Tree Kernels for Semantic Role Labeling

Alessandro Moschitti*
University of Trento

Daniele Pighin**
University of Trento

Roberto Basili†
University of Rome "Tor Vergata"

The availability of large scale data sets of manually annotated predicate–argument structures has recently favored the use of machine learning approaches to the design of automated semantic role labeling (SRL) systems. The main research in this area relates to the design choices for feature representation and for effective decompositions of the task in different learning models. Regarding the former choice, structural properties of full syntactic parses are largely employed as they represent ways to encode different principles suggested by the linking theory between syntax and semantics. The latter choice relates to several learning schemes over global views of the parses. For example, re-ranking stages operating over alternative predicate–argument sequences of the same sentence have shown to be very effective.

In this article, we propose several kernel functions to model parse tree properties in kernel-based machines, for example, perceptrons or support vector machines. In particular, we define different kinds of tree kernels as general approaches to feature engineering in SRL. Moreover, we extensively experiment with such kernels to investigate their contribution to individual stages of an SRL architecture both in isolation and in combination with other traditional manually coded features. The results for boundary recognition, classification, and re-ranking stages provide systematic evidence about the significant impact of tree kernels on the overall accuracy, especially when the amount of training data is small. As a conclusive result, tree kernels allow for a general and easily portable feature engineering method which is applicable to a large family of natural language processing tasks.

1. Introduction

Much attention has recently been devoted to the design of systems for the automatic labeling of semantic roles (SRL) as defined in two important projects: FrameNet (Baker, Fillmore, and Lowe 1998), based on frame semantics, and PropBank (Palmer, Gildea,

* Department of Information Engineering and Computer Science, Via Sommarive, 14 I-38050 Povo (TN). E-mail: moschitti@dit.unitn.it.
** Fondazione Bruno Kessler, Center for Scientific and Technological Research, Department of Information Engineering and Computer Science, Via Sommarive, 18 I-38050 Povo (TN). E-mail: pighin@itc.it.
† Department of Computer Science, Systems and Production, Via del Politecnico, 1 I-00133 RM. E-mail: basili@info.uniroma2.it.

Submission received: 15 July 2006; revised submission received: 1 May 2007; accepted for publication: 19 June 2007.

and Kingsbury 2005), inspired by Levin's verb classes. To annotate natural language sentences, such systems generally require (1) the detection of the target word embodying the predicate and (2) the detection and classification of the word sequences constituting the predicate's arguments.

Previous work has shown that these steps can be carried out by applying machine learning techniques (Carreras and Màrquez 2004, 2005; Litkowski 2004), for which the most important features encoding predicate–argument relations are derived from (shallow or deep) syntactic information. The outcome of this research brings wide empirical evidence in favor of the linking theories between semantics and syntax, for example, Jackendoff (1990). Nevertheless, as no such theory provides a sound and complete treatment, the choice and design of syntactic features to represent semantic structures requires remarkable research effort and intuition.

For example, earlier studies on feature design for semantic role labeling were carried out by Gildea and Jurafsky (2002) and Thompson, Levy, and Manning (2003). Since then, researchers have proposed several syntactic feature sets, where the more recent sets slightly enhanced the older ones.

A careful analysis of such features reveals that most of them are syntactic tree fragments of training sentences, thus a viable way to alleviate the feature design complexity is the adoption of syntactic tree kernels (Collins and Duffy 2002). For example, in Moschitti (2004), the predicate–argument relation is represented by means of the minimal subtree that includes both of them. The similarity between two instances is evaluated by a tree kernel function in terms of common substructures. Such an approach is in line with current research on kernels for natural language learning, for example, syntactic parsing re-ranking (Collins and Duffy 2002), relation extraction (Zelenko, Aone, and Richardella 2003), and named entity recognition (Cumby and Roth 2003; Culotta and Sorensen 2004).

Furthermore, recent work (Haghighi, Toutanova, and Manning 2005; Punyakanok et al. 2005) has shown that, to achieve high labeling accuracy, joint inference should be applied on the whole predicate–argument structure. For this purpose, we need to extract features from the sentence syntactic parse tree that encodes the relationships governing complex semantic structures. This task is rather difficult because we do not exactly know which syntactic clues effectively capture the relation between the predicate and its arguments. For example, to detect the interesting context, the modeling of syntax-/semantics-based features should take into account linguistic aspects like ancestor nodes or semantic dependencies (Toutanova, Markova, and Manning 2004). In this scenario, the automatic feature generation/selection carried out by tree kernels can provide useful insights into the underlying linguistic phenomena. Other advantages coming from the use of tree kernels are the following.

First, we can implement them very quickly as the feature extractor module only requires the writing of a general procedure for subtree extraction. In contrast, traditional SRL systems use more than thirty features (e. g., Pradhan, Hacioglu, Krugler et al. 2005), each of which requires the writing of a dedicated procedure.

Second, their combination with traditional attribute–value models produces more accurate systems, also when using the same machine learning algorithm in the combination, because the feature spaces are very different.

Third, we can carry out feature engineering using kernel combinations and marking strategies (Moschitti et al. 2005a; Moschitti, Pighin, and Basili 2006). This allows us to boost the SRL accuracy in a relatively simple way.

Next, tree kernels generate large tree fragment sets which constitute back-off models for important syntactic features. Using them, the learning algorithm generalizes

better and produces a more accurate classifier, especially when the amount of training data is scarce.

Finally, once the learning algorithm using tree kernels has converged, we can identify the most important structured features of the generated model. One approach for such a reverse engineering process relies on the computation of the explicit feature space, at least for the highest-weighted features (Kudo and Matsumoto 2003). Once the most relevant fragments are available, they can be used to design novel effective attribute–value features (which in turn can be used to design more efficient classifiers, e. g., with linear kernels) and inspire new linguistic theories.

These points suggest that tree kernels should always be applied, at least for an initial study of the problem. Unfortunately, they suffer from two main limitations: (a) poor impact on boundary detection as, in this task, correct and incorrect arguments may share a large portion of the encoding trees (Moschitti 2004); and (b) more expensive running time and limited contribution to the overall accuracy if compared with manually derived features (Cumby and Roth 2003). Point (a) has been addressed by Moschitti, Pighin, and Basili (2006) by showing that a strategy of marking relevant parse-tree nodes makes correct and incorrect subtrees for boundary detection quite different. Point (b) can be tackled by studying approaches to kernel engineering that allow for the design of efficient and effective kernels.

In this article, we provide a comprehensive study of the use of tree kernels for semantic role labeling. For this purpose, we define tree kernels based on the composition of two different feature functions: **canonical mappings**, which map sentence-parse trees in tree structures encoding semantic information, and **feature extraction functions**, which encode these trees in the actual feature space. The latter functions *explode* the canonical trees into all their substructures and, in the literature, are usually referred to as **tree kernels**. For instance, in Collins and Duffy (2002), Vishwanathan and Smola (2002), and Moschitti (2006a) different tree kernels extract different types of tree fragments.

Given the heuristic nature of canonical mappings, we studied their properties by experimenting with them within support vector machines and with the data set provided by CoNLL shared tasks (Carreras and Màrquez 2005). The results show that carefully engineered tree kernels always boost the accuracy of the basic systems. Most importantly, in complex tasks such as the re-ranking of semantic role annotations, they provide an easy way to engineer new features which enhance the state-of-the-art in SRL.

In the remainder of this article, Section 2 presents traditional architectures for SRL and the typical features proposed in literature. Tree kernels are formally introduced in Section 3, and Section 4 describes our modular architecture employing support vector machines along with manually designed features, tree kernels (feature extraction functions), and their combinations. Section 5 presents our structured features (canonical mappings) inducing different kernels that we used for different SRL subtasks. The extensive experimental results obtained on the boundary recognition, role classification, and re-ranking stages are presented in Section 6. Finally, Section 7 summarizes the conclusions.

2. Automatic Shallow Semantic Parsing

The recognition of semantic structures within a sentence relies on lexical and syntactic information provided by early stages of an NLP process, such as lexical analysis, part-of-speech tagging, and syntactic parsing. The complexity of the SRL task mostly depends on two aspects: (a) the information is generally noisy, that is, in a real-world scenario the accuracy and reliability of NLP subsystems are generally not very high; and (b) the

lack of a sound and complete linguistic or cognitive theory about the links between syntax and semantics does not allow an informed, deductive approach to the problem. Nevertheless, the large amount of available lexical and syntactic information favors the application of inductive approaches to the SRL task, which indeed is generally treated as a combination of statistical classification problems.

The next sections define the SRL task more precisely and summarize the most relevant work carried out to address these two problems.

2.1 Problem Definition

The most well-known shallow semantic theories are studied in two different projects: PropBank (Palmer, Gildea, and Kingsbury 2005) and FrameNet (Baker, Fillmore, and Lowe 1998). The former is based on a linguistic model inspired by Levin's verb classes (Levin 1993), focusing on the argument structure of verbs and on the alternation patterns that describe movements of verbal arguments within a predicate structure. The latter refers to the application of frame semantics (Fillmore 1968) in the annotation of predicate–argument structures based on frame elements (semantic roles). These theories have been investigated in two CoNLL shared tasks (Carreras and Màrquez 2004, 2005) and a Senseval-3 evaluation (Litkowski 2004), respectively.

Given a sentence and a predicate word, an SRL system outputs an annotation of the sentence in which the sequences of words that make up the arguments of the predicate are properly labeled, for example:

$$[_{\mathbf{Arg0}} \text{ He}] \ \mathbf{got} \ [_{\mathbf{Arg1}} \text{ his money}] \ [_{\mathbf{C\text{-}V}} \text{ back}]^{1}$$

in response to the input *He got his money back*. This processing requires that: (1) the predicates within the sentence are identified and (2) the word sequences that span the boundaries of each predicate argument are delimited and assigned the proper role label.

The first sub-task can be performed either using statistical methods or hand-crafted lexical and syntactic rules. In the case of verbal predicates, it is quite easy to write simple rules matching regular expressions built on POS tags. The second task is more complex and is generally viewed as a combination of statistical classification problems: The learning algorithms are trained to recognize the extension of predicate arguments and the semantic role they play.

2.2 Models for Semantic Role Labeling

An SRL model and the resulting architecture are largely influenced by the kind of data available for the task. As an example, a model relying on a shallow syntactic parser would assign roles to chunks, whereas with a full syntactic parse of the sentence it would be straightforward to establish a correspondence between nodes of the parse tree and semantic roles. We focused on the latter as it has been shown to be more accurate by the CoNLL 2005 shared task results.

According to the deep syntactic formulation, the classifying instances are pairs of parse-tree nodes which dominate the exact span of the predicate and the target argument. Such pairs are usually represented in terms of attribute–value vectors, where

1 In PropBank notation, Arg0 and Arg1 represent the logical subject and the logical object of the target verbal predicate, respectively. C-V represents the particle of a phrasal-verb predicate.

the attributes describe properties of predicates, arguments, and the way they are related. There is large agreement on an effective set of linguistic features (Gildea and Jurafsky 2002; Pradhan, Hacioglu, Krugler, et al. 2005) that have been employed in the vast majority of SRL systems. The most relevant features are summarized in Table 1.

Once the representation for the predicate–argument pairs is available, a multi-classifier is used to recognize the correct node pairs, namely, nodes associated with correct arguments (given a predicate), and assign them a label (which is the label of the argument). This can be achieved by training a multi-classifier on $n + 1$ classes, where the first n classes correspond to the different roles and the $(n + 1)^{th}$ is a *NARG* (non-argument) class to which non-argument nodes are assigned.

A more efficient solution consists in dividing the labeling process into two steps: boundary detection and argument classification. A **Boundary Classifier** (BC) is a binary classifier that recognizes the tree nodes that exactly cover a predicate argument, that is, that dominate all and only the words that belong to target arguments. Then, such nodes are classified by a **Role Multi-classifier** (RM) that assigns to each example the most appropriate label. This two-step approach (Gildea and Jurafsky 2002) has the advantage of only applying BC on all parse-tree nodes. RM can ignore non-boundary nodes, resulting in a much faster classification. Other approaches have extended this solution and suggested other multi-stage classification models (e. g., Moschitti et al. 2005b in which a four-step hierarchical SRL architecture is described).

After node labeling has been carried out, it is possible that the output of the argument classifier does not result in a consistent annotation, as the labeling scheme may not be compatible with the underlying linguistic model. As an example, PropBank-style annotations do not allow arguments to be nested. This happens when two or more

Table 1
Standard linguistic features employed by most SRL systems.

Feature Name	Description
Predicate	Lemmatization of the predicate word
Path	Syntactic path linking the predicate and an argument, e. g., NN↑NP↑VP↓VBX
Partial path	*Path* feature limited to the branching of the argument
No-direction path	Like *Path*, but without traversal directions
Phrase type	Syntactic type of the argument node
Position	Relative position of the argument with respect to the predicate
Voice	Voice of the predicate, i. e., active or passive
Head word	Syntactic head of the argument phrase
Verb subcategorization	Production rule expanding the predicate parent node
Named entities	Classes of named entities that appear in the argument node
Head word POS	POS tag of the argument node head word (less sparse than Head word)
Verb clustering	Type of verb → direct object relation
Governing Category	Whether the candidate argument is the verb subject or object
Syntactic Frame	Position of the NPs surrounding the predicate
Verb sense	Sense information for polysemous verbs
Head word of PP	Enriched POS of prepositional argument nodes (e. g., PP-for, PP-in)
First and last word/POS	First and last words and POS tags of candidate argument phrases
Ordinal position	Absolute offset of a candidate argument within a proposition
Constituent tree distance	Distance from the predicate with respect to the parse tree
Constituent features	Description of the constituents surrounding the argument node
Temporal Cue Words	Temporal markers which are very distinctive of some roles

overlapping tree nodes, namely, one dominating the other, are classified as positive boundaries.

The simplest solution relies on the application of heuristics that take into account the whole predicate–argument structure to remove the incorrect labels (e. g., Moschitti et al. 2005a; Tjong Kim Sang et al. 2005). A much more complex solution consists in the application of some joint inference model to the whole predicate–argument structure, as in Pradhan et al. (2004). As an example, Haghighi, Toutanova, and Manning (2005) associate a posterior probability with each argument node role assignment, estimate the likelihood of the alternative labeling schemes, and employ a re-ranking mechanism to select the best annotation.

Additionally, the most accurate systems participating in CoNLL 2005 shared task (Pradhan, Hacioglu, Ward et al. 2005; Punyakanok et al. 2005) use different syntactic views of the same input sentence. This allows the SRL system to recover from syntactic parser errors; for example, a prepositional phrase specifying the direct object of the predicate would be attached to the verb instead of the argument. This kind of error prevents some arguments of the proposition from being recognized, as: (1) there may not be a node of the parse tree dominating (all and only) the words of the correct sequence; (2) a badly attached tree node may invalidate other argument nodes, generating unexpected overlapping situations.

The manual design of features which capture important properties of complete predicate–argument structures (also coming from different syntactic views) is quite complex. Tree kernels are a valid alternative to manual design as the next section points out.

3. Tree Kernels

Tree kernels have been applied to reduce the feature design effort in the context of several natural language tasks, for example, syntactic parsing re-ranking (Collins and Duffy 2002), relation extraction (Zelenko, Aone, and Richardella 2003), named entity recognition (Cumby and Roth 2003; Culotta and Sorensen 2004), and semantic role labeling (Moschitti 2004).

On the one hand, these studies show that the kernel ability to generate large feature sets is useful to quickly model new and not well understood linguistic phenomena in learning machines. On the other hand, they show that sometimes it is possible to manually design features for linear kernels that produce higher accuracy and faster computation time. One of the most important causes of such mixed behavior is the inappropriate choice of kernel functions. For example, in Moschitti, Pighin, and Basili (2006) and Moschitti (2006a), several kernels have been designed and shown to produce different impacts on the training algorithms.

In the next sections, we briefly introduce the kernel trick and describe the **subtree** (ST) kernel devised in Vishwanathan and Smola (2002), the **subset tree** (SST) kernel defined in Collins and Duffy (2002), and the **partial tree** (PT) kernel proposed in Moschitti (2006a).

3.1 Kernel Trick

The main concept underlying machine learning for classification tasks is the automatic learning of classification functions based on examples labeled with the class information. Such examples can be described by means of feature vectors in an n dimensional

space over the real numbers, namely, $\Re^n$. The learning algorithm uses space metrics over vectors, for example, the scalar product, to learn an abstract representation of all instances belonging to the target class.

For example, support vector machines (SVMs) are linear classifiers which learn a hyperplane $f(\vec{x}) = \vec{w} \times \vec{x} + b = 0$, separating positive from negative examples. $\vec{x}$ is the feature vector representation of a classifying object o, whereas $\vec{w} \in \Re^n$ and $b \in \Re$ are parameters learned from the data by applying the *Structural Risk Minimization principle* (Vapnik 1998). The object o is mapped to $\vec{x}$ via a feature function $\phi : \mathcal{O} \rightarrow \Re^n$, $\mathcal{O}$ being the set of the objects that we want to classify. o is categorized in the target class only if $f(\vec{x}) \geq 0$.

The kernel trick allows us to rewrite the decision hyperplane as:

$$f(\vec{x}) = \Big(\sum_{i=1..l} y_i \alpha_i \vec{x}_i \Big) \cdot \vec{x} + b = \sum_{i=1..l} y_i \alpha_i \vec{x}_i \cdot \vec{x} + b = \sum_{i=1..l} y_i \alpha_i \phi(o_i) \cdot \phi(o) + b = 0$$

where y_i is equal to 1 for positive examples and -1 for negative examples, $\alpha_i \in \Re$ with $\alpha_i \geq 0$, $o_i \; \forall i \in \{1, .., l\}$ are the training instances and the product $K(o_i, o) = \langle \phi(o_i) \cdot \phi(o) \rangle$ is the kernel function associated with the mapping ϕ.

Note that we do not need to apply the mapping ϕ; we can use $K(o_i, o)$ directly. This allows us, under Mercer's conditions (Shawe-Taylor and Cristianini 2004), to define abstract kernel functions which generate implicit feature spaces. A traditional example is given by the polynomial kernel: $K_p(o_1, o_2) = (c + \vec{x}_1 \cdot \vec{x}_2)^d$, where c is a constant and d is the degree of the polynomial. This kernel generates the space of all conjunctions of feature groups up to d elements.

Additionally, we can carry out two interesting operations:

- kernel combinations, for example, $K_1 + K_2$ or $K_1 \times K_2$
- feature mapping compositions, for example, $K(o_1, o_2) = \langle \phi(o_1) \cdot \phi(o_2) \rangle = \langle \phi_B(\phi_A(o_1)) \cdot \phi_B(\phi_A(o_2)) \rangle$

Kernel combinations are very useful for integrating the knowledge provided by the manually defined features with the knowledge automatically obtained with structural kernels; feature mapping compositions are useful methods to describe diverse kernel classes (see Section 5). In this perspective, we propose to split the mapping ϕ by defining our tree kernel as follows:

- **Canonical Mapping**, $\phi_M()$, in which a linguistic object (e. g., a syntactic parse tree) is transformed into a more meaningful structure (e. g., the subtree corresponding to a verb subcategorization frame).
- **Feature Extraction**, $\phi_S()$, which maps the canonical structure in all its fragments according to different fragment spaces S (e. g., ST, SST, and PT).

For example, given the kernel $K_{ST} = \phi_{ST}(o_1) \cdot \phi_{ST}(o_2)$, we can apply a canonical mapping $\phi_M()$, obtaining $K_{ST}^M = \phi_{ST}(\phi_M(o_1)) \cdot \phi_{ST}(\phi_M(o_2)) = \big(\phi_{ST} \circ \phi_M\big)(o_1) \cdot \big(\phi_{ST} \circ \phi_M\big)(o_2)$, which is a noticeably different kernel, which is induced by the mapping $\big(\phi_{ST} \circ \phi_M\big)$.

In the remainder of this section we start the description of our engineered kernels by defining three different feature extraction mappings based on three different kernel spaces (i. e., ST, SST, and PT).

3.2 Tree Kernel Spaces

The kernels that we consider represent trees in terms of their substructures (fragments). The kernel function detects if a tree subpart (common to both trees) belongs to the feature space that we intend to generate. For this purpose, the desired fragments need to be described. We consider three main characterizations: the subtrees (STs) (Vishwanathan and Smola 2002), the subset trees (SSTs) or all subtrees (Collins and Duffy 2002), and the partial trees (PTs) (Moschitti 2006a).

As we consider syntactic parse trees, each node with its children is associated with a grammar production rule, where the symbol on the left-hand side corresponds to the parent and the symbols on the right-hand side are associated with the children. The terminal symbols of the grammar are always associated with tree leaves.

A **subtree** (ST) is defined as a tree rooted in any non-terminal node along with all its descendants. For example, Figure 1a shows the parse tree of the sentence *Mary brought a cat* together with its six STs. A **subset tree** (SST) is a more general structure because its leaves can be non-terminal symbols. For example, Figure 1(b) shows ten SSTs (out of 17) of the subtree in Figure 1a rooted in `VP`. SSTs satisfy the constraint that grammatical rules cannot be broken. For example, `[VP [V NP]]` is an SST which has two non-terminal symbols, `V` and `NP`, as leaves. On the contrary, `[VP [V]]` is not an SST as it violates the production VP→V NP. If we relax the constraint over the SSTs, we obtain a more general form of substructures that we call **partial trees** (PTs). These can be generated by the application of partial production rules of the grammar; consequently `[VP [V]]` and `[VP [NP]]` are valid PTs. It is worth noting that PTs consider the position of the children as, for example, `[A [B][C][D]]` and `[A [D][C][B]]` only share single children, i.e., `[A [B]]`, `[A [C]]`, and `[A [D]]`.

Figure 1c shows that the number of PTs derived from the same tree as before is still higher (i. e., 30 PTs). These numbers provide an intuitive quantification of the different degrees of information encoded by each representation.

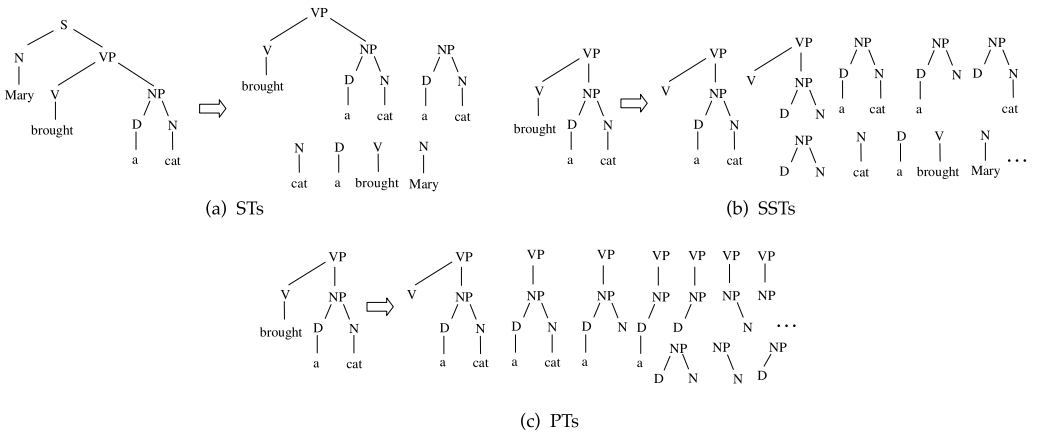

Figure 1
Example of (a) ST, (b) SST, and (c) PT fragments.

3.3 Feature Extraction Functions

The main idea underlying tree kernels is to compute the number of common substructures between two trees T_1 and T_2 without explicitly considering the whole fragment space. In the following, we report on the Subset Tree (SST) kernel proposed in Collins and Duffy (2002). The algorithms to efficiently compute it along with the ST and PT kernels can be found in Moschitti (2006a).

Given two trees T_1 and T_2, let $\{f_1, f_2, ..\} = \mathcal{F}$ be the set of substructures (fragments) and $I_i(n)$ be equal to 1 if f_i is rooted at node n, 0 otherwise. Collins and Duffy's kernel is defined as

$$K(T_1, T_2) = \sum_{n_1 \in N_{T_1}} \sum_{n_2 \in N_{T_2}} \Delta(n_1, n_2) \quad (1)$$

where N_{T_1} and N_{T_2} are the sets of nodes in T_1 and T_2, respectively, and $\Delta(n_1, n_2) = \sum_{i=1}^{|\mathcal{F}|} I_i(n_1) I_i(n_2)$. The latter is equal to the number of common fragments rooted in nodes n_1 and n_2. Δ can be computed as follows:

1. If the productions (i.e. the nodes with their direct children) at n_1 and n_2 are different, then $\Delta(n_1, n_2) = 0$.
2. If the productions at n_1 and n_2 are the same, and n_1 and n_2 only have leaf children (i.e., they are pre-terminal symbols), then $\Delta(n_1, n_2) = 1$.
3. If the productions at n_1 and n_2 are the same, and n_1 and n_2 are not pre-terminals, then $\Delta(n_1, n_2) = \prod_{j=1}^{nc(n_1)} (1 + \Delta(c_{n_1}^j, c_{n_2}^j))$, where $nc(n_1)$ is the number of children of n_1 and c_n^j is the j-th child of n.

Such tree kernels can be normalized and a λ factor can be added to reduce the weight of large structures (refer to Collins and Duffy [2002] for a complete description).

3.4 Related Work

Although the literature on SRL is extensive, there is almost no study of the use of tree kernels for its solution. Consequently, the reported research is mainly based on diverse natural language learning problems tackled by means of tree kernels.

In Collins and Duffy (2002), the SST kernel was experimented with using the voted perceptron for the parse tree re-ranking task. A combination with the original PCFG model improved the syntactic parsing. Another interesting kernel for re-ranking was defined in Toutanova, Markova, and Manning (2004). This represents parse trees as lists of paths (leaf projection paths) from leaves to the top level of the tree. It is worth noting that the PT kernel includes tree fragments identical to such paths.

In Kazama and Torisawa (2005), an interesting algorithm that speeds up the average running time is presented. This algorithm looks for node pairs in which the rooted subtrees share many substructures (**malicious nodes**) and applies a transformation to the trees rooted in such nodes to make the kernel computation faster. The results show a several-hundred-fold speed increase with respect to the basic implementation.

In Zelenko, Aone, and Richardella (2003), two kernels over syntactic shallow parser structures were devised for the extraction of linguistic relations, for example, *person-affiliation*. To measure the similarity between two nodes, the *contiguous string kernel* and the *sparse string kernel* were used. In Culotta and Sorensen (2004) such kernels were slightly generalized by providing a matching function for the node pairs. The time complexity for their computation limited the experiments to a data set of just 200 news items.

In Shen, Sarkar, and Joshi (2003), a tree kernel based on lexicalized tree adjoining grammar (LTAG) for the parse re-ranking task was proposed. The subtrees induced by this kernel are built using the set of elementary trees as defined by LTAG.

In Cumby and Roth (2003), a feature description language was used to extract structured features from the syntactic shallow parse trees associated with named entities. Their experiments on named entity categorization showed that when the description language selects an adequate set of tree fragments the voted perceptron algorithm increases its classification accuracy. The explanation was that the complete tree fragment set contains many irrelevant features and may cause overfitting.

In Zhang, Zhang, and Su (2006), convolution tree kernels for relation extraction were applied in a way similar to the one proposed in Moschitti (2004). The combination of standard features along with several tree subparts, tailored according to their importance for the task, produced again an improvement on the state of the art.

Such previous work, as well as that described previously, show that tree kernels can efficiently represent syntactic objects, for example, constituent parse trees, in huge feature spaces. The next section describes our SRL system adopting tree kernels within SVMs.

4. A State-of-the-Art Architecture for Semantic Role Labeling

A meaningful study of tree kernels for SRL cannot be carried out without a comparison with a state-of-the-art architecture: Kernel models that improve average performing systems are just a technical exercise whose findings would have a reduced value. A state-of-the-art architecture, instead, can be used as a basic system upon which tree kernels should improve. Because kernel functions in general introduce a sensible slowdown with respect to the linear approach, we also have to consider efficiency issues. These aims drove us in choosing the following components for our SRL system:

- SVMs as our learning algorithm; these provide both a state-of-the-art learning model (in terms of accuracy) and the possibility of using kernel functions
- a two-stage role labeling module to improve learning and classification efficiency; this comprises:
  - a feature extractor that can represent candidate arguments using both linear and structured features
  - a boundary classifier (BC)
  - a role multi-classifier (RM), which is obtained by applying the OVA (One vs. All) approach
- a conflict resolution module, that is, a software component that resolves inconsistencies in the annotations using either a rule-based approach or a tree kernel classifier; the latter allows experimentation with

the classification of complete predicate–argument annotations in correct and incorrect structures

- a joint inference re-ranking module, which employs a combination of standard features and tree kernels to rank alternative candidate labeling schemes for a proposition; this module, as shown in Gildea and Jurafsky (2002), Pradhan et al. (2004), and Haghighi, Toutanova, and Manning (2005), is mandatory in order to achieve state-of-the-art accuracy

We point out that we did not use any heuristic to filter out the nodes which are likely to be incorrect boundaries, for example, as done in Xue and Palmer (2004). On the one hand, this makes the learning and classification phases more complex because they involve more instances. On the other hand, our results are not biased by the quality of the heuristics, leading to more meaningful findings.

In the remainder of this section, we describe the main functional modules of our architecture for SRL and introduce some basic concepts about the use of structured features for SRL. Specific feature engineering for the above SRL subtasks is described and discussed in Section 5.

4.1 A Basic Two-Stage Role Labeling System

Given a sentence in natural language, our SRL system identifies all the verb predicates and their respective arguments. We divide this step into three subtasks: (a) predicate detection, which can be carried out by simple heuristics based on part-of-speech information, (b) the detection of predicate–argument boundaries (i. e., the span of their words in the sentence), and (c) the classification of the argument type (e. g., *Arg0* or *ArgM* in PropBank).

The standard approach to learning both the detection and the classification of predicate arguments is summarized by the following steps:

1. Given a sentence from the *training set*, generate a full syntactic parse tree;
2. let $\mathcal{P}$ and $\mathcal{A}$ be the set of predicates and the set of parse-tree nodes (i. e., the potential arguments), respectively;
3. for each pair $\langle p, a\rangle \in \mathcal{P} \times \mathcal{A}$:
   - extract the feature representation, $\phi(p, a)$, (e. g., attribute–values or tree fragments [see Section 3.1]);
   - if the leaves of the subtree rooted in a correspond to all and only the words of one argument of p (i. e., a exactly covers an argument), add $\phi(p, a)$ in E^+ (positive examples), otherwise add it in E^- (negative examples).

For instance, given the example in Figure 2(a), we would consider all the pairs $\langle p, a\rangle$ where p is the node associated with the predicate *took* and a is any other tree node not overlapping with p. If the node a exactly covers the word sequences *John* or *the book*, then $\phi(p, a)$ is added to the set E^+, otherwise it is added to E^-, as in the case of the node *(NN book)*.

The E^+ and E^- sets are used to train the boundary classifier. To train the role multiclassifier, the elements of E^+ can be reorganized as positive $E^+_{arg_i}$ and negative $E^-_{arg_i}$ examples for each role type i. In this way, a binary OVA classifier for each argument

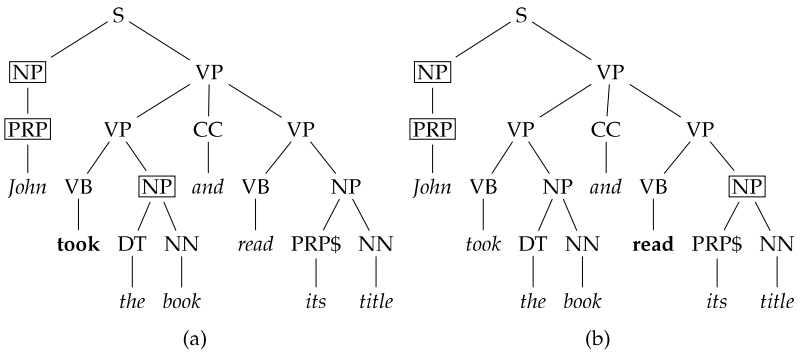

Figure 2
Positive (framed) and negative (unframed) examples of candidate argument nodes for the propositions (a) [$_{Arg0}$ *John*] ***took*** [$_{Arg1}$ *the book*] *and read its title* and (b) [$_{Arg0}$ *John*] *took the book and* ***read*** [$_{Arg1}$ *its title*].

i can be trained. We adopted this solution following Pradhan, Hacioglu, Krugler et al. (2005) because it is simple and effective. In the classification phase, given an unseen sentence, all the pairs $\langle p, a \rangle$ are generated and classified by each individual role classifier C_i. The argument label associated with the maximum among the scores provided by C_i is eventually selected.

The feature extraction function ϕ can be implemented according to different linguistic theories and intuitions. From a technical point of view, we can use ϕ to map $\langle p, a \rangle$ in feature vectors or in structures to be used in a tree kernel function. The next section describes our choices in more detail.

4.2 Linear and Structured Representation

Our feature extractor module and our learning algorithms are designed to cope with both linear and structured features, used for the different stages of the SRL process.

The standard features that we adopted are shown in Table 1. They include:

- the *Phrase Type*, *Predicate Word*, *Head Word*, *Governing Category*, *Position*, and *Voice* defined in Gildea and Jurafsky (2002);
- the *Partial Path*, *No Direction Path*, *Constituent Tree Distance*, *Head Word POS*, *First and Last Word/POS*, *Verb Subcategorization*, and *Head Word of the Noun Phrase in the Prepositional Phrase* proposed in Pradhan, Hacioglu, Krugler et al. (2005); and
- the *Syntactic Frame* defined in Xue and Palmer (2004).

We indicate with structured features the basic syntactic structures extracted from the sentence-parse tree or their canonical transformation (see Section 3.1). In particular, we focus on the minimal spanning tree that includes the predicate along with all of its arguments.

More formally, given a parse tree t, a **node set spanning tree** (NST) over a set of nodes $N_t = \{n_1, \ldots, n_k\}$ is a partial tree of t that (1) is rooted at the deepest level and (2) contains all and only the nodes $n_i \in N_t$, along with their ancestors and descendants. An NST can be built as follows. For any choice of N_t, we call r the lowest common ancestor

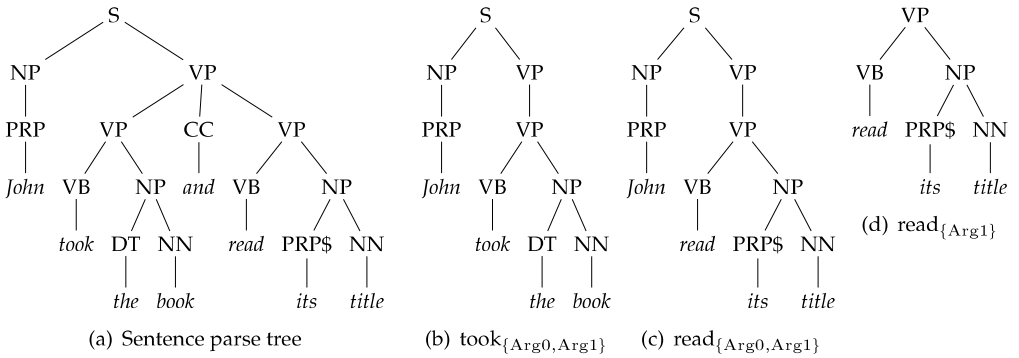

Figure 3
(a) A sentence parse tree, the correct AST_ns associated with two different predicates (b,c), and (d) a correct AST_1 relative to the argument Arg1 *its title* of the predicate *read*.

of $n_1, \ldots, n_k$. Then, from the set of all the descendants of r, we remove all the nodes n_j that: (1) do not belong to N_t and (2) are neither ancestors nor descendants of any node belonging to N_t.

Because predicate arguments are associated with tree nodes, we can define the **predicate argument spanning tree** (AST_n) of a predicate argument node set $A_p = \{a_1, \ldots, a_n\}$ as the NST over these nodes and the predicate node, that is, the node exactly covering the predicate p.[2] An AST_n corresponds to the *minimal* parse subtree whose leaves are all and only the word sequences belonging to the arguments and the predicate. For example, Figure 3a shows the parse tree of the sentence: *John took the book and read its title*. $took_{\{ARG_0,ARG_1\}}$ and $read_{\{ARG_0,ARG_1\}}$ are two AST_n structures associated with the two predicates *took* and *read*, respectively, and are shown in Figure 3b and 3c.

For each predicate, only one NST is a valid AST_n. Careful manipulations of an AST_n can be employed for those tasks that require a representation of the whole predicate–argument structure, for example, overlap resolution or proposition re-ranking.

It is worth noting that the **predicate–argument feature**, or PAF in Moschitti (2004), is a canonical transformation of the AST_n in the subtree including the predicate p and only one of its arguments. For the sake of uniform notation, PAF will be referred to as AST_1 (**argument spanning tree**), the subscript *1* stressing the fact that the structure only encompasses one of the predicate arguments. An example AST_1 is shown in Figure 3d. Manipulations of an AST_1 structure can lead to interesting tree kernels for local learning tasks, such as boundary detection and argument classification.

Regardless of the adopted feature space, our multiclassification approach suffers from the problem of selecting both boundaries and argument roles independently of the whole structures. Thus, it is possible that (a) two labeled nodes refer to the same arguments (node overlaps) and (b) invalid role sequences are generated (e. g., *Arg0, Arg0, Arg0, ...*). Next, we describe our approach to solving such problems.

4.3 Conflict Resolution

We call a **conflict**, or **ambiguity**, or **overlap** resolution a stage of the SRL process which resolves annotation conflicts that invalidate the underlying linguistic model. This

2 The AST_n of a predicate p and its argument nodes $\{a_1, \ldots, a_n\}$, will also be referred to as $p_{\{a_1,\ldots,a_n\}}$.

happens, for example, when both a node and one of its descendants are classified as positive boundaries, namely, they received a role label. We say that such nodes are **overlapping** as their leaf (i. e., word) sequences overlap. Because this situation is not allowed by the PropBank annotation definition, we need a method to select the most appropriate word sequence. Our system architecture can employ one of three different **disambiguation** strategies:

- a basic solution which, given two overlapping nodes, randomly selects one to be removed;
- the following heuristics:
  1. The node causing the major number of overlaps is removed, for example, a node which dominates two nodes labeled as arguments
  2. Core arguments (i. e., arguments associated with the subcategorization frame of the target verb) are always preferred over adjuncts (i. e., arguments that are not specific to verbs or verb senses)
  3. In case the two previous rules do not eliminate all conflicts, the nodes located deeper in the tree are discarded; and
- a tree kernel–based overlap resolution strategy consisting of an SVM trained to recognize non-clashing configurations that often correspond to correct propositions.

The latter approach consists of: (1) a software module that generates all the possible non-overlapping configurations of nodes. These are built using the output of the local node classifiers by generating all the permutations of argument nodes of a predicate and removing the configurations that contain at least one overlap; (2) an SVM trained on such non-overlapping configurations, where the positive examples are correct predicate–argument structures (although eventually not complete) and negative ones are not. At testing time, we classify all the alternative non-clashing configurations. In case more than one structure is selected as correct, we choose the one associated with the highest SVM score.

These disambiguation modules can be invoked after either the BC or the RM classification. The different information available after each phase can be used to design different kinds of features. For example, the knowledge of the candidate role of an argument node can be a key issue in the design of effective conflict resolution methodologies, for example, by eliminating Arg*X*, Arg*X*, Arg*X*, ... sequences. These different approaches are discussed in Section 5.2.

The next section describes a more advanced approach that can eliminate overlaps and choose the most correct annotation for a proposition among a set of alternative labeling schemes.

4.4 A Joint Model for Re-Ranking

The heuristics considered in the previous sections only act when a conflict is detected. In a real situation, many incorrect annotations are generated with no overlaps. To deal with such cases, we need a re-ranking module based on a joint BC and RM model as suggested in Haghighi, Toutanova, and Manning (2005). Such a model is based on (1)

an algorithm to evaluate the most likely labeling schemes for a given predicate, and (2) a re-ranker that sorts the labeling schemes according to their *correctness*.

Step 1 uses the probabilities associated with each possible annotation of parse tree nodes, hence requiring a probabilistic output from BC and RM. As the SVM learning algorithm produces metric values, we applied Platt's algorithm (Platt 1999) to convert them into probabilities, as already proposed in Pradhan, Ward et al. (2005). These posterior probabilities are then combined to generate the n labelings that maximize a likelihood measure. Step 2 requires the training of an automatic re-ranker. This can be designed using a binary classifier that, given two annotations, decides which one is more accurate. We modeled such a classifier by means of three different kernels based on standard features, structured features, and their combination.

4.4.1 Evaluation of the N-best Annotations. First, we converted the output of each node-classifier into a posterior probability conditioned by its output scores (Platt 1999). This method uses a parametric model to fit onto a sigmoid distribution the posterior probability $P(y = 1, f)$, where f is the output of the classifier and the parameters are dynamically adapted to give the best probability output.[3] Second, we selected the n most likely sequences of node labelings. Given a predicate, the likelihood of a labeling scheme (or *state*) s for the K candidate argument nodes is given by:

$$p(s) = \prod_{i=1}^{K} p'_i(l), \quad p'_i(l) = \begin{cases} p_i(l_i)p_i(\text{ARG}) & \text{if } l_i \neq \text{NARG} \\ (1 - p_i(\text{ARG}))^2 & \text{otherwise} \end{cases} \tag{2}$$

where $p_i(l)$ is the probability of node i being assigned the label l, and $p'_i(l)$ is the same probability weighted by the probability $p_i(ARG)$ of the node being an argument. If $l =$ NARG (not an argument) then both terms evaluate to $(1 - p_i(ARG))$ and the likelihood of the NARG label assignment is given by $(1 - p_i(ARG))^2$.

To select the n states associated with the highest probability, we cannot evaluate the likelihood of all possible states because they are exponential in number. In order to reduce the search space we (a) limit the number of possible labelings of each node to m and (b) avoid traversing all the states by applying a Viterbi algorithm to search for the most likely labeling schemes. From each state we generate the states in which a candidate argument is assigned different labels. This operation is bound to output at most n states which are generated by traversing a maximum of $n \times m$ states. Therefore, in the worst case scenario the number of traversed states is $V = n \times m \times k$, k being the number of candidate argument nodes in the tree.

During the search we also enforce overlap resolution policies. Indeed, for any given state in which a node n_j is assigned a label $l \neq NARG$, we generate all; and only the states in which all the nodes that are dominated by n_j are assigned the NARG label.

4.4.2 Modeling an Automatic Re-Ranker. The Viterbi algorithm generates the n most likely annotations for the proposition associated with a predicate p. These can be used to build annotation pairs, $\langle s_i, s_j \rangle$, which, in turn, are used to train a binary classifier that decides if

3 We actually implemented the pseudo-code proposed in Lin, Lin, and Weng (2003) which, with respect to Platt's original formulation, is theoretically demonstrated to converge and avoids some numerical difficulties that may arise.

s_i is more accurate that s_j. Each candidate proposition s_i can be described by a structured feature t_i and a vector of standard features v_i. As a whole, an example e_i is described by the tuple $\langle t_i^1, t_i^2, v_i^1, v_i^2 \rangle$, where t_i^1 and v_i^1 refer to the first candidate annotation, whereas t_i^2 and v_i^2 refer to the second one. Given such data, we can define the following re-ranking kernels:

$$K_{tr}(e_1, e_2) = K_t(t_1^1, t_1^1) + K_t(t_1^2, t_2^2) - K_t(t_1^1, t_2^2) - K_t(t_1^2, t_2^1)$$
$$K_{pr}(e_1, e_2) = K_p(v_1^1, v_2^1) + K_p(v_1^2, v_2^2) - K_p(v_1^1, v_2^2) - K_p(v_1^2, v_2^1)$$

where K_t is one of the tree kernel functions defined in Section 3 and K_p is a polynomial kernel applied to the feature vectors. The final kernel that we use is the following combination:

$$K(e_1, e_2) = \frac{K_{tr}(e_1, e_2)}{|K_{tr}(e_1, e_2)|} + \frac{K_{pr}(e_1, e_2)}{|K_{pr}(e_1, e_2)|}$$

Previous sections have shown how our SRL architecture exploits tree kernel functions to a large extent. In the next section, we describe in more detail our structured features and the engineering methods applied for the different subtasks of the SRL process.

5. Structured Feature Engineering

Structured features are an effective alternative to *standard* features in many aspects. An important advantage is that the target feature space can be completely changed even by small modifications of the applied kernel function. This can be exploited to identify features relevant to learning problems lacking a clear and sound linguistic or cognitive justification.

As shown in Section 3.1, a kernel function is a scalar product $\phi(o_1) \cdot \phi(o_2)$, where ϕ is a mapping in an Euclidean space, and o_1 and o_2 are the target data, for example, parse trees. To make the engineering process easier, we decompose ϕ into a **canonical mapping**, ϕ_M, and a **feature extraction** function, ϕ_S, over the set of incoming parse trees. ϕ_M transforms a tree into a canonical structure equivalent to an entire class of input parses and ϕ_S shatters an input tree into its subparts (e. g., subtrees, subset trees, or partial trees as described in Section 3). A large number of different feature spaces can thus be explored by suitable combinations $\phi = \phi_S \circ \phi_M$ of mappings.

We study different canonical mappings to capture syntactic/semantic aspects useful for SRL. In particular, we define structured features for the different phases of the SRL process, namely, boundary detection, argument classification, conflict resolution, and proposition re-ranking.

5.1 Structures for Boundary Detection and Argument Classification

The AST_1 or PAF structures, already mentioned in Section 4.2, have shown to be very effective for argument classification but not for boundary detection. The reason is that two nodes that encode correct and incorrect boundaries may generate very similar AST_1s and, consequently, have many fragments in common. To solve this problem, we

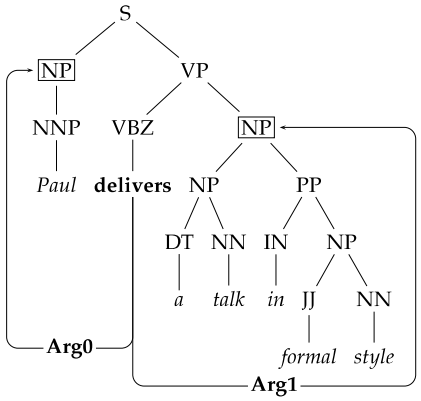

Figure 4
Parse tree of the example proposition [$_{Arg0}$ *Paul*] ***delivers*** [$_{Arg1}$ *a talk in formal style*].

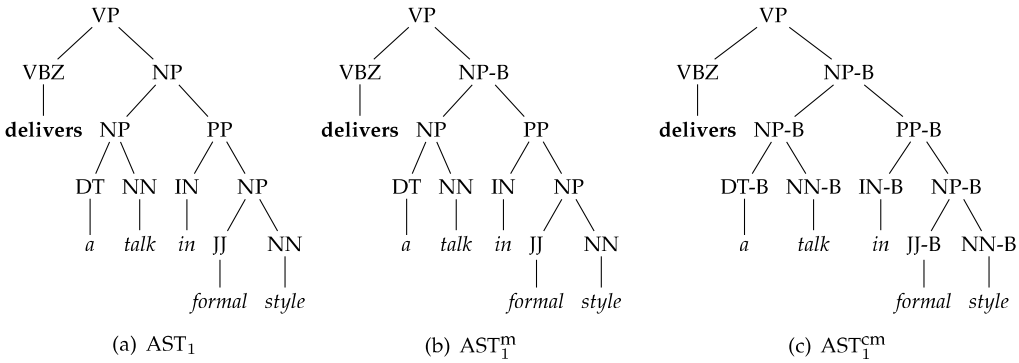

Figure 5
(a) AST_1, (b) AST_1^m, and (c) AST_1^{cm} structures relative to the argument Arg1 *a talk in formal style* of the predicate *delivers* of the example parse tree shown in Figure 4.

specify the node that exactly covers the target argument node by simply marking it (or marking all its descendants) with the label *B*, denoting the boundary property.

For example, Figure 4 shows the parse tree of the sentence *Paul delivers a talk in formal style*, highlighting the predicate with its two arguments, that is, Arg0 and Arg1. Figure 5 shows the AST_1, AST_1^m, and AST_1^{cm}, that is, the basic structure, the structure with the marked argument node, and the completely marked structure, respectively.

To understand the usefulness of node-marking strategies, we can examine Figure 6. This reports the case in which a correct and an incorrect argument node are chosen by also showing the corresponding AST_1 and AST_1^m representations ((a) and (b)). Figure 6c shows that the number of common fragments of two AST_1 structures is 14. This is much larger than the number of common AST_1^m fragments, that is, only 3 substructures (Figure 6d).

Additionally, because the type of a target argument strongly depends on the type and number of the other predicate arguments[4] (Punyakanok et al. 2005; Toutanova,

4 This is true at least for core arguments.

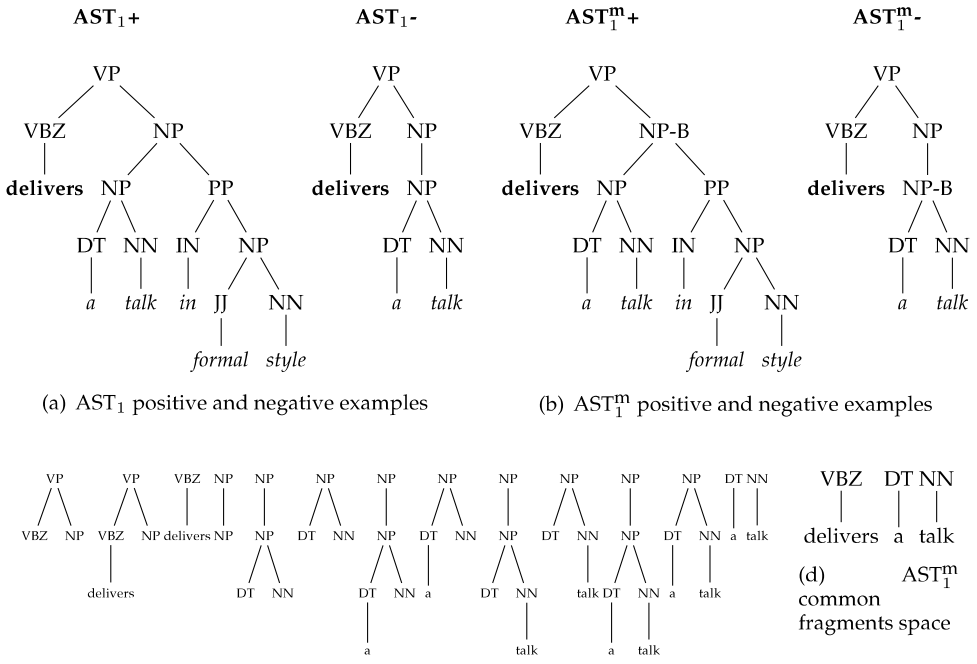

Figure 6
(a) AST_1s and (b) AST_1^ms extracted for the same target argument with their respective (c,b) common fragment spaces.

Haghighi, and Manning 2005), we should extract features from the whole predicate argument structure. In contrast, AST_1s completely neglect the information (i. e., the tree portions) related to non-target arguments.

One way to use this further information with tree kernels is to use the minimum subtree that spans all the predicate–argument structures, that is, the AST_n defined in Section 4.2.

However, AST_ns pose two problems. First, we cannot use them for the boundary detection task since we do not know the predicate–argument structure yet. We can derive the AST_n (its approximation) from the nodes selected by a boundary classifier, that is, the nodes that correspond to potential arguments. Such approximated AST_ns can be easily used in the argument classification stage.

Second, an AST_n is the same for all the arguments in a proposition, thus we need a way to differentiate it for each target argument. Again, we can mark the target argument node as shown in the previous section. We refer to this subtree as a **marked target** AST_n (AST_n^{mt}). However, for large arguments (i. e., spread over a large part of the sentence tree) the substructures' likelihood of being part of different arguments is quite high.

To address this problem, we can mark all the nodes that descend from the target argument node. We refer to this structure as a **completely marked target** AST_n (AST_n^{cmt}). AST_n^{cmt}s may be seen as AST_1s enriched with new information coming from the other arguments (i. e., the non-marked subtrees). Note that if we only consider the AST_1 subtree from a AST_n^{cmt}, we obtain AST_1^{cm}.

5.2 Structured Features for Conflict Resolution

This section describes structured features employed by the tree kernel–based conflict resolution module of the SRL architecture described in Section 4.3. This subtask is performed by means of:

1. A first annotation of potential arguments using a high recall boundary classifier and, eventually, the role information provided by a role multiclassifier (RM).
2. An AST_n classification step aiming at selecting, among the substructures that do not contain overlaps, those that are more likely to encode the correct argument set.

The set of argument nodes recognized by BC can be associated with a subtree of the corresponding sentence parse, which can be classified using tree kernel functions. These should evaluate whether a subtree encodes a correct predicate–argument structure or not. As it encodes features from the whole predicate–argument structure, the AST_n that we introduced in Section 4.2 is a structure that can be employed for this task.

Let A_p be the set of potential argument nodes for the predicate p output by BC; the classifier examples are built as follows: (1) we look for node pairs $\langle n_1, n_2 \rangle \in A_p \times A_p$ where n_1 is the ancestor of n_2 or *vice versa*; (2) we create two node sets $A_1 = A - \{n_1\}$ and $A_2 = A - \{n_2\}$ and classify the two NSTs associated with A_1 and A_2 with the tree kernel classifier to select the most correct set of argument boundaries. This procedure can be generalized to a set of overlapping nodes O with more than two elements, as we simply need to generate all and only the permutations of A's nodes that do not contain overlapping pairs.

Figure 7 shows a working example of such a multi-stage classifier. In (Figure 7a), the BC labels as potential arguments four nodes (circled), three of which are overlapping

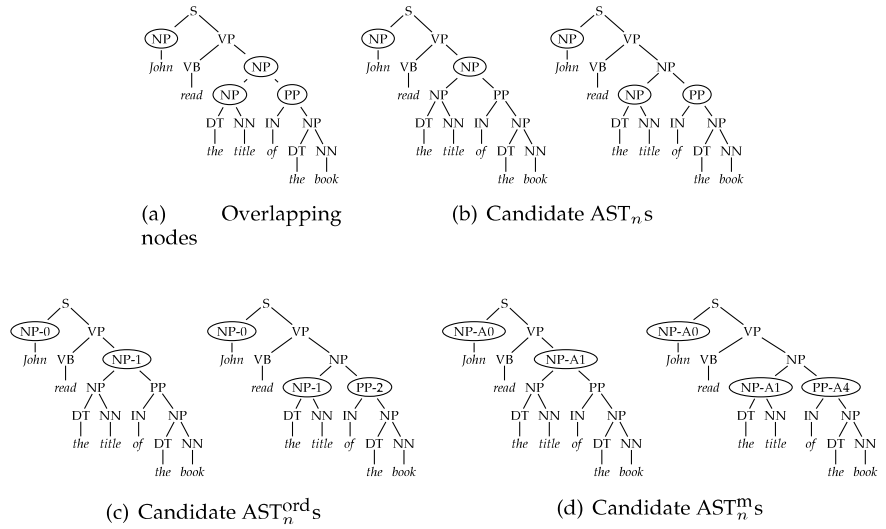

Figure 7
An overlap situation (a) and the candidate solutions resulting from the employment of the different marking strategies.

(in bold circles). The overlap resolution algorithm proposes two solutions (Figure 7b) of which only one is correct. In fact, according to the second solution, the prepositional phrase *of the book* would incorrectly be attached to the verbal predicate, that is, in contrast with the parse tree. The AST_n classifier, applied to the two NSTs, should detect this inconsistency and provide the correct output. Figure 7 also highlights a critical problem the AST_n classifier has to deal with: as the two NSTs are perfectly identical, it is not possible to distinguish between them using only their fragments.

In order to engineer novel features, we simply add the boundary information provided by BC to the NSTs. We mark with a progressive number the phrase type corresponding to an argument node, starting from the leftmost argument. We call the resulting structure an ordinal predicate–argument spanning tree (AST_n^{ord}). For example, in the first NST of Figure 7c, we mark as `NP-0` and `NP-1` the first and second argument nodes, whereas in the second NST, we have a hypothesis of three arguments on three nodes that we transform as `NP-0`, `NP-1`, and `PP-2`.

This simple modification enables the tree kernel to generate features useful for distinguishing between two identical parse trees associated with different argument structures. For example, for the first NST the fragments `[NP-1 [NP PP]]`, `[NP [DT NN]]`, and `[PP [IN NP]]` are generated. They no longer match with the fragments of the second NST `[NP-0 [NP PP]]`, `[NP-1 [DT NN]]`, and `[PP-2 [IN NP]]`.

We also experimented with another structure, the marked predicate–argument spanning tree (AST_n^m), in which each argument node is marked with a role label assigned by a role multi-classifier (RM). Of course, this model requires a RM to classify all the nodes recognized by BC first. An example AST_n^m is shown in Figure 7d.

5.3 Structures for Proposition Re-Ranking

In Section 4.4, we presented our re-ranking mechanism, which is inspired by the joint inference model described in Haghighi, Toutanova, and Manning (2005). Designing structured features for the re-ranking classifier is complex in many aspects. Unlike the other structures that we have discussed so far, the defined mappings should: (1) preserve as much information as possible about the whole predicate–argument structure; (2) focus the learning algorithm on the whole structure; and (3) be able to identify those small differences that distinguish more or less accurate labeling schemes. Among the possible solutions that we have explored, three are especially interesting in terms of accuracy improvement or linguistic properties, and are described hereinafter.

The AST_n^{cm} (*completely marked* AST_n, see Figure 8a) is an AST_n in which each argument node label is enriched with the role assigned to the node by RM. The labels of the descendants of each argument node are modified accordingly, down to pre-terminal nodes. The AST_n^{cmt} is a variant of AST_n^{cm} in which only the target is marked. Marking a node descendant is meant to force substructures matching only among homogeneous argument types. This representation should provide rich syntactic and lexical information about the parse tree encoding the predicate–argument structure.

The PAS (*predicate–argument structure*, see Figure 8b) is a completely different structure that preserves the parse subtrees associated with each argument node while discarding the intra-argument syntactic parse information. Indeed, the syntactic links between the argument nodes are represented as a dummy 1-level tree, which appears in any PAS and therefore does not influence the evaluation of similarity between pairs

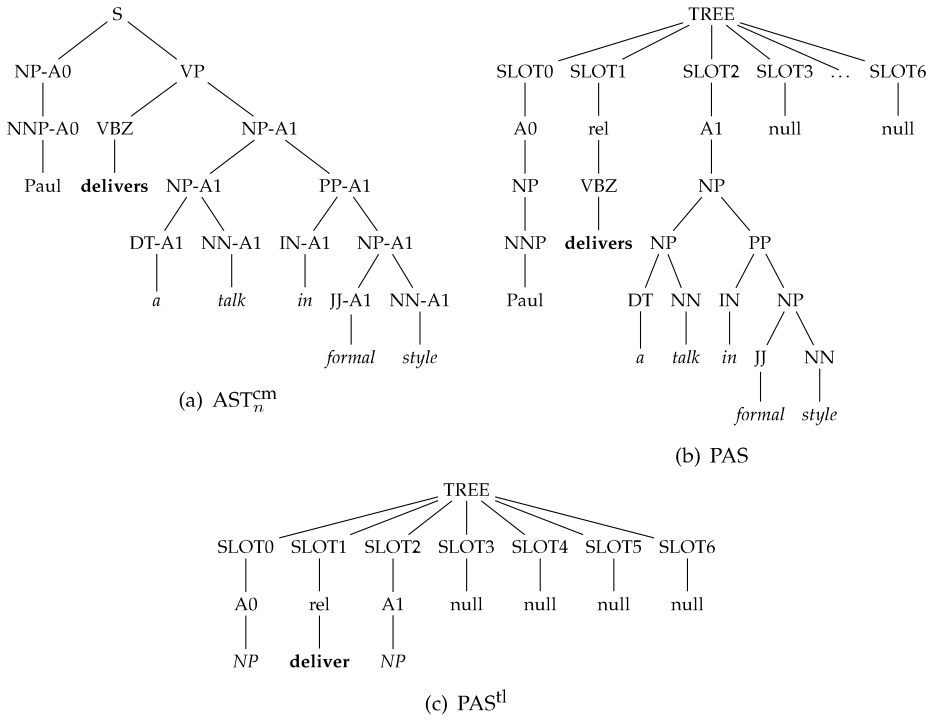

Figure 8
Different representations of the same proposition.

of structures. This structure accommodates the predicate and all the arguments of an annotation in a sequence of seven slots.[5] To each slot is attached an argument label to which in turn is attached the subtree rooted in the argument node. The predicate is represented by means of a pre-terminal node labeled *rel* to which the lemmatization of the predicate word is attached as a leaf node. In general, a proposition consists of m arguments, with $m \leq 6$, where m varies according to the predicate and the context. To guarantee that predicate structures with a different number of arguments are matched in the SST kernel function, we attach a dummy descendant marked *null* to the slots not filled by an argument.

The PAS^{tl} (*type-only, lemmatized PAS*, see Figure 8c) is a specialization of the PAS that only focuses on the syntax of the predicate–argument structure, namely, the type and relative position of each argument, minimizing the amount of lexical and syntactic information derived from the parse tree. The differences with the PAS are that: (1) each slot is attached to a pre-terminal node representing the argument type and a terminal node whose label indicates the syntactic type of the argument; and (2) the predicate word is lemmatized.

The next section presents the experiments used to evaluate the effectiveness of the proposed canonical structures in SRL.

5 We assume that predicate–argument structures cannot be composed by more than six arguments, which is generally true.

6. Experiments

The experiments aim to measure the contribution and the effectiveness of our proposed kernel engineering models and of the diverse structured features that we designed (Section 5). From this perspective, the role of feature extraction functions is not fundamental because the study carried out in Moschitti (2006a) strongly suggests that the SST (Collins and Duffy 2002) kernel produces higher accuracy than the PT kernel when dealing with constituent parse trees, which are adopted in our study.[6] We then selected the SST kernel and designed the following experiments:

(a) A study of canonical functions based on node marking for boundary detection and argument classification, that is, AST_1^m (Section 6.2). Moreover, as the standard features have shown to be effective, we combined them with AST_1^m based kernels on the boundary detection and classification tasks (Section 6.2).

(b) We varied the amount of training data to demonstrate the higher generalization ability of tree kernels (Section 6.3).

(c) Given the promising results of kernel engineering, we also applied it to solve a more complex task, namely, conflict resolution in SRL annotations (see Section 6.4). As this involves the complete predicate–argument structure, we could test advanced canonical functions generating AST_n, AST_n^{ord}, and AST_n^m.

(d) Previous work has shown that re-ranking is very important in boosting the accuracy of SRL. Therefore, we tested advanced canonical mappings, that is, those based on AST_n^{cm}, PAS, and PAS^{tl}, on such tasks (Section 6.5).

6.1 General Setup

The empirical evaluations were mostly carried out within the setting defined in the CoNLL 2005 shared task (Carreras and Màrquez 2005). As a target data set, we used the PropBank[7] and the automatic Charniak parse trees of the sentences of Penn TreeBank 2 corpus[8] (Marcus, Santorini, and Marcinkiewicz 1993) from the CoNLL 2005 shared-task data.[9] We employed the SVM-light-TK software[10], which encodes fast tree kernel evaluation (Moschitti 2006b), and combinations between multiple feature vectors and trees in the SVM-light software (Joachims 1999). We used the default regularization parameter (option `-c`) and $\lambda = 0.4$ (see Moschitti [2004]).

6.2 Testing Canonical Functions Based on Node Marking

In these experiments, we measured the impact of node marking strategies on boundary detection (BD) and the complete SRL task, that is, BD and role classification (RC). We employed a configuration of the architecture described in Section 4 and previously

6 Of course the PT kernel may be much more accurate in processing PAS and PAS^{tl} because these are not simply constituent parse trees. Nevertheless, a study of the PT kernel potential is beyond the purpose of this article.

7 http://www.cis.upenn.edu/∼ace.

8 http://www.cis.upenn.edu/∼treebank.

9 http://www.lsi.upc.edu/∼srlconll/.

10 http://ai-nlp.info.uniroma2.it/moschitti/.

Table 2
Number of arguments (Arguments) and of unrecoverable arguments (Unrecoverable) due to parse tree errors in Sections 2, 3, and 24 of the Penn TreeBank/PropBank.

Sec.	Arguments	Unrecoverable
2	198,373	454 (0.23%)
3	147,193	347 (0.24%)
24	139,454	731 (0.52%)

Table 3
Comparison between different models on Boundary Detection and the complete Semantic Role Labeling tasks. The training set is constituted by the first 1 million instances from Sections 02–06 for the boundary classifier and all arguments from Sections 02–21 for the role multiclassifier (253,129 instances). The performance is measured against Section 24 (149,140 instances).

	Boundary Detection			Semantic Role Labeling		
Kernels	P	R	F1	P	R	F1
AST_1	75.75%	71.68%	73.66	64.71%	61.71%	63.17
AST_1^m	77.32%	74.80%	76.04	66.58%	64.87%	65.71
Poly	82.18%	79.19%	80.66	75.86%	72.60%	73.81
Poly+AST_1	81.74%	80.71%	81.22	74.23%	73.62%	73.92
Poly+AST_1^m	81.64%	80.73%	81.18	74.36%	73.87%	74.11

adopted in Moschitti et al. (2005b), in which the simple conflict resolution heuristic is applied. The results were derived within the CoNLL setting by means of the related evaluator.

In more detail, in the BD experiments, we used the first million instances from the Penn TreeBank Sections 2–6 for training[11] and Section 24 for testing. Our classification model applied to this data replicates the results obtained in the CoNLL 2005 shared task, that is, the highest accuracy in BD among the systems using only one parse tree and one learning algorithm. For the complete SRL task, we used the previous BC and all the available data, that is, the sections from 2 to 21, for training the role multiclassifier.

It is worth mentioning that, as the automatic parse trees contain errors, some arguments cannot be associated with any covering node; thus we cannot extract a tree representation for them. In particular, Table 2 shows the number of arguments (column 2) for sections 2, 3, and 24 as well as the number of arguments that we could not take into account (Unrecoverable) due to the lack of parse tree nodes exactly covering their word spans. Note how Section 24 of the Penn TreeBank (which is not part of the Charniak training set) is much more affected by this problem.

Given this setting, the impact of node marking can be measured by comparing the AST_1 and the AST_1^m based kernels. The results are reported in the rows AST_1 and AST_1^m of Table 3. Columns 2, 3, and 4 show their Precision, Recall, and F1 measure on BD and columns 5, 6, and 7 report the performance on SRL. We note that marking the argument

11 This was the most expensive process in terms of training time, requiring more than one week.

node simplifies the generalization process as it improves both tasks by about 3.5 and 2.5 absolute percentage points, respectively.

However, Row Poly shows that the polynomial kernel using state-of-the-art features (Moschitti et al. 2005b) outperforms AST_1^m by about 4.5 percentage points in BD and 8 points in the SRL task. The main reason is that the employed tree structures do not explicitly encode very important features like the passive voice or predicate position. In Moschitti (2004), these are shown to be very effective especially when used in polynomial kernels. Of course, it is possible to engineer trees including these and other standard features with a canonical mapping, but the aim here is to provide new interesting representations rather than to abide by the simple exercise of representing already designed features within tree kernel functions. In other words, we follow the idea presented in Moschitti (2004), where tree kernels were suggested as a means to derive new features rather than generate a stand-alone feature set.

Rows Poly+AST_1 and Poly+AST_1^m investigate this possibility by presenting the combination of polynomial and tree kernels. Unfortunately, the results on both BD and SRL do not show enough improvement to justify the use of tree kernels; for example, Poly+AST_1^m improves Poly by only 0.52 in BD and 0.3 in SRL. The small improvement is intuitively due to the use of (1) a state-of-the-art model as a baseline and (2) a very large amount of training data which decreases the contribution of tree features. In the next section an analysis in terms of training data will shed some light on the role of tree kernels for BD and RC in SRL.

6.3 The Role of Tree Kernels for Boundary Detection and Argument Classification

The previous section has shown that if a state-of-the-art model[12] is adopted, then the tree kernel contribution is marginal. On the contrary, if a non state-of-the-art model is adopted tree kernels can play a significant role. To verify this hypothesis, we tested the polynomial kernel over the standard feature vector proposed in Gildea and Jurafsky (2002) obtaining an F1 of 67.3, which is comparable with the AST_1^m model, that is 65.71. Moreover, a kernel combination produced a significant improvement of both models reaching an F1 of 70.4.

Thus, the role of tree kernels relates to the design of features for novel linguistic tasks for which the optimal data representation has not yet been developed. For example, although SRL has been studied for many years and many effective features have been designed, representations for languages like Arabic are still not very well understood and raise challenges in the design of effective predicate–argument descriptions.

However, this hypothesis on the usefulness of tree kernels is not completely satisfactory as the huge feature space produced by them should play a more important role in predicate–argument representation. For example, the many fragments extracted by an AST_1 provide a very promising back-off model for the Path feature, which should improve the generalization process of SVMs.

As back-off models show their advantages when the amount of training data is small, we experimented with Poly, AST_1, AST_1^m, Poly+AST_1, and Poly+AST_1^m and

12 The adopted model is the same as used in Moschitti et al. (2005b), which is the most accurate among the systems that use a single learning model, a single source of syntactic information, and no accurate inference mechanism. If tree kernels improved this basic model they would likely improve the accuracy of more complex systems as well.

different bins of training data, starting from a very small set, namely, 10,000 instances (1%) to 1 million (100%) of instances. The results from the BD classifiers and the complete SRL task are very interesting and are illustrated by Figure 9. We note several things.

First, Figure 9a shows that with only 1% of data (i.e., 640 arguments) as positive examples, the F1 on BD of the AST_1^m kernel is surprisingly about 3 percentage points higher than the one obtained by the polynomial kernel (Poly) (i. e., the state of the art). When AST_1^m is combined with Poly the improvement reaches 5 absolute percentage points. This suggests that tree kernels should always be used when small training data sets are available.

Second, although the performance of AST_1 is much lower than all the other models, its combination with Poly produces results similar to Poly+AST_1^m, especially when the amount of training data increases. This, in agreement with the back-off property, indicates that the number of tree fragments is more relevant than their quality.

Third, Figure 9b shows that as we increase training data, the advantage of using tree kernels decreases. This is rather intuitive as (i) in general less accurate data machine learning models trained with enough data can reach the accuracy of the most accurate models, and (ii) if the hypothesis that tree kernels provide back-off models is true, a lot of training data makes them less critical, for example, the probability of finding the Path feature of a test instance in the training set becomes high.

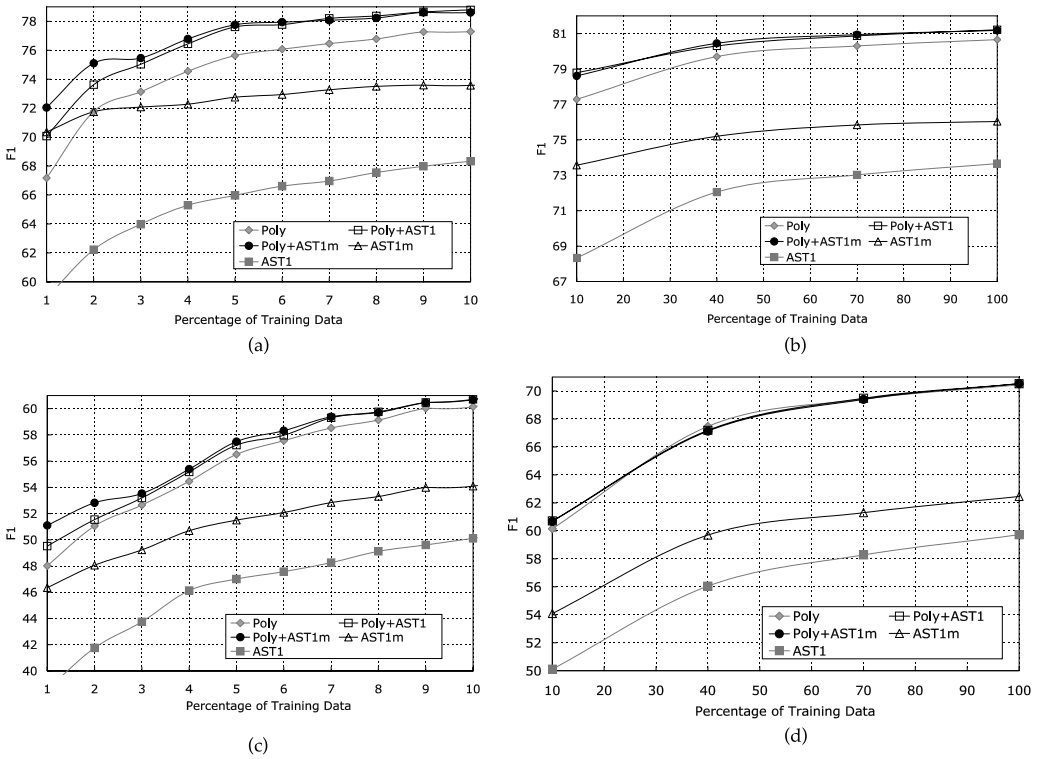

Figure 9
Learning curves for BD (a and b) and the SRL task (c and d), where 100% of data corresponds to 1 million candidate argument nodes for boundary detection and 64,000 argument nodes for role classification.

Table 4
Boundary detection accuracy (F1) on gold-standard parse trees and ambiguous structures employing the different conflict resolution methodologies described in Section 4.3.

RND	HEU	AST_n^{ord}
73.13	71.50	91.11

Finally, Figures 9c and 9d show learning curves[13] similar to Figures 9a and 9b, but with a reduced impact of tree kernels on the Poly model. This is due to the reduced impact of AST_1^m on role classification. Such findings are in agreement with the results in Moschitti (2004), which show that for argument classification the SCF structure (a variant of the AST_n^m) is more effective. Thus a comparison between learning curves of Poly and SCF on RC may show a behavior similar to Poly and AST_1^m for BD.

6.4 Conflict Resolution Results

In these experiments, we are interested in (1) the evaluation of the accuracy of our tree kernel–based conflict resolution strategy and (2) studying the most appropriate structured features for the task.

A first evaluation was carried out over gold-standard Penn TreeBank parses and PropBank annotations. We compared the alternative conflict resolution strategies implemented by our architecture (see Section 4.3), namely the random (RND), the heuristic (HEU), and a tree kernel–based disambiguator working with AST_n^{ord} structures. The disambiguators were run on the output of BC, that is, without any information about the candidate arguments' roles. BC was trained on Sections 2 to 7 with a high-recall linear kernel. We applied it to classify Sections 8 to 21 and obtained 2,988 NSTs containing at least one overlapping node. These structures generated 3,624 positive NSTs (i. e., correct structures) and 4,461 negative NSTs (incorrect structures) in which no overlap is present. We used them to train the AST_n^{ord} classifier. The F1 measure on the boundary detection task was evaluated on the 385 overlapping annotations of Section 23, consisting of 642 argument and 15,408 non-argument nodes.

The outcome of this experiment is summarized in Table 4. We note two points. (1) The RND disambiguator (slightly) outperforms the HEU. This suggests that the heuristics that we implemented were inappropriate for solving the problem. It also underlines how difficult it is to explicitly choose the aspects that are relevant for a complex, non-local task such as overlap resolution. (2) The AST_n^{ord} classifier outperforms the other strategies by about 20 percentage points, that is, 91.11 vs. 73.13 and 71.50. This datum along with the previous one is a good demonstration of how tree kernels can be effectively exploited to describe phenomena whose relevant features are largely unknown or difficult to represent explicitly. It should be noted that a more accurate baseline can be provided by using the Viterbi-style search (see Section 4.4.1). However, the experiments in Section 6.5 show that the heuristics produce the same accuracy (at least when the complete task is carried out).

13 Note that using all training data, all the models reach lower F1s than the respective values shown in Table 3. This happens because the data for training the role multiclassifier is restricted to the first million instances, in other words, about 64,000 out of the total 253,129 arguments.

Table 5
SRL accuracy on different PropBank target sections in terms of F1 measure of the different structured features employed for conflict resolution.

Target section	AST_n	AST_n^{ord}	AST_n^{m}
21	73.7	77.3	78.7
23	68.9	71.2	72.1

These experiments suggest that tree kernels are promising methods for resolving annotation conflicts; thus, we tried to also select the most representative structured features (i. e., AST_n, AST_n^{ord}, or AST_n^{m}) when automatic parse trees are used. We trained BC on Sections 2–8, whereas, to achieve a very accurate argument classifier, we trained a role multi-classifier (RM) on Sections 2–21. Then, we trained the AST_n, AST_n^{ord}, and AST_n^{m} classifiers on the output of BC. To test BC, RM, and the tree kernel classifiers, we ran two evaluations on Section 23 and Section 21.[14]

Table 5 shows the F1 measure for the different tree kernels (columns 2, 3, and 4) for conflict resolution over the NSTs of Sections 21 and 23. Several points should be noted.

(1) The general performance is much lower than that achieved on gold-standard trees, as shown in Table 4. This datum and the gap of about 6 percentage points between Sections 21 and 23 confirm the impact of parsing accuracy on the subtasks of the SRL process.

(2) The ordinal numbering of arguments (AST_n^{ord}) and the role type information (AST_n^{m}) provide tree kernels with more meaningful fragments because they improve the basic model by about 4 percentage points.

(3) The deeper semantic information generated by the argument labels provides useful clues for selecting correct predicate–argument structures because the AST_n^{m} model improves AST_n^{ord} performance on both sections.

6.5 Proposition Re-Ranking Results

In these experiments, Section 23 was used for testing our proposition re-ranking. We employed a BC trained on Sections 2 to 8, whereas RM was trained on Sections 2–12.[15] In order to provide a probabilistic interpretation of the SVM output (see Section 4.4.1), we evaluated each classifier distribution parameter based on its output on Section 12. For computational complexity reasons, we decided to consider the five most likely labelings for each node and the five first alternatives output by the Viterbi algorithm (i. e., $m = 5$ and $n = 5$).

With this set-up, we evaluated the accuracy lower and upper bounds of our system. As our baseline, we consider the accuracy of a re-ranker that always chooses the first alternative output from the Viterbi algorithm, that is, the most likely according to the joint inference model. This accuracy has been measured as 75.91 F1 percentage points; this is practically identical to the 75.89 obtained by applying heuristics to remove overlaps generated by BC.

14 As Section 21 of the Penn TreeBank is part of the Charniak parser training set, the performance derived on its parse trees represents an upper bound for our classifiers, i. e., the results using a nearly ideal syntactic parser and role multiclassifier.

15 In these experiments we did not use tree kernels for BC and RM as we wanted to measure the impact of tree kernels only on the re-ranking stage.

This does not depend on the bad quality of the five top labelings. Indeed, we selected the best alternative produced by the Viterbi algorithm according to the gold-standard score, and we obtained an F1 of 84.76 for $n = 5$. Thus, the critical aspect resides in the selection of the best annotations, which should be carried out by an automatic re-ranker.

Rows 2 and 3 of Table 6 show the number of distinct propositions and alternative annotations output by the Viterbi algorithm for each of the employed sections. In row 3, the number of pair comparisons (i. e., the number of training/test examples for the classifier) is shown.

Using this data, we carried out a complete SRL experiment, which is summarized in Table 7. First, we compared the accuracy of the AST_n^{cm}, PAS, and PAS^{tl}classifiers trained on Section 24 (in row 3, columns 2, 3, and 4) and discovered that the latter structure produces a noticeable F1 improvement, namely, 78.15 vs. 76.47 and 76.77, whereas the accuracy gap between the PAS and the AST_n^{cm} classifiers is very small, namely, 76.77 vs. 76.47 percentage points. We selected the most interesting structured feature, that is, the PAS^{tl}, and extended it with the local (to each argument node) standard features commonly employed for the boundary detection and argument classification tasks, as in Haghighi, Toutanova, and Manning (2005). This richer kernel (PAS^{tl}+STD, column 5) was compared with the PAS^{tl} one. The comparison was performed on two different training sets (rows 2 and 3): In both cases, the introduction of the standard features produced a performance decrement, most notably in the case of Section 12 (i. e., 82.07 vs. 75.06). Our best re-ranking kernel (i. e., the PAS^{tl}) was then employed in a larger experiment, using both Sections 12 and 24 for testing (row 4), achieving an F_1 measure of 78.44.

First, we note that the accuracy of the AST_n^{cm} and PAS classifiers is very similar (i. e., 76.77 vs. 76.47). This datum suggests that the intra-argument syntactic information is not critical for the re-ranking task, as including it or not in the learning algorithm does not lead to noticeable differences.

Second, we note that the PAS^{tl} kernel is much more effective than those based on AST_n^{cm} and PAS, which are always outperformed. This may be due to the fact that

Table 6
Number of propositions, alternative annotations (as output by the Viterbi algorithm), and pair comparisons (i. e., re-ranker input examples) for the PropBank sections used for the experiments.

	Section 12	Section 23	Section 24
Propositions	4,899	5,267	3,248
Alternatives	24,494	26,325	16,240
Comparisons	74,650	81,162	48,582

Table 7
Summary of the proposition re-ranking experiments with different training sets.

Training Section	AST_n^{cm}	PAS	PAS^{tl}	PAS^{tl}+STD
12	–	–	78.27	77.61
24	76.47	76.77	78.15	77.77
12+24	–	–	78.44	–

two AST_n^{cm}s (or PASs) always share a large number of substructures, because most alternative annotations tend to be very similar and the small differences among them only affect a small part of the encoding of syntactic information; on the other hand, the small amount of local parsing information encoded in the PAS^{tl}s enables a good generalization process.

Finally, the introduction of the standard, local standard features in our re-ranking model caused a performance loss of about 0.5 percentage points on both Sections 12 and 24. This fact, which is in contrast with what has been shown in Haghighi, Toutanova, and Manning (2005), might be the consequence of the small training sets that we employed. Indeed, local standard features tend to be very sparse and their effectiveness should be evaluated against a larger data set.

7. Discussions and Conclusions

The design of automatic systems for the labeling of semantic roles requires the solution of complex problems. Among other issues, feature engineering is made difficult by the structured nature of the data, that is, features should represent information expressed by automatically generated parse trees. This raises two main problems: (1) the modeling of effective features, partially solved for some subtasks in previous works, and (2) the implementation of the software for the extraction of a large number of such features.

A system completely (or largely) based on tree kernels alleviates both problems as (1) kernel functions automatically generate features and (2) only a procedure for the extraction of subtrees is needed. Although some of the manually designed features seem to be superior to those derived with tree kernels, their combination still seems worth applying. Moreover, tree kernels provide a back-off model that greatly outperforms state-of-the-art SRL models when the amount of training data is small.

To demonstrate these points, we carried out a comprehensive study of the use of tree kernels for semantic role labeling by designing several canonical mappings. These correspond to the application of innovative tree kernel engineering techniques tailored to different stages of an SRL process. The experiments with these methods and SVMs on the data set provided by the CoNLL 2005 shared task (Carreras and Màrquez 2005) show that, first, tree kernels are a valid support to manually designed features for many stages of the SRL process. We have shown that our improved tree kernel (i.e., the one based on AST_1^m) highly improves accuracy in both boundary detection and the SRL task when the amount of training data is small (e.g., 5 absolute percentage points over a state-of-the-art boundary classifier). In the case of argument classification the improvement is less evident but still consistent, at about 3%.

Second, appropriately engineered tree kernels can replace standard features in many SRL subtasks. For example, in complex tasks such as conflict resolution or re-ranking, they provide an easy way to build new features that would be difficult to describe explicitly. More generally, tree kernels can be used to combine different sources of information for the design of complex learning models.

Third, in the specific re-ranking task, our structured features show a noticeable improvement over our baseline (i. e., about 2.5 percentage points). This could be increased considering that we have not been able to fully exploit the potential of our re-ranking model, whose theoretical upper bound is 6 percentage points away. Still, although we only used a small fraction of the available training data (i. e., only 2 sections out of 22 were used to train the re-ranker) our system's accuracy is in line with state-of-the-art systems (Carreras and Màrquez 2005) that do not employ tree kernels.

Finally, although the study carried out in this article is quite comprehensive, several issues should be considered in more depth in the future:

(a) The tree feature extraction functions ST, SST, and PT should be studied in combination with the proposed canonical mappings. For example, as the PT kernel seems more suitable for the processing of dependency information, it would be interesting to apply it in an architecture using these kinds of syntactic parse trees (e. g., Chen and Rambow 2003). In particular, the combination of different extraction functions on different syntactic views may lead to very good results.

(b) Once the set of the most promising kernels is established, it would be interesting to use all the available CoNLL 2005 data. This would allow us to estimate the potential of our approach by comparing it with previous work on a fairer basis.

(c) The use of fast tree kernels (Moschitti 2006a) along with the proposed tree representations makes the learning and classification much faster, so that the overall running time is comparable with polynomial kernels. However, when used with SVMs their running time on very large data sets (e. g., millions of instances) becomes prohibitive. Exploiting tree kernel–derived features in a more efficient way (e. g., by selecting the most relevant fragments and using them in an explicit space) is thus an interesting line of future research. Note that such fragments would be the product of a reverse engineering process useful to derive linguistic insights on semantic role theory.

(d) As CoNLL 2005 (Punyakanok et al. 2005) has shown that multiple parse trees provide the most important boost to the accuracy of SRL systems, we would like to extend our model to work with multiple syntactic views of each input sentence.

Acknowledgments

This article is the result of research on kernel methods for Semantic Role Labeling which started in 2003 and went under the review of several program committees of different scientific communities, from which it highly benefitted. In this respect, we would like to thank the reviewers of the SRL special issue as well as those of the ACL, CoNLL, EACL, ECAI, ECML, HLT-NAACL, and ICML conferences. We are indebted to Silvia Quarteroni for her help in reviewing the English formulation of an earlier version of this article.

References

Baker, Collin F., Charles J. Fillmore, and John B. Lowe. 1998. The Berkeley FrameNet project. In *COLING-ACL '98: Proceedings of the Conference*, pages 86–90, Montréal, Canada.

Carreras, Xavier and Lluís Màrquez. 2004. Introduction to the CoNLL-2004 shared task: Semantic role labeling. In *HLT-NAACL 2004 Workshop: Eighth Conference on Computational Natural Language Learning (CoNLL-2004)*, pages 89–97, Boston, MA.

Carreras, Xavier and Lluís 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.

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*, pages 41–48, Sapporo, Japan.

Collins, Michael and Nigel Duffy. 2002. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *ACL02*, pages 263–270, Philadelphia, PA.

Culotta, Aron and Jeffrey Sorensen. 2004. Dependency tree kernels for relation extraction. In *ACL04*, pages 423–429, Barcelona, Spain.

Cumby, Chad and Dan Roth. 2003. Kernel methods for relational learning. In *Proceedings of ICML 2003*, pages 107–114, Washington, DC.

Fillmore, Charles J. 1968. The case for case. In Emmon Bach and Robert T. Harms,

editors, *Universals in Linguistic Theory*. Holt, Rinehart, and Winston, New York, pages 1–210.
Gildea, Daniel and Daniel Jurafsky. 2002. Automatic labeling of semantic roles. *Computational Linguistics*, 28(3): 245–288.
Haghighi, Aria, Kristina Toutanova, and Christopher 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.
Jackendoff, Ray. 1990. *Semantic Structures, Current Studies in Linguistics Series*. The MIT Press, Cambridge, MA.
Joachims, Thorsten. 1999. Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods—Support Vector Learning*. MIT Press, Cambridge, MA, pages 169–184.
Kazama, Jun'ichi and Kentaro Torisawa. 2005. Speeding up training with tree kernels for node relation labeling. In *Proceedings of EMNLP 2005*, pages 137–144, Toronto, Canada.
Kudo, Taku and Yuji Matsumoto. 2003. Fast methods for kernel-based text analysis. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 24–31, Sapporo, Japan.
Levin, Beth. 1993. *English Verb Classes and Alternations*. The University of Chicago Press, Chicago, IL.
Lin, H.-T., C.-J. Lin, and R. C. Weng. 2003. A note on Platt's probabilistic outputs for support vector machines. Technical report, National Taiwan University.
Litkowski, Kenneth. 2004. Senseval-3 task: Automatic labeling of semantic roles. In *Senseval-3: Third International Workshop on the Evaluation of Systems for the Semantic Analysis of Text*, pages 9–12, Barcelona, Spain.
Marcus, M. P., B. Santorini, and M. A. Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, 19:313–330.
Moschitti, Alessandro. 2004. A study on convolution kernels for shallow semantic parsing. In *Proceedings of the 42th Conference on Association for Computational Linguistic (ACL-2004)*, pages 335–342, Barcelona, Spain.
Moschitti, Alessandro. 2006a. Efficient convolution kernels for dependency and constituent syntactic trees. In *Proceedings of The 17th European Conference on Machine Learning*, pages 318–329, Berlin, Germany.
Moschitti, Alessandro. 2006b. Making tree kernels practical for natural language learning. In *Proceedings of 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL2006)*, pages 113–120, Treato, Italy.
Moschitti, Alessandro, Bonaventura Coppola, Daniele Pighin, and Roberto Basili. 2005a. Engineering of syntactic features for shallow semantic parsing. In *Proceedings of the ACL Workshop on Feature Engineering for Machine Learning in Natural Language Processing*, pages 48–56, Ann Arbor, MI.
Moschitti, Alessandro, Ana-Maria Giuglea, Bonaventura Coppola, and Roberto Basili. 2005b. Hierarchical semantic role labeling. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 201–204, Ann Arbor, MI.
Moschitti, Alessandro, Daniele Pighin, and Roberto Basili. 2006. Tree kernel engineering in semantic role labeling systems. In *Proceedings of the Workshop on Learning Structured Information in Natural Language Applications, EACL 2006*, pages 49–56, Trento, Italy.
Palmer, Martha, Daniel Gildea, and Paul Kingsbury. 2005. The Proposition Bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1): 71–106.
Platt, J. 1999. Probabilistic outputs for support vector machines and comparison to regularized likelihood methods. In A. J. Smola, P. Bartlett, B. Schoelkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*. MIT Press, Cambridge, MA, pages 61–74.
Pradhan, Sameer, Kadri Hacioglu, Valerie Krugler, Wayne Ward, James H. Martin, and Daniel Jurafsky. 2005a. Support vector learning for semantic argument classification. *Machine Learning*, 60(1–3):11–39.
Pradhan, Sameer, Kadri Hacioglu, Wayne Ward, James H. Martin, and Daniel Jurafsky. 2005b. 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 Martin, and Daniel Jurafsky. 2005c. Semantic role labeling

using different syntactic views. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 581–588, Ann Arbor, MI.

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

Punyakanok, Vasin, Peter Koomen, 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.

Shawe-Taylor, John and Nello Cristianini. 2004. *Kernel Methods for Pattern Analysis*. Cambridge University Press, Cambridge, UK.

Shen, Libin, Anoop Sarkar, and Aravind K. Joshi. 2003. Using LTAG based features in parse reranking. In *Empirical Methods for Natural Language Processing (EMNLP)*, pages 89–96, Sapporo, Japan.

Thompson, Cynthia A., Roger Levy, and Christopher Manning. 2003. A generative model for semantic role labeling. In *14th European Conference on Machine Learning*, pages 397–408, Cavtat, Croatia.

Tjong Kim Sang, Erik, Sander Canisius, Antal van den Bosch, and Toine Bogers. 2005. Applying spelling error correction techniques for improving semantic role labelling. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 229–232, Ann Arbor, MI.

Toutanova, Kristina, Aria Haghighi, and Christopher Manning. 2005. Joint learning improves semantic role labeling. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 589–596, Ann Arbor, MI.

Toutanova, Kristina, Penka Markova, and Christopher Manning. 2004. The leaf path projection view of parse trees: Exploring string kernels for HPSG parse selection. In *Proceedings of EMNLP 2004*, pages 166–173, Barcelona, Spain.

Vapnik, Vladimir N. 1998. *Statistical Learning Theory*. John Wiley and Sons, New York.

Vishwanathan, S. V. N. and A. J. Smola. 2002. Fast kernels on strings and trees. In *Proceedings of Neural Information Processing Systems*, pages 569–576, Vancouver, British Columbia.

Xue, Nianwen and Martha Palmer. 2004. Calibrating features for semantic role labeling. In *Proceedings of EMNLP 2004*, pages 88–94, Barcelona, Spain.

Zelenko, D., C. Aone, and A. Richardella. 2003. Kernel methods for relation extraction. *Journal of Machine Learning Research*, 3:1083–1106.

Zhang, Min, Jie Zhang, and Jian Su. 2006. Exploring syntactic features for relation extraction using a convolution tree kernel. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 288–295, New York, NY.