Inspecting the Structural Biases of Dependency Parsing Algorithms *

Yoav Goldberg and **Michael Elhadad**
Ben Gurion University of the Negev
Department of Computer Science
POB 653 Be'er Sheva, 84105, Israel
yoavg|elhadad@cs.bgu.ac.il

Abstract

We propose the notion of a *structural bias* inherent in a parsing system with respect to the language it is aiming to parse. This structural bias characterizes the behaviour of a parsing system in terms of structures it tends to under- and over- produce. We propose a Boosting-based method for uncovering some of the structural bias inherent in parsing systems. We then apply our method to four English dependency parsers (an Arc-Eager and Arc-Standard transition-based parsers, and first- and second-order graph-based parsers). We show that all four parsers are biased with respect to the kind of annotation they are trained to parse. We present a detailed analysis of the biases that highlights specific differences and commonalities between the parsing systems, and improves our understanding of their strengths and weaknesses.

1 Introduction

Dependency Parsing, the task of inferring a dependency structure over an input sentence, has gained a lot of research attention in the last couple of years, due in part to to the two CoNLL shared tasks (Nivre et al., 2007; Buchholz and Marsi, 2006) in which various dependency parsing algorithms were compared on various data sets. As a result of this research effort, we have a choice of several robust, efficient and accurate parsing algorithms.

These different parsing systems achieve comparable scores, yet produce qualitatively different parses. Sagae and Lavie (2006) demonstrated that a simple combination scheme of the outputs of different parsers can obtain substantially improved accuracies. Nivre and McDonald (2008) explore a parser stacking approach in which the output of one parser is fed as an input to a different kind of parser. The stacking approach also produces more accurate parses.

However, while we know how to produce accurate parsers and how to blend and stack their outputs, little effort was directed toward understanding the behavior of different parsing systems in terms of structures they produce and errors they make. Question such as *which linguistic phenomena are hard for parser Y?* and *what kinds of errors are common for parser Z?*, as well as the more ambitious *which parsing approach is most suitable to parse language X?*, remain largely unanswered.

The current work aims to fill this gap by proposing a methodology to identify systematic biases in various parsing models and proposing and initial analysis of such biases.

McDonald and Nivre (2007) analyze the difference between graph-based and transition-based parsers (specifically the MALT and MST parsers) by comparing the different kinds of errors made by both parsers. They focus on single edge errors, and learn that MST is better for longer dependency arcs while MALT is better on short dependency arcs, that MALT is better than MST in predicting edges further from the root and vice-versa, that MALT has a slight advantage when predicting the parents of nouns and pronouns, and that MST is better at all other word categories. They also conclude that the greedy MALT Parser suffer from error propagation more than the globally optimized

*We would like to thank Reut Tsarfaty for comments and discussions that helped us improve this paper. This work is supported in part by the Lynn and William Frankel Center for Computer Science.

Proceedings of the Fourteenth Conference on Computational Natural Language Learning, pages 234–242,
Uppsala, Sweden, 15-16 July 2010. ©2010 Association for Computational Linguistics

MST Parser.

In what follows, we complement their work by suggesting a different methodology of analysis of parsers behaviour. Our methodology is based on the notion of *structural bias* of parsers, further explained in Section 2. Instead of comparing two parsing systems in terms of the errors they produce, our analysis compares the output of a parsing system with a collection of gold-parsed trees, and searches for common structures which are predicted by the parser more often than they appear in the gold-trees or vice-versa. These kinds of structures represent the bias of the parsing systems, and by analyzing them we can gain important insights into the strengths, weaknesses and inner working of the parser.

In Section 2.2 we propose a Boosting-based algorithm for uncovering these structural biases. Then, in Section 3 we go on to apply our analysis methodology to four parsing systems for English: two transition-based systems and two graph-based systems (Sections 4 and 5). The analysis shows that the different parsing systems indeed possess different biases. Furthermore, the analysis highlights the differences and commonalities among the different parsers, and sheds some more light on the specific behaviours of each system.

Recent work by Dickinson (2010), published concurrently with this one, aims to identify dependency errors in automatically parsed corpora by inspecting *grammatical rules* which appear in the automatically parsed corpora and do not fit well with the grammar learned from a manually annotated treebank. While Dickinson's main concern is with automatic identification of errors rather than characterizing parsers behaviour, we feel that his work shares many intuitions with this one: automatic parsers fail in predictable ways, those ways can be analyzed, and this analysis should be carried out on structures which are larger than single edges, and by inspecting trends rather than individual decisions.

2 Structural Bias

Language is a highly structured phenomena, and sentences exhibit structure on many levels. For example, in English sentences adjectives appear before nouns, subjects tend to appear before their verb, and syntactic trees show a tendency toward right-branching structures.[1]

Different combinations of languages and annotation strategies exhibit different *structural preferences*: under a specific combination of language and annotation strategy some structures are more frequent than others, some structures are illegal and some are very rare.

We argue that parsers also exhibit such structural preferences in the parses they produce. These preferences stem from various parser design decisions. Some of the preferences, such as *projectivity*, are due to explicit design decisions and lie at the core of some parsing algorithms. Other preferences are more implicit, and are due to specific interactions between the parsing mechanism, the feature function, the statistical mechanism and the training data.

Ideally, we would like the structural preferences of a parser trained on a given sample to reflect the general preferences of the language. However, as we demonstrate in Section 3, that it is usually not the case.

We propose the notion of *structural bias* for quantifying the differences in structural preferences between a parsing system and the language it is aiming to parse. The structural bias of a parser with respect to a language is composed of the structures that tend to occur more often in the parser's output than in the language, and vice-versa.

Structural biases are related to but different than common errors. *Parser X makes many PP attachment errors* is a claim about a common error. *Parser X tends to produce low attachment for PPs while the language tends to have high attachment* is a claim about structural bias, which is related to parser errors. *Parser X can never produce structure Y* is a claim about a structural preference of a parser, which may or may not be related to its error patterns.

Structural bias is a vast and vague concept. In order to give a more concrete definition, we pose the following question:

Assuming we are given two parses of the same sentence. Can we tell, by looking at the parses and without knowing the correct parse, which parser produced which parse?

Any predictor which can help in answering this question is an indicator of a structural bias.

[1] As noted by (Owen Rambow, 2010), there is little sense in talking about *the structure of a language* without referring to a specific annotation scheme. In what follow, we assume a fixed annotation strategy is chosen.

Definition: structural bias between sets of trees Given two sets of parse trees, A and B, over the same sentences, a *structural bias* between these sets is the collection of all predictors which can help us decide, for a tree t, whether it belongs to A or to B.

The structural bias between a parsing system and an annotated corpus is then the structural bias between the corpus and the output of the parser on the sentences in the corpus. Note that this definition adheres to the error *vs.* bias distinction given above.

Under this task-based definition, uncovering structural biases between two sets of trees amounts to finding good predictors for discriminating between parses coming from these two sets of trees. In what follows, we present a rich class of structural predictors, and an algorithm for efficiently searching this predictor class for good predictors.

2.1 Representing Structure

A dependency representation of sentences includes words and dependency relations between them (one word is the ROOT of the sentence, and each other word has a single word as its *parent*). Whenever possible, we would like to equate words with their part-of-speech tags, to facilitate generalization. However, in some cases the exact identity of the word may be of interest. When analyzing a language with a relatively fixed word order, such as English, we are also interested in the linear order between words. This includes the *direction* between a parent and its dependent (does the parent appear before or after the dependent in the sentence?), as well as the order among several dependents of the same parent. The *length* of a dependency relation (distance in words between the parent and dependent) may also be structurally interesting.[2]

In order to capture this kind of information, we take a *structural element* of a dependency tree to be any connected subtree, coupled with information about the incoming edge to the root of the subtree. Examples of such structural elements are given in Figure 1. This class of predictors is not complete – it does not directly encode, for instance, information about the number of siblings a node has or the location of the structure relative to the root of the tree. However, we feel it does capture a good deal of linguistic phenomena, and provide a fine balance between expressiveness and tractability.

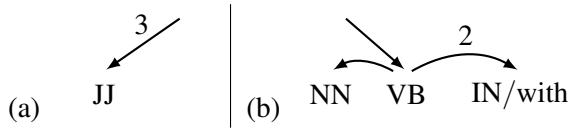


Figure 1: **Structural Elements Examples**. (a) is an adjective with a parent 3 words to its right. (b) is a verb whose parent is on the left, it has a noun dependent on its left, and a preposition dependent 2 words to its right. The lexical item of the preposition is $with$. The lexical items and distance to parent are optional, while all other information is required. There is also no information about other dependents a given word may have.

The class of predictors we consider is the set of all *structural elements*. We seek to find structural elements which appear in many trees of set A but in few trees of set B, or vice versa.

2.2 Boosting Algorithm with Subtree Features

The number of possible predictors is exponential in the size of each tree, and an exhaustive search is impractical. Instead, we solve the search problem using a Boosting algorithm for tree classification using subtree features. The details of the algorithm and its efficient implementation are given in (Kudo and Matsumoto, 2004). We briefly describe the main idea behind the algorithm.

The Boosting algorithm with subtree features gets as input two parse sets with labeled, ordered trees. The output of the algorithm is a set of subtrees t_i and their weights w_i. These weighted subtrees define a linear classifier over trees $f(T) = \sum_{t_i \in T} w_i$, where $f(T) > 0$ for trees in set A and $f(T) < 0$ for trees in set B.

The algorithm works in rounds. Initially, all input trees are given a uniform weight. At each round, the algorithm seeks a subtree t with a *maximum gain*, that is the subtree that classifies correctly the subset of trees with the highest cumulative weight. Then, it re-weights the input trees, so that misclassified trees get higher weights. It continues to repeatedly seek maximum gain subtrees, taking into account the tree weights in the gain calculation, and re-weighting the trees after each iteration. The same subtree can be selected in different iterations.

Kudo and Matsumoto (2004) present an effec-

[2]Relations can also be labeled, and labeling fit naturally in our representation. However, we find the commonly used set of edge labels for English to be lacking, and didn't include edge labels in the current analysis.

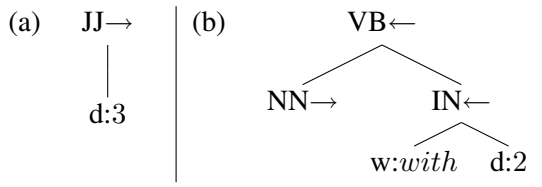


Figure 2: **Encoding Structural Elements as Ordered Trees**. These are the tree encodings of the structural elements in Figure 1. Direction to parent is encoded in the node name, while the optional lexical item and distance to parent are encoded as daughters.

tive branch-and-bound technique for efficiently searching for the maximum gain tree at each round. The reader is referred to their paper for the details.

Structural elements as subtrees The boosting algorithm works on labeled, ordered trees. Such trees are different than dependency trees in that they contain information about nodes, but not about edges. We use a simple transformation to encode dependency trees and structural elements as labeled, ordered trees. The transformation works by concatenating the edge-to-parent information to the node's label for mandatory information, and adding edge-to-parent information as a special child node for optional information. Figure 2 presents the tree-encoded versions of the structural elements in Figure 1. We treat the direction-to-parent and POS tag as required information, while the distance to parent and lexical item are optional.

2.3 Structural Bias Predictors

The output of the boosting algorithm is a set of weighted subtrees. These subtrees are good candidates for structural bias predictors. However, some of the subtrees may be a result of over-fitting the training data, while the weights are tuned to be used as part of a linear classifier. In our application, we disregard the boosting weights, and instead rank the predictors based on their number of occurrences in a validation set. We seek predictors which appear many times in one tree-set but few times in the other tree-set on both the training and the validation sets. Manual inspection of these predictors highlights the structural bias between the two sets. We demonstrate such an analysis for several English dependency parsers below.

In addition, the precision of the learned Boosting classifier on the validation set can serve as a metric for measuring the amount of structural bias between two sets of parses. A high classification accuracy means more structural bias between the two sets, while an accuracy of 50% or lower means that, at least under our class of predictors, the sets are structurally indistinguishable.

3 Biases in Dependency Parsers

3.1 Experimental Setup

In what follows, we analyze and compare the structural biases of 4 parsers, with respect to a dependency representation of English.

Syntactic representation The dependency treebank we use is a conversion of the English WSJ treebank (Marcus et al., 1993) to dependency structure using the procedure described in (Johansson and Nugues, 2007). We use the Mel'čuk encoding of coordination structure, in which the first conjunct is the head of the coordination structure, the coordinating conjunction depends on the head, and the second conjunct depend on the coordinating conjunction (Johansson, 2008).

Data Sections 15-18 were used for training the parsers[3]. The first 4,000 sentences from sections 10-11 were used to train the Boosting algorithm and find structural predictors candidates. Sections 4-7 were used as a validation set for ranking the structural predictors. In all experiments, we used the gold-standard POS tags. We binned the distance-to-parent values to 1,2,3,4-5,6-8 and 9+.

Parsers For graph-based parsers, we used the projective first-order (MST1) and second-order (MST2) variants of the freely available MST parser[4] (McDonald et al., 2005; McDonald and Pereira, 2006). For the transition-based parsers, we used the arc-eager (ARCE) variant of the freely available MALT parser[5] (Nivre et al., 2006), and our own implementation of an arc-standard parser (ARCS) as described in (Huang et al., 2009). The unlabeled attachment accuracies of the four parsers are presented in Table 1.

Procedure For each parser, we train a boosting classifier to distinguish between the gold-standard trees and the parses produced for them by the

[3]Most work on parsing English uses a much larger training set. We chose to use a smaller set for convenience. Training the parsers is much faster, and we can get ample test data without resorting to jackknifing techniques. As can be seen in Table 1, the resulting parsers are still accurate.

[4]http://sourceforge.net/projects/mstparser/

[5]http://maltparser.org/

MST1	MST2	ARCE	ARCS
88.8	89.8	87.6	87.4

Table 1: Unlabeled accuracies of the analyzed parsers

Parser	Train Accuracy	Val Accuracy
MST1	65.4	57.8
MST2	62.8	56.6
ARCE	69.2	65.3
ARCS	65.1	60.1

Table 2: Distinguishing parser output from gold-trees based on structural information

parser. We remove from the training and validation sets all the sentences which the parser got 100% correct. We then apply the models to the validation set. We rank the learned predictors based on their appearances in gold- and parser-produced trees in the train and validation sets, and inspect the highest ranking predictors.

Training the boosting algorithm was done using the `bact`[6] toolkit. We ran 400 iterations of boosting, resulting in between 100 and 250 distinct subtrees in each model. Of these, the top 40 to 60 ranked subtrees in each model were good indicators of structural bias. Our wrapping code is available online[7] in order to ease the application of the method to other parsers and languages.

3.2 Quantitative Analysis

We begin by comparing the accuracies of the boosting models trained to distinguish the parsing results of the various parsers from the English treebank. Table 2 lists the accuracies on both the training and validation sets.

The boosting method is effective in finding structural predictors. All parsers output is distinguishable from English trees based on structural information alone. The ArcEager variant of MALT is the most biased with respect to English. The transition-based parsers are more structurally biased than the graph-based ones.

We now turn to analyze the specific structural biases of the parsing systems. For each system we present some prominent structures which are *under-produced* by the system (these structures appear in the language more often than they are produce by the parser) and some structures which are *over-produced* by the system (these structures are produced by the parser more often than they appear in the language).[8] Specifically, we manually inspected the predictors where the ratio between language and parser was high, ranked by absolute number of occurrences.

4 Transition-based Parsers

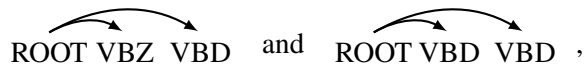
We analyze two transition-based parsers (Nivre, 2008). The parsers differ in the transition systems they adopt. The ARCE system makes use of a transition system with four transitions: LEFT,RIGHT,SHIFT,REDUCE. The semantics of this transition system is described in (Nivre, 2004). The ARCS system adopts an alternative transition system, with three transitions: ATTACHL,ATTACHR,SHIFT. The semantics of the system is described in (Huang et al., 2009). The main difference between the systems is that the ARCE system makes attachments as early as possible, while the ARCS system should not attach a parent to its dependent until the dependent has acquired all its own dependents.

4.1 Biases of the Arc-Eager System

Over-produced structures The over-produced structures of ARCE with respect to English are overwhelmingly dominated by spurious ROOT attachments.

The structures ROOT→“ , ROOT→DT, ROOT→WP are produced almost 300 times by the parser, yet never appear in the language. The structures ROOT→” , ROOT→WRB , ROOT→JJ appear 14 times in the language and are produced hundreds of time by the parser. Another interesting case is ROOT $\overrightarrow{9+}$ NN , produced 180 times by the parser and appearing 7 times in the language. As indicated by the distance marking (9+), nouns are allowed to be heads of sentences, but then they usually appear close to the beginning, a fact which is not captured by the parsing system. Other, less clear-cut cases, are ROOT as the parent of IN, NN, NNS or NNP. Such structures do appear in the language, but are 2-5 times more common in the parser.

A different ROOT attachment bias is captured by



,

[6] http://chasen.org/~taku/software/bact/

[7] http://www.cs.bgu.ac.il/~yoavg/software/

[8] One can think of over- and under- produced structures in terms of the *precision* and *recall* metrics: over-produced structures have low precision, while under-produced structures have low recall.

appearing 3 times in the language and produced over a 100 times by the parser.

It is well known that the ROOT attachment accuracies of transition-based systems is lower than that of graph-based system. Now we can refine this observation: the ARCE parsing system fails to capture the fact that some categories are more likely to be attached to ROOT than others. It also fails to capture the constraint that sentences usually have only one main verb.

Another related class of biases are captured by the structures →VBD $\overrightarrow{9+}$ VBD, →VBD $\overrightarrow{5-7}$ VBD and ROOT→VBZ→VBZ which are produced by the parser twice as many times as they appear in the language. When confronted with embedded sentences, the parser has a strong tendency of marking the first verb as the head of the second one.

The pattern $\overrightarrow{+9}$ IN suggests that the parser prefers high attachment for PPs. The pattern $\overset{9+}{\frown}$ DT← NN captures the bias of the parser toward associating NPs with the preceding verb rather than the next one, even if this preceding verb is far away.

Under-produced structures We now turn to ARCE's under-produced structures. These include the structures IN/*that*← , MD← , VBD← (each 4 times more frequent in the language than in the parser) and VBP← (twice more frequent in the language). MD and *that* usually have their parents to the left. However, in some constructions this is not the case, and the parser has a hard time learning these constructions.

The structure →$→RB appearing 20 times in the language and 4 times in the parser, reflects a very specific construction ("*$ 1.5 up* from $ 1.2"). These constructions pop up as under-produced by all the parsers we analyze.

The structures $\overrightarrow{1}$ RB→IN and →RB→JJ appear twice as often in the language. These stem from constructions such as "not/RB unexpected/JJ", "backed away/RB from/IN", "pushed back/RB in/IN", and are hard for the parser.

Lastly, the structure JJ←NN←NNS←, deviates from the the "standard" NP construction, and is somewhat hard for the parser (39 times parser, 67 in language). However, we will see below that this same construction is even harder for other parsers.

4.2 Biases of the Arc-Standard System

Over-produced structures The over-produced structures of ARCS do not show the spurious ROOT attachment ambiguity of ARCE. They do include ROOT→IN, appearing twice as often in the parser output than in the language.

The patterns ROOT→VBZ$\overrightarrow{9+}$, , →VBP$\overrightarrow{9+}$, , →VBD$\overrightarrow{9+}$VBD and →VB→VBD all reflect the parser's tendency for right-branching structure, and its inability to capture the verb-hierarchy in the sentence correctly, with a clear preference for earlier verbs as parents of later verbs.

Similarly, $\overrightarrow{9+}$NNP and $\overrightarrow{9+}$NNS indicate a tendency to attach NPs to a parent on their left (as an object) rather than to their right (as a subject) even when the left candidate-parent is far away.

Finally, WRB ← MD → VB , produced 48 times by the parser and twice by the language, is the projective parser's way of annotating the correct non-projective structure in which the wh-adverb is dependent on the verb.

Under-produced structures of ARCS include two structures WRB ← VBN and WRB ← VB , which are usually part of non-projective structures, and are thus almost never produced by the projective parser.

Other under-produced structures include appositive NPs:

→IN NN , ,
(*e.g.*, "by Merill , the nation's largest firm , "), and

the structure → NN DT ← NN , which can stand for apposition ("a *journalist*, *the* first *journalist* to ...") or phrases such as "30 %/NN *a month*".

TO usually has its parent on its left. When this is not the case (when it is a part of a quantifier, such as "x *to* y %", or due to fronting: "Due *to* X, we did Y"), the parser is having a hard time to adapt and is under-producing this structure.

Similar to the other parsers, ARCS also under-produces NPs with the structure JJ←NN$\overleftarrow{1}$NN, and the structure →$→RB.

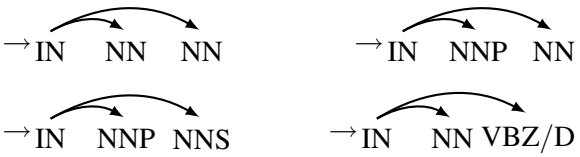
Finally, the parser under-produces the conjunctive structures →NN→CC→NN→IN and →IN→CC→IN.

5 Graph-based Parsers

We analyze the behaviour of two graph-based parsers (McDonald, 2006). Both parsers perform exhaustive search over all projective parse trees, using a dynamic programming algorithm. They differ in the factorizations they employ to make the search tractable. The first-order model employs a single-edge factorization, in which each edge is scored independently of all other edges. The second-order model employs a two-edge factorization, in which scores are assigned to pairs of adjacent edges rather than to a single edge at a time.

5.1 Biases of First-order MST Parser

Over-produced structures of MST1 include:



where the parsers fails to capture the fact that prepositions only have one dependent.

Similarly, in the pattern:

the parser fails to capture that only one phrase should attach to the coordinator, and the patterns

highlight the parser's failing to capture that verbs have only one object.

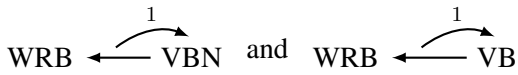
In the structure

, produced by the parser 15 times more than it appears in the language, the parser fails to capture the fact that verbs modified by wh-adverbs are not likely to head a sentence.

All of these over-produced structures are fine examples of cases where MST1 fails due to its edge-factorization assumption.

We now turn to analyzing the structures under-produced by MST1.

Under-produced structures The non-projective structures



clearly cannot be produced by the projective parser, yet they appear over 100 times in the language.

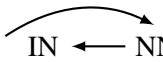
The structure WRB←VBD←VBD which is represented in the language five times more than in the parser, complements the over-produced case in which a verb modified by a wh-adverb heads the sentence.

IN/*that*←, which was under-produced by ARCE is under-produced here also, but less so than in ARCE. →$→RB is also under-produced by the parser.

The structure CC$\overleftarrow{1}$, usually due to conjunctions such as *either*, *nor*, *but* is produced 29 times by the parser and appear 54 times in the language.

An interesting under-produced structure is

. This structure reflects the fact that the parser is having a hard time coordinating "heavy" NPs, where the head nouns are modified by PPs. This bias is probably a result of the "in-between pos-tag" feature, which lists all the pos-tags between the head and dependent. This feature was shown to be important to the parser's overall performance, but probably fails it in this case.

The construction $\overrightarrow{6-8}$JJ, where the adjective functions as an adverb (*e.g.,* "he survived X *unscathed*" or "to impose Y *corporate-wise*") is also under-produced by the parser, as well as 

in which the preposition functions as a determiner/quantifier ("*at* least", "*between*", "more *than*").

Finally, MST1 is under-producing NPs with somewhat "irregular" structures: JJ←NN←NNS or JJ←NN←NNS ("*common stock* purchase *warrants*", "*cardiac bypass patients*"), or JJ←JJ← ("a *good many* short-sellers", "*West German* insurance giant")

5.2 Biases of Second-order MST Parser

Over-produced structures by MST2 are different than those of MST1. The less-extreme edge factorization of the second-order parser successfully prevents the structures where a verb has two objects or a preposition has two dependents.

One over-produced structure,

produced 10 times by the parser and never in the language, is due to one very specific construction, "*bonds due Nov 30 , 1992 ,*" where the second comma should attach higher up the tree.

Another over-produced structure involves the internal structure of proper names: NNP NNP NNP NNP (the "correct" analysis more often makes the last NNP head of all the others).

More interesting are: $\overrightarrow{\ }_{1}$CC→VBD and $\overrightarrow{\ }_{1}$CC→NN→IN . These capture the parser's inability to capture the symmetry of coordinating conjunctions.

Under-produced structures of MST2 are overall very similar to the under-produced structures of MST1.

The structure CC$\overleftarrow{\ }_{1}$ which is under-produced by MST1 is no longer under-produced by MST2. All the other under-produced structures of MST1 reappear here as well.

In addition, MST2 under-produces the structures ROOT→NNP→. (it tends not to trust NNPs as the head of sentences) and $\overrightarrow{\ }_{6-8}TO\overrightarrow{\ }_{1}VB$ (where the parser is having trouble attaching TO correctly to its parent when they are separated by a lot of sentential material).

6 Discussion

We showed that each of the four parsing systems is structurally biased with respect to the English training corpus in a noticeable way: we were able to learn a classifier that can tell, based on structural evidence, if a parse came from a parsing system or from the training corpus, with various success rates. More importantly, the classifier's models are interpretable. By analyzing the predictors induced by the classifier for each parsing system, we uncovered some of the biases of these systems.

Some of these biases (*e.g.,* that transition-based system have lower ROOT-attachment accuracies) were already known. Yet, our analysis refines this knowledge and demonstrates that in the Arc-Eager system a large part of this inaccuracy is not due to finding the incorrect root among valid ambiguous candidates, but rather due to many illegal root attachments, or due to illegal structures where a sentence is analyzed to have two main verbs. In contrast, the Arc-Standard system does not share this spurious root attachment behaviour, and its low root accuracies are due to incorrectly choosing among the valid candidates. A related bias of the Arc-Standard system is its tendency to choose earlier appearing verbs as parents of later occurring verbs.

Some constructions were hard for all the parsing models. For example, While not discussed in the analysis above, all parsers had biased structures containing discourse level punctuation elements (some commas, quotes and dashes) – we strongly believe parsing systems could benefit from special treatment of such markers.

The NP construction (JJ←NN←NNS←) appeared in the analyses of all the parsers, yet were easier for the transition-based parsers than for the graph-based ones. Other NP constructions (discussed above) were hard only for the graph-based parsers.

One specific construction involving the dollar sign and an adverb appeared in all the parsers, and may deserve a special treatment. Similarly, different parsers have different "soft spots" (*e.g.,* "backed away from", "not unexpected" for ARCE, "at least" for MST1, TO← for ARCS, etc.) which may also benefit from special treatments.

It is well known that the first-order edge-factorization of the MST1 parser is too strong. Our analysis reveals some specific cases where this assumptions indeed breaks down. These cases do not appear in the second-order factorization. Yet we show that the second-order model under-produces the same structures as the first-order model, and that both models have specific problems in dealing with coordination structures, specifically coordination of NPs containing PPs. We hypothesize that this bias is due to the "pos-in-between" features used in the MST Parser.

Regarding coordination, the analysis reveals that different parsers show different biases with respect to coordination structures.

7 Conclusions and Future Work

We presented the notion of *structural bias* – specific structures that are systematically over- or under- represented in one set of parse trees relative to another set of parse trees – and argue that different parsing systems exhibit different structural biases in the parses they produced due to various explicit and implicit decisions in parser design. We presented a method for uncovering some of this structural bias, and effectively used it to demonstrate that parsers are indeed biased with respect to the corpus they are trained on, and that different parsers show different biases. We then analyzed the biases of four dependency parsing systems with respect to an English treebank. We ar-

gue that by studying the structural biases of parsing systems we can gain a better understanding on where dependency parsers fail, and how they differ from each other. This understanding can in turn lead us toward designing better parsing systems.

We feel that the current study is just the tip of the iceberg with respect to the analysis of structural bias. Any parsing system for any language and annotation scheme can benefit from such analysis.

References

Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proc. of CoNLL*.

Markus Dickinson. 2010. Detecting errors in automatically-parsed dependency relations. In *Proc. of ACL*.

Liang Huang, Wenbin Jiang, and Qun Liu. 2009. Bilingually-constrained (monolingual) shift-reduce parsing. In *Proc of EMNLP*.

Richard Johansson and Pierre Nugues. 2007. Extended constituent-to-dependency conversion for english. In *Proc of NODALIDA*.

Richard Johansson. 2008. *Dependency-based Semantic Analysis of Natural-language Text*. Ph.D. thesis, Lund University.

Taku Kudo and Yuji Matsumoto. 2004. A Boosting Algorithm for Classification of Semi-Structured Text. In *Proceedings of EMNLP*.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marchinkiewicz. 1993. Building a large annotated corpus of English: The penn treebank. *Computational Linguistics*, 19:313–330.

Ryan McDonald and Joakim Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. In *Proc. of EMNLP*.

Ryan McDonald and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proc of EACL*.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proc of ACL*.

Ryan McDonald. 2006. *Discriminative Training and Spanning Tree Algorithms for Dependency Parsing*. Ph.D. thesis, University of Pennsylvania.

Joakim Nivre and Ryan McDonald. 2008. Integrating graph-based and transition-based dependency parsers. In *Proceedings of ACL*, pages 950–958.

Joakim Nivre, Johan Hall, and Jens Nillson. 2006. MaltParser: A data-driven parser-generator for dependency parsing. In *Proc. of LREC*.

Joakim Nivre, Johan Hall, Sandra Kübler, Ryan Mcdonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *Proc. of EMNLP-CoNLL*.

Joakim Nivre. 2004. Incrementality in deterministic dependency parsing. In *Incremental Parsing: Bringing Engineering and Cognition Together, ACL-Workshop*.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4), December.

Owen Rambow. 2010. The Simple Truth about Dependency and Phrase Structure Representations: An Opinion Piece. In *Proceedings of NAACL*.

Kenji Sagae and Alon Lavie. 2006. Parser combination by reparsing. In *Proceedings of HLT-NAACL*, pages 129–133.