Clustering Words by Syntactic Similarity Improves Dependency Parsing of Predicate-Argument Structures

Kenji Sagae and **Andrew S. Gordon**
Institute for Creative Technologies
University of Southern California
13274 Fiji Way
Marina del Rey, CA 90292
{sagae,gordon}@ict.usc.edu

Abstract

We present an approach for deriving syntactic word clusters from parsed text, grouping words according to their unlexicalized syntactic contexts. We then explore the use of these syntactic clusters in leveraging a large corpus of trees generated by a high-accuracy parser to improve the accuracy of another parser based on a different formalism for representing a different level of sentence structure. In our experiments, we use phrase-structure trees to produce syntactic word clusters that are used by a predicate-argument dependency parser, significantly improving its accuracy.

1 Introduction

Syntactic parsing of natural language has advanced greatly in recent years, in large part due to data-driven techniques (Collins, 1999; Charniak, 2000; Miyao and Tsujii, 2005; McDonald et al., 2005; Nivre et al., 2007) coupled with the availability of large treebanks. Several recent efforts have started to look for ways to go beyond what individual annotated data sets and individual parser models can offer, looking to combine diverse parsing models, develop cross-framework interoperability and evaluation, and leverage the availability of large amounts of text available. Two research directions that have produced promising improvements on the accuracy of data-driven parsing are: (1) combining different parsers using ensemble techniques, such as voting (Henderson and Brill, 1999; Sagae and Lavie, 2006; Hall et al., 2007) and stacking (Nivre and McDonald, 2008; Martins et al., 2008), and (2) semi-supervised learning, where unlabeled data (plain text) is used in addition to a treebank (McClosky et al., 2006; Koo et al., 2008).

In this paper we explore a new way to obtain improved parsing accuracy by using a large amount of unlabeled text and two parsers that use different ways of representing syntactic structure. In contrast to previous work where automatically generated constituent trees were used directly to train a constituent parsing model (McClosky et al., 2006), or where word clusters were derived from a large corpus of plain text to improve a dependency parser (Koo et al., 2008), we use a large corpus of constituent trees (previously generated by an accurate constituent parser), which we use to produce *syntactically derived clusters* that are then used to improve a transition-based parser that outputs dependency graphs that reflect predicate-argument structure where words may be dependents of more than one parent. This type of representation is more general than dependency trees (Sagae and Tsujii, 2008; Henderson et al., 2008), and is suitable for representing both surface relations and long-distance dependencies (such as control, it-cleft and tough movement).

The first contribution of this work is a novel approach for deriving syntactic word clusters from parsed text, grouping words by the general syntactic contexts where they appear, and not by n-gram word context (Brown et al., 1992) or by immediate dependency context (Lin, 1998). Unlike in clustering approaches that rely on *lexical context* (either linear or grammatical) to group words, resulting in a notion of word similarity that blurs syntactic and semantic characteristics of lexical items, we use *unlexicalized syntactic context*, so that words are clustered based only on their syntactic behavior. This way, we attempt to generate clusters that are more conceptually similar to part-of-speech tags or supertags

Proceedings of the 11th International Conference on Parsing Technologies (IWPT), pages 192–201,
Paris, October 2009.

(Bangalore and Joshi, 1999), but organized hierarchically to provide tagsets with varying levels of granularity.

Our second contribution is a methodology for leveraging a high-accuracy parser to improve the accuracy of a parser that uses a different formalism (that represents different structural information), without the need to process the input with both parsers at run-time. In our experiments, we show that we can improve the accuracy of a fast dependency parser for predicate-argument structures by using a corpus which was previously automatically annotated using a highly accurate but considerably slower phrase-structure tree parser. This is accomplished by using the slower parser only to parse the data used to create the syntactic word clusters. During run-time, the dependency parser uses these clusters, which encapsulate syntactic knowledge from the phrase-structure parser. Although our experiments focus on the use of phrase-structure and dependency parsers, the same framework can be easily applied to data-driven parsing using other syntactic formalisms, such as CCG or HPSG.

2 Clustering by Syntactic Similarity

We developed a new approach to clustering words according to their syntactic similarity. Our method involves the use of hierarchical agglomerate clustering techniques using the calculated *syntactic* distance between words. Syntactic distance between words is computed as the cosine distance between vector representations of the frequency of unique parse tree paths emanating from the word in a corpus of parse trees. In this research, we employ a novel encoding of syntactic parse tree paths that includes direction information and non-terminal node labels, but does not include lexical information or part-of-speech tags. Consequently, the resulting hierarchy groups words that appear in similar places in similar parse trees, regardless of its assigned part-of-speech tag. In this section we describe our approach in detail.

2.1 Parse tree path representation

Gordon and Swanson (2007) first described a corpus-based method for calculating a measure of syntactic similarity between words, and demonstrated its utility in improving the performance of a syntax-based Semantic Role Labeling system. The central idea behind their approach was that *parse tree paths* could be used as features for describing a word's grammatical behavior.

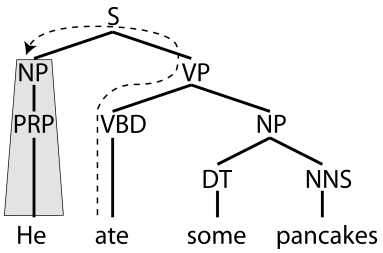

Figure 1: An example parse tree path from the verb *ate* to the argument NP *He*, represented as ↑VBD↑VP↑S↓NP.

Parse tree paths are descriptions of tree transitions from a terminal (e.g. a verb) to a different node in a constituent parse tree of a sentence. Parse tree paths gained popularity in early Semantic Role Labeling research (Gildea and Jurafsky, 2002), where they were used as features describing the relationship between a verb and a particular semantic role label. For example, Figure 1 illustrates a parse tree path between a verb and a semantically related noun phrase.

Gordon and Swanson viewed parse tree paths as features that could be used to describe the syntactic contexts of words in a corpus. In their approach, all of the possible parse tree paths that begin at a given word were identified in a large set of automatically generated constituent parse trees. The normalized frequency counts of unique parse tree paths were combined into a feature vector that describes the location that the given word appears in the set of parse trees. This syntactic profile was then compared with other profiles using a cosine distance function, producing a quantitative value of word similarity. In this manner, the syntactic similarity between the verb "pluck" and the verb "whisk" was calculated as 0.849.

One drawback of the approach of Gordon and Swanson was the inclusion of part-of-speech tags in the encoding of the parse tree paths. As a consequence, the cosine distance between words of different classes was always zero, regardless of their similarities in the remainder of the paths. To correct this problem in our current research, we removed part-of-speech tags from the encoding of parse tree paths, deleting the tag that begins each path and replacing tags when they appear at the end of a path with a generic terminal label.

A second drawback of the approach of Gordon and Swanson is that the path directionality is underspecified. Consider the parse tree paths that

emanate from each of the words "some" and "pancakes" in Figure 1. In the original encoding, the paths for each of these words would be identical (if the part of speech tags were removed), despite their unique locations in this parse tree. To correct this problem in our current research, we elaborated the original set of two path identifiers (↑ and ↓) to a set of six tags that included information about the direction of the transition. Up-Right (↗) and Down-Left (↙) transition are used to and from nodes that are the first constituent of a non-terminal. Up-Left (↖) and Down-Right (↘) transitions are used to and from nodes that are the last constituent of a non-terminal. Transitions to and from all other constituent nodes are labeled Up-Middle (↑) or Down-Middle (↓), accordingly. For example, we represent the parse tree path depicted in Figure 1 as: ↗VP↖S↙NP.

2.2 Profiles for BLLIP WSJ Corpus words

As in the previous work of Gordon and Swanson (2007), we characterize the syntactic properties of words as the normalized frequency of unique parse tree paths emanating from the word in a large corpus of syntactic parse trees.

In our research, we used the Brown Laboratory for Linguistic Information Processing (BLLIP) 1987-89 WSJ Corpus Release 1 (Charniak et al., 2000), which contains approximately 30 million words of Wall Street Journal news articles, parsed with Charniak (2000) parser. Although the trees in the BLLIP corpus are enriched with function tags and empty nodes, we remove this information, leaving only the trees produced by the Charniak parser. We identified the top five thousand most frequent words (or, more generally, *types*, since these also include other sequences of characters, such as numbers and punctuation) in this corpus, treating words that differed in capitalization or in assigned part-of-speech tag as separate types. These five thousand types correspond to approximately 85% of the tokens in the BLLIP corpus. For each token instance of each of these five thousand types, we identified every occurring parse tree path emanating from the token in each of the sentences in which it appeared. The most frequent type was the comma, which appeared 2.2 million times and produced 118 million parse tree paths. The least frequent token in this set was the singular noun "pollution," with 731 instances producing 35,185 parse tree paths.

To generate syntactic profiles for a given type, the frequency of unique parse tree paths was tabulated, and then normalized by dividing this frequency by the number of tokens of that type in the corpus. To reduce the dimensionality of these normalized frequency vectors, parse tree paths that appeared in less than 0.2% of the instances were ignored. This threshold value produced vectors with dimensionality that was comparable across all five thousand types, and small enough to process given our available computational resources. The mean vector size was 2,228 dimensions, with a standard deviation of 734.

2.3 Distance calculation and clustering

Pairwise distances between each of the five thousand types were computed as the cosine distance between their profile vectors. We then grouped similar types using hierarchical agglomerate clustering techniques, where distance between clusters is calculated as mean distance between elements of each cluster (average link clustering).

The three most similar types (the first 2 clustering steps) consisted of the capitalized subordinating conjunctions "Although," "While," and "Though." The two most dissimilar types (the last to be included in any existing cluster) were the symbol "@" and the question mark.

2.4 Cluster label selection

Hierarchical agglomerate clustering produces a binary-branching tree structure, where each branch point is ordered according to a similarity value between 0 and 1. In our clustering of the top five thousand most frequent types in the BLLIP corpus, there are five thousand leaf nodes representing individual tokens, and 4999 branch points that cluster these types into a single tree. We label each of these 4999 branch points, and treat these cluster labels as features of the types that they dominate. For example, the singular noun "house" participates in 114 clusters of increasing size. The syntactic features of this type can therefore be characterized by 114 cluster labels, which overlap with varying degrees with other tokens in the set.

We view these cluster labels as conceptually similar to traditional part-of-speech tags in that they are indicative of the syntactic contexts in which words are likely to appear. Because words are clustered based on their unlexicalized syntactic contexts, the resulting clusters are more likely to reflect purely syntactic information than are clusters derived from lexical context, such as adjacent words (Brown et al., 1992) or immediate head-word (Lin, 1998). However, the extent

to which these syntactic contexts are specified can vary from a more general to a more fine-grained level than that of parts-of-speech. As clusters become more fine-grained, they become more similar to supertags (Bangalore and Joshi, 1999). Clusters that represent more specific syntactic contexts can encode information about, for example, subcategorization. As these labels are derived empirically from a large corpus of syntactic parse trees, they accurately represent syntactic distinctions in real discourse at different granularities, in contrast to the single arbitrary granularity of theoretically derived part-of-speech tags used in existing treebanks (Marcus et al., 1993).

While it is sometimes useful to view types as having multiple part-of-speech tags at different levels of granularity (e.g. the 114 tags for the token "house"), it is often useful to select a single level of granularity to use across all tokens. For example, it is useful to know which one of the 114 cluster labels for "house" to use if exactly 100 part-of-speech distinctions are to be made among tokens in the set. These cluster labels can be identified by slicing the tree at the level for which there are exactly 100 branches, then using the label of the first branch point in each branch as the label for all of its leaf-node types, or the leaf-node itself in the case where no further branching exists. Given our hierarchical clustering, there are five thousand different ways to slice the tree in this manner, yielding sets of cluster labels (and un-clustered types) that vary in size from 1 to 5000. We identified these sets for use in the experiments described in the next sections.

Figure 2 shows a dendrogram representation of the cluster tree when it is sliced to produce exactly 60 clusters, 19 of which are individual types. For the other 41 clusters, we show only the most frequent word in the cluster and the number of additional words in the cluster. The scale line in the lower left of Figure 2 indicates the horizontal length of a calculated similarity between clusters of 0.1.

3 Transition-based dependency parsing with word clusters

The clusters obtained with the approach described in section 2 provide sets of syntactic tags with varying levels of granularity. Previous work by Koo et al. (2008) and Miller et al. (2004) suggests that different levels of cluster granularity may be useful in natural language

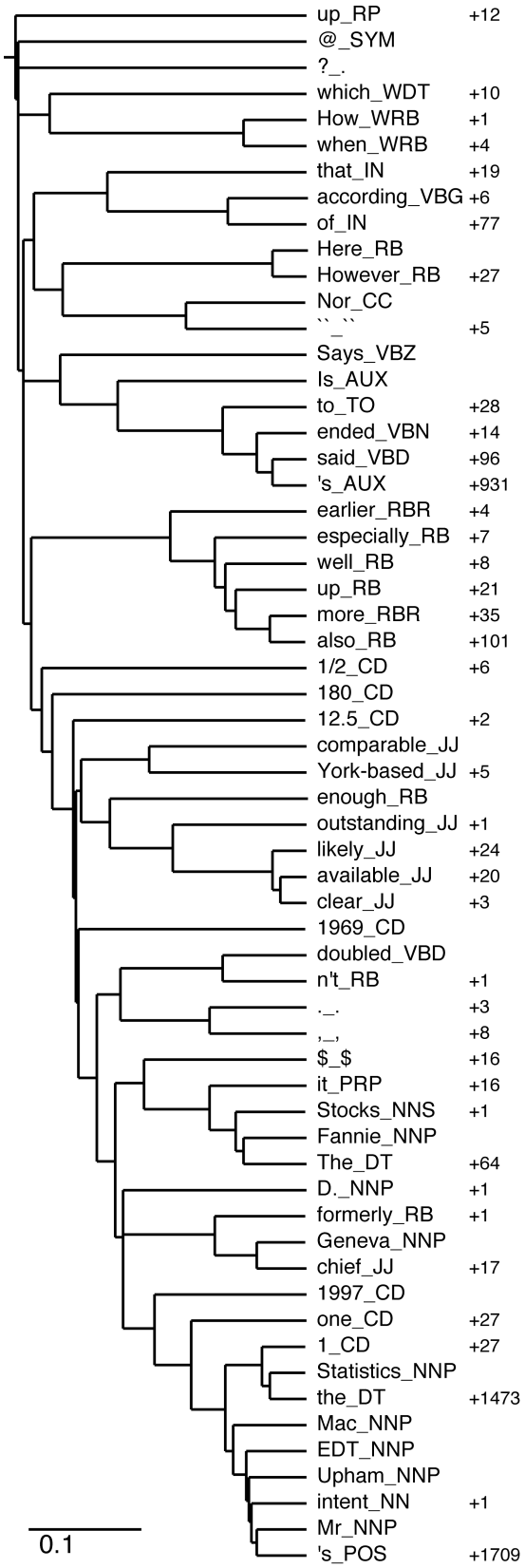

Figure 2: A hierarchical clustering of the top five thousand tokens in the BLLIP corpus, cut at 60 clusters.

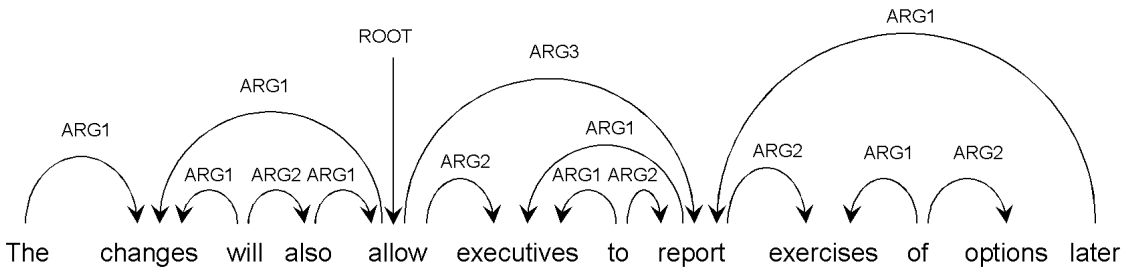

Figure 3: Predicate-argument dependency structure following the HPSG Treebank standard.

processing tasks with discriminative training. We add the syntactic clusters as features in a transition-based parser that uses a classifier to decide among shift/reduce parser actions based on the local context of the decision. This transition-based parsing approach has been found to be efficient and accurate in dependency parsing of surface syntactic dependencies (Yamada and Matsumoto, 2003; Nivre et al., 2004; Hall et al., 2007) and predicate-argument parsing (Henderson et al., 2008; Sagae and Tsujii, 2008).

Our experiments are based on an implementation of Sagae and Tsujii (2008)'s algorithm for basic shift-reduce parsing with multiple heads, which we use to identify predicate-argument dependencies extracted from the HPSG Treebank developed by Miyao et al. (2004). Using this data set allows for a comparison of our results with those obtained in previous work on data-driven HPSG predicate-argument analysis, while demonstrating the use of our clustering approach for cross-framework parser improvement, since the clusters were derived from syntactic trees in Penn Treebank format (as produced by the Charniak parser, without empty nodes, co-indexation or function tags), and used in the identification of HPSG Treebank predicate-argument dependencies. Figure 3 shows a predicate-argument dependency structure following the annotation standard of the HPSG Treebank, where arrows point from head to modifier. We note that unlike in the widely known PropBank (Palmer et al., 2005) predicate-argument structures, argument labels start from ARG1 (not ARG0), and predicate-argument relationships are annotated for all words. One difference between in our implementation is that, instead of maximum entropy classification used by Sagae and Tsujii, we perform parser action classification using the averaged perceptron (Freund and Schapire, 1999; Collins, 2002), which allows for the inclusion of all of Sagae and Tsujii's features, in addition to a set of cluster-based features, while retaining fast training times.

We now describe the parsing approach, starting with the dependency DAG parser that we use as a baseline, followed by how the syntactic cluster features were added to the baseline parser.

3.1 Arc-standard parsing for dependency DAGs

Sagae and Tsujii (2008) describe two algorithms for dependency parsing with words that have multiple heads. Each corresponds to extensions of Nivre (2004)'s arc-standard and arc-eager algorithms for dependency (tree) parsing. In our experiments, we used an implementation of the arc-standard extension.

Nivre's arc-standard dependency parsing algorithm uses a stack to process the input string one word at a time, from left to right, using two general types of parser action: *shift* (push the next input token onto the stack), and *reduce* (create a dependency arc between the top two items on the stack, and pop the item marked as the dependent). Reduce actions are subdivided into *reduce-right* and *reduce-left*, indicating which of the two items on the top of the stack is the head, and which is the dependent in the newly formed dependency arc. These two reduce actions can be further subdivided to reflect what type of dependency arc is created, in the case of labeled dependency parsing. The extension for allowing multiple heads per word consists of the addition a new type of parser action: *attach*, which creates a dependency arc without removing anything from the stack. As with reduce actions, there are two types of attach: *attach-left* which creates a dependency arc between the top two items on the stack such that the item on top is the head, and

right-attach, which creates a dependency arc between the top two items on the stack such that the top item is the dependent, then pops it from the stack and unshifts it back into the input. Finally, this algorithm for unlabeled graphs can be extended to produce labeled dependencies in the same way as Nivre's algorithm, by replacing the reduce and attach actions with sets of actions that perform the reduce or attach operation and also name the label of the arc created. Sagae and Tsujii (2008) provide a more detailed description of the algorithm, including an example that illustrates the new attach actions.

This basic algorithm is only capable of producing labeled directed acyclic graphs where, if the nodes (which correspond to words) are placed on a left to right sequence on a horizontal line in the order in which the words appear in the input sentence, all arcs can be drawn above the nodes without crossing. This corresponds to the notion of projectivity that similarly limits the types of trees produced by Nivre's algorithm. Just as in dependency parsing with tree structures, a way to effectively remove this limitation is the use of pseudo-projective transformations (Nivre and Nilsson, 2005), where arcs that cross have their heads moved towards the root and have their labels edited to reflect this change, often making it reversible. Once crossing arcs have been "lifted" so that no crossing arcs remain, the "projectivized" structures are used to train a parsing model. Projective structures produced by this model can be "deprojectivized" through the use of the edits in the arc labels, in an attempt to produce structures that conform to the scheme in the original data. Sagae and Tsujii also propose a simple *arc reversal* transform, which simply reverses the direction of a dependency arc (editing the arc label to note this change). This transformation, which can be reversed trivially, makes it possible to remove cycles in dependency graphs.

3.2 Baseline features

To create output graph structures for an input sentence, the algorithm described in section 3.1 relies on an oracle that tells it what action to take at each parser state, where the state is the contents of the stack, remaining words in the input, and the dependency arcs formed so far. In grammar-based shift-reduce parsing, this oracle may take the form of a look-up table derived from grammar rules. In our data-driven setting, where the parser learns to choose actions based on examples of correctly parsed data, the (likely imperfect) substitute for the oracle is a classifier that takes features that represent the parser state as input, and produces a matching parser action as output. These features should represent aspects of the parser state that may be informative as to what the corresponding appropriate action is. Our baseline model uses the averaged perceptron with a core set of features derived from the following templates, where *S(n)* denotes the *n-th* item from the top of the stack (for example, *S(1)* is the item on top of the stack), and *I(n)* denotes the next *n-th* input token:

1. For the items *S(1)* and *S(2)*:
    a. the total number of dependents;
    b. the number of dependents to the right of the item;
    c. the number of dependents to the left of the item;
    d. the part-of-speech tag of the rightmost dependent of the item;
    e. the part-of-speech tag of the leftmost dependent of the item;
    f. the arc label of the rightmost dependent of the item;
    g. the arc label of the leftmost dependent of the item;
2. the words in items *S(1), S(2), S(3), I(1)* and *I(2)*;
3. the part-of-speech tags in items *S(1), S(2), S(3), I(1)*, *I(2)* and *I(3)*;
4. the part-of-speech tag of the word i mmediaely to the right of *S(2)*;
5. the part-of-speech tag of the word immediately to the left of *S(1)*;
6. whether an arc exists between *S(1)* and *S(2)*;
7. whether an arc exists between *S(1)* and *I(1)*;
8. the direction of the arc between *S(1)* and *S(2)*, if any;
9. the label of the arc between *S(1)* and *S(2)*, if any;
10. the label of the arc between *S(1)* and *I(1)*, if any;
11. the distance, in linear sequence of words, between *S(1)* and *S(2);*
12. the distance, in linear sequence of words, between *S(1)* and *I(1);*

13. the previous parser action.

In addition to the core set of features, we also use features obtained by concatenating the part-of-speech tags in *S(1), S(2)* and *I(1)* with the features derived from templates 1-6, and additional features derived from selected concatenation of two or three core features.

3.3 Cluster-based features

To take advantage of the clusters that reflect syntactic similarity between words, we assign arbitrary unique labels to each of the hierarchical clusters obtained using the procedure described in section 2. These cluster labels are used to generate additional features that help the parser make its decisions base on the syntactic profile of words. As explained in section 2.4, each there may be several cluster labels (corresponding to clusters of different granularities) associated with each word. To select the set of cluster labels to be used to generate features, we first select a desired granularity for the clusters, and use the set of labels resulting from slicing the cluster tree at the appropriate level, as discussed in section 2.4. We experimented with several levels of cluster granularity using development data, and following Koo et al. (2008), we also experimented with using two sets of cluster labels with different levels of granularity at the same time. Given a specific level of granularity, the cluster-based features we used are:

14. the cluster labels for the words in items *S(1), S(2), S(3), I(1), I(2), I(3);*
15. the cluster labels for the words in the rightmost and leftmost dependents of *S(1)* and *S(2)*;
16. the concatenation of the cluster labels for the words in *S(1), S(2)* and *I(1),* and the features derived from feature templates 1-15.

In experiments where we used two sets of cluster labels corresponding to different levels of granularity, we added all the cluster-based features in 14 and 15 for both sets of labels, and the features in 16 only for the set corresponding to the coarser-grained clusters.

4 Experiments

Following previous experiments with Penn Treebank WSJ data, or annotations derived from it, we used sections 02-21 of the HPSG Treebank as training material, section 22 for development, and section 23 for testing. Only the predicate-argument dependencies were used, not the phrase structures or other information from the HPSG analyses. For all experiments described here, part-of-speech tagging was done separately using a CRF tagger with accuracy of 97.3% on sections 22-24. Our evaluation is based on labeled precision and recall of predicate-argument dependencies. Although accuracy is commonly used for evaluation of dependency parsers, in our task the parser is not restricted to output a fixed number of dependencies. Labeled precision and recall of predicate-argument pairs are also the standard evaluation metrics for data-driven HPSG and CCG parsers (although the predicate-argument pairs extracted from the HPSG Treebank and the CCGBank are specific to their formalisms and not quantitatively comparable).

We started by eliminating cycles from the dependency graphs extracted from the HPSG Treebank by using the arc reversal transform in the following way: for each cycle detected in the data, the shortest arc in the cycle was reversed until no cycles remained. We then applied pseudo-projective transformation to create data that can be used to train our parser, described in section 3. By detransforming the projective graphs generated from gold-standard dependencies, we obtain labeled precision of 98.1% and labeled recall of 97.7%, which is below the accuracy expected for detransformation of syntactic dependency trees. This is expected, since arc crossing occurs more frequently in predicate-argument graphs in the HPSG Treebank than in surface syntactic dependencies.

We first trained a parsing model without cluster-based features, using only the baseline set of features, which was the product of experimentation using the development set. On the test set, this baseline model has labeled precision and recall of 88.7 and 88.2, respectively, slightly below the precision and recall obtained by Sagae and Tsujii on the same data (89.0 precision and 88.5 recall).

We then used the development set to explore the effects of cluster sets with different levels of granularity. The baseline model has precision and recall of 88.6 and 88.0 on the development set. We found that by slicing the cluster tree relatively close to the root, resulting in a set of 50 to 100 distinct cluster labels (corresponding to relatively coarse clusters), we obtain small (0.3 to 0.4), but statistically significant ($p < 0.005$) improvements on precision and recall over the baseline model on the development set. By increasing the number of cluster labels (making the

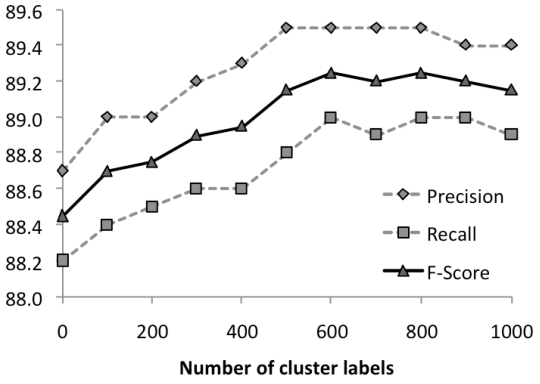

Figure 4: Effect of cluster granularity on labeled the precision and recall of predicate-argument pairs in the development set. The improvement in precision and recall between the baseline (zero cluster labels, where no cluster information is added) and 600 cluster labels is statistically significant ($p < 0.0005$).

distinctions among members of different clusters more fine-grained) in steps of 100, we observed improvements in precision and recall until the point where there were 600 distinct cluster labels. This set of 600 cluster labels produced the highest values of precision and recall (89.5 and 89.0) that we obtained for the development set using only one set of cluster labels. Figure 4 shows how precision, recall and F-score on the development set varied with the number of cluster labels used.

Following Koo et al. (2008), we also experimented with using two sets of cluster labels with different levels of granularity. We found that using the set of 600 labels and an additional set with fewer than 600 labels did not improve or hurt precision and recall. Finer grained clusters with more than 1,000 labels (combined with the set of 600 labels) improved results further. The highest precision and recall figures of 90.1 and 89.6 were obtained with the sets of 600 and 1,400 labels.

We parsed the test set using the best configuration of cluster-based features as determined using the development set (the sets with 600 and 1,400 cluster labels) and obtained 90.2 precision, 89.8 recall and 90.0 f-score, a 13.8% reduction in error over a strong baseline. Table 1 summarizes our results on the test set. For comparison, we also shows results published by Sagae and Tsujii (2008), to our knowledge the highest f-score reported for this test set, and Miyao and Tsujii (2005), who first reported results on this data set.

	Precision	**Recall**	**F-score**
Baseline	88.7	88.2	88.4
Clusters	**90.2**	**89.8**	**90.0**
S & T	89.9	88.5	88.7
Miyao et al.	85.0	84.3	84.6

Table 1: Results obtained on the test set using our baseline model and our best cluster-based features. The results in the bottom two rows are from Sagae and Tsujii (2008) and Miyao and Tsujii (2005).

4.1 Surface dependency parsing with cluster-based features

The parser used in our experiments with HPSG Treebank predicate-argument structures can assign more than one head for a single word, but when the parser is trained using only dependency trees, it behaves in exactly the same way as a parser based on Nivre's arc-standard algorithm, since it never sees examples of attach actions during training. To see whether our clusters can improve surface dependency parsing, and to allow for comparison of our results to a larger body of research on surface dependency parsing, we used dependency trees extracted from the Penn Treebank using the Yamada and Matsumoto (2003) version of the Penn Treebank head-percolation rules to train parsing models that produce dependency trees. However, no tuning of the features or metaparameters was performed; the parser was trained as-is on dependency trees.

We used the standard train, development and test sets splits to train two models, as in our experiments with predicate-argument dependencies: a baseline that uses no cluster information, and a model that uses two sets of clusters that were found to improve results in the development set. The unlabeled accuracy of our baseline model on the test set is 89.96%, which is considerably lower than the best current results. Koo et al. (2008) report 90.84% for a first-order edge-factored model, and 92.02% for a second-order model (and as high as 93.16% with a second-order model enriched with cluster features derived from plain text). Using two sets of clusters, one with 600 and one with 1,200 labels, accuracy improves by 1.32%, to reach 91.28% (a 13.15% reduction in error compared to our baseline). While still below the level of the strongest results for this dataset, it is interesting to see that

the improvement in accuracy over the baseline observed for surface dependency trees is similar to the improvement observed for predicate-argument dependency graphs.

5 Related work

Many aspects of this research were inspired by the recent work of Koo et al. (2008), who reported impressive results on improving dependency parsing accuracy using a second order edge-factored model and word clusters derived from plain text using the Brown et al. (1992) algorithm. Our clustering approach is significantly different, focusing on the use of parsed data to produce strictly syntactic clusters. It is possible that using both types of clusters may be beneficial.

McClosky et al. (2006) used a large corpus of parsed text to obtain improved parsing results through self-training. A key difference in our general framework is that it allows for a parser with one type of syntactic representation to improve the accuracy of a different parser with a different type of formalism. In this regard, our work is related to that of Sagae et al. (2007), who used a stacking-like framework to allow a surface dependency parser to improve an HPSG parser. In that work, however, as in other work that combines different parsers through stacking (Martins et al., 2008; Nivre and McDonald, 2008) or voting (Henderson and Brill, 1999), multiple parsers need to process new text at runtime. In our approach for leveraging diverse parsers, one of the parsers is used only to create a parsed corpus from which we extract clusters of words that have similar syntactic behaviors, and only one parser is needed at run-time.

6 Conclusion

We have presented a novel approach for deriving word clusters based on syntactic similarity, and shown how these word clusters can be applied in a transition-based dependency parser.

Our experiments focused on predicate-argument structures extracted from the HPSG Treebank, which demonstrates that the syntactic clusters are effective in leveraging cross-framework parser representations to improve parsing accuracy. However, we expect that similar accuracy improvements can be obtained in parsing using other frameworks and formalisms, and possibly in other natural language processing tasks.

Acknowledgments

The project or effort described here has been sponsored by the U.S. Army Research, Development, and Engineering Command (RDECOM). Statements and opinions expressed do not necessarily reflect the position or the policy of the United States Government, and no official endorsement should be inferred.

References

Srinivas Bangalore and Aravind K. Joshi. 1999. Supertagging: an approach to almost parsing. *Computational Linguistics* 25, 2 (Jun. 1999), 237-265.

Peter F. Brown, Vincent J. Della Pietra, Peter V. deSouza, Jennifer C. Lai, and Robert L. Mercer. 1992. Class-Based n-gram Models of Natural Language. *Computational Linguistics*, 18(4):467–479.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of the First Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 132–139.

Charniak, E., Blaheta, D., Ge, N., Hall, K., Hale, J., and Johnson, M. (2000) *BLLIP 1987-89 WSJ Corpus Release 1*. Philadelphia, PA: Linguistic Data Consortium.

Michael Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

Michael Collins. 2002. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In *Proceedings of EMNLP*, pages 1–8.

Yoav Freund and Robert E. Schapire. 1999. Large Margin Classification Using the Perceptron Algorithm. *Machine Learning*, 37(3):277–296.

Daniel Gildea and Daniel Jurafsky. 2002. Automatic Labeling of Semantic Roles. *Computational Linguistics* 28(3): 245-288.

Andrew Gordon and Reid Swanson. 2007. Generalizing semantic role annotations across syntactically similar verbs. *Proceedings of the 2007 meeting of the Association for Computational Linguistics (ACL-07)*, Prague, Czech Republic, June 23-30, 2007.

Johan Hall, Jens Nilsson, Joakim Nivre, Gulsen Eryigit, Beata Megyesi, Mattias Nilsson, and Markus Saers. 2007. Single malt or blended? A study in multilingual parser optimization. In *Proceedings of EMNLP-CoNLL*.

James Henderson, Paola Merlo, G. Musillo, and Ivan Titov. 2008. A latent variable model of synchronous parsing for syntactic and semantic dependen-

cies. In *Proceedings of the Shared Task of the Conference on Computational Natural Language Learning (CoNLL)*, pages 178-182. Manchester, UK.

John Henderson and Eric Brill. 1999. Exploiting diversity in natural language processing: combining parsers. In *Proceedings of the Fourth Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Terry Koo, Xavier Carreras and Michael Collins. 2008. Simple semi-supervised dependency parsing. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL-08:HLT),* pages 595-603.

Dekang Lin. 1998. Automatic retrieval and clustering of similar words. In *Proceedings of the 17th international Conference on Computational Linguistics - Volume 2*. Montreal, Quebec, Canada.

Mitchell P. Marcus, Mary A. Marcinkiewicz, Beatrice Santorini. 1993. Building a large annotated corpus of English: The Penn Treebank, *Computational Linguistics*, 19(2), June 1993.

André F. T. Martins, Dipanjan Das, Noah A. Smith, and Eric P. Xing. 2008. Stacking Dependency Parsers. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing, Waikiki*, HI.

David McClosky, Eugene Charniak, and Mark Johnson. 2006. Effective Self-Training for Parsing. In *Proceedings of HLT-NAACL*, pages 152–159.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of ACL*, pages 91–98.

Scott Miller, Jethran Guinness and Alex Zamanian. 2004. Name Tagging withWord Clusters and Discriminative Training. In *Proceedings of HLT-NAACL*, pages 337–342.

Miyao, Yusuke, Takashi Ninomiya, and Jun'ichi Tsujii. 2004. Corpus-oriented grammar development for acquiring a Head-driven Phrase Structure Grammar from the Penn Treebank. In *Proceedings of the International Joint Conference on Natural Language Processing (IJCNLP)*.

Miyao Yusuke and Jun'ichi Tsujii. 2005. Probabilistic disambiguation models for wide-coverage HPSG parsing. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*.

Joakim Nivre.2004. Incrementality in Deterministic Dependency Parsing. In *Incremental Parsing: Bringing Engineering and Cognition Together (Workshop at ACL-2004)*.

Joakim Nivre, Johan Hall, and Jens Nilsson. 2004. Memory-based dependency parsing. In Proceedings of CoNLL, pages 49–56.

Joakim Nivre. and Jens Nilsson. 2005. Pseudo-Projective Dependency Parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 99-106.

Joakim Nivre, Johan Hall, Sandra Kubler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of EMNLP-CoNLL*, pages 915-932.

Nivre, J. and McDonald, R. (2008) Integrating Graph-Based and Transition-Based Dependency Parsers. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL-08: HLT),* 950-958.

Martha Palmer, Dan Gildea and Paul Kingsbury. 2005. The Proposition Bank: A Corpus Annotated with Semantic Roles. *Computational Linguistics*, 31:1.

Kenji Sagae and Alon Lavie. 2006. Parser combination by reparsing. In *Proceedings of NAACL: Short Papers*, pages 129–132.

Kenji Sagae, Yusuke Miyao Jun'ichi and Tsujii. 2007. HPSG Parsing with shallow dependency constraints. In *Proceedings of the 44th Meeting of the Association for Computational Linguistics.*

Kenji Sagae and Jun'ichi Tsujii. 2008. Shift-reduce dependency DAG parsing. In *Proceedings of the International Conference on Computational Linguistics (COLING 2008)*.

Hiroyasu Yamada and Y. Matsumoto. 2003. Statistical Dependency Analysis With Support Vector Machines. In *Proceedings of the Eighth International Workshop on Parsing Technologies (IWPT)*, pages 195–206.