty Rajeswara Rao. Hindi dependency parsing using a combined model of malt and mst, 2012.

Wikipedia. Hindustani grammar, May 2014. URL `http://en.wikipedia.org/wiki/Hindustani\_grammar`.