The Importance of Syntactic Parsing and Inference in Semantic Role Labeling

Vasin Punyakanok*‡
BBN Technologies

Dan Roth**
University of Illinois at
Urbana-Champaign

Wen-tau Yih†‡
Microsoft Research

We present a general framework for semantic role labeling. The framework combines a machine-learning technique with an integer linear programming–based inference procedure, which incorporates linguistic and structural constraints into a global decision process. Within this framework, we study the role of syntactic parsing information in semantic role labeling. We show that full syntactic parsing information is, by far, most relevant in identifying the argument, especially, in the very first stage—the pruning stage. Surprisingly, the quality of the pruning stage cannot be solely determined based on its recall and precision. Instead, it depends on the characteristics of the output candidates that determine the difficulty of the downstream problems. Motivated by this observation, we propose an effective and simple approach of combining different semantic role labeling systems through joint inference, which significantly improves its performance.

Our system has been evaluated in the CoNLL-2005 shared task on semantic role labeling, and achieves the highest F_1 score among 19 participants.

1. Introduction

Semantic parsing of sentences is believed to be an important task on the road to natural language understanding, and has immediate applications in tasks such as information extraction and question answering. **Semantic Role Labeling** (SRL) is a shallow semantic parsing task, in which for each predicate in a sentence, the goal is to identify all constituents that fill a semantic role, and to determine their roles (Agent, Patient, Instrument, etc.) and their adjuncts (Locative, Temporal, Manner, etc.).

* 10 Moulton St., Cambridge, MA 02138, USA. E-mail: vpunyaka@bbn.com.
** Department of Computer Science, University of Illinois at Urbana-Champaign, 201 N. Goodwin Ave., Urbana, IL 61801, USA. E-mail: danr@uiuc.edu.
† One Microsoft Way, Redmond, WA 98052, USA. E-mail: scottyih@microsoft.com.
‡ Most of the work was done when these authors were at the University of Illinois at Urbana-Champaign.

Submission received: 15 July 2006; revised submission received: 3 May 2007; accepted for publication: 19 June 2007.

The PropBank project (Kingsbury and Palmer 2002; Palmer, Gildea, and Kingsbury 2005), which provides a large human-annotated corpus of verb predicates and their arguments, has enabled researchers to apply machine learning techniques to develop SRL systems (Gildea and Palmer 2002; Chen and Rambow 2003; Gildea and Hockenmaier 2003; Pradhan et al. 2003; Surdeanu et al. 2003; Pradhan et al. 2004; Xue and Palmer 2004; Koomen et al. 2005). However, most systems rely heavily on full syntactic parse trees. Therefore, the overall performance of the system is largely determined by the quality of the automatic syntactic parsers of which the state of the art (Collins 1999; Charniak 2001) is still far from perfect.

Alternatively, shallow syntactic parsers (i.e., chunkers and clausers), although they do not provide as much information as a full syntactic parser, have been shown to be more robust in their specific tasks (Li and Roth 2001). This raises the very natural and interesting question of quantifying the importance of full parsing information to semantic parsing and whether it is possible to use only shallow syntactic information to build an outstanding SRL system.

Although PropBank is built by adding semantic annotations to the constituents in the Penn Treebank syntactic parse trees, it is not clear how important syntactic parsing is for an SRL system. To the best of our knowledge, this problem was first addressed by Gildea and Palmer (2002). In their attempt to use limited syntactic information, the parser they used was *very shallow*—clauses were not available and only chunks were used. Moreover, the pruning stage there was very strict—only chunks were considered as argument candidates. This results in over 60% of the actual arguments being ignored. Consequently, the overall recall in their approach was very low.

The use of only shallow parsing information in an SRL system has largely been ignored until the recent CoNLL-2004 shared task competition (Carreras and Màrquez 2004). In that competition, participants were restricted to using only shallow parsing information, which included part-of-speech tags, chunks, and clauses (the definitions of chunks and clauses can be found in Tjong Kim Sang and Buchholz [2000] and Carreras et al. [2002], respectively). As a result, the performance of the best shallow parsing–based system (Hacioglu et al. 2004) in the competition is about 10 points in F_1 below the best system that uses full parsing information (Koomen et al. 2005). However, this is not the outcome of a true and fair quantitative comparison. The CoNLL-2004 shared task used only a subset of the data for training, which potentially makes the problem harder. Furthermore, an SRL system is usually complicated and consists of several stages. It was still unclear how much syntactic information helps and precisely where it helps the most.

The goal of this paper is threefold. First, we describe an architecture for an SRL system that incorporates a level of global inference on top of the relatively common processing steps. This inference step allows us to incorporate structural and linguistic constraints over the possible outcomes of the argument classifier in an easy way. The inference procedure is formalized via an Integer Linear Programming framework and is shown to yield state-of-the-art results on this task. Second, we provide a fair comparison between SRL systems that use full parse trees and systems that only use shallow syntactic information. As with our full syntactic parse–based SRL system (Koomen et al. 2005), our shallow parsing–based SRL system is based on the system that achieves very competitive results and was one of the top systems in the CoNLL-2004 shared task competition (Carreras and Màrquez 2004). This comparison brings forward a careful analysis of the significance of full parsing information in the SRL task, and provides an understanding of the stages in the process in which this information makes the most difference. Finally, to relieve the dependency of the SRL system on the quality of

automatic parsers, we suggest a way to improve semantic role labeling significantly by developing a global inference algorithm, which is used to combine several SRL systems based on different state-of-the-art full parsers. The combination process is done through a joint inference stage, which takes the output of each individual system as input and generates the best predictions, subject to various structural and linguistic constraints.

The underlying system architecture can largely affect the outcome of our study. Therefore, to make the conclusions of our experimental study as applicable as possible to general SRL systems, the architecture of our SRL system follows the most widely used two-step design. In the first step, the system is trained to identify argument candidates for a given verb predicate. In the second step, the system classifies the argument candidates into their types. In addition, it is also a simple procedure to prune obvious non-candidates before the first step, and to use post-processing inference to fix inconsistent predictions after the second step. These two additional steps are also employed by our system.

Our study of shallow and full syntactic information–based SRL systems was done by comparing their impact at each stage of the process. Specifically, our goal is to investigate at what stage full parsing information is most helpful relative to a shallow parsing–based system. Therefore, our experiments were designed so that the compared systems are as similar as possible, and the addition of the full parse tree–based features is the only difference. The most interesting result of this comparison is that although each step of the shallow parsing information–based system exhibits very good performance, the overall performance is significantly inferior to the system that uses full parsing information. Our explanation is that *chaining* multiple processing stages to produce the final SRL analysis is crucial to understanding this analysis. Specifically, the *quality* of the information passed from one stage to the other is a decisive issue, and it is not necessarily judged simply by considering the F-measure. We conclude that, for the system architecture used in our study, the significance of full parsing information comes into play mostly at the pruning stage, where the candidates to be processed later are determined. In addition, we produce a state-of-the-art SRL system by combining different SRL systems based on two automatic full parsers (Collins 1999; Charniak 2001), which achieves the best result in the CoNLL-2005 shared task (Carreras and Màrquez 2005).

The rest of this paper is organized as follows. Section 2 introduces the task of semantic role labeling in more detail. Section 3 describes the four-stage architecture of our SRL system, which includes pruning, argument identification, argument classification, and inference. The features used for building the classifiers and the learning algorithm applied are also explained there. Section 4 explains why and where full parsing information contributes to SRL by conducting a series of carefully designed experiments. Inspired by the result, we examine the effect of inference in a single system and propose an approach that combines different SRL systems based on joint inference in Section 5. Section 6 presents the empirical evaluation of our system in the CoNLL-2005 shared task competition. After that, we discuss the related work in Section 7 and conclude this paper in Section 8.

2. The Semantic Role Labeling (SRL) Task

The goal of the semantic role labeling task is to discover the predicate–argument structure of each predicate in a given input sentence. In this work, we focus only on the verb predicate. For example, given a sentence *I left my pearls to my daughter-in-law in my will*,

the goal is to identify the different arguments of the verb predicate *left* and produce the output:

[$_{A0}$ I] [$_{V}$ *left*] [$_{A1}$ my pearls] [$_{A2}$ to my daughter-in-law] [$_{AM-LOC}$ in my will].

Here A0 represents the *leaver*, A1 represents the *thing left*, A2 represents the *beneficiary*, AM-LOC is an adjunct indicating the location of the action, and V determines the boundaries of the predicate, which is important when a predicate contains many words, for example, a phrasal verb. In addition, each argument can be mapped to a constituent in its corresponding full syntactic parse tree.

Following the definition of the PropBank and CoNLL-2004 and 2005 shared tasks, there are six different types of arguments labeled as A0–A5 and AA. These labels have different semantics for each verb and each of its senses as specified in the PropBank Frame files. In addition, there are also 13 types of adjuncts labeled as AM-*adj* where *adj* specifies the adjunct type. For simplicity in our presentation, we will also refer to these adjuncts as arguments. In some cases, an argument may span over different parts of a sentence; the label C-*arg* is then used to specify the continuity of the arguments, as shown in this example:

[$_{A1}$ The pearls] , [$_{A0}$ I] [$_{V}$ *said*] , [$_{C-A1}$ were left to my daughter-in-law].

In some other cases, an argument might be a relative pronoun that in fact refers to the actual agent outside the clause. In this case, the actual agent is labeled as the appropriate argument type, *arg*, while the relative pronoun is instead labeled as R-*arg*. For example,

[$_{A1}$ The pearls] [$_{R-A1}$ which] [$_{A0}$ I] [$_{V}$ *left*] [$_{A2}$ to my daughter-in-law] are fake.

Because each verb may have different senses producing different semantic roles for the same labels, the task of discovering the complete set of semantic roles should involve not only identifying these labels, but also the underlying sense for a given verb. However, as in all current SRL work, this article focuses only on identifying the boundaries and the labels of the arguments, and ignores the verb sense disambiguation problem.

The distribution of these argument labels is fairly unbalanced. In the official release of PropBank I, core arguments (A0–A5 and AA) occupy 71.26% of the arguments, where the largest parts are A0 (25.39%) and A1 (35.19%). The rest mostly consists of adjunct arguments (24.90%). The continued (C-*arg*) and referential (R-*arg*) arguments are relatively few, occupying 1.22% and 2.63%, respectively. For more information on PropBank and the semantic role labeling task, readers can refer to Kingsbury and Palmer (2002) and Carreras and Màrquez (2004, 2005).

Note that the semantic arguments of the same verb do not *overlap*. We define **overlapping** arguments to be those that share some of their parts. An argument is considered **embedded** in another argument if the second argument completely covers the first one. Arguments are **exclusively overlapping** if they are overlapping but are not embedded.

3. SRL System Architecture

Adhering to the most common architecture for SRL systems, our SRL system consists of four stages: **pruning**, **argument identification**, **argument classification**, and **inference**. In particular, the goal of pruning and argument identification is to identify argument candidates for a given verb predicate. In the first three stages, however, decisions are independently made for each argument, and information across arguments is not

incorporated. The final inference stage allows us to use this type of information along with linguistic and structural constraints in order to make consistent global predictions.

This system architecture remains unchanged when used for studying the importance of syntactic parsing in SRL, although different information and features are used. Throughout this article, when **full parsing** information is available, we assume that the system is presented with the full phrase-structure parse tree as defined in the Penn Treebank (Marcus, Marcinkiewicz, and Santorini 1993) but without trace and functional tags. On the other hand, when only **shallow parsing** information is available, the full parse tree is reduced to only the chunks and the clause constituents.

A **chunk** is a phrase containing syntactically related words. Roughly speaking, chunks are obtained by projecting the full parse tree onto a flat tree; hence, they are closely related to the base phrases. Chunks were not directly defined as part of the standard annotation of the treebank, but, rather, their definition was introduced in the CoNLL-2000 shared task on text chunking (Tjong Kim Sang and Buchholz 2000), which aimed to discover such phrases in order to facilitate full parsing. A **clause**, on the other hand, is the clausal constituent as defined by the treebank standard. An example of chunks and clauses is shown in Figure 1.

3.1 Pruning

When the full parse tree of a sentence is available, only the constituents in the parse tree are considered as argument candidates. Our system exploits the heuristic rules introduced by Xue and Palmer (2004) to filter out simple constituents that are very unlikely to be arguments. This pruning method is a recursive process starting from the target verb. It first returns the siblings of the verb as candidates; then it moves to the parent of the verb, and collects the siblings again. The process goes on until it reaches the root. In addition, if a constituent is a PP (prepositional phrase), its children are also collected. For example, in Figure 1, if the predicate (target verb) is `assume`, the pruning heuristic will output: [$_{\text{PP}}$ *by John Smith who has been elected deputy chairman*], [$_{\text{NP}}$ *John Smith who has been elected deputy chairman*], [$_{\text{VB}}$ *be*], [$_{\text{MD}}$ *will*], and [$_{\text{NP}}$ *His duties*].

3.2 Argument Identification

The argument identification stage utilizes binary classification to identify whether a candidate is an argument or not. When full parsing is available, we train and apply the binary classifiers on the constituents supplied by the pruning stage. When only shallow parsing is available, the system does not have a pruning stage, and also does not have constituents to begin with. Therefore, conceptually, the system has to consider all possible subsequences (i.e., consecutive words) in a sentence as potential argument candidates. We avoid this by using a learning scheme that utilizes two classifiers, one to predict the beginnings of possible arguments, and the other the ends. The predictions are combined to form argument candidates. However, we can employ a simple heuristic to filter out some candidates that are obviously not arguments. The final predication includes those that do not violate the following constraints.

1. Arguments cannot overlap with the predicate.
2. If a predicate is outside a clause, its arguments cannot be embedded in that clause.
3. Arguments cannot exclusively overlap with the clauses.

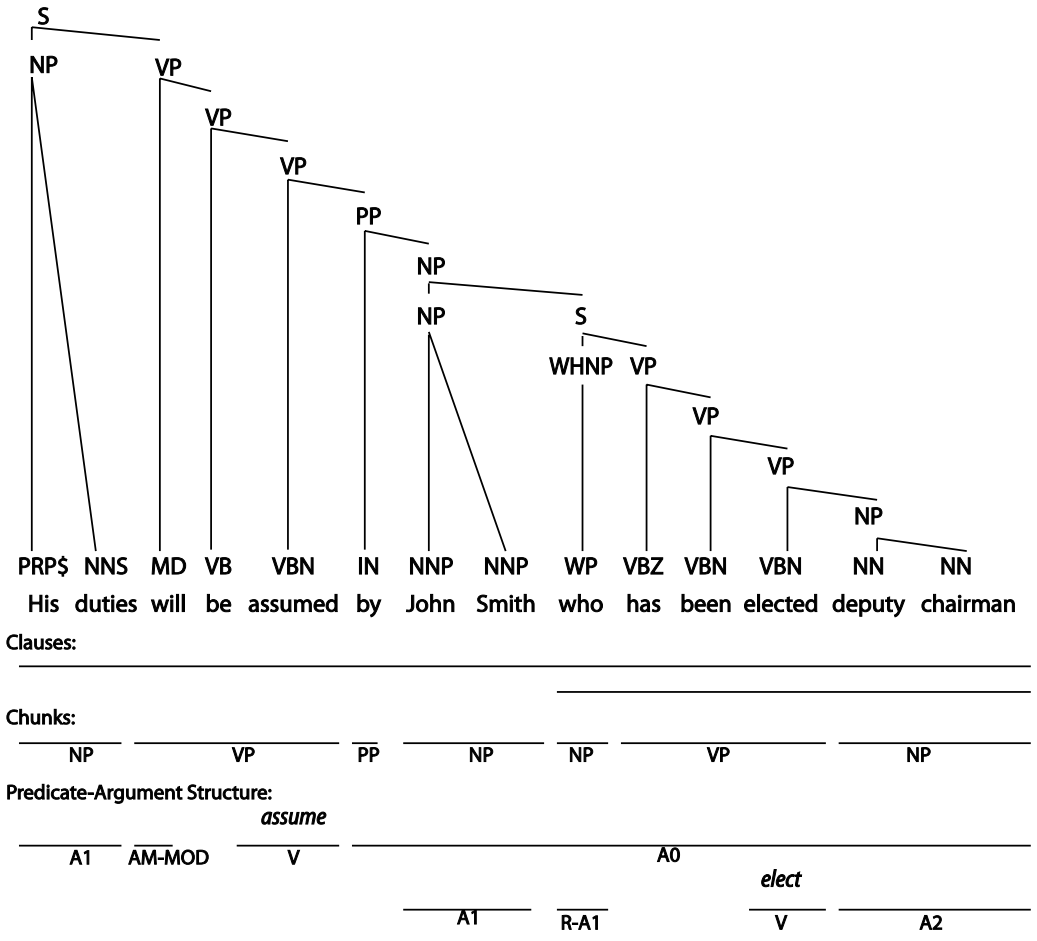

Figure 1
An example of a parse tree and its predicate–argument structure.

The first constraint comes from the definition of this task that the predicate simply cannot take itself or any constituents that contain itself as arguments. The other two constraints are due to the fact that a clause can be treated as a unit that has its own verb–argument structure. If a verb predicate is outside a clause, then its argument can only be the whole clause, but may not be embedded in or exclusively overlap with the clause.

For the argument identification classifier, the features used in full parsing and shallow parsing settings are all binary features, which are described subsequently.

3.2.1 Features Used When Full Parsing is Available. Most of the features used in our system are common features for the SRL task. The creation of PropBank was inspired by the works of Levin (1993) and Levin and Hovav (1996), which discuss the relation between syntactic and semantic information. Following this philosophy, the features aim to indicate the properties of the predicate, the constituent which is an argument candidate, and the relationship between them through the available syntactic information. We explain these features herein. For further discussion of these features, we

refer the readers to the article by Gildea and Jurafsky (2002), which introduced these features.

- **Predicate and POS tag of predicate:** indicate the lemma of the predicate verb and its POS tag.
- **Voice:** indicates passive/active voice of the predicate.
- **Phrase type:** provides the phrase type of the constituent, which is the tag of the corresponding constituent in the parse tree.
- **Head word and POS tag of the head word:** provides the head word of the constituent and its POS tag. We use the rules introduced by Collins (1999) to extract this feature.
- **Position:** describes if the constituent is before or after the predicate, relative to the position in the sentence.
- **Path:** records the tags of parse tree nodes in the traversal path from the constituent to the predicate. For example, in Figure 1, if the predicate is *assume* and the constituent is [$_S$ *who has been elected deputy chairman*], the path is S↑NP↑PP↑VP↓VBN, where ↑ and ↓ indicate the traversal direction in the path.
- **Subcategorization:** describes the phrase structure around the predicate's parent. It records the immediate structure in the parse tree that expands to its parent. As an example, if the predicate is *elect* in Figure 1, its subcategorization is VP→(VBN)-NP while the subcategorization of the predicate *assume* is VP→(VBN)-PP. Parentheses indicate the position of the predicate.

Generally speaking, we consider only the arguments that correspond to some constituents in parse trees. However, in some cases, we need to consider an argument that does not exactly correspond to a constituent, for example, in our experiment in Section 4.2 where the gold-standard boundaries are used with the parse trees generated by an automatic parse. In such cases, if the information on the constituent, such as phrase type, needs to be extracted, the deepest constituent that covers the whole argument will be used. For example, in Figure 1, the phrase type for *by John Smith* is PP, and its path feature to the predicate *assume* is PP↑VP↓VBN.

We also use the following additional features. These features have been shown to be useful for the systems by exploiting other information in the absence of the full parse tree information (Punyakanok et al. 2004), and, hence, can be helpful in conjunction with the features extracted from a full parse tree. They also aim to encode the properties of the predicate, the constituent to be classified, and their relationship in the sentence.

- **Context words and POS tags of the context words:** the feature includes the two words before and after the constituent, and their POS tags.
- **Verb class:** the feature is the VerbNet (Kipper, Palmer, and Rambow 2002) class of the predicate as described in PropBank Frames. Note that a

verb may inhabit many classes and we collect all of these classes as features, regardless of the context-specific sense which we do not attempt to resolve.

- **Lengths:** of the constituent, in the numbers of words and chunks separately.
- **Chunk:** tells if the constituent "is," "embeds," "exclusively overlaps," or "is embedded in" a chunk with its type. For instance, in Figure 1, if the constituents are [$_{NP}$ *His duties*], [$_{PP}$ *by John Smith*], and [$_{VBN}$ *elected*], then their chunk features are "is-NP," "embed-PP & embed-NP," and "embedded-in-VP," respectively.
- **Chunk pattern:** encodes the sequence of chunks from the constituent to the predicate. For example, in Figure 1 the chunk sequence from [$_{NP}$ *His duties*] to the predicate *elect* is VP-PP-NP-NP-VP.
- **Chunk pattern length:** the feature counts the number of chunks in the chunk pattern feature.
- **Clause relative position:** encodes the position of the constituent relative to the predicate in the pseudo-parse tree constructed only from clause constituents, chunks, and part-of-speech tags. In addition, we label the clause with the type of chunk that immediately precedes the clause. This is a simple rule to distinguish the type of clause based on the intuition that a subordinate clause often modifies the part of the sentence immediately before it. Figure 2 shows the pseudo-parse tree of the parse tree in Figure 1. By disregarding the chunks, there are four configurations—"target constituent and predicate are siblings," "target constituent's parent is an ancestor of predicate," "predicate's parent is an ancestor of target word," or "otherwise." This feature can be viewed as a generalization of the Path feature described earlier.
- **Clause coverage:** describes how much of the local clause from the predicate is covered by the target argument.
- **NEG:** the feature is active if the target verb chunk has *not* or *n't*.
- **MOD:** the feature is active when there is a modal verb in the verb chunk. The rules of the NEG and MOD features are used in a baseline SRL system developed by Erik Tjong Kim Sang (Carreras and Màrquez 2004).

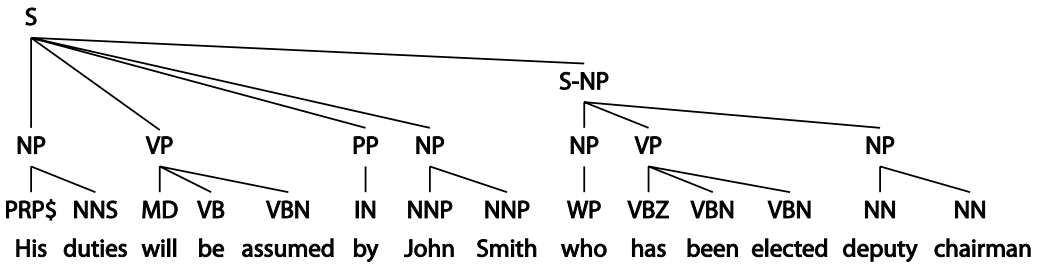

Figure 2
The pseudo-parse tree generated from the parse tree in Figure 1.

In addition, we also use the conjunctions of features which conjoin any two features into a new feature. For example, the conjunction of the predicate and path features for the predicate *assume* and the constituent [$_S$ *who has been elected deputy chairman*] in Figure 1 is (S↑NP↑PP↑VP↓VBN, *assume*).

3.2.2 Features Used When Only Shallow Parsing is Available. Most features used here are similar to those used by the system with full parsing information. However, for features that need full parse trees in their extraction procedures, we either try to mimic them with some heuristic rules or discard them. The details of these features are as follows.

- **Phrase type:** uses a simple heuristic to identify the type of the argument candidate as VP, PP, or NP.
- **Head word and POS tag of the head word:** are the rightmost word for NP, and the leftmost word for VP and PP.
- **Shallow-Path:** records the traversal path in the pseudo-parse tree. This aims to approximate the Path features extracted from the full parse tree.
- **Shallow-Subcategorization:** describes the chunk and clause structure around the predicate's parent in the pseudo-parse tree. This aims to approximate the Subcategorization feature extracted from the full parse tree.

3.3 Argument Classification

This stage assigns labels to the argument candidates identified in the previous stage. A multi-class classifier is trained to predict the types of the argument candidates. In addition, to reduce the excessive candidates mistakenly output by the previous stage, the classifier can also label an argument as "null" (meaning "not an argument") to discard it.

The features used here are the same as those used in the argument identification stage. However, when full parsing is available, an additional feature introduced by Xue and Palmer (2004) is used.

- **Syntactic frame:** describes the sequential pattern of the noun phrases and the predicate in the sentence which aims to complement the Path and Subcategorization features.

The learning algorithm used for training the argument classifier and argument identifier is a variation of the Winnow update rule incorporated in SNoW (Roth 1998; Carlson et al. 1999), a multi-class classifier that is tailored for large scale learning tasks. SNoW learns a sparse network of linear functions, in which the targets (argument border predictions or argument type predictions, in this case) are represented as linear functions over a common feature space; multi-class decisions are done via a winner-take-all mechanism. It improves the basic Winnow multiplicative update rule with a regularization term, which has the effect of separating the data with a large margin separator (Dagan, Karov, and Roth 1997; Grove and Roth 2001; Zhang, Damerau, and Johnson 2002) and voted (averaged) weight vector (Freund and Schapire 1999; Golding and Roth 1999).

The **softmax** function (Bishop 1995) is used to convert raw activation to conditional probabilities. If there are n classes and the raw activation of class i is act_i, the posterior estimation for class i is

$$Prob(i) = \frac{e^{act_i}}{\sum_{1 \leq j \leq n} e^{act_j}}$$

Note that in training this classifier, unless specified otherwise, the argument candidates used to generate the training examples are obtained from the output of the argument identifier, not directly from the gold-standard corpus. In this case, we automatically obtain the necessary examples to learn for class "null."

3.4 Inference

In the previous stages, decisions were always made for each argument independently, ignoring the global information across arguments in the final output. The purpose of the inference stage is to incorporate such information, including both linguistic and structural knowledge, such as "arguments do not overlap" or "each verb takes at most one argument of each type." This knowledge is useful to resolve any inconsistencies of argument classification in order to generate final legitimate predictions. We design an inference procedure that is formalized as a constrained optimization problem, represented as an integer linear program (Roth and Yih 2004). It takes as input the argument classifiers' confidence scores for each type of argument, along with a list of constraints. The output is the optimal solution that maximizes the linear sum of the confidence scores, subject to the constraints that encode the domain knowledge.

The inference stage can be naturally extended to combine the output of several different SRL systems, as we will show in Section 5. In this section we first introduce the constraints and formalize the inference problem for the semantic role labeling task. We then demonstrate how we apply integer linear programming (ILP) to generate the global label assignment.

3.4.1 Constraints over Argument Labeling. Formally, the argument classifiers attempt to assign labels to a set of arguments, $S^{1:M}$, indexed from 1 to M. Each argument S^i can take any label from a set of argument labels, $\mathcal{P}$, and the indexed set of arguments can take a set of labels, $c^{1:M} \in \mathcal{P}^M$. If we assume that the classifiers return a score score$(S^i = c^i)$ that corresponds to the likelihood of argument S^i being labeled c^i then, given a sentence, the unaltered inference task is solved by maximizing the overall score of the arguments,

$$\hat{c}^{1:M} = \underset{c^{1:M} \in \mathcal{P}^M}{\operatorname{argmax}} \operatorname{score}(S^{1:M} = c^{1:M}) = \underset{c^{1:M} \in \mathcal{P}^M}{\operatorname{argmax}} \sum_{i=1}^{M} \operatorname{score}(S^i = c^i) \quad (1)$$

In the presence of global constraints derived from linguistic information and structural considerations, our system seeks to output a *legitimate* labeling that maximizes this score. Specifically, it can be thought of as if the solution space is limited through the use of a filter function, $\mathcal{F}$, which eliminates many argument labelings from consideration.

Here, we are concerned with global constraints as well as constraints on the arguments. Therefore, the final labeling becomes

$$\hat{c}^{1:M} = \underset{c^{1:M} \in \mathcal{F}(\mathcal{P}^M)}{\operatorname{argmax}} \sum_{i=1}^{M} \operatorname{score}(S^i = c^i) \tag{2}$$

When the confidence scores correspond to the conditional probabilities estimated by the argument classifiers, the value of the objective function represents the expected number of correct argument predictions. Hence, the solution of Equation (2) is the one that maximizes this expected value among all legitimate outputs.

The filter function used considers the following constraints:[1]

1. Arguments cannot overlap with the predicate.
2. Arguments cannot exclusively overlap with the clauses.
3. If a predicate is outside a clause, its arguments cannot be embedded in that clause.
4. No overlapping or embedding arguments.
   This constraint holds because semantic arguments are labeled on non-embedding constituents in the syntactic parse tree. In addition, as defined in the CoNLL-2004 and 2005 shared tasks, the legitimate output of an SRL system must satisfy this constraint.
5. No duplicate argument classes for core arguments, such as A0–A5 and AA. The only exception is when there is a conjunction in the sentence. For example,

   [$_{A0}$ I] [$_{V}$ *left*] [$_{A1}$ my pearls] [$_{A2}$ to my daughter] and [$_{A1}$ my gold] [$_{A2}$ to my son].

   Despite this exception, we treat it as a hard constraint because it almost always holds.
6. If there is an R-*arg* argument, then there has to be an *arg* argument. That is, if an argument is a reference to some other argument *arg*, then this referenced argument must exist in the sentence. This constraint is directly derived from the definition of R-*arg* arguments.
7. If there is a C-*arg* argument, then there has to be an *arg* argument; in addition, the C-*arg* argument must occur after *arg*. This is stricter than the previous rule because the order of appearance also needs to be considered. Similarly, this constraint is directly derived from the definition of C-*arg* arguments.
8. Given the predicate, some argument classes are illegal (e.g., predicate *stalk* can take only A0 or A1). This information can be found in PropBank Frames.

1 There are other constraints such as "exactly one V argument per class," or "V–A1–C-V pattern" as introduced by Punyakanok et al. (2004). However, we did not find them particularly helpful in our experiments. Therefore, we exclude those constraints in the presentation here.

This constraint comes from the fact that different predicates take different types and numbers of arguments. By checking the PropBank Frame file of the target verb, we can exclude some core argument labels.

Note that constraints 1, 2, and 3 are actually implemented in the argument identification stage (see Section 3.2). In addition, they need to be explicitly enforced only when full parsing information is not available because the output of the pruning heuristics never violates these constraints.

The optimization problem (Equation (2)) can be solved using an ILP solver by reformulating the constraints as linear (in)equalities over the indicator variables that represent the truth value of statements of the form [*argument i takes label j*], as described in detail next.

3.4.2 Using Integer Linear Programming. As discussed previously, a collection of potential arguments is not necessarily a valid semantic labeling because it may not satisfy all of the constraints. We enforce a legitimate solution using the following inference algorithm. In our context, inference is the process of finding the *best* (according to Equation (1)) valid semantic labels that satisfy all of the specified constraints. We take a similar approach to the one previously used for entity/relation recognition (Roth and Yih 2004), and model this inference procedure as solving an ILP problem.

An **integer linear program** is a **linear program** with integral variables. That is, the cost function and the (in)equality constraints are all linear in terms of the variables. The only difference in an integer linear program is that the variables can only take integers as their values. In our inference problem, the variables are in fact binary. A general binary integer linear programming problem can be stated as follows.

Given a cost vector $\mathbf{p} \in \Re^d$, a collection of variables $\mathbf{u} = (u_1, \ldots, u_d)$ and cost matrices $\mathbf{C}_1 \in \Re^{c_1} \times \Re^d, \mathbf{C}_2 \in \Re^{c_2} \times \Re^d$, where c_1 and c_2 are the numbers of inequality and equality constraints and d is the number of binary variables, the ILP solution $\mathbf{u}^*$ is the vector that maximizes the cost function,

$$\mathbf{u}^* = \underset{\mathbf{u} \in \{0,1\}^d}{\operatorname{argmax}} \mathbf{p} \cdot \mathbf{u}$$

subject to

$$\mathbf{C}_1\mathbf{u} \geq \mathbf{b}_1, \text{ and } \mathbf{C}_2\mathbf{u} = \mathbf{b}_2$$

where $\mathbf{b}_1 \in \Re^{c_1}, \mathbf{b}_2 \in \Re^{c_2}$, and for all $\mathbf{u} \in \{0,1\}^d$.

To solve the problem of Equation (2) in this setting, we first reformulate the original cost function $\sum_{i=1}^{M} \text{score}(S^i = c^i)$ as a linear function over several binary variables, and then represent the filter function $\mathcal{F}$ using linear inequalities and equalities.

We set up a bijection from the semantic labeling to the variable set $\mathbf{u}$. This is done by setting $\mathbf{u}$ to be a set of indicator variables that correspond to the labels assigned to arguments. Specifically, let $u_{ic} = [S^i = c]$ be the indicator variable that represents whether

or not the argument type c is assigned to S^i, and let $p_{ic} = \text{score}(S^i = c)$. Equation (1) can then be written as an ILP cost function as

$$\underset{u_{ic} \in \{0,1\}: \forall i \in [1,M], c \in \mathcal{P}}{\text{argmax}} \sum_{i=1}^{M} \sum_{c \in \mathcal{P}} p_{ic} u_{ic}$$

subject to

$$\sum_{c \in \mathcal{P}} u_{ic} = 1 \qquad \forall i \in [1, M]$$

which means that each argument can take only one type. Note that this new constraint comes from the variable transformation, and is not one of the constraints used in the filter function $\mathcal{F}$.

Of the constraints listed earlier, constraints 1 through 3 can be evaluated on a per-argument basis and, for the sake of efficiency, arguments that violate these constraints are eliminated even before being given to the argument classifier. Next, we show how to transform the constraints in the filter function into the form of linear (in)equalities over $\mathbf{u}$ and use them in this ILP setting. For a more complete example of this ILP formulation, please see Appendix A.

Constraint 4: No overlapping or embedding. If arguments $S^{j_1}, \ldots, S^{j_k}$ cover the same word in a sentence, then this constraint ensures that at most one of the arguments is assigned to an argument type. In other words, at least $k - 1$ arguments will be the special class *null*. If the special class *null* is represented by the symbol ϕ, then for every set of such arguments, the following linear equality represents this constraint.

$$\sum_{i=1}^{k} u_{j_i \phi} \geq k - 1$$

Constraint 5: No duplicate argument classes. Within the same clause, several types of arguments cannot appear more than once. For example, a predicate can only take one A0. This constraint can be represented using the following inequality.

$$\sum_{i=1}^{M} u_{i\text{A0}} \leq 1$$

*Constraint 6: R-*arg *arguments.* Suppose the referenced argument type is A0 and the referential type is R-A0. The linear inequalities that represent this constraint are:

$$\forall m \in \{1, \ldots, M\} : \sum_{i=1}^{M} u_{i\text{A0}} \geq u_{m\text{R-A0}}$$

If there are γ referential types, then the total number of inequalities needed is γM.

*Constraint 7: C-*arg *arguments.* This constraint is similar to the reference argument constraints. The difference is that the continued argument *arg* has to occur before C-*arg*.

Assume that the argument pair is A0 and C-A0, and arguments are sorted by their beginning positions, i.e., if $i < k$, the position of the beginning of S^{j_k} is not before that of the beginning of S^{j_i}. The linear inequalities that represent this constraint are:

$$\forall m \in \{2, \ldots, M\} : \sum_{i=1}^{m-1} u_{j_i \text{A0}} \geq u_{j_m \text{C-A0}}$$

Constraint 8: Illegal argument types. Given a specific verb, some argument types should never occur. For example, most verbs do not have arguments A5. This constraint is represented by summing all the corresponding indicator variables to be 0.

$$\sum_{i=1}^{M} u_{i\text{A5}} = 0$$

Using ILP to solve this inference problem enjoys several advantages. Linear constraints are very general, and are able to represent any Boolean constraint (Guéret, Prins, and Sevaux 2002). Table 1 summarizes the transformations of common constraints (most are Boolean), which are revised from Guéret, Prins, and Sevaux (2002), and can be used for constructing complicated rules.

Previous approaches usually rely on dynamic programming to resolve non-overlapping/embedding constraints (i.e., Constraint 4) when the constraint structure is sequential. However, they are not able to handle more expressive constraints such as those that take long-distance dependencies and counting dependencies into account (Roth and Yih 2005). The ILP approach, on the other hand, is flexible enough to handle more expressive and general constraints. Although solving an ILP problem is NP-hard in the worst case, with the help of today's numerical packages, this problem can usually be solved very quickly in practice. For instance, in our experiments it only took about 10 minutes to solve the inference problem for 4,305 sentences, using

Table 1
Rules of mapping constraints to linear (in)equalities for Boolean variables.

Original constraint	Linear form
exactly k of $x_1, x_2, \cdots, x_n$	$x_1 + x_2 + \cdots + x_n = k$
at most k of $x_1, x_2, \cdots, x_n$	$x_1 + x_2 + \cdots + x_n \leq k$
at least k of $x_1, x_2, \cdots, x_n$	$x_1 + x_2 + \cdots + x_n \geq k$
$a \rightarrow b$	$a \leq b$
$a = \bar{b}$	$a = 1 - b$
$a \rightarrow \bar{b}$	$a + b \leq 1$
$\bar{a} \rightarrow b$	$a + b \geq 1$
$a \leftrightarrow b$	$a = b$
$a \rightarrow b \wedge c$	$a \leq b$ and $a \leq c$
$a \rightarrow b \vee c$	$a \leq b + c$
$b \wedge c \rightarrow a$	$a \geq b + c - 1$
$b \vee c \rightarrow a$	$a \geq (b + c)/2$
$a \rightarrow$ at least k of $x_1, x_2, \cdots, x_n$	$a \leq (x_1 + x_2 + \cdots + x_n)/k$
At least k of $x_1, x_2, \cdots, x_n \rightarrow a$	$a \geq (x_1 + x_2 + \cdots + x_n - (k-1))/(n - (k-1))$

Xpress-MP (2004) running on a Pentium-III 800 MHz machine. Note that ordinary search methods (e.g., beam search) are not necessarily faster than solving an ILP problem and do not guarantee the optimal solution.

4. The Importance of Syntactic Parsing

We experimentally study the significance of syntactic parsing by observing the effects of using full parsing and shallow parsing information at each stage of an SRL system. We first describe, in Section 4.1, how we prepare the data. The comparison of full parsing and shallow parsing on the first three stages of the process is presented in the reverse order (Sections 4.2, 4.3, 4.4). Note that in the following sections, in addition to the performance comparison at various stages, we present also the overall system performance for the different scenarios. In all cases, the overall system performance is derived after the inference stage.

4.1 Experimental Setting

We use PropBank Sections 02 through 21 as training data, Section 23 as testing, and Section 24 as a validation set when necessary. In order to apply the standard CoNLL shared task evaluation script, our system conforms to both the input and output format defined in the shared task.

The goal of the experiments in this section is to understand the effective contribution of full parsing information versus shallow parsing information (i.e., using only the part-of-speech tags, chunks, and clauses). In addition, we also compare performance when using the correct (gold-standard) data versus using automatic parse data. The performance is measured in terms of **precision**, **recall**, and the $\mathbf{F}_1$ measure. Note that all the numbers reported here do not take into account the V arguments as it is quite trivial to predict V and, hence, this gives overoptimistic overall performance if included. When doing the comparison, we also compute the 95% confidence interval of F_1 using the bootstrap resampling method (Noreen 1989), and the difference is considered significant if the compared F_1 lies outside this interval. The automatic full parse trees are derived using Charniak's parser (2001) (version 0.4). In automatic shallow parsing, the information is generated by different state-of-the-art components, including a POS tagger (Even-Zohar and Roth 2001), a chunker (Punyakanok and Roth 2001), and a clauser (Carreras, Màrquez, and Castro 2005).

4.2 Argument Classification

To evaluate the performance gap between full parsing and shallow parsing in argument classification, we assume the argument boundaries are known, and only train classifiers to classify the labels of these arguments. In this stage, the only difference between the uses of full parsing and shallow parsing information is the construction of *phrase type*, *head word*, *POS tag of the head word*, *path*, *subcategorization*, and *syntactic frame* features. As described in Section 3.2.2, most of these features can be approximated using chunks and clauses, with the exception of the syntactic frame feature. It is unclear how this feature can be mimicked because it relies on the internal structure of a full parse tree. Therefore, it does not have a corresponding feature in the shallow parsing case.

Table 2 reports the experimental results of argument classification when argument boundaries are known. In this case, because the argument classifier of our SRL system does not overpredict or miss any arguments, we do not need to train with a *null* class,

Table 2
The accuracy of argument classification when argument boundaries are known.

	Full Parsing	Shallow Parsing
Gold	91.50 ± 0.48	90.75 ± 0.45
Auto	90.32 ± 0.48	89.71 ± 0.50

and we can simply measure the performance using accuracy instead of F_1. The training examples include 90,352 propositions with a total of 332,381 arguments. The test data contain 5,246 propositions and 19,511 arguments. As shown in the table, although the full-parsing features are more helpful than the shallow-parsing features, the performance gap is quite small (0.75% on gold-standard data and 0.61% with the automatic parsers).

The rather small difference in the performance between argument classifiers using full parsing and shallow parsing information almost disappears when their output is processed by the inference stage. Table 3 shows the final results in recall, precision, and F_1, when the argument boundaries are known. In all cases, the differences in F_1 between the full parsing–based and the shallow parsing–based systems are not statistically significant.

Conclusion. When the argument boundaries are known, the performance of the full parsing–based SRL system is about the same as the shallow parsing–based SRL system.

4.3 Argument Identification

Argument identification is an important stage that effectively reduces the number of argument candidates after the pruning stage. Given an argument candidate, an argument identifier is a binary classifier that decides whether or not the candidate should be considered as an argument. To evaluate the influence of full parsing information in this stage, the candidate list used here is the outputs of the pruning heuristic applied on the gold-standard parse trees. The heuristic results in a total number of 323,155 positive and 686,887 negative examples in the training set, and 18,988 positive and 39,585 negative examples in the test set.

Similar to the argument classification stage, the only difference between full parsing– and shallow parsing–based systems is in the construction of some features. Specifically, *phrase type*, *head word*, *POS tag of the head word*, *path*, and *subcategorization* features are approximated using chunks and clauses when the binary classifier is trained using shallow parsing information.

Table 4 reports the performance of the argument identifier on the test set using the direct predictions of the trained binary classifier. The recall and precision of the

Table 3
The overall system performance when argument boundaries are known.

	Full Parsing			Shallow Parsing		
	Prec	Rec	F_1	Prec	Rec	F_1
Gold	91.58	91.90	91.74 ± 0.51	91.14	91.48	91.31 ± 0.51
Auto	90.71	91.14	90.93 ± 0.53	90.50	90.88	90.69 ± 0.53

Table 4
The performance of argument identification after pruning (based on the gold standard full parse trees).

	Full Parsing			Shallow Parsing		
	Prec	Rec	F_1	Prec	Rec	F_1
Gold	96.53	93.57	95.03 ± 0.32	93.66	91.72	92.68 ± 0.38
Auto	94.68	90.60	92.59 ± 0.39	92.31	88.36	90.29 ± 0.43

full parsing–based system are around 2 to 3 percentage points higher than the shallow parsing–based system on the gold-standard data. As a result, the F_1 score is 2.5 percentage points higher. The performance on automatic parse data is unsurprisingly lower but the difference between the full parsing– and the shallow parsing–based systems is as observed previously. In terms of filtering efficiency, around 25% of the examples are predicted as positive. In other words, both argument identifiers filter out around 75% of the argument candidates after pruning.

Because the recall in the argument identification stage sets the upper-bound the recall in argument classification, the threshold that determines when examples are predicted to be positive is usually lowered to allow more positive predictions. That is, a candidate is predicted as positive when its probability estimation is larger than the threshold. Table 5 shows the performance of the argument identifiers when the threshold is 0.1.[2]

Because argument identification is just an intermediate step in a complete system, a more realistic evaluation method is to see how each final system performs. Using an argument identifier with threshold = 0.1 (i.e., Table 5), Table 6 reports the final results in recall, precision, and F_1. The F_1 difference is 1.5 points when using the gold-standard data. However, when automatic parsers are used, the shallow parsing–based system is, in fact, slightly better; although the difference is not statistically significant. This may be due to the fact that chunk and clause predictions are very important here, and shallow parsers are more accurate in chunk or clause predictions than a full parser (Li and Roth 2001).

Conclusion. Full parsing information helps in argument identification. However, when the automatic parsers are used, using the full parsing information may not have better overall results compared to using shallow parsing.

4.4 Pruning

As shown in the previous two sections, the overall performance gaps of full parsing and shallow parsing are small. When automatic parsers are used, the difference is less than 1 point in F_1 or accuracy. Therefore, we conclude that the main contribution of full parsing is in the pruning stage. Because the shallow parsing system does not have enough information for the pruning heuristics, we train two word-based classifiers to replace the pruning stage. One classifier is trained to predict whether a given word is the start (S) of

2 The value was determined by experimenting with the complete system using automatic full parse trees, on the development set. In our tests, lowering the threshold in argument identification always leads to higher overall recall and lower overall precision. As a result, the gain in F_1 is limited.

Table 5
The performance of argument identification after pruning (based on the gold-standard full parse trees) and with threshold = 0.1.

	Full Parsing			Shallow Parsing		
	Prec	Rec	F_1	Prec	Rec	F_1
Gold	92.13	95.62	93.84 ± 0.37	88.54	94.81	91.57 ± 0.42
Auto	89.48	94.14	91.75 ± 0.41	86.14	93.21	89.54 ± 0.47

Table 6
The overall system performance using the output from the pruning heuristics, applied on the gold-standard full parse trees.

	Full Parsing			Shallow Parsing		
	Prec	Rec	F_1	Prec	Rec	F_1
Gold	86.22	87.40	86.81 ± 0.59	84.14	85.31	84.72 ± 0.63
Auto	84.21	85.04	84.63 ± 0.63	86.17	84.02	85.08 ± 0.63

an argument; the other classifier is to predict the end (E) of an argument. If the product of probabilities of a pair of S and E predictions is larger than a predefined threshold, then this pair is considered as an argument candidate. The threshold used here was obtained by using the validation set. Both classifiers use very similar features to those used by the argument identifier as explained in Section 3.2, treating the target word as a constituent. Particularly, the features are predicate, POS tag of the predicate, voice, context words, POS tags of the context words, chunk pattern, clause relative position, and shallow-path. The head word and its POS tag are replaced by the target word and its POS tag. The comparison of using the classifiers and the heuristics is shown in Table 7.

Even without the knowledge of the constituent boundaries, the classifiers seem surprisingly better than the pruning heuristics. Using either the gold-standard data set or the output of automatic parsers, the classifiers achieve higher F_1 scores. One possible reason for this phenomenon is that the accuracy of the pruning strategy is limited by the number of agreements between the correct arguments and the constituents of the parse trees. Table 8 summarizes the statistics of the examples seen by both strategies. The pruning strategy needs to decide which are the potential arguments among all constituents. This strategy is upper-bounded by the number of correct arguments that agree with some constituent. On the other hand, the classifiers do not have this limitation. The number of examples they observe is the total number of words to be processed, and the positive examples are those arguments that are annotated as such in the data set.

Table 7
The performance of pruning using heuristics and classifiers.

	Full Parsing			Classifier Threshold = 0.04		
	Prec	Rec	F_1	Prec	Rec	F_1
Gold	25.94	97.27	40.96 ± 0.51	29.58	97.18	45.35 ± 0.83
Auto	22.79	86.08	36.04 ± 0.52	24.68	94.80	39.17 ± 0.79

Table 8
Statistics of the training and test examples for the pruning stage.

	Words	Arguments	Constituents		Agreements	
			Gold	Auto	Gold	Auto
Train	2,575,665	332,381	4,664,954	4,263,831	327,603	319,768
Test	147,981	19,511	268,678	268,482	19,266	18,301

The *Agreements* column shows the number of arguments that match the boundaries of some constituents.

Note that because each verb is processed independently, a sentence is processed once for each verb in the sentence. Therefore, the words and constituents in each sentence are counted as many times as the number of verbs to be processed.

As before, in order to compare the systems that use full parsing and shallow parsing information, we need to see the impact on the overall performance. Therefore, we built two semantic role systems based on full parsing and shallow parsing information. The full parsing–based system follows the pruning, argument identification, argument classification, and inference stages, as described earlier. For the shallow parsing system, the pruning heuristic is replaced by the word-based pruning classifiers, and the remaining stages are designed to use only shallow parsing as described in previous sections. Table 9 shows the overall performance of the two evaluation systems.

As indicated in the tables, the gap in F_1 between full parsing and shallow parsing–based systems enlarges to more than 11 points on the gold-standard data. At first glance, this result seems to contradict our conclusion in Section 4.3. After all, if the pruning stage of shallow parsing SRL system performs equally well or even better, the overall performance gap in F_1 should be small.

After we carefully examined the output of the word-based classifier, we realized that it filters out easy candidates, and leaves examples that are difficult to the later stages. Specifically, these argument candidates often overlap and differ only in one or two words. On the other hand, the pruning heuristic based on full parsing never outputs overlapping candidates and consequently provides input that is easier for the next stage to handle. Indeed, the following argument identification stage turns out to be good in discriminating these non-overlapping candidates.

Conclusion. The most crucial contribution of full parsing is in the pruning stage. The internal tree structure significantly helps in discriminating argument candidates, which makes the work done by the following stages easier.

Table 9
The overall system performance.

	Full Parsing			Shallow Parsing		
	Prec	Rec	F_1	Prec	Rec	F_1
Gold	86.22	87.40	86.81 ± 0.59	75.34	75.28	75.31 ± 0.76
Auto	77.09	75.51	76.29 ± 0.76	75.48	67.13	71.06 ± 0.80

5. The Effect of Inference

Our inference procedure plays an important role in improving accuracy when the local predictions violate the constraints among argument labels. In this section, we first present the overall system performance when most constraints are not used. We then demonstrate how the inference procedure can be used to combine the output of several systems to yield better performance.

5.1 Inference with Limited Constraints

The inference stage in our system architecture provides a principled way to resolve conflicting local predictions. It is interesting to see whether this procedure improves the performance differently for the full parsing– vs. the shallow parsing–based system, as well as gold-standard vs. automatic parsing input.

Table 10 shows the results of using only constraints 1, 2, 3, and 4. As mentioned previously, the first three constraints are handled before the argument classification stage. Constraint 4, which forbids overlapping or embedding arguments, is required in order to use the official CoNLL-2005 evaluation script and is therefore kept.

By comparing Table 9 with Table 10, we can see that the effect of adding more constraints is quite consistent over the four settings. Precision is improved by 1 to 2 percentage points but recall is decreased a little. As a result, the gain in F_1 is about 0.5 to 1 point. It is not surprising to see this lower recall and higher precision phenomenon after the constraints described in Section 3.4.1 are examined. Most constraints *punish* false non-*null* output, but do not regulate false *null* predictions. For example, an assignment that has two A1 arguments clearly violates the non-duplication constraint. However, if an assignment has no predicted arguments at all, it still satisfies all the constraints.

5.2 Joint Inference

The empirical study in Section 4 indicates that the performance of an SRL system primarily depends on the very first stage—pruning, which is directly derived from the full parse trees. This also means that in practice the quality of the syntactic parser is decisive to the quality of the SRL system. To improve semantic role labeling, one possible way is to combine different SRL systems through a joint inference stage, given that the systems are derived using different full parse trees.

To test this idea, we first build two SRL systems that use Collins's parser (Collins 1999)[3] and Charniak's parser (Charniak 2001), respectively. In fact, these two parsers have noticeably different outputs. Applying the pruning heuristics on the output of Collins's parser produces a list of candidates with 81.05% recall. Although this number is significantly lower than the 86.08% recall produced by Charniak's parser, the union of the two candidate lists still significantly improves recall to 91.37%. We construct the two systems by implementing the first three stages, namely, pruning, argument identification, and argument classification. When a test sentence is given, a joint inference stage is used to resolve the inconsistency of the output of argument classification in these two systems.

We first briefly review the objective function used in the inference procedure introduced in Section 3.4. Formally speaking, the argument classifiers attempt to assign

3 We use the Collins parser implemented by Bikel (2004).

Table 10
The impact of removing most constraints in overall system performance.

	Full Parsing			Shallow Parsing		
	Prec	Rec	F_1	Prec	Rec	F_1
Gold	85.07	87.50	86.27 ± 0.58	73.19	75.63	74.39 ± 0.75
Auto	75.88	75.81	75.84 ± 0.75	73.56	67.45	70.37 ± 0.80

labels to a set of arguments, $S^{1:M}$, indexed from 1 to M. Each argument S^i can take any label from a set of argument labels, $\mathcal{P}$, and the indexed set of arguments can take a set of labels, $c^{1:M} \in \mathcal{P}^M$. If we assume that the argument classifier returns an estimated conditional probability distribution, $Prob(S^i = c^i)$, then, given a sentence, the inference procedure seeks a global assignment that maximizes the objective function denoted by Equation (2), which can be rewritten as follows,

$$\hat{c}^{1:M} = \underset{c^{1:M} \in \mathcal{F}(\mathcal{P}^M)}{\operatorname{argmax}} \sum_{i=1}^{M} Prob(S^i = c^i) \tag{3}$$

where the linguistic and structural constraints are represented by the filter $\mathcal{F}$. In other words, this objective function reflects the expected number of correct argument predictions, subject to the constraints.

When there are two or more argument classifiers from different SRL systems, a joint inference procedure can take the output estimated probabilities for all these candidates as input, although some candidates may refer to the same phrases in the sentence. For example, Figure 3 shows the two candidate sets for a fragment of a sentence, *..., traders say, unable to* ***cool*** *the selling panic in both stocks and futures*. In this example, system A has two argument candidates, a_1 = *traders* and a_4 = *the selling panic in both stocks and futures*; system B has three argument candidates, b_1 = *traders*, b_2 = *the selling panic*, and b_3 = *in both stocks and futures*.

A straightforward solution to the combination is to treat each argument produced by a system as a possible output. Each possible labeling of the argument is associated with a variable which is then used to set up the inference procedure. However, the final prediction will be likely dominated by the system that produces more candidates, which is system B in this example. The reason is that our objective function is the sum of the probabilities of all the candidate assignments.

This bias can be corrected by the following observation. Although system A only has two candidates, a_1 and a_4, it can be treated as if it also has two additional **phantom** candidates, a_2 and a_3, where a_2 and b_2 refer to the same phrase, and so do a_3 and b_3. Similarly, system B has a phantom candidate b_4 that corresponds to a_4. Because system A does not really generate a_2 and a_3, we can assume that these two phantom candidates are predicted by it as "null" (i.e., not an argument). We assign the same prior distribution to each phantom candidate. In particular, the probability of the "null" class is set to be 0.55 based on empirical tests, and the probabilities of the remaining classes are set based on their occurrence frequencies in the training data.

Then, we treat each possible final argument output as a single unit. Each probability estimation by a system can be viewed as evidence in the final probability estimation and, therefore, we can simply average their estimation. Formally, let $\mathcal{S}_i$ be the argument set

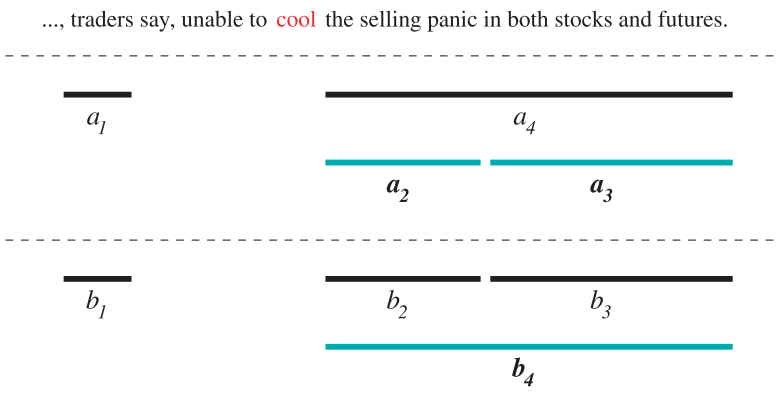

Figure 3
The output of two SRL systems: system A has two candidates, a_1 = *traders* and a_4 = *the selling panic in both stocks and futures*; system B has three argument candidates, b_1 = *traders*, b_2 = *the selling panic*, and b_3 = *in both stocks and futures*. In addition, we create two phantom candidates a_2 and a_3 for system A that correspond to b_2 and b_3 respectively, and b_4 for system B that corresponds to a_4.

output by system i, and $\mathcal{S} = \bigcup_{i=1}^{k} \mathcal{S}_i$ be the set of all arguments where k is the number of systems; let N be the cardinality of $\mathcal{S}$. Our augmented objective function is then:

$$\hat{c}^{1:N} = \underset{c^{1:N} \in \mathcal{F}(\mathcal{P}^N)}{\operatorname{argmax}} \sum_{i=1}^{N} Prob(S^i = c^i) \tag{4}$$

where $S^i \in \mathcal{S}$, and

$$Prob(S^i = c^i) = \frac{1}{k} \sum_{j=1}^{k} Prob_j(S^i = c^i) \tag{5}$$

where $Prob_j$ is the probability output by system j.

Note that we may also treat the individual systems differently by applying different priors (i.e., weights) on the estimated probabilities of the argument candidates. For example, if the performance of system A is much better than system B, then we may want to *trust* system A's output more by multiplying the output probabilities by a larger weight.

Table 11 reports the performance of two individual systems based on Collins's parser and Charniak's parser, as well as the joint system, where the two individual systems are equally weighted. The joint system based on this straightforward strategy significantly improves the performance compared to the two original SRL systems in both recall and precision, and thus achieves a much higher F_1.

6. Empirical Evaluation—CoNLL Shared Task 2005

In this section, we present the detailed evaluation of our SRL system, in the competition on semantic role labeling—the CoNLL-2005 shared task (Carreras and Màrquez

Table 11
The performance of individual and combined SRL systems.

	Prec	Rec	F_1
Collins' parser	75.92	71.45	73.62 ± 0.79
Charniak's parser	77.09	75.51	76.29 ± 0.76
Combined result	80.53	76.94	78.69 ± 0.71

2005). The setting of this shared task is basically the same as it was in 2004, with some extensions. First, it allows much richer syntactic information. In particular, full parse trees generated using Collins's parser (Collins 1999) and Charniak's parser (Charniak 2001) were provided. Second, the *full parsing standard partition* was used—the training set was enlarged and covered Sections 02–21, the development set was Section 24, and the test set was Section 23. Finally, in addition to the *Wall Street Journal* (WSJ) data, three sections of the Brown corpus were used to provide cross-corpora evaluation.

The system we used to participate in the CoNLL-2005 shared task is an enhanced version of the system described in Sections 3 and 5. The main difference was that the joint-inference stage was extended to combine six basic SRL systems instead of two. Specifically for this implementation, we first trained two SRL systems that use Collins's parser and Charniak's parser, respectively, because of their noticeably different outputs. In evaluation, we ran the system that was trained with Charniak's parser five times, with the top-5 parse trees output by Charniak's parser. Together we have six different outputs per predicate. For each parse tree output, we ran the first three stages, namely, pruning, argument identification, and argument classification. Then, a joint-inference stage, where each individual system is weighted equally, was used to resolve the inconsistency of the output of argument classification in these systems.

Table 12 shows the overall results on the development set and different test sets; the detailed results on WSJ section 23 are shown in Table 13. Table 14 shows the results of individual systems and the improvement gained by the joint inference procedure on the development set.

Our system reached the highest F_1 scores on all the test sets and was the best system among the 19 participating teams. After the competition, we improved the system slightly by tuning the weights of the individual systems in the joint inference procedure, where the F_1 scores on WSJ test section and the Brown test set are 79.59 points and 67.98 points, respectively.

Table 12
Overall CoNLL-2005 shared task results.

	Prec.	Rec.	F_1
Development	80.05	74.83	77.35
Test WSJ	**82.28**	**76.78**	**79.44**
Test Brown	73.38	62.93	67.75
Test WSJ+Brown	81.18	74.92	77.92

Table 13
Detailed CoNLL-2005 shared task results on the WSJ test set.

Test WSJ	Prec.	Rec.	F_1
Overall	82.28	76.78	79.44
A0	88.22	87.88	88.05
A1	82.25	77.69	79.91
A2	78.27	60.36	68.16
A3	82.73	52.60	64.31
A4	83.91	71.57	77.25
AM-ADV	63.82	56.13	59.73
AM-CAU	64.15	46.58	53.97
AM-DIR	57.89	38.82	46.48
AM-DIS	75.44	80.62	77.95
AM-EXT	68.18	46.88	55.56
AM-LOC	66.67	55.10	60.33
AM-MNR	66.79	53.20	59.22
AM-MOD	96.11	98.73	97.40
AM-NEG	97.40	97.83	97.61
AM-PNC	60.00	36.52	45.41
AM-TMP	78.16	76.72	77.44
R-A0	89.72	85.71	87.67
R-A1	70.00	76.28	73.01
R-A2	85.71	37.50	52.17
R-AM-LOC	85.71	57.14	68.57
R-AM-TMP	72.34	65.38	68.69

In terms of the computation time, for both the argument identifier and the argument classifier, the training of each model, excluding feature extraction, takes 50–70 minutes using less than 1GB memory on a 2.6GHz AMD machine. On the same machine, the average test time for each stage, excluding feature extraction, is around 2 minutes.

7. Related Work

The pioneering work on building an automatic semantic role labeler was proposed by Gildea and Jurafsky (2002). In their setting, semantic role labeling was treated as a tagging problem on each constituent in a parse tree, solved by a two-stage architecture consisting of an argument identifier and an argument classifier. This is similar to our

Table 14
The results of individual systems and the result with joint inference on the development set.

	Prec.	Rec.	F_1
Charniak-1	75.40	74.13	74.76
Charniak-2	74.21	73.06	73.63
Charniak-3	73.52	72.31	72.91
Charniak-4	74.29	72.92	73.60
Charniak-5	72.57	71.40	71.98
Collins	73.89	70.11	71.95
Joint inference	80.05	74.83	77.35

main architecture with the exclusion of the pruning and inference stages. There are two additional key differences between their system and ours. First, their system used a back-off probabilistic model as its main engine. Second, it was trained on FrameNet (Baker, Fillmore, and Lowe 1998)—another large corpus, besides PropBank, that contains selected examples of semantically labeled sentences.

Later that year, the same approach was applied on PropBank by Gildea and Palmer (2002). Their system achieved 57.7% precision and 50.0% recall with automatic parse trees, and 71.1% precision and 64.4% recall with gold-standard parse trees. It is worth noticing that at that time the PropBank project was not finished and the data set available was only a fraction in size of what it is today. Since these pioneering works, the task has gained increasing popularity and created a new line of research. The two-step constituent-by-constituent architecture became a common blueprint for many systems that followed.

Partly due to the expansion of the PropBank dataset, researchers have gradually made improvement on the performance of automatic SRL systems by using new techniques and new features. Some of the early systems are described in Chen and Rambow (2003), Gildea and Hockenmaier (2003), and Surdeanu et al. (2003). All are based on a two-stage architecture similar to the one proposed by Gildea and Palmer (2002) with the differences in the machine-learning techniques and the features used. The first breakthrough in terms of performance was due to Pradhan et al. (2003), who first viewed the task as a massive classification problem and applied multiple SVMs to it. Their final result (after a few more improvements) reported in Pradhan et al. (2004) achieved 84% and 75% in precision and recall, respectively.

A second significant contribution beyond the two-stage architecture is due to Xue and Palmer (2004), who introduced the pruning heuristics to the two-stage architecture, and remarkably reduced the number of candidate arguments a system needs to consider; this approach was adopted by many systems. Another significant advancement was in the realization that global information can be exploited and benefits the results significantly. Inference based on an integer linear programming technique, which was originally introduced by Roth and Yih (2004) on a relation extraction problem, was first applied to the SRL problem by Punyakanok et al. (2004). It showed that domain knowledge can be easily encoded and contributes significantly through inference over the output of classifiers. The idea of exploiting global information, which is detailed in this paper, was pursued later by other researchers, in different forms.

Besides the constituent-by-constituent based architecture, others have also been explored. The alternative frameworks include representing semantic role labeling as a sequence-tagging problem (Màrquez, Pere Comas, and Català 2005) and tagging the edges in the corresponding dependency trees (Hacioglu 2004). However, the most popular architecture by far is the constituent-by-constituent based multi-stage architecture, perhaps due to its conceptual simplicity and its success. In the CoNLL-2005 shared task competition (Carreras and Màrquez 2005), the majority of the systems followed the constituent-by-constituent based two-stage architecture, and the use of the pruning heuristics was also fairly common.

The CoNLL-2005 shared task also highlighted the importance of system combination, such as our ILP technique when used in joint inference, in order to achieve superior performance. The top four systems, which produced significantly better results than the rest, all used some schemes to combine the output of several SRL systems, ranging from using a fixed combination function (Haghighi, Toutanova, and Manning 2005; Koomen et al. 2005) to using a machine-learned combination strategy (Màrquez, Pere Comas, and Català 2005; Pradhan, Hacioglu, Ward et al. 2005).

The work of Gildea and Palmer (2002) pioneered not only the fundamental architecture of SRL, but was also the first to investigate the interesting question regarding the significance of using full parsing for high quality SRL. They compared their full system with another system that only used chunking, and found that the chunk-based system performed much worse. The precision and recall dropped from 57.7% and 50.0% to 27.6% and 22.0%, respectively. That led to the conclusion that full parsing information is necessary to solving the SRL problem, especially at the stage of argument identification—a finding that is quite similar to ours in this article. However, their chunk-based approach was very weak—only chunks were considered as possible candidates; hence, it is not very surprising that the boundaries of the arguments could not be reliably found. In contrast, our shallow parse–based system does not have these restrictions on the argument boundaries and therefore performs much better at this stage, providing a more fair comparison.

A related comparison can be found also in the work by Pradhan, Hacioglu, Krugler et al. (2005) (their earlier version appeared in Pradhan et al. [2003]), which reported the performance on several systems using different information sources and system architectures. Their shallow parse–based system is modeled as a sequence tagging problem while the full system is a constituent-by-constituent based two-stage system. Due to technical difficulties, though, they reported the results of the chunk-based systems only on a subset of the full data set. Their shallow parse–based system achieved 60.4% precision and 51.4% recall and their full system achieved 80.6% precision and 67.1% recall on the same data set (but 84% precision and 75% recall with the full data set). Therefore, due to the use of different architectures and data set sizes, the questions of "how much one can gain from full parsing over shallow parsing when using the full PropBank data set" and "what are the sources of the performance gain" were left open.

Similarly, in the CoNLL-2004 shared task (Carreras and Màrquez 2004), participants were asked to develop SRL systems with the restriction that only shallow parsing information (i.e., chunks and clauses) were allowed. The performance of the best system was at 72.43% precision and 66.77% recall, which was about 10 points in F_1 lower than the best system based on full parsing in the literature. However, the training examples were derived from only 5 sections and not all the 19 sections usually used in the standard setting. Hence, the question was not yet fully answered.

Our experimental study, on the other hand, is done with a consistent architecture, by considering each stage in a controlled manner, and using the full data set, allowing one to draw direct conclusions regarding the impact of this information source.

8. Conclusion

This paper studies the important task of semantic role labeling. We presented an approach to SRL and a principled and general approach to incorporating global information in natural language decisions. Beyond presenting this approach which leads to a state-of-the-art SRL system, we focused on investigating the significance of using full parse tree information as input to an SRL system adhering to the most common system architecture, and the stages in the process where this information has the most impact. We performed a detailed and fair experimental comparison between shallow and full parsing information and concluded that, indeed, full syntactic information can improve the performance of an SRL system. In particular, we have shown that this information is most crucial in the pruning stage of the system, and relatively less important in the following stages.

In addition, we showed the importance of global inference to good performance in this task, characterized by rich structural and linguistic constraints among the predicted labels of the arguments. Our integer linear programming–based inference procedure is a powerful and flexible optimization strategy that finds the best solution subject to these constraints. As we have shown, it can be used to resolve conflicting argument predictions in an individual system but can also serve as an effective and simple approach to combining different SRL systems, resulting in a significant improvement in performance.

In the future, we plan to extend our work in several directions. By adding more constraints to the inference procedure, an SRL system may be further improved. Currently, the constraints are provided by human experts in advance. Learning both hard and statistical constraints from the data will be our top priority. Some work on combining statistical and declarative constraints has already started and is reported in Roth and Yih (2005). Another issue we want to address is domain adaptation. It has been clearly shown in the CoNLL-2005 shared task that the performance of current SRL systems degrades significantly when tested on a corpus different from the one used in training. This may be due to the underlying components, especially the syntactic parsers which are very sensitive to changes in data genre. Developing a better model that more robustly combines these components could be a promising direction. In addition, although the shallow parsing–based system was shown here to be inferior, shallow parsers were shown to be more robust than full parsers (Li and Roth 2001). Therefore, combining these two systems may bring forward both of their advantages.

Appendix A: An ILP Formulation for SRL

In this section, we show a complete example of the ILP formulation formulated to solve the inference problem as described in Section 3.4.

Example. Assume the sentence is four words long with the following argument candidates, and the following illegal argument types for the predicate of interest.

Sentence: w_1 w_2 w_3 w_4
Candidates: [S^1] [S^2] [S^3] [S^5]
[S^4]
Illegal argument types: A3, A4, A5

Indicator Variables and Their Costs. The followings are the indicator variables and their associated costs set up for the example.

Indicator Variables:

$u_{1\text{A0}}, u_{1\text{A1}}, \ldots, u_{1\text{AM-LOC}}, \ldots, u_{1\text{C-A0}}, \ldots, u_{1\text{R-A0}}, \ldots, u_{1\phi}$
$u_{2\text{A0}}, u_{2\text{A1}}, \ldots, u_{2\text{AM-LOC}}, \ldots, u_{2\text{C-A0}}, \ldots, u_{2\text{R-A0}}, \ldots, u_{2\phi}$
$\vdots$
$u_{5\text{A0}}, u_{5\text{A1}}, \ldots, u_{5\text{AM-LOC}}, \ldots, u_{5\text{C-A0}}, \ldots, u_{5\text{R-A0}}, \ldots, u_{5\phi}$

Costs:

$p_{1\text{A0}}, p_{1\text{A1}}, \ldots, p_{1\text{AM-LOC}}, \ldots, p_{1\text{C-A0}}, \ldots, p_{1\text{R-A0}}, \ldots, p_{1\phi}$
$p_{2\text{A0}}, p_{2\text{A1}}, \ldots, p_{2\text{AM-LOC}}, \ldots, p_{2\text{C-A0}}, \ldots, p_{2\text{R-A0}}, \ldots, p_{2\phi}$
$\vdots$
$p_{5\text{A0}}, p_{5\text{A1}}, \ldots, p_{5\text{AM-LOC}}, \ldots, p_{5\text{C-A0}}, \ldots, p_{5\text{R-A0}}, \ldots, p_{5\phi}$

Objective Function. The objective function can be written as the following.

$$\text{argmax}_{u_{ic} \in \{0,1\}: \forall i \in [1,5], c \in \mathcal{P}} \sum_{i=1}^{5} \sum_{c \in \mathcal{P}} p_{ic} u_{ic}$$

where

$$\mathcal{P} = \{\text{A0}, \text{A1}, \ldots, \text{AM-LOC}, \ldots, \text{C-A0}, \ldots, \text{R-A0}, \ldots, \phi\}$$

subject to

$$\begin{aligned} u_{1\text{A0}} + u_{1\text{A1}} + \ldots + u_{1\text{AM-LOC}} + \ldots + u_{1\text{C-A0}} + \ldots + u_{1\text{R-A0}} + \ldots + u_{1\phi} &= 1 \\ u_{2\text{A0}} + u_{2\text{A1}} + \ldots + u_{2\text{AM-LOC}} + \ldots + u_{2\text{C-A0}} + \ldots + u_{2\text{R-A0}} + \ldots + u_{2\phi} &= 1 \\ &\vdots \\ u_{5\text{A0}} + u_{5\text{A1}} + \ldots + u_{5\text{AM-LOC}} + \ldots + u_{2\text{C-A0}} + \ldots + u_{5\text{R-A0}} + \ldots + u_{5\phi} &= 1 \end{aligned}$$

Additional Constraints. The rest of the constraints can be formulated as the following.

Constraint 4: No overlapping or embedding

$$\begin{aligned} u_{3\phi} + u_{4\phi} &\geq 1 \\ u_{4\phi} + u_{5\phi} &\geq 1 \end{aligned}$$

Constraint 5: No duplicate argument classes

$$\begin{aligned} u_{1\text{A0}} + u_{2\text{A0}} + \ldots + u_{5\text{A0}} &\leq 1 \\ u_{1\text{A1}} + u_{2\text{A1}} + \ldots + u_{5\text{A1}} &\leq 1 \\ u_{1\text{A2}} + u_{2\text{A2}} + \ldots + u_{5\text{A2}} &\leq 1 \end{aligned}$$

Constraint 6: R-*arg* arguments

$$\begin{aligned} u_{1\text{A0}} + u_{2\text{A0}} + \ldots + u_{5\text{A0}} &\geq u_{1\text{R-A0}} \\ u_{1\text{A0}} + u_{2\text{A0}} + \ldots + u_{5\text{A0}} &\geq u_{2\text{R-A0}} \\ &\vdots \\ u_{1\text{A0}} + u_{2\text{A0}} + \ldots + u_{5\text{A0}} &\geq u_{5\text{R-A0}} \\ \\ u_{1\text{A1}} + u_{2\text{A1}} + \ldots + u_{5\text{A1}} &\geq u_{1\text{R-A1}} \\ &\vdots \\ \\ u_{1\text{AM-LOC}} + u_{2\text{AM-LOC}} + \ldots + u_{5\text{AM-LOC}} &\geq u_{1\text{R-AM-LOC}} \\ &\vdots \end{aligned}$$

Constraint 7: C-*arg* arguments

$$\begin{aligned} u_{1\text{A0}} &\geq u_{2\text{C-A0}} \\ u_{1\text{A0}} + u_{2\text{A0}} &\geq u_{3\text{C-A0}} \\ &\vdots \\ u_{1\text{A0}} + u_{2\text{A0}} + \ldots + u_{4\text{A0}} &\geq u_{5\text{C-A0}} \\ \\ u_{1\text{A1}} &\geq u_{2\text{C-A1}} \\ &\vdots \\ \\ u_{1\text{AM-LOC}} &\geq u_{2\text{C-AM-LOC}} \\ &\vdots \end{aligned}$$

Constraint 8: Illegal argument types

$$\begin{aligned} u_{1A3} + u_{2A3} + \ldots + u_{5A3} &= 0 \\ u_{1A4} + u_{2A4} + \ldots + u_{5A4} &= 0 \\ u_{1A5} + u_{2A5} + \ldots + u_{5A5} &= 0 \end{aligned}$$

Acknowledgments

We thank Xavier Carreras and Lluís Màrquez for the data and scripts, Szu-ting Yi for her help in improving our joint inference procedure, and Nick Rizzolo as well as the anonymous reviewers for their comments and suggestions. We are also grateful to Dash Optimization for the free academic use of Xpress-MP and AMD for their equipment donation. This research is supported by the Advanced Research and Development Activity (ARDA)'s Advanced Question Answering for Intelligence (AQUAINT) Program, a DOI grant under the Reflex program, NSF grants ITR-IIS-0085836, ITR-IIS-0085980, and IIS-9984168, EIA-0224453, and an ONR MURI Award.

References

Baker, Collin F., Charles J. Fillmore, and John B. Lowe. 1998. The Berkeley Framenet project. In *Proceedings of COLING-ACL*, pages 86–90, Montreal, Canada.

Bikel, Daniel M. 2004. Intricacies of Collins' parsing model. *Computational Linguistics*, 30(4):479–511.

Bishop, Christopher M., 1995. *Neural Networks for Pattern Recognition*, chapter 6.4: Modelling conditional distributions, page 215. Oxford University Press, Oxford, UK.

Carlson, Andrew J., Chad M. Cumby, Jeff L. Rosen, and Dan Roth. 1999. The SNoW learning architecture. Technical Report UIUCDCS-R-99-2101, UIUC Computer Science Department.

Carreras, Xavier and Llúis Màrquez. 2004. Introduction to the CoNLL-2004 shared tasks: Semantic role labeling. In *Proceedings of CoNLL-2004*, pages 89–97, Boston, MA.

Carreras, Xavier and Llúis Màrquez. 2005. Introduction to the CoNLL-2005 shared task: Semantic role labeling. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 152–164, Ann Arbor, MI.

Carreras, Xavier, Llúis Màrquez, and Jorge Castro. 2005. Filtering–ranking perceptron learning for partial parsing. *Machine Learning*, 60:41–71.

Carreras, Xavier, Llúis Màrquez, Vasin Punyakanok, and Dan Roth. 2002. Learning and inference for clause identification. In *Proceedings of the 13th European Conference on Machine Learning (ECML-2002)*, pages 35–47, Helsinki, Finland.

Charniak, Eugene. 2001. Immediate-head parsing for language models. In *Proceedings of the 39th Annual Meeting of the Association of Computational Linguistics (ACL-2001)*, pages 116–123, Toulouse, France.

Chen, John and Owen Rambow. 2003. Use of deep linguistic features for the recognition and labeling of semantic arguments. In *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing (EMNLP-2003)*, pages 41–48, Sapporo, Japan.

Collins, Michael. 1999. *Head-driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, Computer Science Department, University of Pennsylvania, Philadelphia, PA.

Dagan, Ido, Yael Karov, and Dan Roth. 1997. Mistake-driven learning in text categorization. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing (EMNLP-1997)*, pages 55–63, Providence, RI.

Even-Zohar, Yair and Dan Roth. 2001. A sequential model for multi-class classification. In *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing (EMNLP-2001)*, pages 10–19, Pittsburgh, PA.

Freund, Yoav and Robert E. Schapire. 1999. Large margin classification using the Perceptron algorithm. *Machine Learning*, 37(3):277–296.

Gildea, Daniel and Julia Hockenmaier. 2003. Identifying semantic roles using combinatory categorial grammar. In *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing (EMNLP-2003)*, pages 57–64, Sapporo, Japan.

Gildea, Daniel and Daniel Jurafsky. 2002. Automatic labeling of semantic roles. *Computational Linguistics*, 28(3):245–288.

Gildea, Daniel and Martha Palmer. 2002. The necessity of parsing for predicate argument recognition. In *Proceedings of the 40th Annual Meeting of the Association of Computational Linguistics (ACL-2002)*, pages 239–246, Philadelphia, PA.

Golding, Andrew R. and Dan Roth. 1999. A Winnow based approach to context-sensitive spelling correction. *Machine Learning*, 34(1-3):107–130.

Grove, Adam J. and Dan Roth. 2001. Linear concepts and hidden variables. *Machine Learning*, 42(1–2):123–141.

Guéret, Christelle, Christian Prins, and Marc Sevaux. 2002. *Applications of Optimization with Xpress-MP*. Dash Optimization. Translated and revised by Susanne Heipcke. http://www.dashoptimization.com/home/downloads/book/booka4.pdf.

Hacioglu, Kadri. 2004. Semantic role labeling using dependency trees. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING)*, Geneva, Switzerland.

Hacioglu, Kadri, Sameer Pradhan, Wayne Ward, James H. Martin, and Daniel Jurafsky. 2004. Semantic role labeling by tagging syntactic chunks. In *Proceedings of CoNLL-2004*, pages 110–113, Boston, MA.

Haghighi, Aria, Kristina Toutanova, and Christopher D. Manning. 2005. A joint model for semantic role labeling. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 173–176, Ann Arbor, MI.

Kingsbury, Paul and Martha Palmer. 2002. From Treebank to PropBank. In *Proceedings of LREC-2002*, Las Palmas, Canary Islands, Spain.

Kipper, Karin, Martha Palmer, and Owen Rambow. 2002. Extending PropBank with VerbNet semantic predicates. In *Proceedings of Workshop on Applied Interlinguas*, Tiburon, CA.

Koomen, Peter, Vasin Punyakanok, Dan Roth, and Wen-tau Yih. 2005. Generalized inference with multiple semantic role labeling systems. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 181–184, Ann Arbor, MI.

Levin, Beth. 1993. *English Verb Classes and Alternations: A Preliminary Investigation*. University of Chicago Press, Chicago.

Levin, Beth and Malka R. Hovav. 1996. From lexical semantics to argument realization. Unpublished manuscript.

Li, Xin and Dan Roth. 2001. Exploring evidence for shallow parsing. In *Proceedings of CoNLL-2001*, pages 107–110, Toulouse, France.

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

Màrquez, Llúis, Jesus Giménez Pere Comas, and Neus Català. 2005. Semantic role labeling as sequential tagging. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 193–196, Ann Arbor, MI.

Noreen, Eric W. 1989. *Computer-Intensive Methods for Testing Hypotheses*. New York: John Wiley & Sons.

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

Pradhan, Sameer, Kadri Hacioglu, Valerie Krugler, Wayne Ward, James H. Martin, and Daniel Jurafsky. 2005. Support vector learning for semantic argument classification. *Machine Learning*, 60:11–39.

Pradhan, Sameer, Kadri Hacioglu, Wayne Ward, James H. Martin, and Daniel Jurafsky. 2003. Semantic role parsing adding semantic structure to unstructured text. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM 2003)*, pages 629–632, Melbourne, FL.

Pradhan, Sameer, Kadri Hacioglu, Wayne Ward, James H. Martin, and Daniel Jurafsky. 2005. Semantic role chunking combining complementary syntactic views. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 217–220, Ann Arbor, MI.

Pradhan, Sameer, Wayne Ward, Kadri Hacioglu, James H. Martin, and Daniel Jurafsky. 2004. Shallow semantic parsing using support vector machines. In *Proceedings of NAACL-HLT 2004*, pages 233–240, Boston, MA.

Punyakanok, Vasin, Dan Roth, Wen-tau Yih, and Dav Zimak. 2004. Semantic role labeling via integer linear programming inference. In *Proceedings the 20th International Conference on Computational Linguistics (COLING)*, pages 1346–1352, Geneva, Switzerland.

Punyakanok, Vasin and Dan Roth. 2001. The use of classifiers in sequential inference. In

Todd K. Leen, Thomas G. Dietterich, and Volker Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 995–1001. MIT Press.

Roth, Dan. 1998. Learning to resolve natural language ambiguities: A unified approach. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pages 806–813, Madison, WI.

Roth, Dan and Wen-tau Yih. 2004. A linear programming formulation for global inference in natural language tasks. In *Proceedings of CoNLL-2004*, pages 1–8, Boston, MA.

Roth, Dan and Wen-tau Yih. 2005. Integer linear programming inference for conditional random fields. In *Proceedings of the 22nd International Conference on Machine Learning (ICML-2005)*, pages 737–744, Bonn, Germany.

Surdeanu, Mihai, Sanda Harabagiu, John Williams, and Paul Aarseth. 2003. Using predicate-argument structures for information extraction. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, pages 8–15, Sapporo, Japan.

Tjong Kim Sang, Erik F. and Sabine Buchholz. 2000. Introduction to the CoNLL-2000 shared task: Chunking. In *Proceedings of CoNLL-2000 and LLL-2000*, pages 127–132, Lisbon, Portugal.

Xpress-MP. 2004. Dash Optimization. Xpress-MP. http://www.dashoptimization.com.

Xue, Nianwen and Martha Palmer. 2004. Calibrating features for semantic role labeling. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing (EMNLP-2004)*, pages 88–94, Barcelona, Spain.

Zhang, Tong, Fred Damerau, and David Johnson. 2002. Text chunking based on a generalization of Winnow. *Journal of Machine Learning Research*, 2:615–637.