Multi-Argument Classification for Semantic Role Labeling

Chi-San Althon Lin
Department of Computer Science
Waikato University
Hamilton, New Zealand
cl123@cs.waikato.ac.nz

Tony C. Smith
Department of Computer Science
Waikato University
Hamilton, New Zealand
tcs@cs.waikato.ac.nz

Abstract

This paper describes a Multi-Argument Classification (MAC) approach to Semantic Role Labeling. The goal is to exploit dependencies between semantic roles by simultaneously classifying all arguments as a pattern. Argument identification, as a pre-processing stage, is carried at using the improved Predicate-Argument Recognition Algorithm (PARA) developed by Lin and Smith (2006). Results using standard evaluation metrics show that multi-argument classification, achieving 76.60 in F_1 measurement on WSJ 23, outperforms existing systems that use a single parse tree for the CoNLL 2005 shared task data. This paper also describes ways to significantly increase the speed of multi-argument classification, making it suitable for real-time language processing tasks that require semantic role labeling.

1 Introduction

The Conference on Natural Language Learning (CoNLL) has organized different shared tasks since 1999, including syntactic chunking, clause identification, name entity recognition, semantic role labeling (SRL), and multi-lingual dependency parsing. The goal of the problem of SRL as posed for CoNLL 2005 (Carreras and Marquez, 2005) is to recognize all the arguments of given predicates in a sentence and label them with appropriate semantic roles. Arguments related to a predicate are typically phrases in the sentence that form a relationship with the predicate. This relationship is called a semantic role. Generally speaking, SRL is a two step process (though some existing systems address it as a single task). Firstly, all arguments for a predicate must be identified with exact word spans—so-called *argument identification*. Secondly, these arguments must be labelled with correct semantic roles—referred to as *argument classification*.

Existing systems for semantic role labeling use machine learning methods to assign roles one-at-a-time to candidate arguments. There are several drawbacks to this general approach. First, more than one candidate can be assigned the same role, which is undesirable. Second, the search for each candidate argument is exponential with respect to the number of words in the sentence. Third, single-role assignment cannot take advantage of dependencies known to exist between semantic roles of predicate arguments, such as their relative juxtaposition. And fourth, execution times for existing algorithms are excessive, making them unsuitable for real-time use.

This paper seeks to obviate these problems by approaching semantic role labeling as a multi-argument classification process. It observes that the only valid arguments to a predicate are unembedded constituent phrases that do not overlap the predicate. Given that semantic role labeling occurs after parsing, this paper uses the Predicate-Argument Recognition Algorithm (PARA) by Lin and Smith (2006) that systematically traverses the parse tree when looking for arguments, thereby eliminating the vast majority of impossible candidates.

2 Syntax-Driven Argument Identification

Conventional argument identifiers, such as the one developed by Gildea and Palmer (2002), take all nodes in a parse tree, including each word in a sentence, as potential arguments (*pa*). Whether a potential argument is classified as a valid semantic argument depends on a probability estimation such as that given by Gildea and Palmer, (2002) or similar. Such a recognizer is a binary classifier, utilizing the distribution observed in the training data to learn how to predict future novel semantic arguments. Information from a parse tree is forwarded as features to the argument recognizer to help formulate a model to make correct predictions. The most-frequently used features for semantic arguments are the Path, Headword, Phrase type, and Predicate itself, as summarized in Table 1.

Predicate (*pr*) – The given predicate lemma (an uninflected, untensed verb).
Path (*path*) – The syntactic path through the parse tree from the constituent to the given predicate.
Head Word (*hw*) – The syntactic head of the phrase. (The head is normally simply the last noun of the rightmost subordinate noun phrase).

Table 1. Features used in semantic argument identification.

The statistical argument recognizer from Palmer et al. (2005) utilizes the following formula to estimate the probability of a potential argument:

$$\begin{aligned} P(pa \mid path, hw, predicate) = \\ &\lambda_1 * P(pa \mid path) + \\ &\lambda_2 * P(pa \mid path, predicate) + \\ &\lambda_3 * P(pa \mid hw, predicate) \end{aligned}$$

where $\Sigma i\ \lambda_i = 1$.

Traditional argument recognizers have to spend time on each phrase and word to find possible semantic arguments. In order to reduce computational time, Xue and Palmer (2004) describe a pruning strategy to filter out constituents that are clearly not semantic arguments to the predicate. Then they classify the candidates derived from the pruning strategy as either semantic arguments or non-arguments. Finally they use a role classifier to label candidate arguments with semantic roles (Xue and Palmer, 2004). This pruning strategy has been widely used by systems in CoNLL2005 (Punyakanok et al., 2005; Tsai et al., 2005) to reduce training and testing time. Results (like Tsai et al. 2005) show this pruning strategy helps eliminate large portions of the training data (about 61% in Tsai et al. 2005) without sacrificing overall performance. Tsai et al. (2005) claim their systems with the pruning strategy achieve 93% of the correct arguments (or coverage) in training sets.

Generally speaking, valid arguments are non-overlapping and not embedded within each other. State-of-the-art syntactic parsers such as Collins (1999) or Charniak (2000) already solve the overlapping problem and their output provides an ideal structure for finding arguments. The residual problem is to select valid semantic arguments from these non-overlapping constituents of the parse trees. Cursory examination of hand-corrected parses reveals that upper-most nodes that do not include predicates are all valid potential arguments. PARA (Lin and Smith, 2006) was developed in accordance with this observation. The hypothesis is that upper-most nodes in the parse tree that do not include predicates are the potentially valid arguments and need not be rediscovered during argument identification.

PARA has been slightly modified so that it now ignores phrasal nodes that contain just punctuation symbols (an occasional error produced by automatic parsers). This turns out to improve PARA's performance quite significantly, as the following results for the CoNLL 2005 data demonstrate.

Approach	P	R	F_1
PARA-Imp	82.90	82.30	82.60
Moschitti	83.38	81.31	82.33
Palmer	81.30	80.62	80.96
Surdeanu	84.91	76.28	80.36
PARA	82.45	73.94	77.96

Table 2. Comparison of argument identification.

Table 2 shows comparison of argument identification on WSJ23 for four different approaches and the modified PARA (PARA-Imp). These results[1] are based on the official evaluation script[2] offered for the CoNLL shared tasks. The table shows the modification to PARA improves performance from

[1] All arguments are labeled with A0 except predicates.
[2] http://www.lsi.upc.edu/~srlconll/home.html

F_1: 77.96 to 82.60. The improved PARA also outperforms existing approaches using the same syntactic parses as input. It achieves the goal of a direct mapping from syntactic parses to unlabelled semantic arguments *without* the need for training by utilizing the output from a state-of-art parser, such as Charniak's (2000). The new PARA is fast and accurate, and can be used as a stand-alone preprocessor for the problem of Predicate-Argument Recognition or joined with other ML recognizers to increase the overall performance.

Utilizing the new PARA for argument identification, the following section introduces a new technique for multi-argument classification—one that outperforms existing systems in the SRL shared task, given single syntactic information (i.e. one parse tree per sentence).

3 Multi-Argument Classification

Approaches to argument classification are described in detail in the proceedings of CoNLL 2004 and CoNLL 2005 shared tasks. Many address argument classification using Machine Learning (ML) approaches; as with the SNoW learning architecture (Punyakanok et al., 2004, 2005), Support Vector Machines (Moschitti et al., 2005), and so on. The general trend is to try to increase performance by adding more features.

In contrast, this paper applies the concept of Multi-Argument Classification (MAC) to achieve better performance without additional features. MAC is based on the idea of exploiting relationships between roles in predicate-argument structures (e.g. [A0 V A1], [A1 V], etc). Such a relationship is called a ***semantic role dependency***. The relationship in the predicate-argument structure exhibits **semantic role dependency** manifest in the sequential order, count and juxtaposition of different core roles (like A0, or A1) in the predicate-argument list. Generally speaking, there is only one core role in each predicate structure[3]. This can serve as useful information for role classification, as demonstrated in the following classification model.

[3] There is rare situation happened with more than one core role.

3.1 Classification Model

Gildea and Jurafsky (2002) calculate the probability of the optimal role assignment r^* for each sentence as follows.

$$r^* = \operatorname{argmax}_{r_{1\_n}} P(\{r_{1\ldots n}\} \mid predicate) \prod_i \frac{P(r_i \mid \{f_i\}, predicate)}{P(r_i \mid predicate)}$$

$P(r_i \mid \{f_i\}, predicate)$ is the probability of a constituent's role given the above features for the constituent and the predicate. More detail is given in Gildea and Jurasky (2002).

This is a typical ML approach for maximizing the probability of the optimal role assignment to assign roles for each sentence without utilizing role dependency learned from the training data. In Multi-Argument Classification, the optimal probability applied with the role dependency relationship learned from the training data is as follows.

$$r^* = \underset{\{r_{1\ldots n}\}}{\operatorname{argmax}} P(\{r_{1\ldots n}\} \mid predicate) \prod_i \frac{P(\{r_i\} \mid \{f_i\}, predicate)}{P(\{r_i\} \mid predicate)}$$

where $\{r_{1\ldots n}\}$ is a sequential role list learned from the training data, $\{r_i\}$ is the i-th role in $\{r_{1\ldots n}\}$, $P(\{r_{1\ldots n}\} \mid predicate)$ represents the probability of an overall assignment of the role list $\{r_{1\ldots n}\}$ to each of the n constituents or semantic arguments of a sentence, given the *predicate* and the various features $\{f_i\}$of each of the constituents.

The role list $\{r_{1\ldots n}\}$ denotes there are n arguments in a test sentence; but the number of arguments in any training sentence may vary. To compare instances of different lengths, we add a mapping function to convert the role list $\{r_{1\ldots m}\}$ of a training sentence to the role list $\{r_{1\ldots n}\}$ of a test sentence as follows:

$$M: \{r_{1\ldots m}\}_j \rightarrow \{r_{1\ldots n}\}$$

where $\{r_{1\ldots m}\}_j$ is the role list with m arguments of a training sentence j and $\{r_{1\ldots n}\}$ is the role list with n arguments of the test (i.e. query) sentence. The basic principle of this mapping function is to map m arguments of a training sentence to n arguments of the query.

By replacing $\{r_{1\ldots n}\}$ with $M\{r_{1\ldots m}\}_j$ and $\{r_i\}$ with $\{r_{ki}\}$ in the previous formula, the probability formula for MAC is shown as follows.

$$r^* = \underset{M\{r_{1\ldots m}\}_j}{\operatorname{argmax}} P(M\{r_{1\ldots m}\}_j \mid predicate) \prod_i \frac{P(\{r_{kj}\} \mid \{f_i\}, predicate)}{P(\{r_{kj}\} \mid predicate)}$$

where $M\{r_{1...m}\}_j$ is the role list generated by the mapping function M from the j-th training sentences with m arguments of the training data to the role list $\{r_{1...n}\}$ for the test sentence with n arguments, and $\{r_{ki}\}$ denotes the k-th role of $\{r_{1...m}\}_j$ ($1 <= k <= m$) corresponding to the i-th argument of $\{r_{1...n}\}$. The details of the mapping algorithm are described in the next section.

3.2 Mapping Algorithm

There are four considerations essential to the function that maps a knowledge pattern learned from the training data to a new query sentence: *i)* where to start matching two patterns; *ii)* how to deal with different numbers of arguments between the knowledge and query patterns, *iii)* how to compute similarity between an argument in a knowledge pattern and an argument in a query pattern, and *iv)* how to measure the quality of the matching.

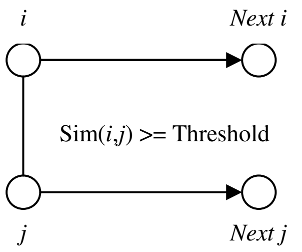
i) Where to start

The first consideration is solved by looking for the most common instance in a knowledge pattern and a query one, given the predicate. In this discussion, an instance is an argument in a predicate-argument structure.

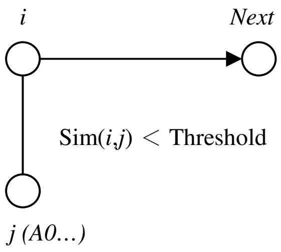
ii) Mapping of different arguments

Empirically there is very low coverage or recall (about 0.46) to match query sentences with training sentences that have the same number of arguments. This paper proposes an alternative way to increase the coverage. The principle is based on semantic role dependency, in which core roles (like A0 or A1) are regarded as *more essential* than adjuncts (like AM-TMP, or AM-LOC). We need to estimate similarity between an argument (i.e. instance) in a knowledge pattern and an argument in the query. If two instances (one in the knowledge pattern and the other in the query) are considered highly similar (Case 1), we can try to match the next instances in both patterns. If two instances are not similar, there are two kinds of situation. One is to match the current instance in the knowledge pattern with the next instance in the query (Case 2). The other is to match the next instance in the knowledge pattern with the current instance in the query (Case 3). The two final circumstances are unmatched instances in the query (Case 4), and unmatched instances in the knowledge pattern (Case 5). These five cases are more formally described as follows:

Case 1: if there exists a query instance i and a corresponding knowledge instance j, and both instances are similar (or their similarity is no less than a threshold), try to match the next instance in the knowledge pattern with the next instance in the query. This is the case when two instances are considered highly similar, then try to match the next instances in the query and knowledge patterns.

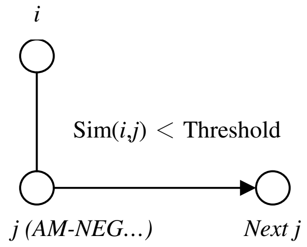


Case 2: if there exists a query instance i and a corresponding knowledge instance j, both instances are not similar (or their similarity is below a threshold) and the role of the knowledge instance j appears to be one of the core roles (i.e. A0 to A5 and AA), as opposed to a non-core or adjunctive role, try to match the current instance in this knowledge pattern with the next instance in the query. The reason to keep the current knowledge instance is to try to increase the coverage. It is rare to have two patterns match exactly due to inherent data sparseness. For example, a query sentence "They will come" and a training sentence, "They come" is not matched due to different number of arguments. But they can be considered highly similar if the second argument "will" in the query sentence is skipped during matching. Such a skipped argument can be labeled latter by other approaches.



Case 3: if there exists a query instance i and a corresponding knowledge instance j, both instances are not similar and the role of the instance j in the knowledge pattern appears to be a non-core label (e.g. AM-MOD AM-NEG or AM-DIS), try to match the next instance in the knowledge pattern with the current instance in the query. Such non-

core roles are optional to a query pattern—which is to say that not all sentences have them. This means they can be skipped. This is the complement situation to Case 2 where all non-core or adjunctive roles in the knowledge pattern can be skipped in order to increase coverage.



Case 4: if there does not exist an argument *j* in the knowledge pattern, keep a default probability to query instance *i* to avoid zero frequency.

Case 5: skip all extra corresponding knowledge arguments.

After mapping, all patterns from the pattern base are compared to role lists with the same number of arguments as the query. It remains now to measure the similarity of each argument in the query role list with each corresponding argument in the role list of each pattern from the knowledge base.

iii) Similarity Function

The calculation of similarity between an instance in the knowledge pattern and its corresponding instance in the query is based on the feature space. The distance between two points (i.e. instances in the feature space) is estimated by Euclidean distance as follows.

Distance metric (Euclidean distance):

$$D(x_i, x_j) = \sqrt{\Sigma(a_r(x_i))-a_r(x_j))^2}$$

for r =1 to *n* (i.e. *n* different classifications), where $a_r(x)$ means the r-th feature of an instance x. (Features used are described in Section 3.5.) If instances, x_i and x_j, are identical, then $D(x_i , x_j)=0$; Otherwise, $D(x_i , x_j)$ represents the vector distance between x_i and x_j. x_i is an instance in the query and x_j is an instance in the knowledge pattern.

Therefore the similarity function is defined as Sim(i, j) = (number of features - D(i, j)) / (number of features)

If all features are the same between two instances, D(i, j) is zero and Sim(i, j) is 1.0. If there are different features between two instances (for example two features are not the same), the score of similarity by Sim(i, j) will be less than 1.0. For example, if there are five features for calculation and two different, D(i, j) is two and Sim(i, j) is 0.6, which is (5 – 2) / 5. The threshold utilized in the first three cases is initially set to 1.0, which means all features in the knowledge instance must be the same with the ones of the query pattern. The threshold value was arrived at through trial and error.

iv) What is the quality estimation

The fourth issue mentioned early in this section is to find the quality of a match between a pattern in a training sentence and a query pattern in the query sentence by the formula given in Section 3.1, except that argument maximization is not used.

Once all quality probabilities for patterns in the training data are calculated, the system selects the pattern from the training sentence with the highest quality probability.

3.3 Unlabeled Arguments

MA is designed for matching two patterns with different arguments. It helps to increase the overall coverage from 0.46 (if only marching patterns with the same number of arguments) to 0.78. This is still not good enough compared to statistical singular-argument classifiers (SAC). The cause of low coverage is sparseness of data. For example, a skipped argument like "will" in the query of Case 2 can be labeled by other approaches (i.e. existing classifiers). Thus we propose a simple argument labeler to fill unlabeled arguments.

$$\underset{r}{\text{argmax}}\ P(r \mid \{f\}, \textit{predicate})$$

where $P(r \mid \{f\}, \textit{predicate})$ represents the probability of an assignment of role *r* (excluding any core role that already appears in the label list to avoid duplication of core roles) to each of the unlabeled arguments of a sentence after MA, given the *predicate* and the features {*f*} (including headword, distance, voice, preposition, phrase type and path) of the argument. By handling unmatched arguments with this simple argument labeler, the recall rises from 0.78 to 0.86.

3.4 Complete PM Model

The complete model for Pattern-Matching (PM) is thus a combination of MAC and SAC. PM tries to find all suitable patterns from the training data using the mapping algorithm described in Section 3.2, selects the best one from the pattern base according to the quality probabilities from the mapping algorithm using MAC, and classifies any unlabelled arguments in the best pattern with SAC like a simple argument labeler in Section 3.3.

```
Procedure of Pattern-Matching with SAC
For all knowledge patterns
    apply Mapping Algorithm for the query and
    knowledge patterns
Select the best knowledge pattern according to
their quality probabilities
Use SAC to classify the unlabelled arguments
```

Figure 1. Procedure of the PM model and SAC.

The goal of selection is to find the knowledge patterns with the highest Quality, calculated by MA described in Section 3.2. The procedure for PM is shown in Figure 1.

In the testing stage for the system, PARA is used as an argument recognizer to identify predicates and arguments related to predicates. It forwards the predicates and their arguments to classification. Argument classification in the system includes two role classifiers, a multi-argument classifier, Pattern-Matching (PM) described and a statistical singular argument classifier modified from Palmer et al. (2005). The modification includes two extra features, preposition and distance described as follows.

3.5 Features

Features used in this paper are predicate, voice, phrase type, distance, headword, path and preposition, as shown in Figure 2.

Predicate	–	The given predicate lemma.
Voice	–	Whether the predicate is realized as an active or passive construction.
Phrase Type	–	The syntactic category (NP, PP, S, etc.) of the phrase corresponding to the semantic argument.
Distance	–	The relative displacement from the predicate, measured in intervening constituents (negative if the constituent is to the left of or prior to, positive if it is to the right of or after, the predicate).
Head Word	–	The syntactic head of the phrase.
Path	–	The syntactic path through the parse tree, from the parse constituent to the predicate being classified.
Preposition	–	The preposition of an argument in a PP such as *during, at, with*, and so on.

Figure 2. Features used for experimentation..

4 Experiments and Results

Data used in this chapter is that released on March 2005 for CoNLL-2005[4], which includes Wall Street Journal sections with Charniak's (2000) and Collins' (1999) parse-trees. Charniak's parse tree is accepted as input to the system due to its better performance on WSJ (Carreras and Marquez, 2005). Evaluation is carried out using the official evaluation script from CoNLL 2005, *srl-eval.pl* which provides precision, recall and F_1 measure of the predicated arguments. Predicates are given in the CoNLL shared tasks.

Table 3 shows the results for several approaches, when used with known arguments (i.e. the systems are given the correct arguments for role classification). All training data (WSJ02-21) with Charniak's parses are included. The modified version of the classifier from Palmer et al. (2005) (*Palmer-Imp)* provides 85.59 in F_1 and the performance of the basic model (*PM without Palmer*) estimation is F_1: 1.16 improved compared to *Palmer* itself. The complete model (*PM*), combined with *Palmer*, achieves the best results on Precision (88.89), Recall (87.65), and F_1 measurement (88.27) and offers the best solution on all test datasets compared to *Palmer-Imp*. It suggests *PM*, utilizing role dependencies existing in semantic roles, helps to increase F_1 by 3.0 over *Palmer-Imp*.

Table 4 shows the result using all features (*ALL*), and the contribution of each feature in Precision (P), Recall (R), and F_1 measurements.

Approach	P	R	F_1
Pamler-Imp	85.53	85.65	85.59
PM without Palmer-Imp	87.67	85.85	86.75
PM	88.89	87.65	88.27

Table 3. Results obtained by different algorithms on WSJ Section 24 with known arguments.

[4] http://www.lsi.upc.edu/~srlconll/soft.html

	P	R	F_1
ALL	88.89	87.65	88.27
- Preposition	84.77	83.03	83.89
- Phrase Type	85.21	83.03	84.11
- Head Word	85.52	83.93	84.72
- Path	87.12	85.89	86.50
- Voice	88.52	87.02	87.77
- Distance	88.81	87.56	88.18

Table 4. Contribution of each feature on WSJ 24, with known arguments.

Test dataset	P	R	F_1
WSJ 24	75.88	72.98	74.40
WSJ 23	78.04	75.20	76.60
Brown	69.33	63.44	66.25

Table 5. Results for different test datasets with Charniak's parses and PARA-Imp.

Test WSJ23	Precision	Recall	$F_{\beta=1}$
Overall	78.04%	75.20%	76.60
A0	84.31%	85.18%	84.74
A1	78.86%	76.98%	77.91
A2	70.83%	61.26%	65.70
A3	68.84%	54.91%	61.09
A4	66.67%	62.75%	64.65
A5	100.00%	60.00%	75.00
AM-ADV	59.07%	55.34%	57.14
AM-CAU	64.91%	50.68%	56.92
AM-DIR	35.53%	31.76%	33.54
AM-DIS	76.25%	76.25%	76.25
AM-EXT	50.00%	37.50%	42.86
AM-LOC	62.54%	51.52%	56.50
AM-MNR	59.33%	51.74%	55.28
AM-MOD	97.42%	95.83%	96.61
AM-NEG	95.18%	94.35%	94.76
AM-PNC	46.39%	39.13%	42.45
AM-PRD	0.00%	0.00%	0.00
AM-REC	0.00%	0.00%	0.00
AM-TMP	73.58%	72.49%	73.03
R-A0	85.84%	86.61%	86.22
R-A1	80.28%	73.08%	76.51
R-A2	80.00%	50.00%	61.54
R-A3	0.00%	0.00%	0.00
R-A4	0.00%	0.00%	0.00
R-AM-ADV	0.00%	0.00%	0.00
R-AM-CAU	0.00%	0.00%	0.00
R-AM-EXT	0.00%	0.00%	0.00
R-AM-LOC	73.68%	66.67%	70.00
R-AM-MNR	25.00%	16.67%	20.00
R-AM-TMP	62.69%	80.77%	70.59

Table 6. Details for each semantic role on WSJ 23, with Charniak's parses and PARA.

Preposition, Phrase Type , and Head word are the three features whose removal decreases the performance of the complete system by a large amount. The distance feature plays a key role in overall performance of *Palmer-Imp* but is the least influential in *PM* because of the usage of multi-argument classification. When using *PM*, the related distance is implicitly included when matching two patterns. The path feature is the fourth most influential factor on performance for role classification, and the voice feature has the least detrimental effect, along with the distance feature, on the performance of this system. Both features (path and voice) have the same influence in *PM* and *Palmer-Imp*.

Table 5 shows performance (on WSJ 24, WSJ 23 and the Brown corpus) of the complete model (PM) using auto parses (Charniak's parser) and PARA as the pre-processor to recognize all related arguments. It also shows the results on WSJ 23 are about F_1:2.0 better than that by WSJ 24. This increase is because the performance by PARA on WSJ 23 is about F_1:2.0 better than WSJ 24. The results on the Brown corpus show the performance drops by more than 10 points in F_1 compared to WSJ 23. This is caused by propagating process-errors described in Carreras and Marquez, (2005). Table 5 also shows such errors affect results even more in the domain of the Brown corpus. Another area for future work is to look for ways to minimize the impact of different domains.

The results on WSJ 23 for each role are shown in Table 6. Generally speaking, performance on core roles is better than on adjuncts, except for the modal, and negation tags. This is because there are more training examples for core roles than for adjuncts.

Experimental results show that execution times for *PM* and *Palmer-Imp* are about 3.0 and 0.8 seconds per sentence respectively. To increase speed, we introduce a controlling strategy called the Maximum Suitable Pattern (MSP) number. MSP limits how many suitable patterns must be found for a query pattern before searching/comparing can stop. The MSP formula is:

$$r^* = \underset{M\{r_{1\ldots m}\}_j}{\overset{\text{Suitable(j)} <= \text{MSP}}{\operatorname{argmax}}} P(M\{r_{1\ldots m}\}_j \mid predicate) \prod_i \frac{P(\{r_{kj}\} \mid \{f_i\}, predicate)}{P(\{r_{kj}\} \mid predicate)}$$

where Suitable(j) denotes the number of suitable knowledge patterns found.

Once PM has found enough suitable patterns (Suitable(j) > MSP), PM stops matching knowledge patterns in the pattern base. A knowledge pattern with at least one instance that has similarity probability greater than the threshold is defined as a suitable one.

Table 7 shows different results for various values of Maximum-Suitable Pattern (MSP) and suggests no improvement after 100 matches. Note that all accuracy differences appear insignificant, but the execution time per sentence (T) increases as the MSP value does, suggesting an MSP between 10 and 20. All execution time are calculated based on a P4 3.0 GHz CPU and 1G RAM Linux machine.

MSP	P	R	F_1	T
30000	89.68	89.20	89.44	2.949
10000	89.68	89.20	89.44	2.943
1000	89.67	89.17	89.42	2.433
100	89.71	89.39	89.55	1.235
50	89.73	89.39	89.56	1.035
20	89.84	89.54	89.69	0.858
10	89.78	89.34	89.56	0.788
2	89.13	88.58	88.86	0.609
1	89.00	88.32	88.66	0.591

Table 7. Results for different MSP values obtained on WSJ 23, with known arguments

System	P	R	F_1	NoF
PARA+PM	78.04	75.20	76.60	7
Surdeanu	80.32	72.95	76.46	31
Tsai	82.77	70.90	76.38	25
Moschitti	76.55	75.24	75.89	14
PARA+Palmer-Imp	71.18	70.90	73.49	7

Table 8. Results for different systems on WSJ 23 listed in the CoNLL 2005 shared task.

Table 8 shows comparative results for various systems using the same input. Surdeanu et al. (2005), Tsai et al. (2005), and Moschitti et al. (2005) are systems only using Charniak's parses listed in CoNLL 2005 shared task. The modified system (*PARA+Palmer_Imp*) is the combination of PARA and *Palmer-Modified.* Even using fewer features, the combination of *PARA* and *PM* offers a more accurate system for SRL compared to systems using the same input. It also becomes one of the top-performing systems in the CoNLL 2005 shared task compared to systems using far more features and multiple parses. It suggests that exploiting role dependencies helps improve accuracy in SRL.

References

Carreras, X. and Marquez, L. (2005). Introduction to the CoNLL-2005 Shared Task: Semantic Role Labeling. In *Proceedings of CoNLL-2005.*

Charniak, E. (2000). A Maximum-Entropy-Inspired Parser. In *Proceedings of NAACL-2000.*

Collins, M. (1999). Head-Driven Statistical Models for Natural Language Parsing. *PhD Dissertation, University of Pennsylvania.*

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

Gildea, D. and Palmer, M. (2002). The Necessity of Parsing for Predicate Argument Recognition . In *Proceedings of ACL 2002, Philadelphia, USA.*

Lin, C.S. A. and Smith, T. C. (2006). A Tree-based Algorithm for Predicate-Argument Recognition. In *Bulletin of Association for Computing Machinery New Zealand (ACM_NZ), volumn 2, issue 1.*

Moschitti, A., Giuglea, A.-M., Coppola, B., and Basili, R. (2005). Semantic role labeling using support vector machines. In *Proceedings of CoNLL-2005.*

Palmer, M., Gildea, D., abd Kingsbury, P., (2005). The Propostin Bank: An Annotated Corpus of Semantic Roles. In *Proceedings of ACL: Volume 31, Number 1. p72-105.*

Punyakanok, V., Koomen, P., Roth, D., and Yih, W. T. (2005). Generalized inference with multiple semantic role labeling systems. In *Proceedings of CoNLL-2005.*

Surdeanu, M. and Turmo, J. (2005). Semantic role labeling using complete syntactic analysis. In *Proceedings of CoNLL-2005.*

Tsai, T.-H., Wu, C.-W., Lin, Y.-C. and Hsu, W.-L. (2005). Exploiting Full Parsing Information to Label Semantic Roles Using an Ensemble of ME and SVM via Integer Linear Programming. In *Proceedings of CoNLL 2005.*

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