Detecting Parser Errors Using Web-based Semantic Filters

Alexander Yates **Stefan Schoenmackers** **Oren Etzioni**
University of Washington
Computer Science and Engineering
Box 352350
Seattle, WA 98195-2350
{ayates, stef, etzioni} @cs.washington.edu

Abstract

NLP systems for tasks such as question answering and information extraction typically rely on statistical parsers. But the efficacy of such parsers can be surprisingly low, particularly for sentences drawn from heterogeneous corpora such as the Web. We have observed that incorrect parses often result in wildly implausible semantic interpretations of sentences, which can be detected automatically using semantic information obtained from the Web.

Based on this observation, we introduce *Web-based semantic filtering*—a novel, domain-independent method for automatically detecting and discarding incorrect parses. We measure the effectiveness of our filtering system, called WOODWARD, on two test collections. On a set of TREC questions, it reduces error by 67%. On a set of more complex Penn Treebank sentences, the reduction in error rate was 20%.

1 Introduction

Semantic processing of text in applications such as question answering or information extraction frequently relies on statistical parsers. Unfortunately, the efficacy of state-of-the-art parsers can be disappointingly low. For example, we found that the Collins parser correctly parsed just 42% of the list and factoid questions from TREC 2004 (that is, 42% of the parses had 100% precision and 100% recall on labeled constituents). Similarly, this parser produced 45% correct parses on a subset of 100 sentences from section 23 of the Penn Treebank.

Although statistical parsers continue to improve their efficacy over time, progress is slow, particularly for Web applications where training the parsers on a "representative" corpus of hand-tagged sentences is not an option. Because of the heterogeneous nature of text on the Web, such a corpus would be exceedingly difficult to generate.

In response, this paper investigates the possibility of detecting parser errors by using semantic information obtained from the Web. Our fundamental hypothesis is that *incorrect parses often result in wildly implausible semantic interpretations of sentences, which can be detected automatically in certain circumstances.* Consider, for example, the following sentence from the Wall Street Journal: "That compares with per-share earnings from continuing operations of 69 cents." The Collins parser yields a parse that attaches "of 69 cents" to "operations," rather than "earnings." By computing the mutual information between "operations" and "cents" on the Web, we can detect that this attachment is unlikely to be correct.

Our WOODWARD system detects parser errors as follows. First, it maps the tree produced by a parser to a *relational conjunction* (RC), a logic-based representation language that we describe in Section 2.1. Second, WOODWARD employs four distinct methods for analyzing whether a conjunct in the RC is likely to be "reasonable" as described in Section 2.

Our approach makes several assumptions. First, if the sentence is absurd to begin with, then a correct parse could be deemed incorrect. Second, we require a corpus whose content overlaps at least in part with the content of the sentences to be parsed. Otherwise, much of our semantic analysis is impossible.

In applications such as Web-based question answering, these assumptions are quite natural. The questions are *about* topics that are covered extensively on the Web, and we can assume that most questions link verbs to nouns in reasonable combinations. Likewise, when using parsing for information extraction, we would expect our assumptions to hold as well.

Our contributions are as follows:

1. We introduce *Web-based semantic filtering*—a novel, domain-independent method for detecting and discarding incorrect parses.
2. We describe four techniques for analyzing relational conjuncts using semantic information obtained from the Web, and assess their efficacy both separately and in combination.
3. We find that WOODWARD can filter good parses from bad on TREC 2004 questions for a reduction of 67% in error rate. On a harder set of sentences from the Penn Treebank, the reduction in error rate is 20%.

The remainder of this paper is organized as follows. We give an overview of related work in Section 1.1. Section 2 describes semantic filtering, including our RC representation and the four Web-based filters that constitute the WOODWARD system. Section 3 presents our experiments and results, and section 4 concludes and gives ideas for future work.

1.1 Related Work

The problem of detecting parse errors is most similar to the idea of parse reranking. Collins (2000) describes statistical techniques for reranking alternative parses for a sentence. Implicitly, a reranking method detects parser errors, in that if the reranking method picks a new parse over the original one, it is classifying the original one as less likely to be correct. Collins uses syntactic and lexical features and trains on the Penn Treebank; in contrast, WOODWARD uses semantic features derived from the web. See section 3 for a comparison of our results with Collins'.

Several systems produce a semantic interpretation of a sentence on top of a parser. For example, Bos et al. (2004) build semantic representations from the parse derivations of a CCG parser, and the English Resource Grammar (ERG) (Toutanova et al., 2005) provides a semantic representation using minimal recursion semantics. Toutanova et al. also include semantic features in their parse selection mechanism, although it is mostly syntax-driven. The ERG is a hand-built grammar and thus does not have the same coverage as the grammar we use. We also use the semantic interpretations in a novel way, checking them against semantic information on the Web to decide if they are plausible.

NLP literature is replete with examples of systems that produce semantic interpretations and use semantics to improve understanding. Several systems in the 1970s and 1980s used hand-built augmented transition networks or semantic networks to prune bad semantic interpretations. More recently, people have tried incorporating large lexical and semantic resources like WordNet, FrameNet, and PropBank into the disambiguation process. Allen (1995) provides an overview of some of this work and contains many references. Our work focuses on using statistical techniques over large corpora, reducing the need for hand-built resources and making the system more robust to changes in domain.

Numerous systems, including Question-Answering systems like MULDER (Kwok et al., 2001), PiQASso (Attardi et al., 2001), and Moldovan et al.'s QA system (2003), use parsing technology as a key component in their analysis of sentences. In part to overcome incorrect parses, Moldovan et al.'s QA system requires a complex set of relaxation techniques. These systems would greatly benefit from knowing when parses are correct or incorrect. Our system is the first to suggest using the output of a QA system to classify the input parse as good or bad.

Several researchers have used pointwise mutual information (PMI) over the Web to help make syntactic and semantic judgments in NLP tasks. Volk (2001) uses PMI to resolve preposition attachments in German. Lapata and Keller (2005) use web counts to resolve preposition attachments, compound noun interpretation, and noun countability detection, among other things. And Markert et al. (2003) use PMI to resolve certain types of anaphora. We use PMI as just one of several techniques for acquiring information from the Web.

2 Semantic Filtering

This section describes semantic filtering as implemented in the WOODWARD system. WOODWARD consists of two components: a semantic interpreter that takes a parse tree and converts it to a conjunction of first-order predicates, and a sequence of four increasingly sophisticated methods that check semantic plausibility of conjuncts on the Web. Below, we describe each component in turn.

1. What(NP1) ∧	**are(VP1, NP1, NP2)** ∧	states(NP2) ∧	**producing(VP2, NP2, NP3)** ∧		oil(NP3) ∧	in(PP1, NP2, U.S.)
2. What(NP1) ∧		states(NP2) ∧		**producing(VP1, NP3, NP2, NP1)** ∧	oil(NP3) ∧	in(PP1, NP2, U.S.)

Figure 2: **Example relational conjunctions. The first RC is the correct one for the sentence "What are oil producing states in the U.S.?" The second is the RC derived from the Collins parse in Figure 1. Differences between the two RCs appear in bold.**

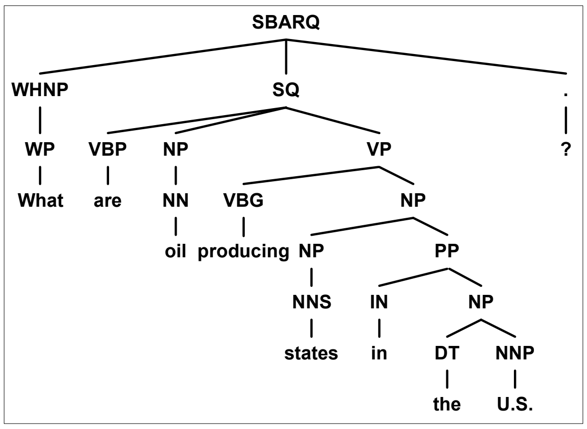

Figure 1: **An *incorrect* Collins Parse of a TREC question. The parser treats "producing" as the main verb in the clause, rather than "are".**

2.1 Semantic Interpreter

The semantic interpreter aims to make explicit the relations that a sentence introduces, and the arguments to each of those relations. More specifically, the interpreter identifies the main verb relations, preposition relations, and semantic type relations in a sentence; identifies the number of arguments to each relation; and ensures that for every argument that two relations share in the sentence, they share a variable in the logical representation. Given a sentence and a Penn-Treebank-style parse of that sentence, the interpreter outputs a conjunction of First-Order Logic predicates. We call this representation a *relational conjunction* (RC). Each relation in an RC consists of a relation name and a tuple of variables and string constants representing the arguments of the relation. As an example, Figure 1 contains a sentence taken from the TREC 2003 corpus, parsed by the Collins parser. Figure 2 shows the correct RC for this sentence and the RC derived automatically from the incorrect parse.

Due to space constraints, we omit details about the algorithm for converting a parse into an RC, but Moldovan et al. (2003) describe a method similar to ours.

2.2 Semantic Filters

Given the RC representation of a parsed sentence as supplied by the Semantic Interpreter, we test the parse using four web-based methods. Fundamentally, the methods all share the underlying principle that some form of co-occurrence of terms in the vast Web corpus can help decide whether a proposed relationship is semantically plausible.

Traditional statistical parsers also use co-occurrence of lexical heads as features for making parse decisions. We expand on this idea in two ways: first, we use a corpus several orders of magnitude larger than the tagged corpora traditionally used to train statistical parses, so that the fundamental problem of data sparseness is ameliorated. Second, we search for targeted patterns of words to help judge specific properties, like the number of complements to a verb. We now describe each of our techniques in more detail.

2.3 A PMI-Based Filter

A number of authors have demonstrated important ways in which search engines can be used to uncover semantic relationships, especially Turney's notion of *pointwise mutual information* (PMI) based on search-engine hits counts (Turney, 2001). WOODWARD's PMI-Based Filter (PBF) uses PMI scores as features in a learned filter for predicates. Following Turney, we use the formula below for the PMI between two terms $t1$ and $t2$:

$$PMI(t1, t2) = log\left(\frac{P(t1 \wedge t2)}{P(t1)P(t2)}\right) \qquad (1)$$

We use PMI scores to judge the semantic plausibility of an RC conjunct as follows. We construct a number of different phrases, which we call *discriminator phrases*, from the name of the relation and the head words of each argument. For example, the prepositional attachment "operations of 65 cents" would yield phrases like "operations of" and "operations of * cents". (The '*' character is a wildcard in the Google interface; it can match any single word.) We then collect hitcounts for each discriminator phrase, as well as for the relation name and each argument head word, and compute a PMI score for each phrase, using the phrase's hitcount as the numerator in Equation 1.

Given a set of such PMI scores for a single relation, we apply a learned classifier to decide if the PMI scores justify calling the relation implausible.

This classifier (as well as all of our other ones) is trained on a set of sentences from TREC and the Penn Treebank; our training and test sets are described in more detail in section 3. We parsed each sentence automatically using Daniel Bikel's implementation of the Collins parsing model,[1] trained on sections 2–21 of the Penn Treebank, and then applied our semantic interpreter algorithm to come up with a set of relations. We labeled each relation by hand for correctness. Correct relations are positive examples for our classifier, incorrect relations are negative examples (and likewise for all of our other classifiers). We used the LIBSVM software package[2] to learn a Gaussian-kernel support vector machine model from the PMI scores collected for these relations. We can then use the classifier to predict if a relation is correct or not depending on the various PMI scores we have collected.

Because we require different discriminator phrases for preposition relations and verb relations, we actually learn two different models. After extensive experimentation, optimizing for training set accuracy using leave-one-out cross-validation, we ended up using only two patterns for verbs: "$noun\ verb$" ("$verb\ noun$" for non-subjects) and "$noun$ * $verb$" ("$verb$ * $noun$" for non-subjects). We use the PMI scores from the argument whose PMI values add up to the lowest value as the features for a verb relation, with the intuition being that the relation is correct only if every argument to it is valid.

For prepositions, we use a larger set of patterns. Letting $arg1$ and $arg2$ denote the head words of the two arguments to a preposition, and letting $prep$ denote the preposition itself, we used the patterns "$arg1\ prep$", "$arg1\ prep$ * $arg2$", "$arg1$ $prep$ the $arg2$", "$arg1$ * $arg2$", and, for verb attachments, "$arg1$ it $prep\ arg2$" and "$arg1$ them $prep\ arg2$". These last two patterns are helpful for preposition attachments to strictly transitive verbs.

2.4 The Verb Arity Sampling Test

In our training set from the Penn Treebank, 13% of the time the Collins parser chooses too many or too few arguments to a verb. In this case, checking the PMI between the verb and each argument independently is insufficient, and there is not enough data to find hitcounts for the verb and all of its arguments at once. We therefore use a different type of filter in order to detect these errors, which we call the Verb Arity Sampling Test (VAST).

Instead of testing a verb to see if it can take a *particular* argument, we test if it can take a certain *number* of arguments. The verb predicate *producing(VP1, NP3, NP2, NP1)* in interpretation 2 of Figure 2, for example, has too many arguments. To check if this predicate can actually take three noun phrase arguments, we can construct a common phrase containing the verb, with the property that *if the verb can take three NP arguments, the phrase will often be followed by a NP in text, and vice versa*. An example of such a phrase is "which it is producing." Since "which" and "it" are so common, this phrase will appear many times on the Web. Furthermore, for verbs like "producing," there will be very few sentences in which this phrase is followed by a NP (mostly temporal noun phrases like "next week"). But for verbs like "give" or "name," which can accept three noun phrase arguments, there will be significantly more sentences where the phrase is followed by a NP.

The VAST algorithm is built upon this observation. For a given verb phrase, VAST first counts the number of noun phrase arguments. The Collins parser also marks clause arguments as being essential by annotating them differently. VAST counts these as well, and considers the sum of the noun and clause arguments as the number of *essential arguments*. If the verb is passive and the number of essential arguments is one, or if the verb is active and the number of essential arguments is two, VAST performs no check. We call these *strictly transitive* verb relations. If the verb is passive and there are two essential arguments, or if the verb is active and there are three, it performs the ditransitive check below. If the verb is active and there is one essential argument, it does the intransitive check described below. We call these two cases collectively *nontransitive* verb relations. In both cases, the checks produce a single real-valued score, and we use a linear kernel SVM to identify an appropriate threshold such that predicates above the threshold have the correct arity.

The ditransitive check begins by querying Google for two hundred documents containing the phrase "which it $verb$" or "which they $verb$". It downloads each document and identifies the sentences containing the phrase. It then POS-tags and NP-chunks the sentences using a maximum entropy tagger and chunker. It filters out any sen-

[1] http://www.cis.upenn.edu/∼dbikel/software.html

[2] http://www.csie.ntu.edu.tw/∼cjlin/libsvm/

tences for which the word "which" is preceded by a preposition. Finally, if there are enough sentences remaining (more than ten), it counts the number of sentences in which the verb is directly followed by a noun phrase chunk, which we call an extraction. It then calculates the ditransitive score for verb v as the ratio of the number of extractions E to the number of filtered sentences F:

$$ditransitiveScore(v) = \frac{E}{F} \tag{2}$$

The intransitive check performs a very similar set of operations. It fetches up to two hundred sentences matching the phrases "but it $verb$" or "but they $verb$", tags and chunks them, and extracts noun phrases that directly follow the verb. It calculates the intransitive score for verb v using the number of extractions E and sentences S as:

$$intransitiveScore(v) = 1 - \frac{E}{S} \tag{3}$$

2.5 TextRunner Filter

TextRunner is a new kind of web search engine. Its design is described in detail elsewhere (Cafarella et al., 2006), but we utilize its capabilities in WOODWARD. TextRunner provides a search interface to a set of over a billion triples of the form *(object string, predicate string, object string)* that have been extracted automatically from approximately 90 million documents to date. The search interface takes queries of the form $(string1, string2, string3)$, and returns all tuples for which each of the three tuple strings contains the corresponding query string as a substring.

TextRunner's object strings are very similar to the standard notion of a noun phrase chunk. The notion of a predicate string, on the other hand, is loose in TextRunner; a variety of POS sequences will match the patterns for an extracted relation. For example, a search for tuples with a predicate containing the word 'with' will yield the tuple *(risks, associated with dealing with, waste wood)*, among thousands of others.

TextRunner embodies a trade-off with the PMI method for checking the validity of a relation. Its structure provides a much more natural search for the purpose of verifying a semantic relationship, since it has already arranged Web text into predicates and arguments. It is also much faster than querying a search engine like Google, both because we have local access to it and because commercial search engines tightly limit the number of queries an application may issue per day. On the other hand, the TextRunner index is at present still about two orders of magnitude smaller than Google's search index, due to limited hardware.

The TextRunner semantic filter checks the validity of an RC conjunct in a natural way: it asks TextRunner for the number of tuples that match the argument heads and relation name of the conjunct being checked. Since TextRunner predicates only have two arguments, we break the conjunct into trigrams and bigrams of head words, and average over the hitcounts for each. For predicate $P(A_1, \ldots, A_n)$ with $n \geq 2$, the score becomes

$$\begin{aligned} &TextRunnerScore = \\ &\quad \frac{1}{n-1}\sum_{i=2}^{n} hits(A_1, P, A_i) \\ &\quad + \frac{1}{n}(hits(A_1, P,) + \sum_{i=2}^{n} hits(, P, A_i)) \end{aligned}$$

As with PBF, we learn a threshold for good predicates using the LIBSVM package.

2.6 Question Answering Filter

When parsing questions, an additional method of detecting incorrect parses becomes available: use a question answering (QA) system to find answers. If a QA system using the parse can find an answer to the question, then the question was probably parsed correctly.

To test this theory, we implemented a lightweight, simple, and fast QA system that directly mirrors the semantic interpretation. It relies on TextRunner and KnowItNow (Cafarella et al., 2005) to quickly find possible answers, given the *relational conjunction* (RC) of the question. KnowItNow is a state of the art Information Extraction system that uses a set of domain independent patterns to efficiently find hyponyms of a class.

We formalize the process as follows: define a question as a set of variables X_i corresponding to noun phrases, a set of noun type predicates $T_i(X_i)$, and a set of relational predicates $P_i(X_{i1}, ..., X_{ik})$ which relate one or more variables and constants. The conjunction of type and relational predicates is precisely the RC.

We define an answer as a set of values for each variable that satisfies all types and predicates

$$ans(x_1, ..., x_n) = \bigwedge_i T_i(x_i) \wedge \bigwedge_j P_j(x_{j1}, ..., x_{jk})$$

The algorithm is as follows:

1. Compute the RC of the question sentence.

2. $\forall i$ find instances of the class T_i for possible values for X_i, using KnowItNow.

3. $\forall j$ find instances of the relation predicate $P_j(x_{j1}, ..., x_{jk})$. We use TextRunner to efficiently find objects that are related by the predicate P_j.

4. Return all tuples that satisfy $ans(x_1, ..., x_n)$

The QA semantic filter runs the Question Answering algorithm described above. If the number of returned answers is above a threshold (1 in our case), it indicates the question has been parsed correctly. Otherwise, it indicates an incorrect parse. This differs from the TextRunner semantic filter in that it tries to find subclasses and instances, rather than just argument heads.

2.7 The WOODWARD Filter

Each of the above semantic filters has its strengths and weaknesses. On our training data, TextRunner had the most success of any of the methods on classifying verb relations that did not have arity errors. Because of sparse data problems, however, it was less successful than PMI on preposition relations. The QA system had the interesting property that when it predicted an interpretation was correct, it was always right; however, when it made a negative prediction, its results were mixed.

WOODWARD combines the four semantic filters in a way that draws on each of their strengths. First, it checks if the sentence is a question that does not contain prepositions. If so, it runs the QA module, and returns true if that module does.

After trying the QA module, WOODWARD checks each predicate in turn. If the predicate is a preposition relation, it uses PBF to classify it. For nontransitive verb relations, it uses VAST. For strictly transitive verb relations, it uses TextRunner. WOODWARD accepts the RC if every relation is predicted to be correct; otherwise, it rejects it.

3 Experiments

In our experiments we tested the ability of WOODWARD to detect bad parses. Our experiments proceeded as follows: we parsed a set of sentences, ran the semantic interpreter on them, and labeled each parse and each relation in the resulting RCs for correctness. We then extracted all of the necessary information from the Web and TextRunner. We divided the sentences into a training and test set, and trained the filters on the labeled RCs from the training sentences. Finally, we ran each of the filters and WOODWARD on the test set to predict which parses were correct. We report the results below, but first we describe our datasets and tools in more detail.

3.1 Datasets and Tools

Because question-answering is a key application, we began with data from the TREC question-answering track. We split the data into a training set of 61 questions (all of the TREC 2002 and TREC 2003 questions), and a test set of 55 questions (all list and factoid questions from TREC 2004). We preprocessed the questions to remove parentheticals (this affected 3 training questions and 1 test question). We removed 12 test questions because the Collins parser did not parse them as questions,[3] and that error was too easy to detect. 25 training questions had the same error, but we left them in to provide more training data.

We used the Penn Treebank as our second data set. Training sentences were taken from section 22, and test sentences from section 23. Because PBF is time-consuming, we took a subset of 100 sentences from each section to expedite our experiments. We extracted from each section the first 100 sentences that did not contain conjunctions, and for which all of the errors, if any, were contained in preposition and verb relations.

For our parser, we used Bikel's implementation of the Collins parsing model, trained on sections 2-21 of the Penn Treebank. We only use the top-ranked parse for each sentence. For the TREC data only, we first POS-tagged each question using Ratnaparkhi's MXPOST tagger. We judged each of the TREC parses manually for correctness, but scored the Treebank parses automatically.

3.2 Results and Discussion

Our semantic interpreter was able to produce the appropriate RC for every parsed sentence in our data sets, except for a few minor cases. Two idiomatic expressions in the WSJ caused the semantic interpreter to find noun phrases outside of a clause to fill gaps that were not actually there. And in several sentences with infinitive phrases, the semantic interpreter did not find the extracted subject of the infinitive expression. It turned out that none of these mistakes caused the filters to reject correct parses, so we were satisfied that our results mainly reflect the performance of the filters, rather than the interpreter.

[3]That is, the root node was neither SBARQ nor SQ.

Relation Type	num. correct	num. incorrect	PBF acc.	VAST acc.	TextRunner acc.
Nontrans. Verb	41	35	0.54	0.66	0.52
Other Verb	126	68	0.72	N/A	0.73
Preposition	183	58	0.73	N/A	0.76

Table 1: **Accuracy of the filters on three relation types in the TREC 2004 questions and WSJ data.**

			Baseline			WOODWARD			
	sents.	parser eff.	filter prec.	filter rec.	F1	filter prec.	filter rec.	F1	red. err.
trec	43	54%	0.54	1.0	0.70	0.82	1.0	0.90	**67%**
wsj	100	45%	0.45	1.0	0.62	0.58	0.88	0.70	**20%**

Table 2: **Performance of WOODWARD on different data sets. Parser efficacy reports the percentage of sentences that the Collins parser parsed correctly. See the text for a discussion of our baseline and the precision and recall metrics. We weight precision and recall equally in calculating F1. Reduction in error rate (red. err.) reports the relative decrease in error (error calculated as $1 - $F1) over baseline.**

In Table 1 we report the accuracy of our first three filters on the task of predicting whether a relation in an RC is correct. We break these results down into three categories for the three types of relations we built filters for: strictly transitive verb relations, nontransitive verb relations, and preposition relations. Since the QA filter works at the level of an entire RC, rather than a single relation, it does not apply here. These results show that the trends on the training data mostly held true: VAST was quite effective at verb arity errors, and TextRunner narrowly beat PBF on the remaining verb errors. However, on our training data PBF narrowly beat TextRunner on preposition errors, and the reverse was true on our test data.

Our QA filter predicts whether a full parse is correct with an accuracy of 0.76 on the 17 TREC 2004 questions that had no prepositions. The Collins parser achieves the same level of accuracy on these sentences, so the main benefit of the QA filter for WOODWARD is that it never misclassifies an incorrect parse as a correct one, as was observed on the training set. This property allows WOODWARD to correctly predict a parse is correct whenever it passes the QA filter.

Classification accuracy is important for good performance, and we report it to show how effective each of WOODWARD's components is. However, it fails to capture the whole story of a filter's performance. Consider a filter that simply predicts that every sentence is incorrectly parsed: it would have an overall accuracy of 55% on our WSJ corpus, not too much worse than WOODWARD's classification accuracy of 66% on this data. However, such a filter would be useless because it filters out every correctly parsed sentence.

Let the *filtered set* be the set of sentences that a filter predicts to be correctly parsed. The performance of a filter is better captured by two quantities related to the filtered set: first, how "pure" the filtered set is, or how many good parses it contains compared to bad parses; and second, how wasteful the filter is in terms of losing good parses from the original set. We measure these two quantities using metrics we call *filter precision* and *filter recall*. Filter precision is defined as the ratio of correctly parsed sentences in the filtered set to total sentences in the filtered set. Filter recall is defined as the ratio of correctly parsed sentences in the filtered set to correctly parsed sentences in the unfiltered set. Note that these metrics are quite different from the labeled constituent precision/recall metrics that are typically used to measure statistical parser performance.

Table 2 shows our overall results for filtering parses using WOODWARD. We compare against a baseline model that predicts every sentence is parsed correctly. WOODWARD outperforms this baseline in precision and F1 measure on both of our data sets.

Collins (2000) reports a decrease in error rate of 13% over his original parsing model (the same model as used in our experiments) by performing a discriminative reranking of parses. Our WSJ test set is a subset of the set of sentences used in Collins' experiments, so our results are not directly comparable, but we do achieve a roughly similar decrease in error rate (20%) when we use our filtered precision/recall metrics. We also measured the labeled constituent precision and recall of both the original test set and the filtered set, and found a decrease in error rate of 37% according to this metric (corresponding to a jump in F1 from 90.1 to 93.8). Note that in our case, the error is reduced by throwing out bad parses, rather than trying to fix them. The 17% difference between the two decreases in error rate is probably due to the fact that WOODWARD is more likely to detect the worse parses in the original set, which contribute a proportionally larger share of error in labeled constituent precision/recall in the original test set.

WOODWARD performs significantly better on the TREC questions than on the Penn Treebank data. One major reason is that there are far more clause adjuncts in the Treebank data, and adjunct errors are intrinsically harder to detect. Consider the Treebank sentence: "The S&P pit stayed locked at its 30-point trading limit as the Dow average ground to its final 190.58 point loss Friday." The parser incorrectly attaches the clause beginning "as the Dow ... " to "locked", rather than to "stayed." Our current methods aim to use key words in the clause to determine if the attachment is correct. However, with such clauses there is no single key word that can allow us to make that determination. We anticipate that as the paradigm matures we and others will design filters that can use more of the information in the clause to help make these decisions.

4 Conclusions and Future Work

Given a parse of a sentence, WOODWARD constructs a representation that identifies the key semantic relationships implicit in the parse. It then uses a set of Web-based sampling techniques to check whether these relationships are plausible. If any of the relationships is highly implausible, WOODWARD concludes that the parse is incorrect. WOODWARD successfully detects common errors in the output of the Collins parser including verb arity errors as well as preposition and verb attachment errors. While more extensive experiments are clearly necessary, our results suggest that the paradigm of *Web-based semantic filtering* could substantially improve the performance of statistical parsers.

In future work, we hope to further validate this paradigm by constructing additional semantic filters that detect other types of errors. We also plan to use semantic filters such as WOODWARD to build a large-scale corpus of automatically-parsed sentences that has higher accuracy than can be achieved today. Such a corpus could be used to re-train a statistical parser to improve its performance. Beyond that, we plan to embed semantic filtering into the parser itself. If semantic filters become sufficiently accurate, they could rule out enough erroneous parses that the parser is left with just the correct one.

Acknowledgements

This research was supported in part by NSF grant IIS-0312988, DARPA contract NBCHD030010, ONR grant N00014-02-1-0324 as well as gifts from Google, and carried out at the University of Washington's Turing Center.

References

J. Allen. 1995. *Natural Language Understanding*. Benjamin/Cummings Publishing, Redwood City, CA, 2nd edition.

G. Attardi, A. Cisternino, F. Formica, M. Simi, and A. Tommasi. 2001. PiQASso: Pisa Question Answering System. In *TREC*.

J. Bos, S. Clark, M. Steedman, J. R. Curran, and J. Hockenmaier. 2004. Wide-coverage semantic representations from a CCG parser. In *COLING*.

Michael J. Cafarella, Doug Downey, Stephen Soderland, and Oren Etzioni. 2005. KnowItNow: Fast, scalable information extraction from the web. In *HLT-EMNLP*.

M. J. Cafarella, M. Banko, and O. Etzioni. 2006. Relational web search. *UW Tech Report 06-04-02*.

M. Collins. 2000. Discriminative reranking for natural language parsing. In *ICML*, pages 175–182.

C. C. T. Kwok, O. Etzioni, and D. S. Weld. 2001. Scaling question answering to the web. In *WWW*.

M. Lapata and F. Keller. 2005. Web-based models for natural language processing. *ACM Transactions on Speech and Language Processing*, 2:1–31.

K. Markert, N. Modjeska, and M. Nissim. 2003. Using the web for nominal anaphora resolution. In *EACL Workshop on the Computational Treatment of Anaphora*.

D. Moldovan, C. Clark, S. Harabagiu, and S. Maiorano. 2003. Cogex: A logic prover for question answering. In *HLT*.

K. Toutanova, C. D. Manning, D. Flickinger, and S. Oepen. 2005. Stochastic HPSG parse disambiguation using the Redwoods Corpus. *Journal of Logic and Computation*.

P.D. Turney. 2001. Mining the Web for Synonyms: PMI–IR versus LSA on TOEFL. *Lecture Notes in Computer Science*, 2167:491–502.

M. Volk. 2001. Exploiting the WWW as a corpus to resolve PP attachment ambiguities. In *Corpus Linguistics*.