Transition-based Semantic Role Labeling Using Predicate Argument Clustering

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
mpalmer@colorado.edu

Abstract

This paper suggests two ways of improving semantic role labeling (SRL). First, we introduce a novel transition-based SRL algorithm that gives a quite different approach to SRL. Our algorithm is inspired by shift-reduce parsing and brings the advantages of the transition-based approach to SRL. Second, we present a self-learning clustering technique that effectively improves labeling accuracy in the test domain. For better generalization of the statistical models, we cluster verb predicates by comparing their predicate argument structures and apply the clustering information to the final labeling decisions. All approaches are evaluated on the CoNLL'09 English data. The new algorithm shows comparable results to another state-of-the-art system. The clustering technique improves labeling accuracy for both in-domain and out-of-domain tasks.

1 Introduction

Semantic role labeling (SRL) has sparked much interest in NLP (Shen and Lapata, 2007; Liu and Gildea, 2010). Lately, dependency-based SRL has shown advantages over constituent-based SRL (Johansson and Nugues, 2008). Two main benefits can be found. First, dependency parsing is much faster than constituent parsing, whereas constituent parsing is usually considered to be a bottleneck to SRL in terms of execution time. Second, dependency structure is more similar to predicate argument structure than phrase structure because it specifically defines relations between a predicate and its arguments with labeled arcs. Unlike constituent-based SRL that maps phrases to semantic roles, dependency-based SRL maps headwords to semantic roles because there is no phrasal node in dependency structure. This may lead to a concern about getting the actual semantic chunks back, but Choi and Palmer (2010) have shown that it is possible to recover the original chunks from the headwords with minimal loss, using a certain type of dependency structure.

Traditionally, either constituent or dependency-based, semantic role labeling is done in two steps, argument identification and classification (Gildea and Jurafsky, 2002). This is from a general belief that each step requires a different set of features (Xue and Palmer, 2004), and training these steps in a pipeline takes less time than training them as a joint-inference task. However, recent machine learning algorithms can deal with large scale vector spaces without taking too much training time (Hsieh et al., 2008). Furthermore, from our experience in dependency parsing, handling these steps together improves accuracy in identification as well as classification (unlabeled and labeled attachment scores in dependency parsing). This motivates the development of a new semantic role labeling algorithm that treats these two steps as a joint inference task.

Our algorithm is inspired by shift-reduce parsing (Nivre, 2008). The algorithm uses several transitions to identify predicates and their arguments with semantic roles. One big advantage of the transition-based approach is that it can use previously identified arguments as features to predict the next argument. We apply this technique to our approach and achieve comparable results to another state-of-the-art system evaluated on the same data sets.

Proceedings of the ACL 2011 Workshop on Relational Models of Semantics (RELMS 2011), pages 37–45,
Portland, Oregon, USA, June 23, 2011.

NO-PRED	$(\ \lambda_1\ ,\ \lambda_2,\ j,\ \lambda_3,\ [i\|\lambda_4],\ A\ ) \Rightarrow (\ [\lambda_1\|j],\ \lambda_2,\ i,\ \lambda_3,\ \lambda_4\ ,\ A\ )$ $\exists j.\ \text{oracle}(j) \neq \text{predicate}$
SHIFT	$(\ \lambda_1\ ,\ \lambda_2,\ j,\ [i\|\lambda_3],\ \lambda_4,\ A\ ) \Rightarrow (\ [\lambda_2\|j],\ [\ ]\ ,\ i,\ [\ ]\ ,\ \lambda_3,\ A\ )$ $\exists j.\ \text{oracle}(j) = \text{predicate} \quad \wedge \quad \lambda_1 = [\ ] \quad \wedge \quad \lambda_4 = [\ ]$
NO-ARC$^{\leftarrow}$	$(\ [\lambda_1\|i],\ \lambda_2\ ,\ j,\ \lambda_3,\ \lambda_4,\ A\ ) \Rightarrow (\ \lambda_1\ ,\ [i\|\lambda_2],\ j,\ \lambda_3,\ \lambda_4,\ A\ )$ $\exists j.\ \text{oracle}(j) = \text{predicate} \quad \wedge \quad \exists i.\text{oracle}(i,j) = \{i \not\leftarrow j\}$
NO-ARC$^{\rightarrow}$	$(\ \lambda_1,\ \lambda_2,\ j,\ \lambda_3\ ,\ [i\|\lambda_4],\ A\ ) \Rightarrow (\ \lambda_1,\ \lambda_2,\ j,\ [\lambda_3\|i],\ \lambda_4\ ,\ A\ )$ $\exists j.\ \text{oracle}(j) = \text{predicate} \quad \wedge \quad \exists i.\text{oracle}(i,j) = \{j \not\rightarrow i\}$
LEFT-ARC$^{\leftarrow}_{\text{L}}$	$(\ [\lambda_1\|i],\ \lambda_2\ ,\ j,\ \lambda_3,\ \lambda_4,\ A\ ) \Rightarrow (\ \lambda_1\ ,\ [i\|\lambda_2],\ j,\ \lambda_3,\ \lambda_4,\ A \cup \{i \overset{\text{L}}{\leftarrow} j\}\ )$ $\exists j.\ \text{oracle}(j) = \text{predicate} \quad \wedge \quad \exists i.\text{oracle}(i,j) = \{i \overset{\text{L}}{\leftarrow} j\}$
RIGHT-ARC$^{\rightarrow}_{\text{L}}$	$(\ \lambda_1,\ \lambda_2,\ j,\ \lambda_3\ ,\ [i\|\lambda_4],\ A\ ) \Rightarrow (\ \lambda_1,\ \lambda_2,\ j,\ [\lambda_3\|i],\ \lambda_4\ ,\ A \cup \{j \overset{\text{L}}{\rightarrow} i\}\ )$ $\exists j.\ \text{oracle}(j) = \text{predicate} \quad \wedge \quad \exists i.\text{oracle}(i,j) = \{j \overset{\text{L}}{\rightarrow} i\}$

Table 1: Transitions in our bidirectional top-down search algorithm. For each row, the first line shows a transition and the second line shows preconditions of the transition.

For better generalization of the statistical models, we apply a self-learning clustering technique. We first cluster predicates in test data using automatically generated predicate argument structures, then cluster predicates in training data by using the previously found clusters as seeds. Our experiments show that this technique improves labeling accuracy for both in-domain and out-of-domain tasks.

2 Transition-based semantic role labeling

Dependency-based semantic role labeling can be viewed as a special kind of dependency parsing in the sense that both try to find relations between word pairs. However, they are distinguished in two major ways. First, unlike dependency parsing that tries to find some kind of relation between any word pair, semantic role labeling restricts its search only to top-down relations between predicate and argument pairs. Second, dependency parsing requires one head for each word, so the final output is a tree, whereas semantic role labeling allows multiple predicates for each argument. Thus, not all dependency parsing algorithms, such as a maximum spanning tree algorithm (Mcdonald and Pereira, 2006), can be naively applied to semantic role labeling.

Some transition-based dependency parsing algorithms have been adapted to semantic role labeling and shown good results (Henderson et al., 2008; Titov et al., 2009). However, these algorithms are originally designed for dependency parsing, so are not necessarily customized for semantic role labeling. Here, we present a novel transition-based algorithm dedicated to semantic role labeling. The key difference between this algorithm and most other transition-based algorithms is in its directionality. Given an identified predicate, this algorithm tries to find top-down relations between the predicate and the words on both left and right-hand sides, whereas other transition-based algorithms would consider words on either the left or the right-hand side, but not both. This bidirectional top-down search makes more sense for semantic role labeling because predicates are always assumed to be the heads of their arguments, an assumption that cannot be generalized to dependency parsing, and arguments can appear either side of the predicate.

Table 1 shows transitions used in our algorithm. All parsing states are represented as tuples (λ_1, λ_2, p, λ_3, λ_4, A), where $\lambda_{1..4}$ are lists of word indices and p is either a word index of the current predicate candidate or $\sharp$ indicating no predicate candidate. $\lambda_{1,4}$ contain indices to be compared with p and $\lambda_{2,3}$ contain indices already compared with p. A is a set of labeled arcs representing previously identified arguments with respect to their predicates. $\leftarrow$ and $\rightarrow$ indicate parsing directions. L is a semantic role label, and i, j represent indices of their corresponding word tokens. The initial state is ([ ], [ ], 1, [ ], $[2, \ldots, n]$, $\emptyset$), where w_1 and w_n are the first and the last words in a sentence, respectively. The final state is (λ_1, λ_2, $\sharp$, [ ], [ ], A), i.e., the algorithm terminates when there is no more predicate candidate left.

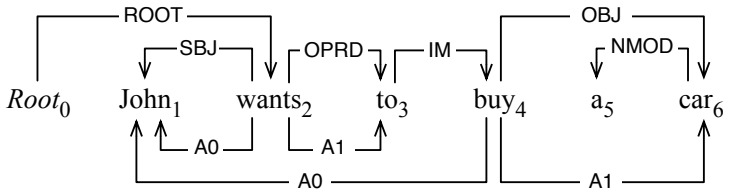

Figure 1: An example of a dependency tree with semantic roles. The upper and lower arcs stand for syntactic and semantic dependencies, respectively. SBJ, OBJ, OPRD, IM, NMOD stand for a subject, object, object predicative, infinitive marker, and noun-modifier. A0, A1 stand for ARG0, ARG1 in PropBank (Palmer et al., 2005).

	Transition	λ_1	λ_2	p	λ_3	λ_4	A
0		[ ]	[ ]	1	[ ]	[2..6]	$\emptyset$
1	NO-PRED	[1]	[ ]	2	[ ]	[3..6]	
2	LEFT-ARC	[ ]	[1]	2	[ ]	[3..6]	$A \cup \{1 \leftarrow \text{A0} - 2\}$
3	RIGHT-ARC	[ ]	[1]	2	[3]	[4..6]	$A \cup \{2 - \text{A1} \rightarrow 3\}$
4	NO-ARC	[ ]	[1]	2	[3..4]	[5..6]	
5	NO-ARC	[ ]	[1]	2	[3..5]	[6]	
6	NO-ARC	[ ]	[1]	2	[3..6]	[ ]	
7	SHIFT	[1..2]	[ ]	3	[ ]	[4..6]	
8	NO-PRED	[1..3]	[ ]	4	[ ]	[5..6]	
9	NO-ARC	[1..2]	[3]	4	[ ]	[5..6]	
10	NO-ARC	[1]	[2..3]	4	[ ]	[5..6]	
11	LEFT-ARC	[ ]	[1..3]	4	[ ]	[5..6]	$A \cup \{1 \leftarrow \text{A0} - 4\}$
12	NO-ARC	[ ]	[1..3]	4	[5]	[6]	
13	RIGHT-ARC	[ ]	[1..3]	4	[5..6]	[ ]	$A \cup \{4 - \text{A1} \rightarrow 6\}$
14	SHIFT	[1..4]	[ ]	5	[ ]	[6]	
15	NO-PRED	[1..5]	[ ]	6	[ ]	[ ]	
16	NO-PRED	[1..6]	[ ]	$\sharp$	[ ]	[ ]	

Table 2: Parsing states generated by our algorithm for the example in Figure 1.

The algorithm uses six kinds of transitions. NO-PRED is performed when an oracle identifies w_j as not a predicate. All other transitions are performed when w_j is identified as a predicate. SHIFT is performed when both λ_1 and λ_4 are empty, meaning that there are no more argument candidates left for the predicate w_j. NO-ARC is performed when w_i is identified as not an argument of w_j. LEFT-ARC$_\text{L}$ and RIGHT-ARC$_\text{L}$ are performed when w_i is identified as an argument of w_j with a label L. These transitions can be performed in any order as long as their preconditions are satisfied. For our experiments, we use the following generalized sequence:

$$[\ (\text{NO-PRED})* \Rightarrow (\text{LEFT-ARC}_\text{L}^{\leftarrow} | \text{NO-ARC}^{\leftarrow})* \Rightarrow (\text{RIGHT-ARC}_\text{L}^{\rightarrow} | \text{NO-ARC}^{\rightarrow})* \Rightarrow \text{SHIFT}\ ]*$$

Notice that this algorithm does not take separate steps for argument identification and classification. By adding the NO-ARC transitions, we successfully merge these two steps together without decrease in labeling accuracy.[1] Since each word can be a predicate candidate and each predicate considers all other words as argument candidates, a worst-case complexity of the algorithm is $O(n^2)$. To reduce the complexity, Zhao et al. (2009) reformulated a pruning algorithm introduced by Xue and Palmer (2004) for dependency structure by considering only direct dependents of a predicate and its ancestors as argument candidates. This pruning algorithm can be easily applied to our algorithm: the oracle can prefilter such dependents and uses the information to perform NO-ARC transitions without consulting statistical models.

[1] We also experimented with the traditional approach of building separate classifiers for identification and classification, which did not lead to better performance in our case.

Table 2 shows parsing states generated by our algorithm. Our experiments show that this algorithm gives comparable results against another state-of-the-art system.

3 Predicate argument clustering

Some studies showed that verb clustering information could improve performance in semantic role labeling (Gildea and Jurafsky, 2002; Pradhan et al., 2008). This is because semantic role labelers usually perform worse on verbs not seen during training, for which the clustering information can provide useful features. Most previous studies used either bag-of-words or syntactic structure to cluster verbs; however, this may or may not capture the nature of predicate argument structure, which is more semantically oriented. Thus, it is preferable to cluster verbs by their predicate argument structures to get optimized features for semantic role labeling.

In this section, we present a self-learning clustering technique that effectively improves labeling accuracy in the test domain. First, we perform semantic role labeling on the test data using the algorithm in Section 2. Next, we cluster verbs in the test data using predicate argument structures generated by our semantic role labeler (Section 3.2). Then, we cluster verbs in the training data using the verb clusters we found in the test data (Section 3.3). Finally, we re-run our semantic role labeler on the test data using the clustering information. Our experiments show that this technique gives improvement to labeling accuracy for both in and out-of domain tasks.

3.1 Projecting predicate argument structure into vector space

Before clustering, we need to project the predicate argument structure of each verb into vector space. Two kinds of features are used to represent these vectors: semantic role labels and joined tags of semantic role labels and their corresponding word lemmas. Figure 2 shows vector representations of predicate argument structures of verbs, *want* and *buy*, in Figure 1.

Initially, all existing and non-existing features are assigned with a value of 1 and 0, respectively. However, assigning equal values to all existing features is not necessarily fair because some features have

Verb	A0	A1	...	john:A0	to:A1	car:A1	...
want	1	1	0s	1	1	0	0s
buy	1	1	0s	1	0	1	0s

Figure 2: Projecting the predicate argument structure of each verb into vector space.

higher confidence, or are more important than the others; e.g., `ARG0` and `ARG1` are generally predicted with higher confidence than modifiers, nouns give more important information than some other grammatical categories, etc. Instead, we assign each existing feature with a value computed by the following equations:

$$s(l_j|v_i) = \frac{1}{1 + exp(-score(l_j|v_i))}$$

$$s(m_j, l_j) = \begin{cases} 1 \quad (w_j \neq \text{noun}) \\ exp(\frac{count(m_j, l_j)}{\sum_{\forall k} count(m_k, l_k)}) \end{cases}$$

v_i is the current verb, l_j is the j'th label of v_i, and m_j is l_j's corresponding lemma. $score(l_j|v_i)$ is a score of l_j being a correct argument label of v_i; this is always 1 for training data and is provided by our statistical models for test data. Thus, $s(l_j|v_i)$ is an approximated probability of l_j being a correct argument label of v_i, estimated by the logistic function. $s(m_j, l_j)$ is equal to 1 if w_j is not a noun. If w_j is a noun, it gets a value ≥ 1 given a maximum likelihood of m_j being co-occurred with l_j.[2]

With the vector representation, we can apply any kind of clustering algorithm (Hofmann and Puzicha, 1998; Kamvar et al., 2002). For our experiments, we use k-best hierarchical clustering for test data, and k-means clustering for training data.

3.2 Clustering verbs in test data

Given automatically generated predicate argument structures in the test data, we apply k-best hierarchical clustering; that is, a relaxation of classical hierarchical agglomerative clustering (from now on, HAC; Ward (1963)), to find verb clusters. Unlike HAC that merges a pair of clusters at each iteration, k-best hierarchical clustering merges k-best pairs at

[2]Assigning different weights for nouns resulted in more meaningful clusters in our experiments. We will explore additional grammatical category specific weighting schemes in future work.

each iteration (Lo et al., 2009). Instead of merging a fixed number of k-clusters, we use a threshold to dynamically determine the top k-clusters. Our studies indicate that this technique produces almost as fine-grained clusters as HAC, yet converges much faster.

Our algorithm for k-best hierarchical clustering is presented in Algorithm 1. th_{up} is a threshold that determines which k-best pairs are to be merged (in our case, k_{up} = 0.8). $sim(c_i, c_j)$ is a similarity between clusters c_i and c_j. For our experiments, we use cosine similarity with average-linkage. It is possible that other kinds of similarity metrics would work better, which we will explore as future work. Conditions in line 15 ensure that each cluster is merged with at most one other cluster at each iteration, and conditions in line 17 force at least one cluster to be merged with one other cluster at each iteration. Thus, the algorithm is guaranteed to terminate after at most $(n-1)$ iterations.

When the algorithm terminates, it returns a set of one cluster with different hierarchical levels. For our experiments, we set another threshold, th_{low}, for early break-out: if there is no cluster pair whose similarity is greater than th_{low}, we terminate the algorithm (in our case, th_{low} = 0.7). A cluster set generated by this early break-out contains several unit clusters that are not merged with any other cluster. All of these unit clusters are discarded from the set to improve set quality. This is reasonable because our goal is not to cluster all verbs but to find a useful set of verb clusters that can be mapped to verbs in training data, which can lead to better performance in semantic role labeling.

input : $C = [c_1, .., c_n]$: c_i is a unit cluster.
$th_{up} \in \mathbb{R}$: threshold.
output: $\hat{C} = [c_1, .., c_m]$: c_j is a unit or merged cluster, where $m \leq n$.

```
1  begin
2      while |C| > 1 do
3          L ← list()
4          for i ∈ [1, |C| − 1] do
5              for j ∈ [i + 1, |C|] do
6                  t ← (i, j, sim(c_i, c_j))
7                  L.add(t)
8              end
9          end
10         descendingSortBySimilarity(L)
11         S ← set()
12         for k ∈ [1, |L|] do
13             t ← L.get(k)
14             i ← t(0); j ← t(1); sim ← t(2)
15             if i ∈ S or j ∈ S then
16                 continue
17             if k = 1 or sim > th_up then
18                 C.add(c_i ∪ c_j); S.add(i, j)
19                 C.remove(c_i, c_j)
20             else
21                 break
22             end
23         end
24     end
25 end
```

Algorithm 1: k-best hierarchical clustering.

3.3 Clustering verbs in training data

Given the verb clusters we found in the test data, we search for verbs that are similar to these clusters in the training data. K-means clustering (Hartigan, 1975) is a natural choice for this case because we already know k-number of center clusters to begin with. Each verb in the training data is compared with all verb clusters in the test data, and merged with the cluster that gives the highest similarity. To maintain the quality of the clusters, we use the same threshold, th_{low}, to filter out verbs in the training data that are not similar enough to any verb cluster in the test data. By doing so, we keep only verbs that are more likely to be helpful for semantic role labeling.

4 Features

4.1 Baseline features

For a baseline approach, we use features similar to ones used by Johansson and Nugues (2008). All features are assumed to have dependency structures as input. Table 3 shows n-gram feature templates used for our experiments (f: form, m: lemma, p: POS tag, d: dependency label). w_{arg} and w_{pred} are the current argument and predicate candidates. $hd(w)$ stands for the head of w, $lm(w)$, $rm(w)$ stand for the leftmost, rightmost dependents of w, and $ls(w)$, $rs(w)$ stand for the left-nearest, right-nearest siblings of w, with respect to the dependency structures. Some of these features can be presented as a joined feature; e.g., a combination of w_{arg}'s POS tag and lemma.

Word tokens	Features
w_{arg}, w_{pred}	f,m,p,d
$w_{arg\pm1}$, hd, lm, rm, ls, rs (w_{arg})	m,p
$w_{pred\pm1}$, hd, lm, rm (w_{pred})	m,p

Table 3: N-gram feature templates.

Besides the n-gram features, we use several structural features such as dependency label set, subcategorization, POS path, dependency path, and dependency depth. Dependency label set features are derived by collecting all dependency labels of w_{pred}'s direct dependents. Unlike Johansson and Nugues, we decompose subcategorization features into two parts: one representing the left-hand side and the other representing the right-hand side dependencies of w_{pred}. For the predicate *wants* in Figure 3, we generate $\overleftarrow{\text{SBJ}}$ and $\overrightarrow{\text{OPRD}}$ as separate subcategorization features.

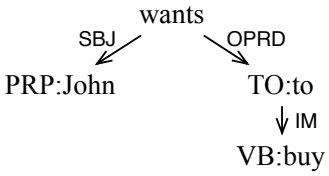

Figure 3: Dependency structure used for subcategorization, path, and depth features.

We also decompose path features into two parts: given the lowest common ancestor (LCA) of w_{arg} and w_{pred}, we generate path features from w_{arg} to the LCA and from the LCA to w_{pred}, separately. For example, the predicate *buy* and the argument *John* in Figure 3 have a LCA at *wants*, so we generate two sets of path features, {↑PRP, ↓TO↓VB} with POS tags, and {↑SBJ, ↓OPRD↓IM} with dependency labels. Such decompositions allow more generalization of those features; even if one part is not matched to the current parsing state, the other part can still participate as a feature. Throughout our experiments, these generalized features give slightly higher labeling accuracy than ungeneralized features although they form a smaller feature space.

In addition, we apply dependency path features to w_{pred}'s highest verb chain, which often shares arguments with the predicate (e.g., *John* is a shared argument of the predicate *buy* and its highest verb chain *wants*). To retrieve the highest verb chain, we apply a simple heuristic presented below. The function `getHighestVerbChain` takes a predicate, `pred`, as input and returns its highest verb chain, `vNode`, as output. If there is no verb chain for the predicate, it returns `null` instead. Note that this heuristic is designed to work with dependency relations and labels described by the CoNLL'09 shared task (Hajič et al., 2009).

```
func getHighestVerbChain(pred)
  vNode = pred;
  regex = "CONJ|COORD|IM|OPRD|VC";

  while (regex.matches(vNode.deprel))
    vNode = vNode.head;

  if (vNode != pred) return vNode;
  else               return null;
```

Dependency depth features are a reduced form of path features. Instead of specifying POS tags or dependency labels, we indicate paths with their depths. For instance, *John* and *buy* in Figure 3 have a dependency depth feature of ↑1↓2, which implies that the depth between *John* and its LCA (*wants*) is 1, and the depth between the LCA and *buy* is 2.

Finally, we use four kinds of binary features: if w_{arg} is a syntactic head of w_{pred}, if w_{pred} is a syntactic head of w_{arg}, if w_{pred} is a syntactic ancestor of w_{arg}, and if w_{pred}'s verb chain has a subject. Each feature gets a value of 1 if true; otherwise, it gets a value of 0.

4.2 Dynamic and clustering features

All dynamic features are derived by using previously identified arguments. Two kinds of dynamic features are used for our experiments. One is a label of the very last predicted numbered argument of w_{pred}. For instance, the parsing state 3 in Table 2 uses a label `A0` as a feature to make its prediction, $wants \xrightarrow{\text{A1}} to$, and the parsing states 4 to 6 use a label `A1` as a feature to make their predictions, NO-ARC's. With this feature, the oracle can narrow down the scope of expected arguments of w_{pred}. The other is a previously identified argument label of w_{arg}. The existence of this feature implies that w_{arg} is already identified as an argument of some other predicate. For instance, when w_{arg} = $John$ and w_{pred} = buy in Table 2, a label `A0` is used as a feature to make the prediction, $John \xleftarrow{\text{A0}} buy$, because $John$ is already identified as an `A0` of $wants$.

Finally, we use w_{pred}'s cluster ID as a feature. The dynamic and clustering features combine a very small portion of the entire feature set, but still give a fair improvement to labeling accuracy.

5 Experiments

5.1 Corpora

All models are trained on Wall Street Journal sections 2-21 and developed on section 24 using automatically generated lemmas and POS tags, as distributed by the CoNLL'09 shared task (Hajič et al., 2009). CoNLL'09 data contains semantic roles for both verb and noun predicates, for which we use only ones related to verb predicates. Furthermore, we do not include predicate sense classification as a part of our task, which is rather a task of word sense disambiguation than semantic role labeling.

For in-domain and out-of-domain evaluations, WSJ section 23 and the Brown corpus are used, also distributed by CoNLL'09. To retrieve automatically generated dependency trees as input to our semantic role labeler, we train our open source dependency parser, called ClearParser[3], on the training set and run the parser on the evaluation sets. ClearParser uses a transition-based dependency parsing algorithm that gives near state-of-the-art results (Choi and Palmer, 2011), and mirrors our SRL algorithm.

5.2 Statistical models

We use Liblinear L2-L1 SVM for learning; a linear classification algorithm using L2 regularization and L1 loss function. This algorithm is designed to handle large scale data: it assumes the data to be linearly separable so does not use any kind of kernel space (Hsieh et al., 2008). As a result, it significantly reduces training time compared to typical SVM, yet performs accurately. For our experiments, we use the following learning parameters: c = 0.1 (cost), e = 0.2 (termination criterion), B = 0 (bias).

Since predicate identification is already provided in the CoNLL'09 data, we do not train NO-PRED. SHIFT does not need to be trained in general because the preconditions of SHIFT can be checked deterministically without consulting statistical models. NO-ARC$^{\leftarrow}$ and LEFT-ARC$^{\leftarrow}_{\text{L}}$ are trained together using the one-vs-all method as are NO-ARC$^{\rightarrow}$ and RIGHT-ARC$^{\rightarrow}_{\text{L}}$. Even with multi-classifications, it takes less than two minutes for the entire training using Liblinear.

[3] http://code.google.com/p/clearparser/

5.3 Accuracy comparisons

Tables 4 and 5 show accuracy comparisons between three models evaluated on the WSJ and Brown corpora, respectively. 'Baseline' uses the features described in Section 4.1. '+Dynamic' uses all baseline features and the dynamic features described in Section 4.2. '+Cluster' uses all previous features and the clustering feature. Even though our baseline system already has high performance, each model shows an improvement over its previous model (very slight for '+Cluster'). The improvement is greater for the out-of-domain task, implying that the dynamic and clustering features help more on new domains. The differences between 'Baseline' and '+Dynamic' are statistically significant for both in and out-of domain tasks (Wilcoxon signed-rank test, treating each sentence as an individual event, $p \leq 0.025$).

	Task	P	R	F1
Baseline	AI	92.57	88.44	90.46
	AI+AC	87.20	83.31	85.21
+Dynamic	AI	92.38	88.76	90.54
	AI+AC	87.33	83.91	85.59*
+Cluster	AI	92.62	88.90	**90.72**
	AI+AC	87.43	83.92	**85.64**
JN (2008)	AI+AC	88.46	83.55	85.93

Table 4: Labeling accuracies evaluated on the WSJ (P: precision, R: recall, F1: F1-score, all in %). 'AI' and 'AC' stand for argument identification and argument classification, respectively.

	Task	P	R	F1
Baseline	AI	90.96	81.57	86.01
	AI+AC	77.11	69.14	72.91
+Dynamic	AI	90.90	82.25	86.36
	AI+AC	77.41	70.05	73.55*
+Cluster	AI	90.87	82.43	**86.44**
	AI+AC	77.47	70.28	**73.70**
JN (2008)	AI+AC	77.67	69.63	73.43

Table 5: Labeling accuracies evaluated on the Brown.

We also compare our results against another state-of-the-art system. Unfortunately, no other system

has been evaluated with our exact environmental settings. However, Johansson and Nugues (2008), who showed state-of-the-art performance in CoNLL'08, evaluated their system with settings very similar to ours. Their task was exactly the same as ours; given predicate identification, they evaluated their dependency-based semantic role labeler for argument identification and classification on the WSJ and Brown corpora, distributed by the CoNLL'05 shared task (Carreras and Màrquez, 2005). Since the CoNLL'05 data was not dependency-based, they applied heuristics to build dependency-based predicate argument structures. Their converted data may appear to be a bit different from the CoNLL'09 data we use (e.g., hyphenated words are tokenized by the hyphens in CoNLL'09 data whereas they are not in CoNLL'05 data), but semantic role annotations on headwords should look very similar.

Johansson and Nugues's results are presented as JN (2008) in Tables 4 and 5. Our final system shows comparable results against this system. These results are meaningful in two ways. First, JN used a graph-based dependency parsing algorithm that gave higher parsing accuracy for these test sets than the transition-based dependency parsing algorithm used in ClearParser (about 0.9% better in labeled attachment score). Even with poorer parse output, our SRL system performed as well as theirs. Furthermore, our system used only one set of features, which makes the feature engineering easier than JN's approach that used different sets of features for argument identification and classification.

6 Conclusion and future work

This paper makes two contributions. First, we introduce a transition-based semantic role labeling algorithm that shows comparable performance against another state-of-the-art system. The new algorithm takes advantage of using previous predictions as features to make the next predictions. Second, we suggest a self-learning clustering technique that improves labeling accuracy slightly in both the domains. The clustering technique shows potential for improving performance in other new domains.

These preliminary results are promising; however, there is still much room for improvement. Since our algorithm is transition-based, many existing techniques such as k-best ranking (Zhang and Clark, 2008) or dynamic programming (Huang and Sagae, 2010) designed to improve transition-based parsing can be applied. We can also apply different kinds of clustering algorithms to improve the quality of the verb clusters. Furthermore, more features, such as named entity tags or dependency labels, can be used to form a better representation of feature vectors for the clustering.

One of the strongest motivations for designing our transition-based SRL system is to develop a joint-inference system between dependency parsing and semantic role labeling. Since we have already developed a dependency parser, ClearParser, based on a parallel transition-based approach, it will be straightforward to integrate this SRL system with the parser. We will also explore the possiblity of adding empty categories during semantic role labeling.

7 Related work

Nivre (2008) introduced several transition-based dependency parsing algorithms that have been widely used. Johansson and Nugues (2008) and Zhao et al. (2009) presented dependency-based semantic role labelers showing state-of-the-art performance for the CoNLL'08 and '09 shared tasks in English. Scheible (2010) clustered predicate argument structures using EM training and the MDL principle. Wagner et al. (2009) used predicate argument clustering to improve verb sense disambiguation.

Acknowledgments

We gratefully acknowledge the support of the National Science Foundation Grants CISE-IIS-RI-0910992, Richer Representations for Machine Translation, a subcontract from the Mayo Clinic and Harvard Children's Hospital based on a grant from the ONC, 90TR0002/01, Strategic Health Advanced Research Project Area 4: Natural Language Processing, 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

X. Carreras and L. Màrquez. 2005. Introduction to the conll-2005 shared task: semantic role labeling. In *Proceedings of the Ninth Conference on Computational Natural Language Learning*.

J. D. Choi and M. Palmer. 2010. Retrieving correct semantic boundaries in dependency structure. In *Proceedings of ACL workshop on Linguistic Annotation*.

J. D. Choi and M. Palmer. 2011. Getting the most out of transition-based dependency parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*.

D. Gildea and D. Jurafsky. 2002. Automatic labeling of semantic roles. *Computational Linguistics*, 28(3).

J. Hajič, M. Ciaramita, R. Johansson, D. Kawahara, M. A. Martí, L. Màrquez, A. Meyers, J. Nivre, S. Padó, J. Štěpánek, P. Straňák, M. Surdeanu, N. Xue, and Y. 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: Shared Task*.

J. A. Hartigan. 1975. *Clustering Algorithms*. New York: John Wiley & Sons.

J. Henderson, P. Merlo, G. Musillo, and I. Titov. 2008. A latent variable model of synchronous parsing for syntactic and semantic dependencies. In *Proceedings of the Twelfth Conference on Computational Natural Language Learning*.

T. Hofmann and J. Puzicha. 1998. Statistical models for co-occurrence data. Technical report, Massachusetts Institute of Technology.

C. Hsieh, K. Chang, C. Lin, S. S. Keerthi, and S. Sundararajan. 2008. A dual coordinate descent method for large-scale linear svm. In *Proceedings of the 25th international conference on Machine learning*.

L. Huang and K. Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*.

R. Johansson and P. Nugues. 2008. Dependency-based semantic role labeling of PropBank. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*.

S. D. Kamvar, D. Klein, and C. D. Manning. 2002. Interpreting and extending classical agglomerative clustering algorithms using a model-based approach. In *Proceedings of the 9th International Conference on Machine Learning*.

D. Liu and D. Gildea. 2010. Semantic role features for machine translation. In *Proceedings of the 23rd International Conference on Computational Linguistics*.

C. Lo, J. Luo, and M. Shieh. 2009. Hardware/software codesign of resource constrained real-time systems. In *Proceedings of the 5th International Conference on Information Assurance and Security*.

R. Mcdonald and F. Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of the Annual Meeting of the European American Chapter of the Association for Computational Linguistics*.

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

M. Palmer, D. Gildea, and P. Kingsbury. 2005. The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1).

S. Pradhan, W. Ward, and J. H. Martin. 2008. Towards robust semantic role labeling. *Computational Linguistics: Special Issue on Semantic Role Labeling*, 34(2).

C. Scheible. 2010. An evaluation of predicate argument clustering using pseudo-disambiguation. In *Proceedings of the 7th conference on International Language Resources and Evaluation*.

D. Shen and M. Lapata. 2007. Using semantic roles to improve question answering. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing and on Computational Natural Language Learning*.

I. Titov, J. Henderson, P. Merlo, and G. Musillo. 2009. Online graph planarisation for synchronous parsing of semantic and syntactic dependencies. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*.

W. Wagner, H. Schmid, and S. Schulte im Walde. 2009. Verb sense disambiguation using a predicate-argument-clustering model. In *Proceedings of the CogSci Workshop on Distributional Semantics beyond Concrete Concepts*.

J. H. Ward. 1963. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58(301).

N. Xue and M. Palmer. 2004. Calibrating features for semantic role labeling. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Y. Zhang and S. Clark. 2008. A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

H. Zhao, W. Chen, and C. Kit. 2009. Semantic dependency parsing of NomBank and PropBank: An efficient integrated approach via a large-scale feature selection. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.