Retrieving Correct Semantic Boundaries in Dependency Structure

Jinho D. Choi
Department of Computer Science
University of Colorado at Boulder
choijd@colorado.edu

Martha Palmer
Department of Linguistics
University of Colorado at Boulder
martha.palmer@colorado.edu

Abstract

This paper describes the retrieval of correct semantic boundaries for predicate-argument structures annotated by dependency structure. Unlike phrase structure, in which arguments are annotated at the phrase level, dependency structure does not have phrases so the argument labels are associated with head words instead: the subtree of each head word is assumed to include the same set of words as the annotated phrase does in phrase structure. However, at least in English, retrieving such subtrees does not always guarantee retrieval of the correct phrase boundaries. In this paper, we present heuristics that retrieve correct phrase boundaries for semantic arguments, called semantic boundaries, from dependency trees. By applying heuristics, we achieved an F1-score of 99.54% for correct representation of semantic boundaries. Furthermore, error analysis showed that some of the errors could also be considered correct, depending on the interpretation of the annotation.

1 Introduction

Dependency structure has recently gained wide interest because it is simple yet provides useful information for many NLP tasks such as sentiment analysis (Kessler and Nicolov, 2009) or machine translation (Gildea, 2004). Although dependency structure is a kind of syntactic structure, it is quite different from phrase structure: phrase structure gives phrase information by grouping constituents whereas dependency structure gives dependency relations between pairs of words. Many dependency relations (e.g., subject, object) have high correlations with semantic roles (e.g., agent, patient), which makes dependency structure suitable for representing semantic information such as predicate-argument structure.

In 2009, the Conference on Computational Natural Language Learning (CoNLL) opened a shared task: the participants were supposed to take dependency trees as input and produce semantic role labels as output (Hajič et al., 2009). The dependency trees were automatically converted from the Penn Treebank (Marcus et al., 1993), which consists of phrase structure trees, using some heuristics (cf. Section 3). The semantic roles were extracted from the Propbank (Palmer et al., 2005). Since Propbank arguments were originally annotated at the phrase level using the Penn Treebank and the phrase information got lost during the conversion to the dependency trees, arguments are annotated on head words instead of phrases in dependency trees; the subtree of each head word is assumed to include the same set of words as the annotated phrase does in phrase structure. Figure 1 shows a dependency tree that has been converted from the corresponding phrase structure tree.

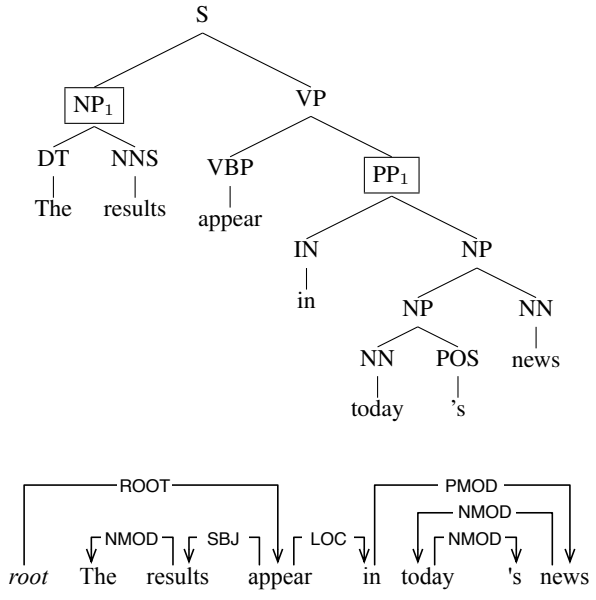


Figure 1: Phrase vs. dependency structure

Proceedings of the Fourth Linguistic Annotation Workshop, ACL 2010, pages 91–99,
Uppsala, Sweden, 15-16 July 2010.

In the phrase structure tree, arguments of the verb predicate *appear* are annotated on the phrases: NP_1 as ARG_0 and PP_1 as ARGM-LOC. In the dependency tree, the arguments are annotated on the head words instead: *results* as the ARG_0 and *in* as the ARGM-LOC. In this example, both PP_1 and the subtree of *in* consist of the same set of words {*in*, *today*, *'s*, *news*} (as is the case for NP_1 and the subtree of *results*); therefore, the phrase boundaries for the semantic arguments, called semantic boundaries, are retrieved correctly from the dependency tree.

Retrieving the subtrees of head words usually gives correct semantic boundaries; however, there are cases where the strategy does not work. For example, if the verb predicate is a gerund or a past-participle, it is possible that the predicate becomes a syntactic child of the head word annotated as a semantic argument of the predicate. In Figure 2, the head word *plant* is annotated as ARG_1 of the verb predicate *owned*, where *owned* is a child of *plant* in the dependency tree. Thus, retrieving the subtree of *plant* would include the predicate itself, which is not the correct semantic boundary for the argument (the correct boundary would be only {*The*, *plant*}).

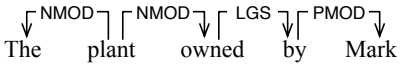


Figure 2: Past-participle example

For such cases, we need some alternative for retrieving the correct semantic boundaries. This is an important issue that has not yet been thoroughly addressed. In this paper, we first show how to convert the Penn Treebank style phrase structure to dependency structure. We then describe how to annotate the Propbank arguments, already annotated in the phrase structure, on head words in the dependency structure. Finally, we present heuristics that correctly retrieve semantic boundaries in most cases. For our experiments, we used the entire Penn Treebank (Wall Street Journal). Our experiments show that it is possible to achieve an F1-score of 99.54% for correct representation of the semantic boundaries.

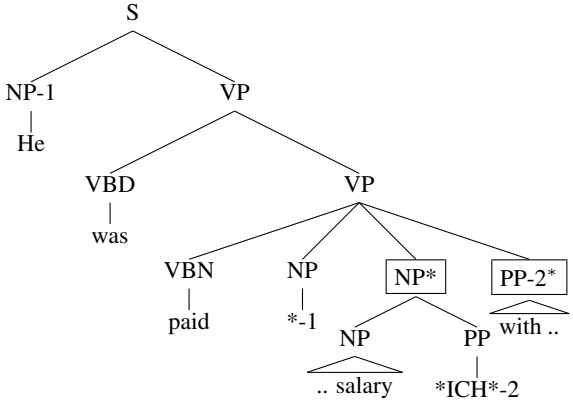
2 Related work

Ekeklint and Nivre (2007) tried to retrieve semantic boundaries by adding extra arcs to dependency trees, so the structure is no longer a tree but a graph. They experimented with the same corpus, the Penn Treebank, but used a different dependency conversion tool, Penn2Malt.[1] Our work is distinguished from theirs because we keep the tree structure but use heuristics to find the boundaries. Johansson (2008) also tried to find semantic boundaries for evaluation of his semantic role labeling system using dependency structure. He used heuristics that apply to general cases whereas we add more detailed heuristics for specific cases.

3 Converting phrase structure to dependency structure

We used the same tool as the one used for the CoNLL'09 shared task to automatically convert the phrase structure trees in the Penn Treebank to the dependency trees (Johansson and Nugues, 2007). The script gives several options for the conversion; we mostly used the default values except for the following options:[2]

- **splitSlash=false**: do not split slashes. This option is taken so the dependency trees preserve the same number of word-tokens as the original phrase structure trees.
- **noSecEdges=true**: ignore secondary edges if present. This option is taken so all siblings of verb predicates in phrase structure become children of the verbs in dependency structure regardless of empty categories. Figure 3 shows the converted dependency tree, which is produced when the secondary edge (*ICH*) is not ignored, and Figure 4 shows the one produced by ignoring the secondary edge. This option is useful because NP* and PP-2* are annotated as separate arguments of the verb predicate *paid* in Propbank (NP* as ARG_1 and PP-2* as ARGM-MNR).



[1] http://stp.lingfil.uu.se/ nivre/research/Penn2Malt.html
[2] http://nlp.cs.lth.se/software/treebank_converter/

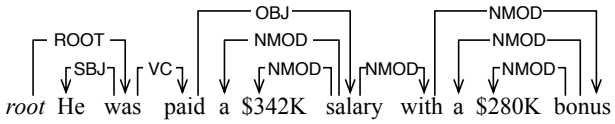


Figure 3: When the secondary edge is not ignored

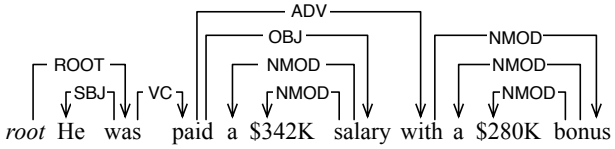


Figure 4: When the secondary edge is ignored

Total 49,208 dependency trees were converted from the Penn Treebank. Although it was possible to apply different values for other options, we found them not helpful in finding correct semantic boundaries of Propbank arguments. Note that some of non-projective dependencies are removed by ignoring the secondary edges. However, it did not make all dependency trees projective; our methods can be applied for either projective or non-projective dependency trees.

4 Adding semantic roles to dependency structure

4.1 Finding the head words

For each argument in the Propbank annotated on a phrase, we extracted the set of words belonging to the phrase. Let this set be S_p. In Figure 1, PP_1 is the ARGM-LOC of *appear* so S_p is {*in*, *today*, *'s*, *news*}. Next, we found a set of head words, say S_d, whose subtrees cover all words in S_p (e.g., S_d = {*in*} in Figure 1). It would be ideal if there existed one head word whose subtree covers all words in S_p, but this is not always the case. It is possible that S_d needs more than one head word to cover all the words in S_p.

Figure 5 shows an algorithm that finds a set of head words S_d whose subtrees cover all words in S_p. For each word w in S_p, the algorithm checks if w's subtree gives the maximum coverage (if w's subtree contains more words than any other subtree); if it does, the algorithm adds w to S_d, removes all words in w's subtree from S_p, then repeats the search. The search ends when all words in S_p are covered by some subtree of a head word in S_d. Notice that the algorithm searches for the minimum number of head words by matching the maximum coverages.

Input: S_p = a set of words for each argument in the Propbank
Output: S_d = a set of head words whose subtrees cover all words in S_p

Algorithm: getHeadWords(S_p)
$S_d = \{\}$
while $S_p \neq \emptyset$ **do**
  max = None
  foreach $w \in S_p$ **do**
    if $|subtree(w)| > |subtree(max)|$ **then**
      $max = w$
  end
  S_d.add(max)
  S_p.removeAll($subtree(max)$)
end
return S_d

Figure 5: Finding the min-set of head words

The algorithm guarantees to find the min-set S_d whose subtrees cover all words in S_p. This gives 100% recall for S_d compared to S_p; however, the precision is not guaranteed to be as perfect. Section 5 illustrates heuristics that remove the over-generated words so we could improve the precision as well.

4.2 Ignoring empty categories

As described in Figures 3 and 4, dependency trees do not include any empty categories (e.g., null elements, traces, PRO's): the empty categories are dropped during the conversion to the dependency trees. In the Penn Treebank, 11.5% of the Propbank arguments are annotated on empty categories. Although this is a fair amount, we decided to ignore them for now since dependency structure is not naturally designed to handle empty categories. Nonetheless, we are in the process of finding ways of automatically adding empty categories to dependency trees so we can deal with the remaining of 11.5% Propbank arguments.

4.3 Handling disjoint arguments

Some Propbank arguments are disjoint in the phrase structure so that they cannot be represented as single head words in dependency trees. For example in Figure 6, both NP-1^* and S^* are ARG_1 of the verb predicate *continued* but there is no head word for the dependency tree that can represent both phrases. The algorithm in Figure 5 naturally

handles this kind of disjoint arguments. Although words in S_p are not entirely consecutive ({*Yields*, *on*, *mutual*, *funds*, *to*, *slide*}), it iteratively finds both head words correctly: *Yields* and *to*.

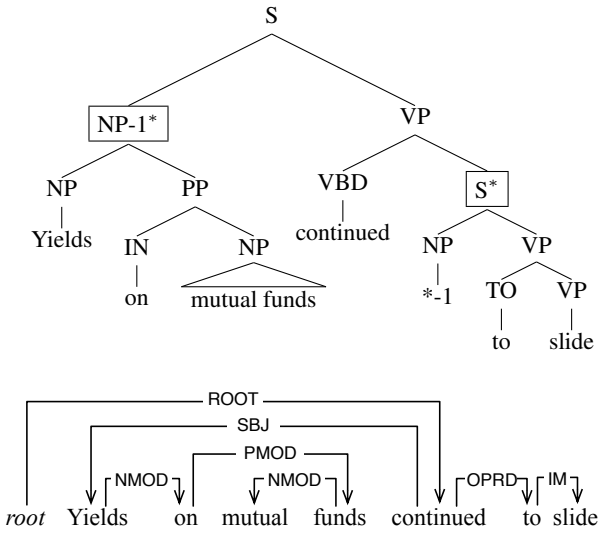


Figure 6: Disjoint argument example

5 Retrieving fine-grained semantic boundaries

There are a total of 292,073 Propbank arguments in the Penn Treebank, and only 88% of them map to correct semantic boundaries from the dependency trees by taking the subtrees of head words. The errors are typically caused by including more words than required: the recall is still 100% for the error cases whereas the precision is not. Among several error cases, the most critical one is caused by verb predicates whose semantic arguments are the parents of themselves in the dependency trees (cf. Figure 2). In this section, we present heuristics to handle such cases so we can achieve precision nearly as good as the recall.

5.1 Modals

In the current dependency structure, modals (e.g., *will*, *can*, *do*) become the heads of the main verbs. In Figure 7, *will* is the head of the verb predicate *remain* in the dependency tree; however, it is also an argument (ARGM-MOD) of the verb in Propbank. This can be resolved by retrieving only the head word, but not the subtree. Thus, only *will* is retrieved as the ARGM-MOD of *remain*.

Modals can be followed by conjuncts that are also modals. In this case, the entire coordination is retrieved as ARGM-MOD (e.g., {*may*, *or*, *may*, *not*} in Figure 8).

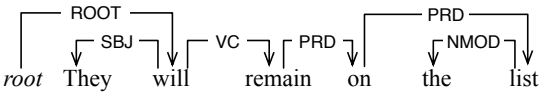


Figure 7: Modal example 1

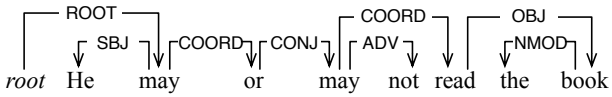


Figure 8: Modal example 2

5.2 Negations

Negations (e.g., *not*, *no longer*) are annotated as ARGM-NEG in Propbank. In most cases, negations do not have any child in dependency trees, so retrieving only the negations themselves gives the correct semantic boundaries for ARGM-NEG, but there are exceptions. One is where a negation comes after a conjunction; in which case, the negation becomes the parent of the main verb. In Figure 9, *not* is the parent of the verb predicate *copy* although it is the ARGM-NEG of the verb.

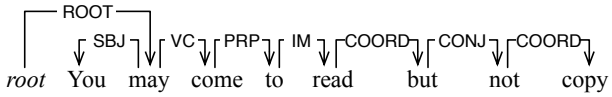


Figure 9: Negation example 1

The other case is where a negation is modified by some adverb; in which case, the adverb should also be retrieved as well as the negation. In Figure 10, both *no* and *longer* should be retrieved as the ARGM-NEG of the verb predicate *oppose*.

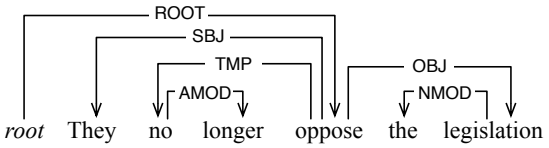


Figure 10: Negation example 2

5.3 Overlapping arguments

Propbank does not allow overlapping arguments. For each predicate, if a word is included in one argument, it cannot be included in any other argument of the predicate. In Figure 11, *burdens* and *in the region* are annotated as ARG_1 and ARGM-LOC of the verb predicate *share*, respectively. The arguments were originally annotated as two separate phrases in the phrase structure tree; however,

in became the child of *burdens* during the conversion, so the subtree of *burdens* includes the subtree of *in*, which causes overlapping arguments.

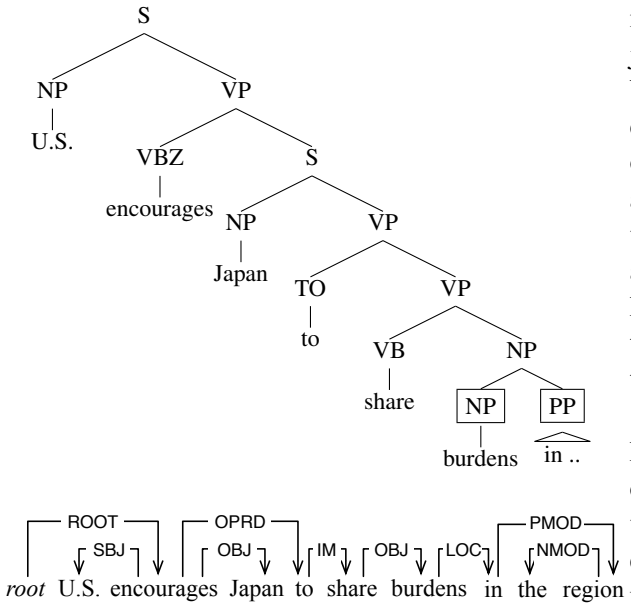


Figure 11: Overlapping argument example 1

When this happens, we reconstruct the dependency tree so *in* becomes the child of *share* instead of *burdens* (Figure 12). By doing so, taking the subtrees of *burdens* and *in* no longer causes overlapping arguments.[3]

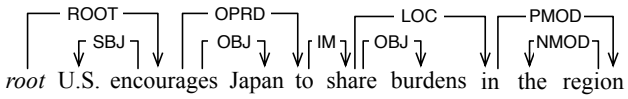


Figure 12: Overlapping argument example 2

5.4 Verb predicates whose semantic arguments are their syntactic heads

There are several cases where semantic arguments of verb predicates become the syntactic heads of the verbs. The modals and negations in the previous sections are special cases where the semantic boundaries can be retrieved correctly without compromising recall. The following sections describe other cases, such as relative clauses (Section 5.4.2), gerunds and past-participles (Section 5.4.3), that may cause a slight decrease in recall by finding more fine-grained semantic boundaries. In these cases, the subtree of the verb predicates are excluded from the semantic arguments.

[3]This can be considered as a Treebank/Propbank disagreement, which is further discussed in Sectino 6.2.

5.4.1 Verb chains

Three kinds of verb chains exist in the current dependency structure: auxiliary verbs (including modals and *be*-verbs), infinitive markers, and conjunctions. As discussed in Section 5.1, verb chains become the parents of their main verbs in dependency trees. This indicates that when the subtree of the main verb is to be excluded from semantic arguments, the verb chain needs to be excluded as well. This usually happens when the main verbs are used within relative clauses. In addition, more heuristics are needed for retrieving correct semantic boundaries for relative clauses, which are further discussed in Section 5.4.2.

The following figures show examples of each kind of verb chain. It is possible that multiple verb chains are joined with one main verb. In this case, we find the top-most verb chain and exclude its entire subtree from the semantic argument. In Figure 13, *part* is annotated as ARG_1 of the verb predicate *gone*, chained with the auxiliary verb *be*, and again chained with the modal *may*. Since *may* is the top-most verb chain, we exclude its subtree so only *a part* is retrieved as the ARG_1 of *gone*.

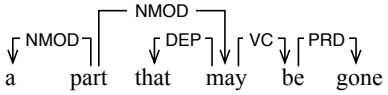


Figure 13: Auxiliary verb example

Figure 14 shows the case of infinitive markers. *those* is annotated as ARG_0 of the verb predicate *leave*, which is first chained with the infinitive marker *to* then chained with the verb *required*. By excluding the subtree of *required*, only *those* is retrieved as the ARG_0 of *leave*.

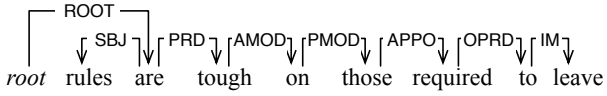


Figure 14: Infinitive marker example

Figure 15 shows the case of conjunctions. *people* is annotated as ARG_0 of the verb predicate *exceed*, which is first chained with *or* then chained with *meet*. By excluding the subtree of *meet*, only *people* is retrieved as the ARG_0 of *exceed*.

When a verb predicate is followed by an object complement (OPRD), the subtree of the object complement is not excluded from the semantic argument. In Figure 16, *distribution* is annotated as

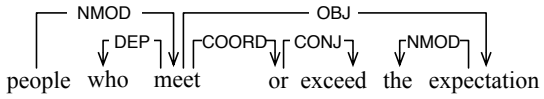


Figure 15: Conjunction example

ARG_1 of the verb predicate *expected*. By excluding the subtree of *expected*, the object complement *to occur* would be excluded as well; however, Propbank annotation requires keeping the object complement as the part of the argument. Thus, *a distribution to occur* is retrieved as the ARG_1 of *expected*.

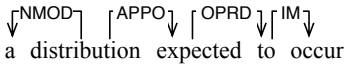


Figure 16: Object complement example

5.4.2 Relative clauses

When a verb predicate is within a relative clause, Propbank annotates both the relativizer (if present) and its antecedent as part of the argument. For example in Figure 15, *people* is annotated as ARG_0 of both *meet* and *exceed*. By excluding the subtree of *meet*, the relativizer *who* is also excluded from the semantic argument, which is different from the original Propbank annotation. In this case, we keep the relativizer as part of the ARG_0; thus, *people who* is retrieved as the ARG_0 (similarly, *a part that* is retrieved as the ARG_0 of *gone* in Figure 13).

It is possible that a relativizer is headed by a preposition. In Figure 17, *climate* is annotated as ARGM-LOC of the verb predicate *made* and the relativizer *which* is headed by the preposition *in*. In this case, both the relativizer and the preposition are included in the semantic argument. Thus, *the climate in which* becomes the ARGM-LOC of *made*.

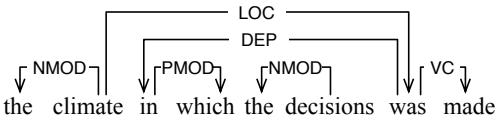


Figure 17: Relativizer example

5.4.3 Gerunds and past-participles

In English, when gerunds and past-participles are used without the presence of *be*-verbs, they often function as noun modifiers. Propbank still treats them as verb predicates; however, these verbs become children of the nouns they modify in the dependency structure, so the heuristics discussed in Section 5.4 and 5.4.1 need to be applied to find the correct semantic boundaries. Furthermore, since these are special kinds of verbs, they require even more rigorous pruning.

When a head word, annotated to be a semantic argument of a verb predicate, comes after the verb, every word prior to the verb predicate needs to be excluded from the semantic argument. In Figure 18, *group* is annotated as ARG_0 of the verb predicate *publishing*, so all words prior to the predicate (*the Dutch*) need to be excluded. Thus, only *group* is retrieved as the ARG_0 of *publishing*.

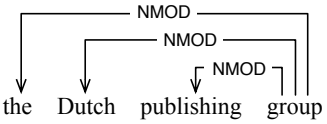


Figure 18: Gerund example

When the head word comes before the verb predicate, the subtree of the head word, excluding the subtree of the verb predicate, is retrieved as the semantic argument. In Figure 19, *correspondence* is annotated as ARG_1 of the verb predicate *mailed*, so the subtree of *correspondence*, excluding the subtree of *mailed*, is retrieved to be the argument. Thus, *correspondence about incomplete 8300s* becomes the ARG_1 of *mailed*.

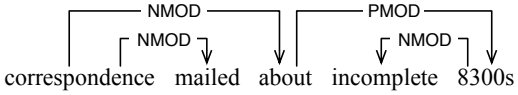


Figure 19: Past-participle example 1

When the subtree of the verb predicate is immediately followed by comma-like punctuation (e.g., comma, colon, semi-colon, etc.) and the head word comes before the predicate, every word after the punctuation is excluded from the semantic argument. In Figure 20, *fellow* is annotated as ARG_1 of the verb predicate *named*, so both the subtree of the verb (*named John*) and every word after the comma (*, who stayed for years*) are excluded from the semantic argument. Thus, only *a fellow* is retrieved as the ARG_1 of *named*.

5.5 Punctuation

For evaluation, we built a model that excludes punctuation from semantic boundaries for two reasons. First, it is often not clear how punctuation

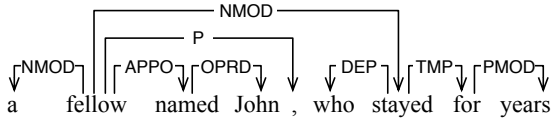


Figure 20: Past-participle example 2

needs to be annotated in either Treebank or Propbank; because of that, annotation for punctuation is not entirely consistent, which makes it hard to evaluate. Second, although punctuation gives useful information for obtaining semantic boundaries, it is not crucial for semantic roles. In fact, some of the state-of-art semantic role labeling systems, such as ASSERT (Pradhan et al., 2004), give an option for omitting punctuation from the output. For these reasons, our final model ignores punctuation for semantic boundaries.

6 Evaluations

6.1 Model comparisons

The following list describes six models used for the experiments. Model I is the baseline approach that retrieves all words in the subtrees of head words as semantic boundaries. Model II to VI use the heuristics discussed in the previous sections. Each model inherits all the heuristics from the previous model and adds new heuristics; therefore, each model is expected to perform better than the previous model.

- I - all words in the subtrees (baseline)
- II - modals + negations (Sections 5.1, 5.2)
- III - overlapping arguments (Section 5.3)
- IV - verb chains + relative clauses (Sections 5.4.1, 5.4.2)
- V - gerunds + past-participles (Section 5.4.3)
- VI - excluding punctuations (Section 5.5)

The following list shows measurements used for the evaluations. $gold(arg)$ is the gold-standard set of words for the argument arg. $sys(arg)$ is the set of words for arg produced by our system. $c(arg_1, arg_2)$ returns 1 if arg_1 is equal to arg_2; otherwise, returns 0. T is the total number of arguments in the Propbank.

$$
\begin{aligned}
Accuracy &= \frac{1}{T} \cdot \sum_{\forall arg} c(gold(arg), sys(arg)) \\
Precision &= \frac{1}{T} \cdot \sum_{\forall arg} \frac{|gold(arg) \cap sys(arg)|}{|sys(arg)|} \\
Recall &= \frac{1}{T} \cdot \sum_{\forall arg} \frac{|gold(arg) \cap sys(arg)|}{|gold(arg)|} \\
F1 &= \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}
\end{aligned}
$$

Table 1 shows the results from the models using the measurements. As expected, each model shows improvement over the previous one in terms of accuracy and F1-score. The F1-score of Model VI shows improvement that is statistically significant compared to Model I using t-test ($t = 149.00$, $p < 0.0001$). The result from the final model is encouraging because it enables us to take full advantage of dependency structure for semantic role labeling. Without finding the correct semantic boundaries, even if a semantic role labeling system did an excellent job finding the right head words, we would not be able to find the actual chunks for the arguments. By using our approach, finding the correct semantic boundaries is no longer an issue for using dependency structure for automatic semantic role labeling.

Model	Accuracy	Precision	Recall	F1
I	88.00	92.51	100	96.11
II	91.84	95.77	100	97.84
III	92.17	97.08	100	98.52
IV	95.89	98.51	99.95	99.23
V	97.00	98.94	99.95	99.44
VI	**98.20**	99.14	99.95	**99.54**

Table 1: Model comparisons (in percentage)

6.2 Error analysis

Although each model consistently shows improvement on the precision, the recall is reduced a bit for some models. Specifically, the recalls for Models II and III are not 100% but rather 99.9994% and 99.996%, respectively. We manually checked all errors for Models II and III and found that they are caused by inconsistent annotations in the gold-standard. For Model II, Propbank annotation for ARGM-MOD was not done consistently with con-

junctions. For example in Figure 8, instead of annotating *may or may not* as the ARGM-MOD, some annotations include only *may* and *may not* but not the conjunction *or*. Since our system consistently included the conjunctions, they appeared to be different from the gold-standard, but are not errors.

For Model III, Treebank annotation was not done consistently for adverbs modifying negations. For example in Figure 10, *longer* is sometimes (but rarely) annotated as an adjective where it is supposed to be an adverb. Furthermore, *longer* sometimes becomes a child of the verb predicate *oppose* (instead of being the child of *no*). Such annotations made our system exclude *longer* as a part of ARGM-NEG, but it would have found them correctly if the trees were annotated consistently.

There are a few cases that caused errors in Models IV and V. The most critical one is caused by PP (prepositional phrase) attachment. In Figure 21, *enthusiasm* is annotated as ARG_1 of the verb predicate *showed*, so our system retrieved the subtree of *enthusiasm*, excluding the subtree of *showed*, as the semantic boundary for the ARG_1 (e.g., *the enthusiasm*). However, Propbank originally annotated both *the enthusiasm* and *for stocks* as the ARG_1 in the phrase structure tree (so the prepositional phrase got lost in our system).

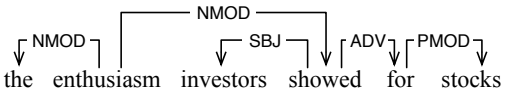


Figure 21: PP-attachment example 1

This happens when there is a disagreement between Treebank and Propbank annotations: the Treebank annotation attached the PP (*for stocks*) to the verb (*showed*) whereas the Propbank annotation attached the PP to the noun (*enthusiasm*). This is a potential error in the Treebank. In this case, we can trust the Propbank annotation and reconstruct the tree so the Treebank and Propbank annotations agree with each other. After the reconstruction, the dependency tree would look like one in Figure 22.

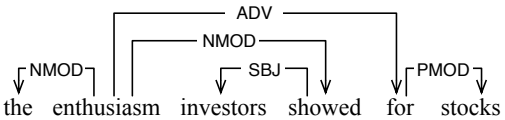


Figure 22: PP-attachment example 2

7 Conclusion and future work

We have discussed how to convert phrase structure trees to dependency trees, how to find the minimum-set of head words for Propbank arguments in dependency structure, and heuristics for retrieving fine-grained semantic boundaries. By using our approach, we correctly retrieved the semantic boundaries of 98.2% of the Propbank arguments (F1-score of 99.54%). Furthermore, the heuristics can be used to fix some of the inconsistencies in both Treebank and Propbank annotations. Moreover, they suggest ways of reconstructing dependency structure so that it can fit better with semantic roles.

Retrieving correct semantic boundaries is important for tasks like machine translation where not only the head words but also all other words matter to complete the task (Choi et al., 2009). In the future, we are going to apply our approach to other corpora and see how well the heuristics work. In addition, we will try to find ways of automatically adding empty categories to dependency structure so we can deal with the full set of Propbank arguments.

Acknowledgments

Special thanks are due to Professor Joakim Nivre of Uppsala University and Claire Bonial of the University of Colorado at Boulder for very helpful insights. We gratefully acknowledge the support of the National Science Foundation Grants CISE-CRI-0551615, Towards a Comprehensive Linguistic Annotation and CISE-CRI 0709167, Collaborative: A Multi-Representational and Multi-Layered Treebank for Hindi/Urdu, and a grant from the Defense Advanced Research Projects Agency (DARPA/IPTO) under the GALE program, DARPA/CMO Contract No. HR0011-06-C-0022, subcontract from BBN, Inc. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

Jinho D. Choi, Martha Palmer, and Nianwen Xue. 2009. Using parallel propbanks to enhance word-alignments. In *Proceedings of ACL-IJCNLP workshop on Linguistic Annotation (LAW'09)*, pages 121–124.

Susanne Ekeklint and Joakim Nivre. 2007. A dependency-based conversion of propbank. In *Proceedings of NODALIDA workshop on Building Frame Semantics Resources for Scandinavian and Baltic Languages (FRAME'07)*, pages 19–25.

Daniel Gildea. 2004. Dependencies vs. constituents for tree-based alignment. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP'04)*, pages 214–221.

Jan Hajič, Massimiliano Ciaramita, Richard Johansson, Daisuke Kawahara, Maria Antònia Martí, Lluís Màrquez, Adam Meyers, Joakim Nivre, Sebastian Padó, Jan Štěpánek, Pavel Straňák, Mihai Surdeanu, Nianwen Xue, and Yi Zhang. 2009. The conll-2009 shared task: Syntactic and semantic dependencies in multiple languages. In *Proceedings of the 13th Conference on Computational Natural Language Learning (CoNLL'09)*, pages 1–18.

Richard Johansson and Pierre Nugues. 2007. Extended constituent-to-dependency conversion for english. In *Proceedings of the 16th Nordic Conference of Computational Linguistics (NODALIDA'07)*.

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

Jason S. Kessler and Nicolas Nicolov. 2009. Targeting sentiment expressions through supervised ranking of linguistic configurations. In *Proceedings of the 3rd International AAAI Conference on Weblogs and Social Media (ICWSM'09)*.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330.

Martha Palmer, Daniel Gildea, and Paul Kingsbury. 2005. The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1):71–106.

Sameer S. Pradhan, Wayne Ward, Kadri Hacioglu, James H. Martin, and Daniel Jurafsky. 2004. Shallow semantic parsing using support vector machines. In *Proceedings of the Human Language Technology Conference/North American chapter of the Association for Computational Linguistics annual meeting (HLT/NAACL'04)*.