Applications of Data-Driven Dependency Rules

A thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science
in
Computer Science and Engineering by Research

by

VARUN KUCHIBHOTLA
200702052
varun.k@research.iiit.ac.in

International Institute of Information Technology Hyderabad
500 032, India

July 2024

To my family for their perpetual support.

Acknowledgments

I'm very grateful to my advisor Prof. Dipti Misra Sharma and Siva Reddy, whose guidance was instrumental at all stages of this thesis. Thanks to Anil, my parter in crime and the primary author of our research papers. Want to take this moment to thank my immediate and extended family whose support and encouragement played a huge part in me not losing hope. Special thanks to Dr. P. Kumaraguru for his efforts towards completion of the course for long-pending dual degree students.

International Institute of Information Technology Hyderabad
Hyderabad, India

CERTIFICATE

This is to certify that work presented in this thesis proposal titled ***Applications of Data-Driven Dependency Rules*** by *Varun Kuchibhotla* has been carried out under my supervision and is not submitted elsewhere for a degree.

Date

Advisor: Prof. Dipti Misra Sharma

Abstract

In this paper, we present an approach to integrate unlexicalised grammatical features into Malt dependency parser. Malt parser is a lexicalised parser, and like every lexicalised parser, it is prone to data sparseness. We aim to address this problem by providing features from an unlexicalised parser. Contrary to lexicalised parsers, unlexicalised parsers are known for their robustness. We build a simple unlexicalised grammatical parser with POS tag sequences as grammar rules. We use the features from the grammatical parser as additional features to Malt. We achieved improvements of about 0.17-0.30 percent for UAS on both English and Hindi state-of-the-art Malt results.

Word sketches are one-page automatic, corpus- based summaries of a word's grammatical and collocational behaviour. These are widely used for studying a language and in lexicography. Sketch Engine is a leading corpus tool which takes as input a corpus and generates word sketches for the words of that language. It also generates a thesaurus and 'sketch differences', which specify similarities and differences between near-synonyms. In this paper, we present the functionalities of Sketch Engine for Hindi. We collected HindiWaC, a web crawled corpus for Hindi with 240 million words. We lemmatized, POS tagged the corpus and then loaded it into Sketch Engine.

Contents

List of Figures

List of Tables

Related Publications

1. Anil Krishna Eragani, Varun Kuchibhotla: **Improving malt dependency parser using a simple grammar-driven unlexicalised dependency parser.** ALP 2014: 211-214

2. Anil Krishna Eragani, Varun Kuchibhotla, Dipti Misra Sharma, Siva Reddy, Adam Kilgarriff: **Hindi Word Sketches.** ICON 2014: 328-335

Chapter *1*

Introduction to Key Terms

Parsing is a fundamental task in natural language processing (NLP) that involves analyzing the grammatical structure of sentences. It aims to assign a hierarchical structure to sentences that reflects their syntactic relationships, enabling deeper understanding and facilitating subsequent NLP tasks such as semantic analysis, machine translation, and information extraction.

1.1 Phrase Structure Parsing

Phrase structure parsing, also known as constituency parsing, focuses on decomposing sentences into nested phrases according to a formal grammar. The grammar typically consists of rules that define how smaller units (like words and phrases) can be combined to form larger units (like sentences). One of the most widely used formalisms for phrase structure parsing is context-free grammar (CFG). CFGs define syntactic rules using symbols (non-terminal and terminal) and production rules. For example, a simple rule might state that a sentence (S) consists of a noun phrase (NP) followed by a verb phrase (VP).

Phrase structure parsing typically employs algorithms such as top-down recursive descent parsing, bottom-up shift-reduce parsing, and chart parsing. These algorithms traverse or build parse trees according to the grammar rules, attempting to match the input sentence against possible parse structures. The output is a parse tree where nodes represent constituents (phrases) and edges denote syntactic relationships (e.g., subject-object relationships).

1.2 Dependency Parsing

Dependency parsing, on the other hand, focuses on identifying relationships between words in a sentence based on directed links (dependencies) rather than hierarchical structures. A dependency parser aims to generate a dependency tree where each word (except for a root node) is directly linked to another word, indicating a syntactic relationship such as subject, object, modifier, etc. Dependency parsing is particularly useful for languages with relatively free word order or for applications requiring efficient syntactic analysis.

Dependency parsing algorithms can be broadly classified into transition-based and graph-based approaches. Transition-based parsers operate by applying a sequence of actions (transitions) to construct the dependency tree incrementally. These parsers are efficient and suited for online processing but require careful design of transition rules. In contrast, graph-based parsers formulate parsing as an optimization problem, where the goal is to find the best dependency tree based on learned weights and features. The Maximum Spanning Tree (MST) algorithm is a prominent example of a graph-based approach used in dependency parsing.

1.3 Focus of our Work

At the end of the day, both the methods above return a tree. i.e., the following properties are upheld:

- Every node has a single parent
- All the nodes are connected
- There are no cycles present in the final structure

Owing to the tree structure where every node is a word and every edge is a dependency relation, this parse very closely aligns with the semantic meaning of the sentence. One issue here is that the tree is basically rooted at the *finite verb*, and looking at it does not make the source sentence obvious - especially for free-word-order languages.

All our work is focused on the dependency parsers, which will be outlined in the sections to come.

1.3.1 Grammar Driven Parsing

Grammar-driven parsing is a fundamental approach in natural language processing (NLP) that involves analyzing sentences based on predefined grammatical rules and structures. At its core, this method relies on formal grammatical frameworks like context-free grammar (CFG), dependency grammar, or phrase structure grammar. These frameworks define syntactic rules that dictate how words and phrases can be combined to form grammatically correct sentences. During parsing, these rules are applied to analyze the syntactic structure of sentences, aiming to generate hierarchical representations such as parse trees or dependency trees.

One of the key advantages of grammar-driven parsing is its ability to enforce linguistic constraints and ensure that the parsed output adheres to grammatical rules. This systematic approach provides a clear and structured representation of the relationships between words and phrases within sentences, which is essential for many downstream NLP tasks such as machine translation, information retrieval, and semantic analysis. By relying on formal grammatical theories, grammar-driven parsers can handle a wide range of syntactic phenomena and linguistic variations across different languages.

However, grammar-driven parsing also faces challenges, particularly in languages with flexible word order or complex syntactic structures that may not be fully captured by a single formal grammar. As a result, hybrid approaches that combine grammar-driven techniques with statistical or machine learning methods have emerged to address these challenges effectively. These hybrid models integrate the strengths of both approaches, leveraging grammatical rules for structural integrity while incorporating statistical patterns to enhance parsing accuracy and adaptability to diverse linguistic contexts.

Chapter *2*

Journey and Motivation

We had started off comparing two of the most popular dependency parsers at the time, Malt and MST.

MaltParser is a versatile and efficient dependency parser that excels in syntactic analysis tasks within natural language processing (NLP). Developed by Johan Hall, Jens Nilsson, and Joakim Nivre, MaltParser employs a transition-based parsing approach to construct dependency trees from raw text. Unlike traditional graph-based parsers, MaltParser utilizes a deterministic parsing algorithm that iteratively applies a set of parsing actions based on predefined transition rules. These rules guide the parser through the decision-making process to systematically build the syntactic structure of sentences. This method ensures both accuracy and computational efficiency, making MaltParser a popular choice for parsing tasks ranging from part-of-speech tagging to machine translation.

The Maximum Spanning Tree (MST) parser is a dependency parser that operates on the principle of finding the maximum spanning tree over a graph representation of a sentence. Developed primarily by Ryan McDonald and Fernando Pereira, the MST parser formulates the parsing task as an optimization problem where the goal is to select a tree structure that maximizes a scoring function based on learned weights and features. This approach contrasts with transition-based parsers by directly constructing a global structure rather than incrementally building it through transitions.

On experimenting with 2 data sets with different relations and POS tag counts, we found that Malt parser is very good for short distance dependencies, whereas the MST parser is good for handling long distance dependencies (>9 words between parent and child). One other finding is that MST outperformed Malt for JJP, VG, POF tags, whereas Malt performs better on nouns and pronouns.

We then tried feeding the intermediate output of Malt parser into MST and vice-versa. The majority of the errors were not in head-identification, but in labeling the relation, so we tried to build a custom tree by combining the outputs of both the parsers based on sentence lenghts. This lead to minor improvements, but there was a lot of manual intervention needed, and we could not come up with a generic checklist of items to automate it.

So we started to think about how generic rules can be used to nudge lexical parsers in the right direction when they encounter unseen patterns in the input sentence, or parent/child pairs separated by long distances.

Chapter *3*

Auto-extracted Grammar

We primarily worked on building a data driven grammar, like sketch grammar. We tried building our own parser using this grammar, but after development, we found that the accuracies were quite low, so we started to use the grammar in other ways like improving the performance of Malt by feeding features obtained from this during training or building the Sketches for Hindi. These are detailed in the further sections.

The grammar itself is a collection of regular expressions over part-of-speech tags, written in Corpus Query Language (CQL). Example:

k1 (doer): 2:[tag="NN"] [tag="PSP:ने"] [tag="JJ"]? 1:[tag="VM"]

where 1 is head, 2 is child.

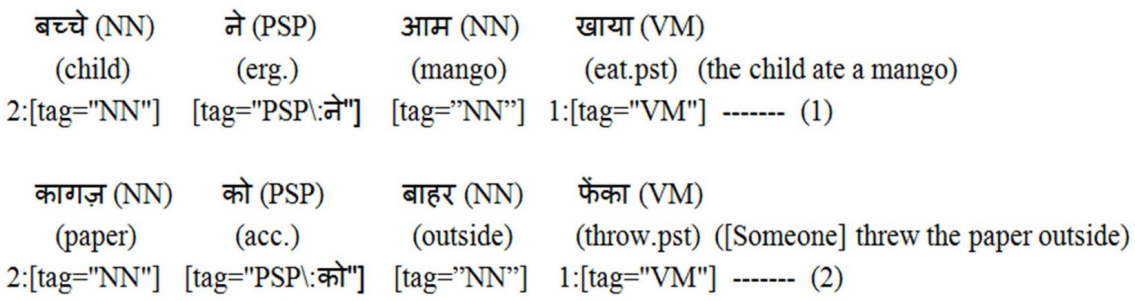
Writing any grammar with a high coverage is a tough task due to the idiosyncracies of languages or the word order they enforce. So following a general approach of what MST parser does, we wanted to automatically extract a grammar purley based on POS-tag sequences from an annotated corpus (HDT-v0.5). Based on the gold data, we extract a sequence of POS tags starting and ending with the parent/child maintaining the order in which they appear in the sentence. We do this for all dependency relations (parent-child pair and deprel combination) for all sentences. We end up with a bunch of rules like so:

- k1 - 2:[tag="NNP"] [tag="PSP"] [tag="NN"] [tag="PSP"] 1:[tag="VM"]
- k2 - 2:[tag="NN"] 1:[tag="VM"]
- k7 - 2:[tag="NN"] [tag="PSP"] [tag="NN"] 1:[tag="VM"]

One contrast here when compared to MST's non lexicalized rules is that MST maintains context of what comes before and after the immediate context (elements not just in between the parent and child).

3.1 Assigning Weights

Since we are extracting so many rules, when these are applied to a test sentence, they would definitely result in a slightly dense graph of various parent-child dependency relationships. So



we need a method to pick one over the other, so we apply back these rules on the annotated data and compute

- how many times the sequence itself was found in the data - here, we only care about the parent-child connection and not the dependency relation (label of the edge). Call this N.
- how many times the sequence was found in the data with the correct dependency relation. Call this M.

We then compute a precision of each of these rules based on the source data they were extracted from, as M/N.

During the time, large-scale annotated data was not available, so some of the more complex rules did not result in many other matches than the sample sentence they were extraced from. To rule out such outliers, we put in some basic constraints to include the rules in further steps.

- it should have a precision of at least 0.05.
- total matches (N) should be > 4
- max rule length is 7 tokens

3.2 Lexicalization and Generalization

Hindi, being a free word order language, with the subject/object being omitted in colloquial speech, presents us with one challenge. Both k1 (doer) and k2 (object) have very similar patterns in which they occur

As can be seen above, two very similar sentence structures give rise to different relations. We found the only way to disambiguate PSPs (postpositions) and CCOFs (conjunctions) was to incorporate some lexical element to those tags. As evident in the image above, using the lexical items ने and कोhelps with this issue.

One other issue stemming from the small annotated data sets is very low frequencies for longer rules (where the parent and child are separated by 6-7 words). These relations were

automatically ignored by the constraints in the previous section. So we decided to include regular expression patterns in the rules under reasonable conditions. This results in rules like

2:[tag="NNP"] [ ]6,10 1:[tag="VM"]

We then remove all the rules which are covered by this generic rule, and compute the precision for this one in the source data.

3.3 Parsing Attempt

Using the above grammar rules, we tried our hand at parsing. The general approach is to apply all the rules on the input sentence, and resolve the resultant graph into a tree using some constraints.

- a node cannot have more than 1 parent.
- there cannot be any cycles in the result. It should be a proper tree.
- a parent cannot have two children with the same dependency relation.
  - if the relation is a conjunction, there can be multiple children as long as all the children are of the same type (all nouns; or all verbs)
  - in any other case, a parent can only have 1 child of a type

Before lexicalising the PSP and CCOF tags, we achieved a meager 30% labeled and unlabeled accuracy on test data. But after including the lexical features, it went up to around 70%. The main challenge is still the relatively small sample from which these rules were extracted.

The accuracies we got were below the state-of-the art parsers at that time, which were averaging at around mid-80%. We thought the generic nature of these would lend themselves as prompts to a lexical parser to better pick its options. These efforts are detailed in the upcoming chapters.

Chapter 4

Improving Malt Dependency Parser

4.1 Introduction

Dependency Parsing has gained popularity due to its ease of representation for any language including free word order languages. With shared tasks like CoNLL [1, 2] and others [3], dependency treebanks for many languages came into existence, and thereby many dependency parsing techniques. Among these, Malt parser [4] has become one of the popular dependency parsers due to its ease of training, capability of handling various features, faster training and parsing rates. We use Malt Parser in place of other higher-order graph-based parsers as we want to improve greedy dependency parsing, which is still one of the most efficient parsing methods available. Malt is a transition-based parser with greedy local search. Malt uses lexicalised information i.e., the word itself, along with the features specified, to take a best local decision while parsing. While lexicalised information is useful, the algorithm is prone to difficulties when the words are unseen since the algorithm has to base its decision only on the features seen. The features, in general, are unigram, bigram or trigram of various linguistic properties around a fixed window from the current word.

There have been many attempts of improving a parser's performance using features from other parsers. [5] used features from [6] and [7] parsers which improved MSTParser's UAS by 1.7%. [8] and [9] used a technique called blending where output of different parsers is used to generate the final parse. [10] improved CCG parser using dependency features. They first extracted n-best parsers for a sentence using a CCG parser. Then dependency features from a dependency parser are provided to a re-ranker.

Unlexicalised grammar-driven parsers [11,12] are well known for their robustness in handling unseen words, and are also competitive in performance with statistical data driven lexicalised parsers. We aim to integrate information from a simple grammar-driven unlexicalised parser into Malt, and thereby increase the robustness of Malt.

We define our unlexicalised grammar as a set of rules corresponding to each dependency relation type. Each grammatical rule is a sequence of POS tags between (and including) the head and child words. We extract these rules from the training data. To parse test data, we

index	word	POS tag	1st best relation	2nd best	3rd best
1	The	DT	4:det:0.89159	3:det:0.870516	2:det:0.846192
2	luxury	NN	3:nn:0.734832	4:nn:0.715789	4:amod:0.1166
3	auto	NN	4:nn:0.734832	6:nn:0.139535	
4	maker	NN	7:nsubj:0.273	6:nn:0.169154	6:amod:0.1293
5	last	JJ	6:amod:0.7330	4:amod:0.2280	
6	year	NN	7:nsubj:0.567	4:tmod:0.1268	
7	sold	VBD	0:ROOT:0.453		
8	1,214	CD	9:num:0.96036	7:dobj:0.185424	
9	cars	NNS	7:dobj:0.6859		
10	in	IN	9:prep:0.6236	7:prep:0.479452	
11	the	DT	12:det:0.2726		
12	U.S.	NNP	10:pobj:0.228		

Figure 4.1 Top 3 parents for each word. The format of each relation above is parent node index : relation name : precision of the rule applied. For example, the 2nd best relation on word index 3 i.e. *auto* is *6:nn:0.139535* indicating that word index 6 is the parent with dependency relation *nn* with confidence 0.139535 (i.e. precision of rule **nn** *2:NN NN JJ 1:NN*. Note: These are not the features themselves. Features are constructed from these relations, e.g. feature **n-Rels** of word index 4 (i.e. maker) are 1-nsubj, 2-nn, 3-amod

apply these rules on the test data, and assign the n-best dependency relations based on the rules' probabilities. These n-best relations are given to Malt as features.

In our experiments with English and Hindi on lexicalised data, we achieved state-of-the-art results over the previous best known settings for Malt.

4.2 Simple Grammar-Driven Parser

The goal of our grammar-driven parser is to identify all possible dependency relations of a word in a given sentence. We extract rules for each dependency relation type from the training data. Each rule is defined as a sequence of POS tags surrounding the head and child words. We describe these steps in detail below.

4.2.1 Dependency Grammar

Our dependency grammar differs from the commonly used grammars like CFGs and demand frames. Inspired from RASP [13, 14], we define a grammar rule as a POS tag sequence, also called as Sketch Grammars [15, 16]. Sketch Grammars are popular with lexicographic and corpus linguistics community, and are used to identify collocations of a word with a given grammatical relation [17]. We use Sketch Grammar to identify words in syntactic relations in a given sentence. For example, a grammar rule for the relation "direct object" (**dobj**) is *1:"VBD" "DT"? "JJ"* 2:NN*, which specifies that if a verb is followed by an optional determiner and any number of adjectives and followed by a noun, then the noun is the object of the verb. The

head and child are identified by 1: and 2:. Each rule may often be matched by more than one relation creating ambiguity. Yet they tend to capture the most common behaviour in the language.

Given a training data with dependency annotations, we extract dependency grammar rules (i.e. sketch grammar) automatically for each relation type, based on the POS tags appearing in between the dependent words. For example, from the sentence, *Rowling wrote the book on Potter*, we extract rules of type,

- **nsubj** - *2:"NNP" 1:"VBD"*
- **dobj** - *1:"VBD" "DT" 2:"NN"*
- **prep** - *1:"VBD" "DT" "NN" 2:"PP"*

In the above example, relation names are in bold, and the rules are italicised. Though sketch grammars support regular expressions, generating a compact rule with regular expressions is our future work, and in this paper we work only with plain tag sequences (our initial attempts of using regular expressions bombarded the search space, and so we reverted to simple grammatical rules). We only use the POS tags of words between head and child words including head and child POS tags (In the camera ready we hope to extend the context to the left and right of the target words).

Since Hindi follows the SOV (Subject-Object-Verb) word order, some inter-chunk relations can only be represented by long, and hence, less frequent and less statistically significant rules. So, we generalised the rules for inter-chunk relations by only including the chunk head (the parent in each local word group).

4.2.2 Weighted Dependency Grammar

After extracting all the dependency rules, we apply each rule on training data, and compute its precision. For example, if the rule **dobj:** *1:"VBD" "DT" 2:"NN"* is applied on training data, we get all the (1:VBD, 2:NN) pairs where the rule holds, say N pairs. Out of these N pairs, if M of them are seen correctly with *dobj* relation in the training data, then precision of the rule is $\frac{M}{N}$. We use precision scores as weights for disambiguation when rules are applied on test data.

For Hindi, whenever possible, we take the average of the weights from both the type of rules (inter-chunk and plain tag sequences)

4.2.3 Grammar-Driven Parser

For each sentence in the test data, we apply all the rules extracted in the above steps. If we define a sentence as a set of nodes in a graph, each rule is like an edge connecting two nodes in the graph, with relation name as the edge label. For each child node (i.e. indicated by 2:

Language	*Experiment/FEATS*	*LAS*	*UAS*
English	Stanford Baseline (BL)	87.71	90.17
	Stanford BL + {5-posTags, 5-ScoreRanges}	**87.90****	**90.34****
	† Stanford Baseline (BL)	79.45	83.39
	† Stanford BL + {3-posTags, 3-ScoreRanges}	**79.90****	**83.85****
	CoNLL Baseline (BL)	88.73	90.00
	CoNLL BL + {2-Rels, 2-posTags}	**88.88***	**90.20****
	† CoNLL Baseline (BL)	83.31	85.68
	† CoNLL BL + {4-Rels, 4-posTags, 4-ScoreRanges}	**84.01****	**86.29****
Hindi	Baseline (BL)	87.97	93.40
	BL + {5-Rels, 5-posTags}	**88.25****	**93.70****
	† Baseline (BL)	83.33	91.79
	† BL + {3-Rels, 3-posTags}	**84.16****	**92.65****

Table 4.1 Impact of new feature set on Malt baseline. McNemar's test, ** = $p < 0.01$, * = $p < 0.05$
† - Unlexicalised

in an applied rule), we select its n-best parent nodes (i.e. indicated by 1: in an applied rule) by sorting the rules based on their weights (in descending order). In Figure 4.1, for each word, we show its 3 best parent nodes. One can apply algorithms like MST (or ILP) to retrieve a dependency tree from the graph [18]. Since we aim to use n-best relations of each child node as features to Malt, we do not perform MST or ILP. Though the parser is noisy, we hope that the features from grammar could still help Malt.

4.3 Integrating Grammatical Features with Malt

As mentioned in section 4.2, we use the intermediate output of our parser, namely, the n-best relations for each word, to generate features which we integrate with Malt. For each word in the sentence, we specify its possible parents, relations and the scores derived from our parser. As far as we know, the only way to encode the predicted head of a word in a feature is through InputArc and InputArcDir constructors, but we chose not to use them as they only work on 1-best predictions. Instead, we specify the features of parent word as parent indicating features to the child word. Similarly, it is not possible to specify scores from our parser as numerical features to Malt. Instead we convert score to a range based on the magnitude of the score. Our feature set from the grammar-driven parser is as follows.

1. **n-Rels:** Relations between the current word and the nth-best parents specified by the grammar-driven parser.

2. **n-posTags:** POS tags of the n-best parents.

3. **n-ScoreRanges:** The range in which score of the n-best parents falls in. We define the ranges as follows. 1 for 90-100%, 2 for 80-90% and {3,4,5,6} for 80-0% in splits of 20.

If n is 3, then we have one feature each for the 1st, 2nd and 3rd best feature value.

In our experiments, we add the above features from grammar-driven parser to the existing state-of-the-art Malt feature set (refer next section) to form a new feature set, and tune over the complete feature set using development data to get the best settings.

We also experiment with Malt by disabling the lexical features. Since our grammar rules are purely syntactic, this would serve as a control experiment indicating whether we better the non-lexical part of Malt Parser with the aforementioned features.

4.4 Experiments

In this section we describe the datasets we used for English and Hindi and our experimental settings.

4.4.1 Data and Tools

For English, we experimented with two different dependency annotation schemes, namely, CoNLL which has a label set of 11 labels, and Stanford which has a much larger label set of 48 labels, both of which are widely popular. We used Penn Treebank [19] with standard splits - sections 02-21 for training, section 22 for development and section 23 for testing. We extracted Stanford dependencies and CoNLL dependencies from Penn Treebank using Stanford Parser built-in converter with basic projective option and Penn2Malt[1] respectively.

For Hindi, we used the Hindi Dependency Treebank (HDT-v0.5) which was released for the Coling2012 shared task on dependency parsing [3]. This treebank contains 12,041 training, 1,233 development and 1,828 testing sentences with an average of 22 words per sentence.

We use predicted POS tags for English, and the gold POS tags for Hindi.

4.4.2 Final Feature Set and Malt Settings

We use state-of-the-art English and Hindi feature sets of Malt as our baselines [3, 20]. The state-of-the-art Malt features for English are based on word and POS. For Hindi, apart from word, lemma and POS, the features also include morphological properties such as case markers for nouns; tense, aspect and modality markers for verbs; and chunk properties such as chunk label and chunk type (indicates if the word is head or non-head of the chunk).

In our models, in addition to the baseline feature set, we also include the features from our grammar-driven parser as described in Section 4.3. These features are provided in the FEATS

[1] `http://w3.msi.vxu.se/nivre/research/Penn2Malt.html`

Language	*Experiment*	*LAS*				*UAS*		
		1-5	*6-10*	*>10*		*1-5*	*6-10*	*>10*
English	Stanford Baseline (BL)	89.76	70.69	72.17		91.89	73.54	74.17
	Stanford BL+BestFeat	89.88	70.46	73.52		91.98	73.22	75.72
	CoNLL Baseline (BL)	90.61	72.64	73.18		91.86	73.34	73.43
	CoNLL BL+BestFeat	90.69	73.00	73.82		91.98	73.70	74.17
Hindi	Baseline (BL)	91.33	73.59	71.42		96.21	82.82	78.14
	BL+BestFeat	91.57	73.86	72.10		96.44	83.24	78.99

Table 4.2 Distance wise accuracies with our improved feature set
1-5,6-10 and >10 represent the parent-child disntance

column of CONLL format[2]. We tried various combinations of features: baseline features + combinations of n-Rels, n-posTags, n-ScoreRanges.

All the experiments are carried out using Malt Parser version 1.7.2[3]. We use *nivrestandard* as the parsing algorithm and *liblinear* as the learner in our experiments. The rest of the settings are set to default.

4.5 Results

Table 4.1 displays the results of our experiments with English and Hindi on test data. We chose the best performing feature set based on the results on development data. As mentioned before, we also ran Malt by disabling lexical features - these results are marked by †. All the improvements on this data are statistically significant (McNemar's significance: $p < 0.01$).

On lexicalised data for English, we obtained an improvement of 0.19% LAS and 0.17% UAS on Stanford dependencies, and 0.15% LAS and 0.20% UAS on CoNLL scheme. For Hindi, we achieved an improvement of 0.28% LAS and 0.30% UAS. All these improvements are statistically significant ($p < 0.01$ for all except CoNLL-LAS with $p < 0.05$ using McNemar's test).

In Table 4.2, we also present the distance wise improvements of dependency relations. We observed improvements on all distance ranges (except 6-10 on Stanford), which shows that our grammar is effective even for long distance relations.

[2] `http://nextens.uvt.nl/depparse-wiki/DataFormat`
[3] Malt Parser v1.7.2 can be downloaded at `http://www.maltparser.org/download.html`

Chapter *5*

Hindi Word Sketches

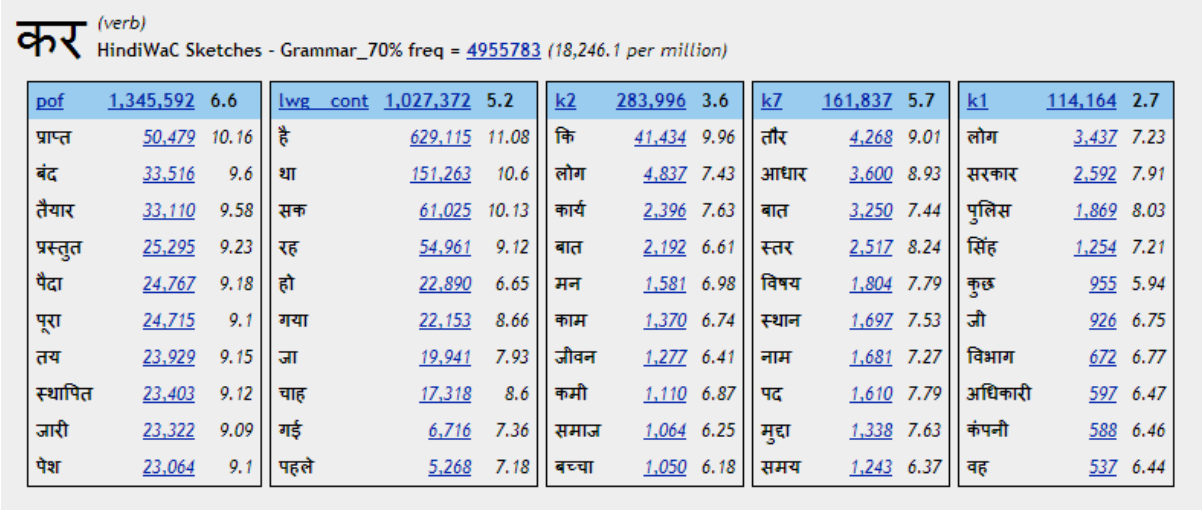
5.1 Introduction

A language corpus is simply a collection of texts, so-called when it is used for language research. Corpora can be used for all sorts of purposes: from literature to language learning; from discourse analysis to grammar to language change to sociolinguistic or regional variation; from translation to technology.

Corpora are becoming more and more important, because of computers. On a computer, a corpus can be searched and explored in all sorts of ways. Of course that requires the right app. One leading app for corpus querying is the Sketch Engine (Kilgarriff et al., 2004). The Sketch Engine has been in daily use for writing dictionary entries since 2004, first at Oxford University Press, more recently at Cambridge University Press, Collins, Macmillan, and in National Language Institutes for Czech, Dutch, Estonian, Irish, Slovak and Slovene. It is also in use for all the other purposes listed above. On logging in to the Sketch Engine, the user can explore corpora for sixty languages. In many cases the corpora are the largest and best available for the language. For Indian languages, there are the corpora for Bengali, Gujarati, Hindi, Malayalam, Tamil and Telugu. The largest is for Hindi with 240 million words – we would be referring to it as HindiWaC in the rest of the paper.

The function that gives the Sketch Engine its name is the '**word sketch**', a one-page, automatically-derived summary of a word's grammatical and collocational behaviour, as in Figure 5.1. Since the images in this paper are screen shots taken from Sketch Engine, translations and gloss have not been provided for the Hindi words in the images.

In this paper we first introduce the main functions of the Sketch Engine, with Hindi examples. We then describe how we built and processed HindiWaC, and set it up in the Sketch Engine.



कर (verb)
HindiWaC Sketches - Grammar_70% freq = 4955783 (18,246.1 per million)

pof	1,345,592	6.6	lwg__cont	1,027,372	5.2	k2	283,996	3.6	k7	161,837	5.7	k1	114,164	2.7
प्राप्त	50,479	10.16	है	629,115	11.08	कि	41,434	9.96	तौर	4,268	9.01	लोग	3,437	7.23
बंद	33,516	9.6	था	151,263	10.6	लोग	4,837	7.43	आधार	3,600	8.93	सरकार	2,592	7.91
तैयार	33,110	9.58	सक	61,025	10.13	कार्य	2,396	7.63	बात	3,250	7.44	पुलिस	1,869	8.03
प्रस्तुत	25,295	9.23	रह	54,961	9.12	बात	2,192	6.61	स्तर	2,517	8.24	सिंह	1,254	7.21
पैदा	24,767	9.18	हो	22,890	6.65	मन	1,581	6.98	विषय	1,804	7.79	कुछ	955	5.94
पूरा	24,715	9.1	गया	22,153	8.66	काम	1,370	6.74	स्थान	1,697	7.53	जो	926	6.75
तय	23,929	9.15	जा	19,941	7.93	जीवन	1,277	6.41	नाम	1,681	7.27	विभाग	672	6.77
स्थापित	23,403	9.12	चाह	17,318	8.6	कमी	1,110	6.87	पद	1,610	7.79	अधिकारी	597	6.47
जारी	23,322	9.09	गई	6,716	7.36	समाज	1,064	6.25	मुद्दा	1,338	7.63	कंपनी	588	6.46
पेश	23,064	9.1	पहले	5,268	7.18	बच्चा	1,050	6.18	समय	1,243	6.37	वह	537	6.44

Figure 5.1 Word sketches for the verb कर (do)

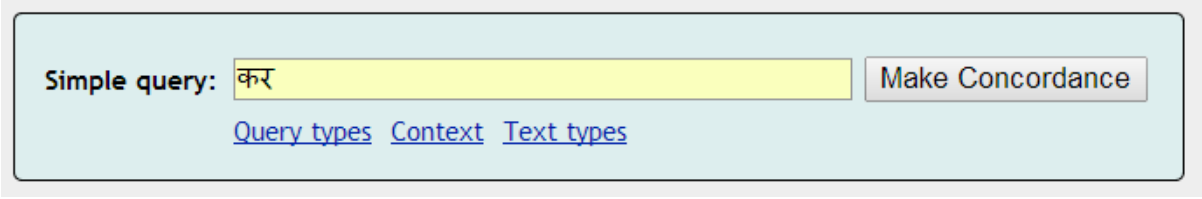


Figure 5.2 Concordance query

5.2 The Sketch Engine For Hindi

5.2.1 The Simple Concordance Query Function

A Simple concordance query shows the word as it is used in different texts. Figure 5.2.1 shows the query box, while Figure 5.2.1 shows its output. A simple search query for a word such as कर (do) searches for the lemma as well as the words which have कर (do) as the lemma, so कर (do), कयी (did), करने (to do), करते ([they] will do), etc. are all retrieved. Figure 5.2.1 shows the first 20 results out of the retrieved 5 million results.

5.2.2 The Frequency Functions

The Sketch Engine interface provides easy access to tools for visualizing different aspects of the word frequency (see Figure 5.2.2). *The Frequency Node forms* function on the left hand menu in Figure 4 shows which of the returned forms are most frequent.

Thus we have immediately discovered that the commonest forms of the lemma कर (do) are कर (do), कयी (did) and करने (to do).

Figure 5.3 Concordance results

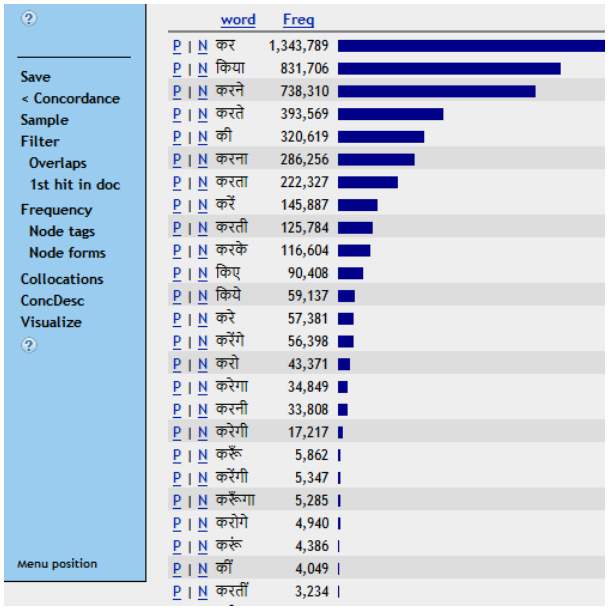


Figure 5.4 Frequency of word forms

Figure 5.5 Word list

The **p/n** links are for positive and negative examples. Clicking on **p** gives a concordance for the word form, while clicking on **n** gives the whole concordance *except* for the word form.

5.2.3 The Word List function

The Word List function allows the user to make frequency lists of many types (words, lemmas, tags). Figure 5.2.3 shows the most frequent words in the corpus. In addition to most frequent words, keywords of any target corpus can be extracted. This is done by comparing frequent words from the target corpus with the frequent words from a general purpose corpus. Figure 5.2.3 displays the keywords of a Hindi Election corpus, where this is the target corpus, and the general purpose corpus is the HindiWaC.

Almost every keyword closely relates to the trend of news articles in the 2014 Indian Parliament elections. Since the Hindi Election Corpus is of small size, the frequency-per-million column contains projected values. These are significantly higher than the same words in the HindiWaC since the Election Corpus is domain specific.

5.2.4 The Word Sketch and Collocation Concordance functions

The Word Sketch function is invaluable for finding collocations. The word sketches of the word (people) for three dependency relations are shown in Figure 5.2.4.

The dependency relations that we use are based on the Paninian framework (Begum et al., 2008). Three of the most common dependency relations given by this model are as follows:

	Hindi Election Corpus		HindiWaC		
word	Freq	Freq/mill	Freq	Freq/mill	Score
केजरीवाल	234	7801.8	2,097	31.9	8.5
पार्टी	209	6968.3	17,336	263.6	6.3
मोदी	161	5367.9	4,884	74.3	5.9
सरकार	222	7401.7	59,774	908.8	4.4
कांग्रेस	130	4334.3	18,838	286.4	4.1
चुनाव	115	3834.2	11,435	173.9	4.1
राहुल	105	3500.8	8,324	126.6	4.0
सम्मेलन	84	2800.7	4,093	62.2	3.6
गुजरात	83	2767.3	5,203	79.1	3.5
कहा	285	9502.2	136,695	2078.3	3.4
भाजपा	82	2734.0	9,501	144.5	3.3
गांधी	88	2934.0	15,291	232.5	3.2
दिल्ली	107	3567.5	30,665	466.2	3.1
प्रधानमंत्री	75	2500.6	12,890	196.0	2.9
मंत्री	74	2467.2	17,495	266.0	2.7
वाराणसी	54	1800.4	1,499	22.8	2.7
उन्होंने	130	4334.3	65,402	994.4	2.7
नरेंद्र	44	1467.0	1,502	22.8	2.4
मुख्यमंत्री	53	1767.1	10,822	164.5	2.4
विधानसभा	44	1467.0	3,764	57.2	2.3

Figure 5.6 Frequency list of the whole corpus for Words and Keywords extracted automatically from Hindi Election Corpus by comparing it with Hindi Web Corpus

लोग *(noun)*

HindiWaC Sketches - Grammar_70% freq = 568946 *(2,094.7 per million)*

nmod adj	240,145	4.3	k1 inv	22,290	6.4	nmod	12,337	9.8
कुछ	24,854	10.33	जम	159	7.05	मानसिकता	104	7.01
उन	11,086	10.12	मान	1,117	6.42	ज्यादातर	52	6.04
ऐसे	10,530	9.98	कह	2,606	6.0	मुताबिक	73	6.04
कई	11,367	9.66	मार	265	5.83	अधिकतर	40	5.83
सभी	7,664	9.28	मर	152	5.77	जी	86	5.8
सब	5,960	9.21	जता	76	5.73	रसूख	21	5.73
आम	5,412	9.16	सोच	277	5.72	सोच	65	5.51
जिन	4,453	9.1	मचा	57	5.56	आय	37	5.49
लाख	4,871	9.1	समझा	141	5.53	राशि	86	5.45
इन	7,082	9.02	बता	616	5.53	रख	550	5.36
अन्य	4,844	8.76	सराह	36	5.5	गुजर	57	5.29
स्थानीय	3,444	8.72	पूछ	172	5.48	पहुंचने	33	5.22
यह	11,814	8.67	घेर	48	5.42	बस	30	5.11
कम	3,329	8.46	देख	727	5.14	मर	81	5.03
हजारा	2,322	8.22	थाना	49	5.08	विचारधारा	29	5.03
सारा	2,851	8.2	उठा	166	5.02	विपरीत	24	4.99
ज्यादातर	1,958	7.98	चुन	68	4.99	आमदनी	15	4.86
अधिक	2,581	7.97	बोल	193	4.99	पैसा	49	4.83
अनेक	2,215	7.93	बैठ	142	4.95	छवि	25	4.8
ज्यादा	2,257	7.87	बजा	42	4.95	कमा	31	4.78

Figure 5.7 Word Sketch results for लोग (people)

- k1: agent and/or doer
- k2: object and/or theme
- k3: instrument

These relations are syntactico-semantic in nature, and differ slightly from the equivalent thematic roles mentioned above. More about how we get the word sketches shown in Figure 5.2.4 is explained in Section 5.3.

In figure 5.2.4, the three dependency relations shown are:

- nmod_adj: nounmodifier_adjective
- k1_inv: doer_inverse
- nmod: nounmodifier

The word sketch function assigns weights to each of the collocates and also to the dependency relations.

Clicking on the number after the collocate gives a concordance of the combination (Figure 5.2.4).

5.2.5 The Bilingual Word Sketch function

A new function has been added recently to the Word Sketch, which is the Bilingual Word Sketch. This allows the user to see word sketches for two words side by side in different languages. Figure 5.2.5 shows a comparison between लाल (red) and red. Interestingly, the usage of word red in Hindi and English are very diverse. The only common noun which is modified by red in both languages is गुलाब (rose).

5.2.6 Distributional Thesaurus and Sketch Diff

The Sketch Engine also offers a distributional thesaurus, where, for the input word, the words 'sharing' most collocates are presented. Figure 5.2.6 shows the top entries in similarity to कर (do). The top result is हो(be). Clicking on it takes us to a 'sketch diff', a report that shows the similarities and differences between the two words - the right half of the image.

The red results occur most frequently with हो(be), the green ones with कर (do). The ones on white occur equally with both.

5.3 Building and processing HindiWaC and loading it into the Sketch Engine

HindiWaC was built using the Corpus Factory procedure (Kilgarriff et al., 2010). A first tranche was built in 2009, with the crawling process repeated and more data added in 2011, and

Figure 5.8 Concordance for लोग (people) in combination with its gramrel "nmod"

लाल (adjective) Alternativ HindiWac Sketches - G			red (noun) ukWaC Frec		
modifier of	6,082	7.8	modifies	76,712	3.1
मिर्च	579	10.95	squirrel	1,549	9.12
रंग	938	10.64	tape	2,781	9.11
किला	337	10.53	wine	2,668	8.63
किताब	303	9.83	herring	853	8.4
शास्त्री	168	9.6	deer	952	8.26
चंदन	160	9.51	brick	1,070	8.14
बत्ती	112	9.12	sandstone	753	8.09
वस्त्र	148	9.08	pepper	818	8.04
फूल	136	8.64	flag	1,087	7.9
पुष्प	62	8.15	cell	2,590	7.81
धागा	54	8.01	meat	938	7.62
गुलाब	54	8.01	carpet	639	7.59
यादव	64	7.87	kite	434	7.37
डोरा	45	7.86	onion	528	7.36
कपड़ा	76	7.86	light	2,350	7.34
बाबू	47	7.68	rose	375	7.03

Figure 5.9 Adjective results of a bilingual word sketch for Hindi लाल (red) and English red. English translations of some of the Hindi words are: chilli, colour, fort, flower, rose, cloth, Shastri

कर (verb) HindiWac Sketches - (

Lemma	Score	Freq
हो	0.653	4,452,572
दे	0.594	1,714,785
ले	0.52	1,020,412
रह	0.491	2,086,780
रख	0.485	316,618
आ	0.476	978,940
है	0.468	9,658,560
बना	0.467	440,902
देख	0.444	527,703
था	0.411	2,460,144
लगा	0.403	355,352
जान	0.389	506,661
कह	0.372	914,047
मिल	0.344	431,473
लग	0.337	483,869
मान	0.335	233,821

Figure 5.10 Thesaurus search showing entries similar to कर (do) (left) and Sketch Diff comparing collocates of कर (do) and हो(be) (right)

again in 2014. Corpus Factory method can be briefly described as follows: several thousands of target language search queries are generated from Wikipedia, and are submitted to Microsoft Bing search engine. The corresponding hit pages for each query are downloaded. The pages are filtered using a language model. Boilerplate text is removed using body text extraction, deduplication to create clean corpus. We use jusText and Onion tools (Pomikálek, 2011) for body-text extraction and deduplication tools respectively.

The text is then tokenized, lemmatized and POS-tagged using the tools downloaded from http://sivareddy.in/downloads (Reddy and Sharoff, 2011). The tokenizer found here is installed in the Sketch Engine.

5.3.1 Sketch Grammar for Hindi

A sketch grammar is a grammar for the language, based on regular expressions over part-ofspeech tags. It underlies the word sketches and is written in the Corpus Query Language (CQL). Sketch grammar is designed particularly to identify head-and-dependent pairs of words (e.g., खा [eat] and राम [Ram]) in specified grammatical relations (here, k1 [doer]), in order that the dependent can be entered into the head's word sketch and vice versa.

Sketch Grammars are popular with lexicographic and corpus linguistics community, and are used to identify collocations of a word with a given grammatical relation (Kilgarriff and Rundell, 2002). We use Sketch Grammar to identify words in syntactic relations in a given sentence. For example, a grammar rule for the relation "k1" (doer) is 2:"NN" "PSP ने" "JJ"? 1:"VM", which specifies that if a noun is followed by a PSP and an optional adjective and followed by a verb, then the noun is the kartha/subject of the verb. The head and child are identified by 1: and 2: respectively. Each rule may often be matched by more than one relation creating ambiguity. Yet they tend to capture the most common behavior in the language.

Writing a full-fledged sketch grammar with high coverage is a difficult task even for language experts, as it would involve capturing all the idiosyncrasies of a language. Even though such hand-written rules tend to be more accurate, the recall of the rules is very low. In this paper, the grammar we use is a collection of POS tag sequences (rules) which are automatically extracted from an annotated Treebank, Hindi Dependency Treebank (HDT-v0.5), which was released for the Coling2012 shared task on dependency parsing (Sharma et al., 2012). This treebank uses IIIT tagset described in (Bharati et al., 2006). This method gives us a lot of rules based on the syntactic ordering of the words. Though these rules do not have all the lexical cues of a language, the hope is that, when applied on a large-scale web corpus, the correct matches (sketches) of the rules automatically become statistically more frequent, and hence more significant.

From the above mentioned treebank (HDT) we extract dependency grammar rules (i.e. sketch grammar) automatically for each dependency relation, based on the POS tags appearing in between the dependent words (inclusive). For example, from the sentence,

राम(Ram) ने(erg.) कमरे(room) में (inside) आम(mango) खाया(eat)

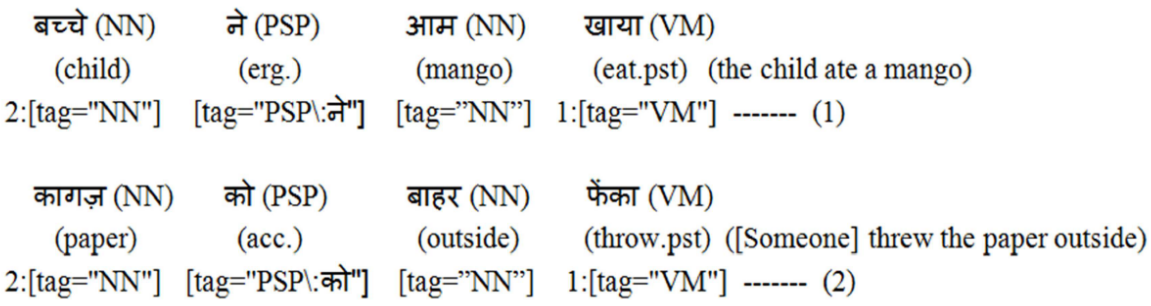


Figure 5.11 A sample of similar rules for different dependency relations

Ram ate [a] mango in [the] room
we extract rules of the type: (k1[doer], k2 [object], k7[location])

- **k1** - 2:[tag="NNP"] [tag="PSP ने"] [tag="NN"] [tag="PSP में"] 1:[tag="VM"]
- **k2** - 2:[tag="NN"] 1:[tag="VM"]
- **k7** - 2:[tag="NN"] [tag="PSP में"] [tag="NN"] 1:[tag="VM"]

In the above example, relation names are in bold with one of the corresponding rules for each of them.

We do include a few lexical features associated with the POS tags **PSP** (post-position) and **CC** (conjunction) in order to disambiguate between different dependency relations. For example, in both the relations k1 (doer) and k2 (object) we have the rules (1) and (2) given below respectively in Figure 5.3.1:

In (1), the ergative marker indicates that the noun (NN) is the doer of the verb (VM). In (2), the accusative marker indicates that the noun (NN) is the object of the verb (VM). Also, (2) is not a complete sentence – the doer has not been mentioned, and only the part of the sentence that the rule is applied to is shown.

If in the rules, the PSP POS tags didn't contain the lexical features, both the rules would have been the same, and hence both the rules have been applied on both the sentences, making the word sketches erroneous.

By lexicalizing the PSP POS tag, the rule(s) formed are now less ambiguous, and more accurate.

Chapter *6*

Conclusion

We presented an easy-to-extract non lexical grammar using POS tag sequences, and some ways of disambiguating the rules and making them more generic for increased coverage. This is useful for languages which do not have large annotated corpora.

Using the above grammar, we have shown that the performance of lexical parsers (in this case, Malt) can be improved significantly - whilst also increasing the recall. The grammar can be made more concise/precise/generic depending on the need, and can most likely be used to improve parsing performance on languages with sparse annotated data.

We prepared and used HindiWac, a large corpus for Hindi in Sketch Engine. Thich offers us valuable insights not only into the behaviour of words and relations in Hindi, but also how similar words are used across different languages.

Bibliography

[1] S. Buchholz and E. Marsi, "Conll-x shared task on multilingual dependency parsing," in *Proceedings of the Tenth Conference on Computational Natural Language Learning*. Association for Computational Linguistics, 2006, pp. 149–164.

[2] J. Nilsson, S. Riedel, and D. Yuret, "The conll 2007 shared task on dependency parsing," in *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL*. sn, 2007, pp. 915–932.

[3] A. Bharati, P. Mannem, and D. M. Sharma, "Hindi Parsing Shared Task," in *Proceedings of Coling Workshop on Machine Translation and Parsing in Indian Languages*, Kharagpur, India, 2012.

[4] J. Nivre, J. Hall, J. Nilsson, A. Chanev, G. Eryigit, S. Kübler, S. Marinov, and E. Marsi, "Maltparser: A language-independent system for data-driven dependency parsing," *Natural Language Engineering*, vol. 13, no. 2, pp. 95–135, 2007.

[5] R. McDonald, "Discriminative learning and spanning tree algorithms for dependency parsing," Ph.D. dissertation, Philadelphia, PA, USA, 2006.

[6] M. Collins, "Head-driven statistical models for natural language parsing," *Computational linguistics*, vol. 29, no. 4, pp. 589–637, 2003.

[7] E. Charniak, "A maximum-entropy-inspired parser," in *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*. Association for Computational Linguistics, 2000, pp. 132–139.

[8] K. Sagae and A. Lavie, "Parser combination by reparsing," in *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*. New York City, USA: Association for Computational Linguistics, June 2006, pp. 129–132. [Online]. Available: http://www.aclweb.org/anthology/N/N06/N06-2033

[9] J. Hall, J. Nilsson, J. Nivre, G. Eryigit, B. Megyesi, M. Nilsson, and M. Saers, "Single malt or blended? a study in multilingual parser optimization," in *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, Prague, Czech Republic, June 2007, pp. 933–939. [Online]. Available: http://www.aclweb.org/anthology/D/D07/D07-1097

[10] S. M. Kim, D. Ng, M. Johnson, and J. Curran, "Improving combinatory categorial grammar parse reranking with dependency grammar features," in *Proceedings of COLING 2012*. Mumbai, India: The COLING 2012 Organizing Committee, December 2012, pp. 1441–1458. [Online]. Available: http://www.aclweb.org/anthology/C12-1088

[11] D. Klein and C. D. Manning, "Accurate unlexicalized parsing," in *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*. Association for Computational Linguistics, 2003, pp. 423–430.

[12] T. Briscoe and J. Carroll, "Evaluating the accuracy of an unlexicalized statistical parser on the parc depbank," in *Proceedings of the COLING/ACL on Main conference poster sessions*. Association for Computational Linguistics, 2006, pp. 41–48.

[13] T. Briscoe, "An introduction to tag sequence grammars and the rasp system parser," *Computer Laboratory Technical Report*, vol. 662, 2006.

[14] T. Briscoe, J. Carroll, and R. Watson, "The second release of the rasp system," in *Proceedings of the COLING/ACL on Interactive presentation sessions*. Association for Computational Linguistics, 2006, pp. 77–80.

[15] A. Kilgarriff, P. Rychly, P. Smrz, and D. Tugwell, "Itri-04-08 the sketch engine," *Information Technology*, vol. 105, p. 116, 2004.

[16] K. Ivanova, U. Heid, S. S. Im Walde, A. Kilgarriff, and J. Pomikálek, "Evaluating a german sketch grammar: A case study on noun phrase case." in *LREC*, 2008.

[17] A. Kilgarriff and D. Tugwell, "Sketching words," *Lexicography and Natural Language Processing: A Festschrift in Honour of BTS Atkins*, pp. 125–137, 2002.

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

[19] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of english: The penn treebank," *Computational linguistics*, vol. 19, no. 2, pp. 313–330, 1993.

[20] Y. Zhang and J. Nivre, "Analyzing the effect of global learning and beam-search on transition-based dependency parsing," in *Proceedings of COLING 2012: Posters*. Mumbai, India: The COLING 2012 Organizing Committee, December 2012, pp. 1391–1400. [Online]. Available: http://www.aclweb.org/anthology/C12-2136