Semantic Role Labeling Using a Grammar-Driven Convolution Tree Kernel

Min Zhang, Wanxiang Che, GuoDong Zhou, Aiti Aw, Chew Lim Tan, *Senior Member, IEEE*, Ting Liu, and Sheng Li

***Abstract*—Convolution tree kernel has shown promising results in semantic role labeling (SRL). However, this kernel does not consider much linguistic knowledge in kernel design and only performs hard matching between subtrees. To overcome these constraints, this paper proposes a grammar-driven convolution tree kernel for SRL by introducing more linguistic knowledge. Compared with the standard convolution tree kernel, the proposed grammar-driven kernel has two advantages: 1) grammar-driven approximate substructure matching, and 2) grammar-driven approximate tree node matching. The two approximate matching mechanisms enable the proposed kernel to better explore linguistically motivated structured knowledge. Experiments on the CoNLL-2005 SRL shared task and the PropBank I corpus show that the proposed kernel outperforms the standard convolution tree kernel significantly. Moreover, we present a composite kernel to integrate a feature-based polynomial kernel and the proposed grammar-driven convolution tree kernel for SRL. Experimental results show that our composite kernel-based method significantly outperforms the previously best-reported ones.**

***Index Terms*—Dynamic programming, grammar-driven convolution tree kernel, natural languages, semantic role labeling.**

I. Introduction

SEMANTIC parsing maps a natural language sentence into a formal representation of its meaning. Due to the difficulty in deep semantic parsing, previous work mainly focuses on *shallow semantic parsing*, which assigns a simple structure (such as WHO did WHAT to WHOM, WHEN, WHERE, WHY, HOW, etc.) to each predicate in a sentence. As a particular case of *shallow semantic parsing*, the well-defined *semantic role labeling* (SRL) issue has been drawing more and more attention in recent years due to its importance in deep natural language processing (NLP) applications, such as question answering [1], information extraction [2], and coreference resolution [3]. Given a sentence and each predicate (either a target verb or a noun), SRL recognizes and maps all the constituents in the sentence into their corresponding semantic arguments (roles) of the predicate or *non*-argument. In PropBank context [4], these semantic arguments include core arguments (e.g., A0 for *Agent* and A1 for *Patient*) and adjuncts (e.g., AM-LOC for *Locative* and AM-TMP for *Temporal*). Fig. 1 shows an example from the PropBank I corpus on SRL [4]. Given the sentence "She bought the silk in China." and the predicate "bought," a SRL system needs to label the sentence as "[$_{\text{Arg0}}$ She] bought [$_{\text{Arg1}}$ the silk] [$_{\text{ArgM-LOC}}$ in China].".

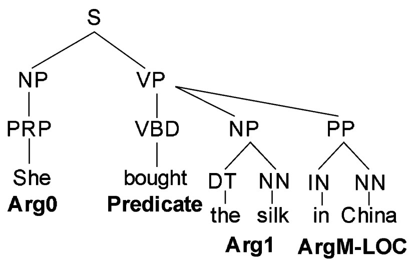

Fig. 1. Semantic role labeling in a syntactic tree representation.

Generally, SRL takes a parse tree as input and determines proper labels for semantic arguments of a given predicate. Given a sentence and its parse tree, all predicates in the sentence can be recognized with simple heuristic rules in practice [5], [6], and for each recognized predicate, their semantic arguments are identified and labeled independently. From learning algorithm viewpoint, SRL can be recast as a classification problem and done in two steps: semantic role identification, which identifies each syntactic constituent in a sentence as either a semantic argument of a given predicate or not, and semantic role classification, which classifies each identified semantic argument into a specific semantic role. With the availability of large annotated corpora [4], [7], data-driven techniques, including both feature-based and kernel-based methods, have been extensively studied for SRL [5], [6]. A feature-based method maps a single predicate-argument structure[1] to a flat feature vector and computes the similarity between two feature vectors (i.e.,

[1]A predicate-argument structure is part of a parse tree related with the predicate and the argument, including the predicate word, the argument phrase structure and the path that links the predicate and the argument.

Manuscript received June 29, 2007; revised November 23, 2007. Published August 13, 2008 (projected). This work was supported by the Science and Engineering Research Council (SERC) under the Agency for Science, Technology, and Research (A*STAR), Singapore. The work of W. Che, T. Liu, and S. Li was supported in part by the National Natural Science Foundation of China under Project 60736044. The work of G. Zhou was supported in part by the the National Natural Science Foundation of China under Project 60673041 and the "863" National High-Tech Research and Development of China Project under Project 2006AA01Z147. This paper is a much extended version of Zhang *et al.* (2007) at ACL'07. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Mari Ostendorf.

M. Zhang and A. Aw are with the Institute for Infocomm Research, Singapore 119613 (e-mail: mzhang@i2r.a-star.edu.sg; aaiti@i2r.a-star.edu.sg).

W. Che, T. Liu, and S. Li are with the Harbin Institute of Technology, Harbin 150001, China (e-mail: car@ir.hit.edu.cn; tliu@ir.hit.edu.cn; lisheng@hit.edu.cn).

G. Zhou is with the School of Computer Science and Engineering, Soochow University, Suzhou 215006, China (e-mail: gdzhou@suda.edu.cn).

C. L. Tan is with the National University of Singapore, Singapore 119077 (e-mail: tancl@comp.nus.edu.sg).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TASL.2008.2001104

two predicate-argument structures) using dot product while, in kernel-based methods, a kernel function is used to directly measure the similarity between two predicate-argument structures without using the feature vector representations (The kernel trick is introduced in detail in Section IV-A). Although feature-based methods represent the state-of-the-art and have achieved much success in SRL, kernel methods have more potential in capturing structured knowledge than feature-based methods. Moreover, it is well known that syntactic structure knowledge in a parse tree plays an important role in SRL and is thereby worth exploring [8], [9]. Besides SRL, tree kernels have been applied to much more natural language tasks, including syntactic parsing reranking [10], relation extraction [11], [12], named entity recognition [13], [14], etc.

In the literature, various tree kernels have been proposed, such as the convolution tree kernel [15], the PT kernel [16], the labeled order tree kernel [17], [18], the path-sensitive parse tree kernel [11], the path-sensitive dependency tree kernel [14], the shortest path kernel over dependency trees [19], [20], and the LTAG-based tree kernel [21]. As a representative tree kernel, the convolution tree kernel implicitly takes subtrees as its features and counts the number of common subtrees as the similarity between two predicate-argument structures. This kernel was first proposed in [15] for syntactic parsing and has achieved promising results in SRL [22], [23] and other NLP applications, such as parsing [15], relation extraction [12], and question classification in question answering [24]. However, the tree kernel only performs hard matching between two subtrees without considering linguistic knowledge. This makes the kernel fail to handle similar phrase structures (e.g., "buy a car" versus "buy a *red* car") and quasi-synonymic grammar tags (e.g., the POS variations between "*high*/JJ degree/NN" and "*higher*/JJR degree/NN").[2] In natural language, some rewrite rules are generalizations of others. For example, "NP $\rightarrow$ DT JJ NN" is a specialized version of "NP $\rightarrow$ DT NN." The same applies to POSs and other grammar tags. However, the standard convolution tree kernel is unable to capture such linguistic knowledge. To overcome these shortcomings, the PT kernel generalizes the standard convolution tree kernel by allowing partial matching between subtrees. Compared with the standard convolution tree kernel, the PT kernel generates much more substructures implicitly. However, evaluation on SRL [16] shows that the PT kernel performs worse than the standard convolution tree kernel. It is not surprising due to its nongrammar-driven nature, and hence allowing many nonlinguistically motivated (normally noisy) substructures to be generated.

This paper addresses the above-mentioned issues by directly incorporating linguistic knowledge into the convolution tree kernel. To the best of our knowledge, this is the first attempt to integrate linguistic knowledge into a tree kernel (i.e., the convolution tree kernel in this paper). In particular, we propose a grammar-driven convolution tree kernel to better explore linguistically motivated substructures. Experimental results show that the proposed kernel significantly outperforms the standard convolution tree kernel on the CoNLL-2005 SRL shared task [6]. Furthermore, we present a composite kernel to integrate a state-of-the-art feature-based polynomial kernel and our grammar-driven tree kernel. Experimental results show that our composite kernel-based method outperforms previously best reported ones.

The remainder of the paper is organized as follows: Section II introduces the public-available SRL corpora. In Section III, we review the previous work, while Section IV discusses our new grammar-driven convolution tree kernel. The experimental results are shown in Section V. Finally, we conclude our work and present our future work in Section VI.

II. Semantic Annotation and Corpora

There are two major kinds of corpora available for semantic argument annotation: FrameNet [7], which uses predicate specific labels (such as JUDGE and JUDGEE) in semantic annotation (called frames) and PropBank [4], [25], which uses predicate independent labels (such as A0, A1). FrameNet can be viewed as the application of frame semantics [26] in the annotation of predicate argument structures based on frame elements (semantic role), while PropBank is based on a linguistic model inspired by Levin's verb classes [27], focusing on the argument structure of verbs and on the alternation patterns that describe movements of verbal arguments within a predicate structure [28]. Another difference between them is that semantic arguments in PropBank are annotated consistently with their syntactic alternations, while FrameNet roles are assigned according to semantic considerations rather than syntactic aspects. In this paper, we will only consider PropBank due to its large scale, popularity, and compatibility with the Penn TreeBank corpus [29].

Currently, there are two PropBank style corpora in English, the PropBank I corpus (LDC2004T14) and the CoNLL 2005 data on the SRL shared task [6] with only two minor differences between them: 1) the PropBank I corpus considers trace nodes while the CoNLL 2005 data does not, and 2) the CoNLL 2005 data annotates discontinuous and coreferential arguments while the PropBank I corpus does not. In this paper, we will focus on the CoNLL 2005 data, with a few exceptions on the PropBank I corpus for a fair comparison with related work.

The semantic roles covered by the CoNLL 2005 data are classified into four clusters [6].

1) Numbered Core Arguments (A0–A5, AA): Arguments defining verb-specific roles, e.g., A0 (for the *agent*) and A1 (for the *patient* or the *theme* of the proposition). However, no consistent generalization can be made across different verbs or even different senses of the same verb for higher numbered arguments.
2) Adjuncts (AM-): General arguments that any verb may take optionally. There are 13 types of adjuncts, such as AM-ADV (for general-purpose adverbs), AM-TMP (for temporal).
3) References (R-): For arguments realized in other parts of the sentences.
4) Verbs (V): For the predicate of the proposition.

[2]Please refer to http://www.cis.upenn.edu/~treebank/ for the detailed definitions of the grammar tags used in the paper.

For details, please refer to [6, Sec. 3.1], [4, Sec. 3], and [30, Sec. 2].

III. Previous Work

Labeling of semantic roles is important for many NLP applications. Traditional understanding systems [31] relied on handcrafted grammars to realize semantic roles and thus had limited coverage. Due to the recent availability of large human-annotated corpora, data-driven techniques, including both feature-based and kernel-based methods, have been extensively studied in SRL. Normally, SRL is recast as a classification problem in the literature. That is, in training, the annotated semantic argument instances are used to learn a classifier while, in testing, the learned classifier is applied to input instances to determine their argument classes (including *non*-argument) and thus label each possible arguments.

A. Feature-Based Methods for SRL

Feature-based methods rely on a flat feature vector to represent an object. [32] extracted dozens of basic flat features (such as Predicate, Phrase Type, Parse Tree Path, Position, Voice, Head Word, and Verb Sub-Categorization) from a syntactic parse tree, clustered them into several subsets according to their linguistic nature and linearly interpolated the contribution of each feature subset to the SRL task in the FrameNet [7]. Since then, much research work has been pursued to explore more features (including Constituent-related features, Predicate-related features, and Predicate–Constituent features) by applying different learning algorithms and tagging strategies [30], [33]–[37]. As a representative, [30] made an extensive study on the feature-based SRL using a support vector machine (SVM) classifier [38]. For the interested, please refer to [28, Table 1] for a summary of the standard linguistic features employed by most SRL systems. Although feature-based methods represent the state-of-the-art for SRL, they cannot model syntactic structured information effectively. For example, the widely used Parse Tree Path feature in feature-based methods is so sensitive to small changes in syntactic parse trees that a pair of predicate-argument structures will have two different Parse Tree Path features even if they differ only by one node, resulting in poor model generalization problems [22].

B. Kernel-Based Methods for SRL

As an alternative to feature-based methods, kernel methods [38] have the potential to better model structured objects due to its ability to directly measure the similarity between two structured objects without explicitly enumerating their substructures. In literature, many kernels have been proposed in the machine learning community and applied in NLP. In particular, [39] proposed the well-known convolution kernels for a discrete structure. Motivated by this work, more and more kernels are proposed and explored in NLP [11], [15], [40], [41], such as the *convolution Tree Kernel* [15], the *String Subsequence Kernel* (SSK) [40], the *Syllables-based String Kernel* [42], and the *Graph Kernel* (HDAG Kernel) [43].

Of special interest here, [22] extracted the Predicate Argument Feature (PAF) portion from a parse tree, which includes salient substructures of the predicate–argument structure, and computed the similarity between two PAFs using the convolution tree kernel. Under the same framework, [23] further separated the PAF into a Path portion and a Constituent Structure portion and combined the two convolution tree kernels over these two separated portions into a single composite kernel. [23] showed that such method significantly increased the performance on the CoNLL-2005 SRL data.

One major problem with the convolution tree kernel is that it only performs hard matching without considering linguistic knowledge. Therefore, it fails to handle similar phrase structures and quasi-synonymic grammar tags. In order to overcome these shortcomings, this paper proposes a grammar-driven convolution tree kernel to better explore grammatical substructures by considering those nonidentical substructures but with similar syntactic properties.

IV. Grammar-Driven Convolution Tree Kernel

In this section, we first briefly review the kernel trick and the standard convolution tree kernel and then define our grammar-driven convolution tree kernel.

A. Kernel Trick

For a linear classification algorithm, it is usually to learn a hyper plane $f(\vec{x}) = \vec{w}^T\vec{x} + b = 0$ to separate two class objects. $\vec{x}$ is a flat feature vector representation of a classifying object o in an n-dimensional space over the real numbers, i.e., $\mathfrak{R}^n$. $\vec{w} \in \mathfrak{R}^n$ and $b \in \mathfrak{R}$ are parameters learned from training data. The object o is mapped to $\vec{x}$ via a feature function $\phi : \mathfrak{O} \rightarrow \mathfrak{R}^n$, $\mathfrak{O}$ being the set of the objects. The kernel trick allows us to rewrite the hyper plane as

$$\begin{aligned} f(\vec{x}) &= \sum_i (y_i\alpha_i\vec{x}_i)^T\vec{x} + b \\ &= \sum_i y_i\alpha_i\vec{x}_i^T\vec{x} + b \\ &= \sum_i y_i\alpha_i\phi(o_i)^T\phi(o) + b \\ &= 0 \end{aligned}$$

where y_i is the label of training object o_i, which is equal to 1 for positive examples and -1 for negative examples, $\alpha_i \geq 0$, and the product $K(o_i, o) = \langle\phi(o_i)^T\phi(o)\rangle$ is the kernel (similarity) function associated with the mapping ϕ. Note that we do not need to apply the mapping ϕ explicitly, the $K(o_i, o)$ can be used directly. This allows us, under Mercer's conditions [44], to define abstract kernel (similarity) functions which generate implicit feature spaces.

B. Convolution Tree Kernel

Convolution (parse) tree kernel [15] counts the number of common subtrees as the syntactic similarity between two parse trees, where a parse tree T is implicitly represented by a vector of integer counts of each subtree type (regardless of its ancestors)

$$\Phi(T) = (\#\text{subtree}_1(T), \ldots, \#\text{subtree}_n(T))$$

where $\#\text{subtree}_i(T)$ is the occurrence number of the ith subtree type (subtree_i) in T. Fig. 2 illustrates a parse tree "NP $\rightarrow$ some/DT red/JJ cars/NNS" with all of its 11 subtrees covered

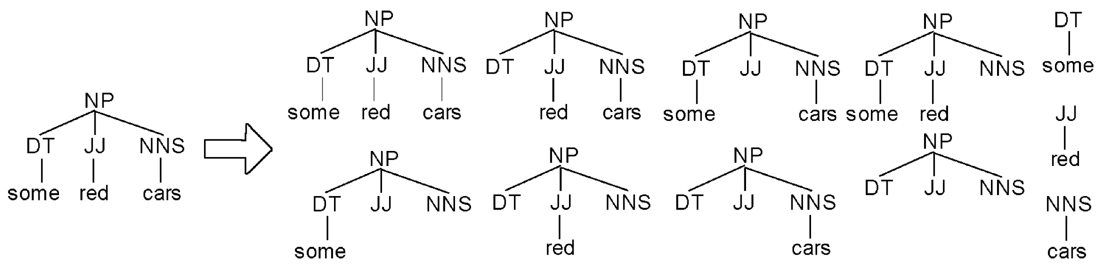

Fig. 2. Syntactic parse tree and all of its 11 subtrees covered by the convolution tree kernel.

by convolution tree kernel. Since the number of different subtrees increases exponentially with the parse tree size, it is computationally infeasible to use the feature vector $\Phi(T)$ directly. To solve this computational issue, [15] proposed the following parse tree kernel to calculate the dot product between the above high dimensional vectors implicitly:

$$\begin{aligned} K(T_1,T_2) &= \langle \Phi(T_1), \Phi(T_2)\rangle \\ &= \sum_i \#\text{subtree}_i(T_1)\cdot \#\text{subtree}_i(T_2) \\ &= \sum_i \left(\left(\sum_{n_1\in N_1} I_{\text{subtree}_i}(n_1)\right) \cdot \left(\sum_{n_2\in N_2} I_{\text{subtree}_i}(n_2)\right)\right) \\ &= \sum_{n_1\in N_1}\sum_{n_2\in N_2} \Delta(n_1,n_2) \end{aligned}$$

where N_1 and N_2 are the sets of nodes in trees T_1 and T_2, respectively, and $I_{\text{subtree}_i}(n)$ is a function whose value is 1 iff there is a subtree$_i$ rooted at node n and zero otherwise, and $\Delta(n_1,n_2)$ is the number of the common subtrees rooted at n_1 and n_2, i.e.,

$$\Delta(n_1,n_2) = \sum_i I_{\text{subtree}_i}(n_1)\cdot I_{\text{subtree}_i}(n_2).$$

Furthermore, $\Delta(n_1,n_2)$ can be computed in polynomial time by applying following recursive rules.

R-1: if the productions (CFG rules) at n_1 and n_2 are different, then $\Delta(n_1,n_2)=0$.

R-2: else if both n_1 and n_2 are preterminals (POS tags), $\Delta(n_1,n_2)=1\times\lambda$.

R-3: else

$$\Delta(n_1,n_2) = \lambda \times \prod_{j=1}^{nc(n_1)} \left(1+\Delta\left(ch(n_1,j), ch(n_2,j)\right)\right)$$

where $nc(n_1)$ is the number of children of n_1, $ch(n,j)$ is the jth child of node n. Please note that a decay factor λ $(0<\lambda \leqq 1)$ is introduced to make the kernel value less variable with respect to the subtree sizes. This corresponds to a modified kernel $K(T_1,T_2)=\sum_i \lambda^{\text{size}_i}\cdot\#\text{subtree}_i(T_1)\cdot\#\text{subtree}_i(T_2)$[3], where size$_i$ is the number of rules in the ith subtree. In addition, rule **R-3** holds because given two nodes with the same children, one can construct common subtrees using these children and common subtrees of further offspring in a recursive way. The time complexity for computing this kernel is $O(|N_1|\cdot|N_2|)$.[4]

C. Grammar-Driven Approximate Substructure Matching

The standard convolution tree kernel requires exact matching between two contiguous phrase structures. It means that only identical phrase structures are matchable. This constraint may be too strict. For example, "NP → DT JJ NN" (NP → *a red car*) and "NP → DT NN" (NP → *a car*) in two parse trees contribute nothing to the standard conventional tree kernel although they share much similar syntactic structure property and thus should play the same semantic role for a given predicate. Therefore, we propose a grammar-driven approximate matching mechanism to capture the similarity between such kinds of structures for SRL by constructing a reduced rule set with optional nodes, e.g., [JJ] in "NP → DT [JJ] NP" and [ADVP] in "VP → VB [ADVP] PP." For convenience, we call "NP → DT JJ NP" the original rule and "NP → DT [JJ] NP" the reduced rule. Here, we define two grammar-driven criteria to determine reduced rules.

1) The reduced rules must be grammatical. This means that a reduced rule should be valid, i.e., occurring in the original rule set, which are collected from training data. For example, "NP → DT [JJ] NP" is acceptable since "NP → DT NP" also occurs in the original rule set, while "VP → [VB ADVP] PP" is invalid since "VP → PP" does not occur in the original rule set.
2) A reduced rule must keep the head child of its corresponding original rule and has at least two children. This is to ensure that the reduced rules retain much of the

[3]Due to the exponential nature of the original kernel (when $\lambda=1$), larger matchable subtrees will have much more influence, then by analogy with a Gaussian kernel we say that the original kernel is very peaked [15]. If we use the kernel to construct a model which is a linear combination of trees, as one would with an SVM or the perceptron, the output will be dominated by the most similar tree, and so the model will behave like a nearest neighbor rule [15]. Experiments in [15], [22], [41] empirically verify this point. To overcome this problem, the decay factor λ is introduced to downweight the contribution of subtrees exponentially with their sizes. For details, please refer to [15, p. 4].

[4][45] proposed a suffix-tree-based algorithm that can compute a tree kernel in linear complexity. Unfortunately, their algorithm is not applicable to the tree kernel (as discussed in this paper) since their algorithm requires all the leaf nodes in a subtree to be terminal nodes (i.e., lexical word in a parse tree) while the leaf nodes in our subtrees can be nonterminals (e.g., phrase tags NP and VP, and POS tags NN and NNS).

semantics of the corresponding original rules and to avoid over-generation of reduced rules.

In this paper, the original rule set is extracted from a training corpus. By defining a small set of optional nodes for each type of left-hand phrases manually, we can automatically build a set of reduced rules from the original rule set (Later in Section V, we will discuss this issue in more details). For two rules A and B with the same left-hand phrase label to be matched, A should be equal to B after zero or more deletions of optional nodes. Obviously, the head child of A must be aligned to the head child of B. Given the reduced rule set, we can formulate the approximate substructure matching mechanism as follows:

$$M(r_1, r_2) = \sum_{i,j} \left(I_T\left(T_{r_1}^i, T_{r_2}^j\right) \times \lambda_1^{a_i+b_j} \right) \tag{1}$$

where

- r_1 is a production rule, representing a subtree of depth one,[5] and likewise for r_2.
- $T_{r_1}^i$ is the ith variation of the subtree r_1 by removing zero ore more optional nodes, and likewise for $T_{r_2}^j$. The training algorithm of a kernel method converges only when using a proper kernel. To make sure that the new kernel is a proper kernel, all the possible variations of the original subtrees should be considered.
- $I_T(\cdot,\cdot)$ is a binary function that is 1 iff the two subtrees are identical and zero otherwise.
- λ_1 $(0 \leqq \lambda_1 \leqq 1)$ is a weight to penalize the removal of optional nodes, while a_i and b_j denote the numbers of removed optional nodes in subtrees $T_{r_1}^i$ and $T_{r_2}^j$, respectively.

$M(r_1, r_2)$ returns the similarity between two subtrees r_1 and r_2 by summing up the similarities between their possible variations. Please note that all the possible variations are taken into account in (1) in order to guarantee that the new kernel is a proper one. Under the approximate matching mechanism, two subtrees are matchable with proper weighting if they are identical after removing zero or more optional nodes. In this way, "NP $\rightarrow$ *a red car*" and "NP $\rightarrow$ *a car*" is matchable with a penalty λ_1 in our new kernel. This means that one such co-occurrence of the two structures contributes λ_1 to our proposed kernel while it contributes nothing to the standard one. Therefore, with the approximate substructure matching mechanism, our method would be able to explore more linguistically appropriate substructures than the standard one.

D. Grammar-Driven Approximate Node Matching

The convolution tree kernel only considers exact matching between two (terminal/nonterminal) nodes. However, some similar POSs or phrase tags may represent similar roles, such as NN (*dog*), NNS (*dogs*), and NP (*the dog*). In order to capture this phenomenon, we allow approximate matching between node features for better coverage by introducing some equivalent node feature sets:

- JJ, JJR, JJS;
- VB, VBD, VBG, VBN, VBP, VBZ;
- NN, NNS, NNP, NNPS, NX.[6]

where node features in the same line are treated as analogous and can match each other. Similar to the approximate substructure matching mechanism, we introduce a new parameter λ_2 $(0 \leqq \lambda_2 \leq 1$ to weight approximate node matching. We call it node feature *mutation*[7] and the approximate node matching mechanism can be formulated as

$$M(f_1, f_2) = \sum_{i,j} \left(I_f\left(f_1^i, f_2^j\right) \times \lambda_2^{a_i+b_j} \right) \tag{2}$$

where f_1 is a node feature, f_1^i is the ith mutation of f_1, and a_i is zero iff f_1^i and f_1 are identical and 1 otherwise, and likewise for f_2 and b_j. $I_f(\cdot,\cdot)$ is a function that is 1 iff the two features are identical and zero otherwise. Equation (2) sums over all combinations of feature mutations as the node feature similarity. Same as (1), all the possibilities are taken into account in (2) to guarantee that the new kernel is a proper kernel.

E. Grammar-Driven Convolution Tree Kernel

Both the approximate substructure and node matching mechanisms are grammar-driven, i.e., they retain most of the underlying linguistic constraints and keep the semantic meanings of the original rules. Given these two approximate matching mechanisms, we can define our grammar-driven convolution tree kernel step by step. First, we can represent a parse tree T using a feature vector implicitly

$$\Phi'(T) = (\#\text{subtree}_1(T), \ldots, \#\text{subtree}_n(T))$$

where $\#\text{subtree}_i(T)$ is the occurrence number of the ith subtree type (subtree_i) in T. Please note that, unlike the standard convolution tree kernel, we relax the condition for the occurrence of a subtree by allowing both original and reduced rules (via the approximate substructure matching), and node feature mutations (via the approximate node matching). In other words, the criteria by which a subtree is said to occur are modified. For example, one occurrence of the subtree "NP $\rightarrow$ DT JJ NP" contributes 1 and λ_1 to its counterparts "NP $\rightarrow$ DT JJ NP," and "NP $\rightarrow$ DT NP," respectively, in the new kernel, while it only contributes 1 to "NP $\rightarrow$ DT JJ NP" in the standard one.

Then, we can define the grammar-driven kernel between two parse trees

$$\begin{aligned} K_G(T_1, T_2) &= \langle \Phi'(T_1), \Phi'(T_2) \rangle \\ &= \sum_i \#\text{subtree}_i(T_1) \cdot \#\text{subtree}_i(T_2) \\ &= \sum_i \left(\left(\sum_{n_1 \in N_1} I'_{\text{subtree}_i}(n_1) \right) \right. \\ &\qquad \left. \cdot \left(\sum_{n_2 \in N_2} I'_{\text{subtree}_i}(n_2) \right) \right) \\ &= \sum_{n_1 \in N_1} \sum_{n_2 \in N_2} \Delta'(n_1, n_2) \end{aligned} \tag{3}$$

[5]Equation (1) is defined over subtrees of depth one. Multilayer subtree approximate matching can be achieved recursively. We will discuss it later in Section IV-E

[6]In Penn TreeBank II, the phrase tag NX standards for a kind of noun phrase (NP) that is used to mark the head of certain complex NPs.

[7]In this paper, nonterminals are manually grouped according to their syntactic similarity and a mutation of a nonterminal in a group is used to refer to another nonterminal in the group.

where N_1 and N_2 are the sets of nodes in trees T_1 and T_2, respectively. Given a and b are the total numbers of removed optional nodes and mutated node features at node n, respectively, $I'_{\mathrm{subtree}_i}(n)$ is $\lambda_1^a \cdot \lambda_2^b$ iff there is a subtree$_i$ rooted at node n and zero otherwise. Moreover, $\Delta'(n_1, n_2)$ counts the weighted number of common subtrees rooted at n_1 and n_2, i.e.,

$$\Delta'(n_1, n_2) = \sum_i I'_{\mathrm{subtree}_i}(n_1) \cdot I'_{\mathrm{subtree}_i}(n_2). \tag{4}$$

Obviously, $\Delta'(n_1, n_2)$ can be further computed using following recursive rules.

R-A: if n_1 and n_2 are pre-terminals, then

$$\Delta'(n_1, n_2) = \lambda \times M(f_1, f_2) \tag{5}$$

where f_1 and f_2 are features of nodes n_1 and n_2, respectively, $M(f_1, f_2)$ is defined in (2) and, similar to the standard convolution tree kernel, the decay factor λ $(0 \leq \lambda \leq 1)$ is introduced to make the kernel less variable to the subtree size.

R-B: else if both n_1 and n_2 are the same nonterminals, then generate all variations of subtrees of *depth two* rooted at n_1 and n_2 (denoted by T_{n_1} and T_{n_2}, respectively) by removing different optional nodes

$$\Delta'(n_1, n_2) = \lambda \times \sum_{i,j} \left(I_T\left(T^i_{n_1}, T^j_{n_2}\right) \times \lambda_1^{a_i + b_j} \times \prod_{k=1}^{nc(n_1,i)} (1 + \Delta'(ch(n_1, i, k), ch(n_2, j, k))) \right) \tag{6}$$

where

- $T^i_{n_1}$ and $T^j_{n_2}$ denote the ith and jth variations of subtrees T_{n_1} and T_{n_2}, respectively;
- $I_T(\cdot, \cdot)$ is a function that is 1 iff the two subtrees are identical and zero otherwise;
- a_i and b_j stand for the number of removed optional nodes in subtrees $T^i_{n_1}$ and $T^j_{n_2}$, respectively;
- $nc(n_1, i)$ returns the number of children of n_1 in its ith subtree variation $T^i_{n_1}$;
- $ch(n_1, i, k)$ is the kth child of node n_1 in its ith variation subtree $T^i_{n_1}$, and likewise for $ch(n_2, j, k)$;
- similar to **R-A**, λ $(0 < \lambda \leqq 1)$ is also included here to make the kernel less variable to the subtree size.

R-C: else $\Delta'(n_1, n_2) = 0$

R-A accounts for the approximate node matching while **R-B** accounts for the approximate substructure matching. In **R-B**, (6) is able to carry out multilayer subtree approximate matching by its recursive nature, while (1) is only effective for two-layer subtrees. Moreover, we reformulate (4) into (5) and (6) for efficient computation. Since (4) is a dot product, we can easily prove that our kernel defined in (3) is a valid kernel using the proof as given in [39].

It is interesting to observe the difference in the substructures covered by the standard convolution tree kernel and the grammar-driven convolution tree kernel. Fig. 3 shows that the grammar-driven kernel can capture additional 17 grammatical substructures with proper weighting.

F. Efficient Computation of the Grammar-Driven Convolution Tree Kernel

One major problem with the grammar-driven convolution tree kernel is its computational complexity, which is exponential in nature due to the approximate substructure and node matching mechanisms. In particular, computing (6) requires exponential time if computed by brute force since all variations of the current subtree have to be explicitly generated by removing one or more optional nodes although the number of reduced rules is bounded by the number of rules in the original Treebank. Here, we present an efficient dynamic programming (DP) algorithm to compute the grammar-driven kernel in polynomial time. Similar to the partial tree (PT) kernel [16], we recast (6) as finding common subtrees with possible optional nodes between two parse trees and rewrite it as

$$\Delta'(n_1, n_2) = \lambda \times \sum_{p=lx}^{lm} \Delta_p\left(c_{n_1}, c_{n_2}\right) \tag{7}$$

where

- c_{n_1} and c_{n_2} are the child node sequences of n_1 and n_2;
- $\Delta_p(\cdot, \cdot)$ evaluates the number of common subtrees with exactly p children (at least including all nonoptional nodes) rooted at n_1 and n_2;
- $lx = \max\{np(c_{n_1}), np(c_{n_2})\}$ and $np(\cdot)$ is the number of non-optional nodes;
- $lm = \min\{l(c_{n_1}), l(c_{n_2})\}$ and $l(\cdot)$ returns the number of children;
- the decay factor λ $(0 < \lambda \leq 1)$ is introduced to make the kernel less variable to the subtree size, similar to the standard convolution tree kernel.

Now the problem becomes how to calculate $\Delta_p(c_{n_1}, c_{n_2})$ efficiently. In this paper, we propose a dynamic programming algorithm similar to the one used in the PT kernel. Let us first look at the DP algorithm in the PT kernel. Given two child node sequences $c_1 a = c_{n_1}$ and $c_2 b = c_{n_2}$ (a and b are their last children, respectively), the DP algorithm in the PT kernel evaluates $\Delta_p(c_{n_1}, c_{n_2})$ recursively as follows:[8]

$$\begin{aligned} \Delta_p\left(c_{n_1}, c_{n_2}\right) &= \Delta_p(c_1 a, c_2 b) \\ &= I(a, b) \times (1 + \Delta'(a, b)) \times \Delta_{p-1}(c_1, c_2) \\ &\quad + \Delta_p(c_1 a, c_2) + \Delta_p(c_1, c_2 b) - \Delta_p(c_1, c_2) \end{aligned} \tag{8}$$

with the following two stopping criteria

$$\Delta_p(c_1, c_2) = 0, \ \forall \text{ node sequence pair } c_1 \text{ and } c_2, \quad \text{if } \min(|c_1|, |c_2|) < p \tag{9}$$

$$\Delta_0(c_1, c_2) = 1, \ \forall \text{ any node sequence pair } c_1 \text{ and } c_2 \tag{10}$$

where $I(a, b)$ is a binary function that is 1 iff two nodes are matchable and zero otherwise. Obviously, (8) is a typical DP algorithm and (9) and (10) are two stopping criteria of the DP algorithm. Moreover, (8) shows the following.

[8]Thanks to A. Moschitti for giving us his source code of the PT kernel. Based on the source code, we reformulated the PT kernel as (8)–(10) instead of his original complicated equations. In addition, for clarity, we did not consider penalizing the length of the child sequence. Moreover, we find a typo in their original formula: the last plus sign "+" in the (5) in his paper [16] should be a subtraction sign "−".

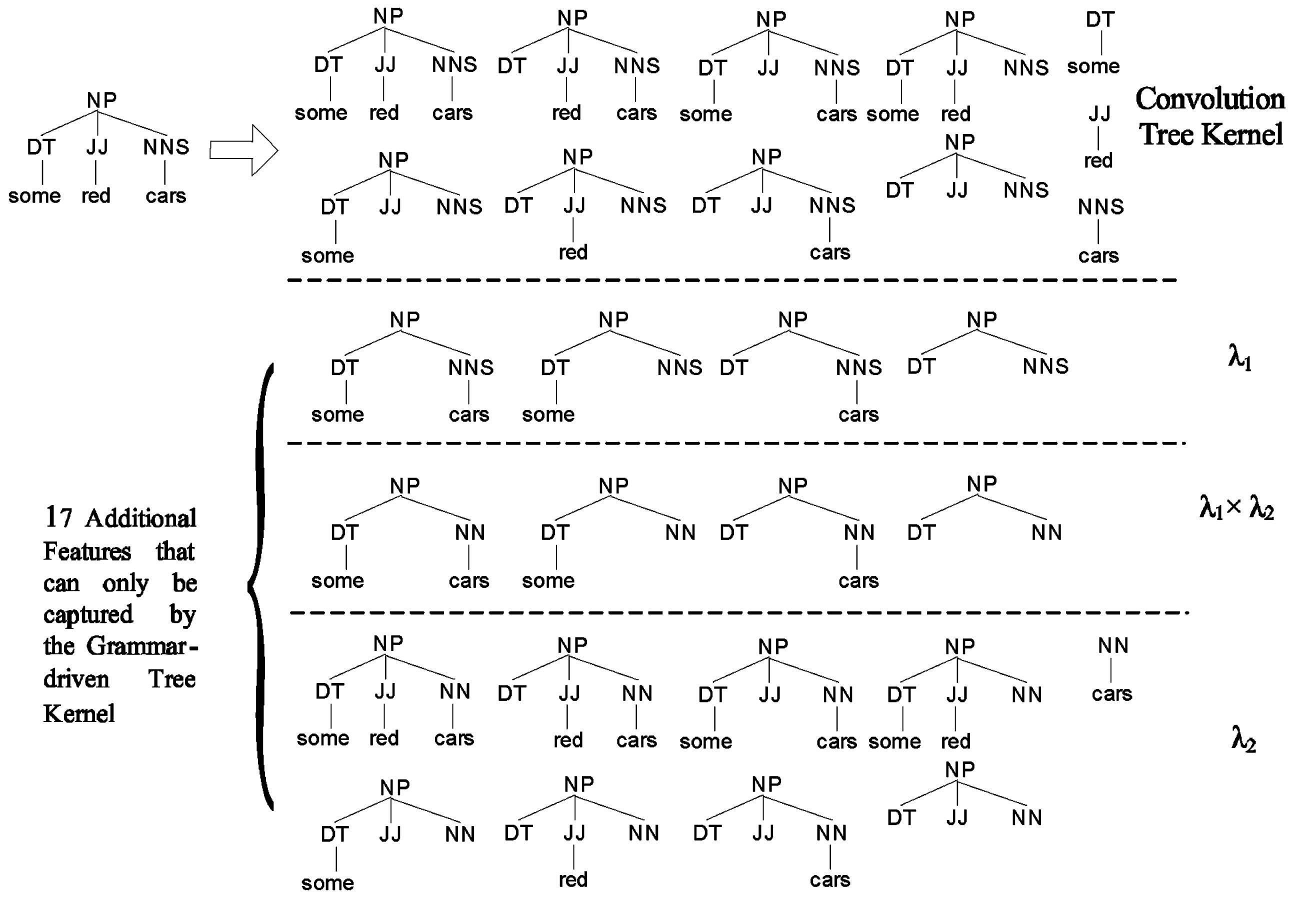
Convolution Tree Kernel
17 Additional Features that can only be captured by the Grammar-driven Tree Kernel
λ1
λ1× λ2
λ2
NP
DT
JJ
NNS
NN
some
red
cars

Fig. 3. Syntactic parse tree and all of its 28 substructures covered by the grammar-driven convolution tree kernel (compared with the standard convolution tree kernel).

- If the last two children nodes a and b are matchable, then we can 1) use $\Delta'(a,b)$ to further count the number of common subtrees rooted at the matchable node pair and 2) use $\Delta_{p-1}(c_1,c_2)$ to evaluate the number of common subtrees with exactly $p-1$ children between c_1 and c_2.
- Besides, $\Delta_p(\cdot,\cdot)$ is called twice over the two shorter node sequence pairs with nodes a and b removed, respectively. Since $\Delta_p(c_1,c_2)$ will be called twice in $\Delta_{p-1}(c_1a,c_2)$ and $\Delta_{p-1}(c_1,c_2b)$ in the next recursive call, we subtract one in (8) to avoid the duplicate calculation. Indeed, this kind of double-counting problem appears frequently in NLP, and is sometime known as "spurious ambiguity" [46].
- The time complexity of computing $\Delta'(n_1,n_2)$ is $O(p|c_{n_1}|\cdot|c_{n_2}|)$ since for any node pair a and b, $\Delta'(a,b)$ is called only once.

Compared with the PT kernel, the grammar-driven tree kernel has two special characteristics.

C1: the grammar-driven kernel only skips optional nodes while the PT kernel has no restriction on node skipping.

C2: the grammar-driven kernel penalizes removed optional nodes only (including both interior and exterior skipped nodes) while the PT kernel weights the length of subsequences (all interior skipped nodes are counted in, but exterior nodes are ignored).

By taking these two factors into consideration, $\Delta_p(c_1,c_2)$ can be computed in the grammar-driven kernel as follows:

$$\begin{aligned}\Delta_p(c_{n_1},c_{n_2}) &= \Delta_p(c_1a,c_2b)\\ &= I(a,b)\times(1+\Delta'(a,b))\times\Delta_{p-1}(c_1,c_2)\\ &\quad+\Delta_p(c_1a,c_2)\times\mathrm{opt}(b)\times\lambda_1\\ &\quad+\Delta_p(c_1,c_2b)\times\mathrm{opt}(a)\times\lambda_1\\ &\quad-\Delta_p(c_1,c_2)\times\mathrm{opt}(a)\times\mathrm{opt}(b)\times\lambda_1^2\end{aligned}$$

with the following two stopping criteria:

$$\begin{aligned}\Delta_p(c_1,c_2) &= 0, \quad \forall \text{ node sequence pair } c_1 \text{ and } c_2,\\ &\qquad \text{if } \min(|c_1|,|c_2|)<p\\ \Delta_0(c_1,c_2) &= 1\times\mathrm{opt}(c_1)\times\mathrm{opt}(c_2)\times\lambda_1^{|c_1|+|c_2|},\\ &\qquad \forall \text{ any node sequence pair } c_1 \text{ and } c_2\end{aligned}$$

where $\mathrm{opt}(w)$ is a binary function, which is 0 if nonoptional nodes are found in the node sequence w and 1 otherwise (for C1); λ_1 is the weight to reflect skipped optional nodes, and the power of λ_1 is the number of skipped optional nodes (for C2). Compared with the PT kernel, the grammar-driven kernel introduces the function $\mathrm{opt}(\cdot)$ and the penalty λ_1. This is to ensure

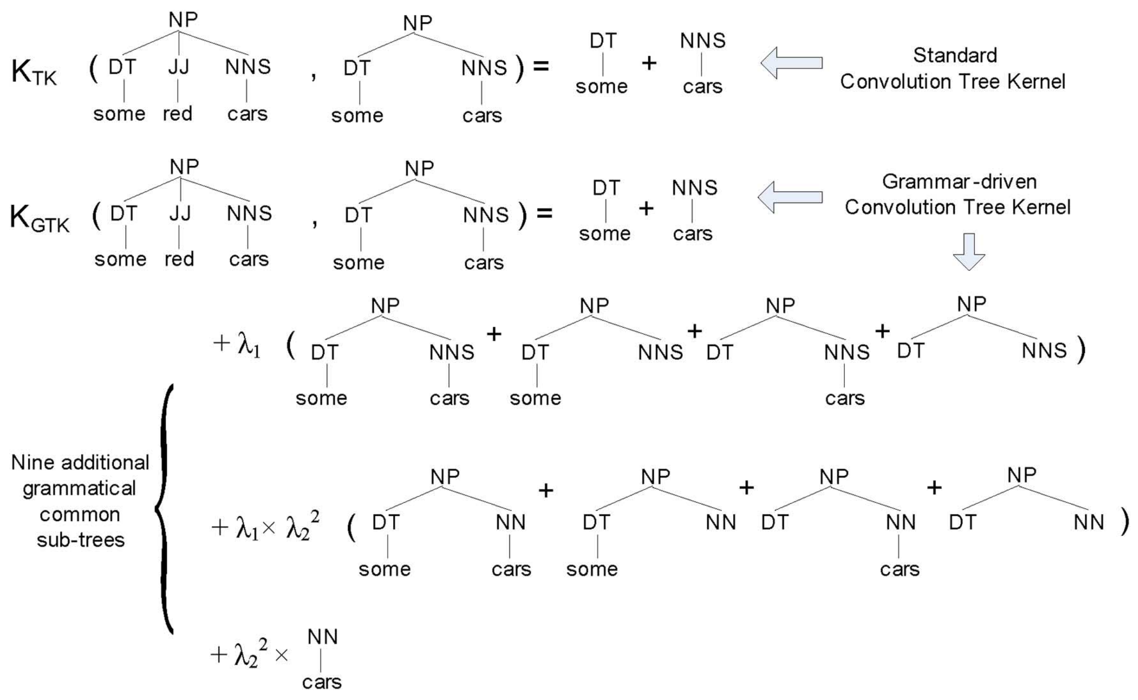
K_TK
NP DT JJ NNS some red cars
NP DT NNS some cars
DT some + NNS cars
Standard Convolution Tree Kernel
K_GTK
Grammar-driven Convolution Tree Kernel
+λ1
Nine additional grammatical common sub-trees
+λ1× λ2²
NP DT NN some cars
+λ2² × NN cars

Fig. 4. Comparison of the convolution tree kernel and the grammar-driven kernel.

that the grammar-driven kernel only skips optional nodes and penalizes those subtrees with skipped optional nodes.

It is clear that we can compute $\Delta'(n_1, n_2)$ in $O(p|c_{n_1}|\cdot|c_{n_2}|)$. This means that the computational complexity of the grammar-driven convolution tree kernel is $O(p\rho^2|N_1|\cdot|N_2|)$ in the worst case, where $\rho = \max_i(|c_{n_i}|)$ is the maximum branching factor of the two trees. Note that the average ρ in natural language parse trees is normally very small and the overall complexity can be reduced by avoiding the computation of node pairs with unmatchable labels (refer to **R-C** in Section IV-E).

G. Comparison With Previous Work

To provide better understanding of our grammar-driven kernel, this section compares it in detail with related tree kernels one by one, such as the convolution tree kernel [15], the PT kernel [16], the labeled order tree kernel [17], [18], the LTAG-based tree kernel [21], the path-sensitive parse tree kernel [11], the path-sensitive dependency tree kernel [14], and the shortest path kernel over dependency trees [19], [20].

The conventional convolution tree kernel [15] is a special case of our grammar-driven kernel. From the kernel function viewpoint, our kernel considers not only exact matching but also approximate matching. From feature exploration viewpoint, although they explore the same substructures (defined recursively by the phrase parse rules), their substructure values are different since our kernel captures similar substructures in a more linguistically appropriate way. Fig. 4 compares the standard kernel and the grammar-driven kernel on two similar trees. It shows that the standard kernel is only able to derive two common subtrees, while the grammar-driven kernel can obtain additional nine common substructures.

Compared with the PT kernel [16] which allows partial matching between subtrees, both the standard and grammar-driven tree kernels generate much fewer substructures. Fig. 5 compares the PT kernel with the standard convolution tree kernel. In some sense, the grammar-driven tree kernel is a special case of the PT kernel. The main difference is that the PT kernel is not grammar-driven, and hence allowing many nonlinguistically motivated substructures to be matched. This may compromise the performance in SRL [16] since some of the generated nongrammatical substructures may be noisy due to the lack of linguistic interpretations and constraints. Another difference is that the PT kernel does not allow approximate node matching. As a result, our kernel exploits linguistic knowledge than the PT kernel.

The labeled order tree kernel [17], [18] generalizes the convolution kernel over labeled order trees. It is much more adaptable than the PT kernel and can explore much larger substructures than the PT kernel through exploiting label mutations and elastic structure matching. However, the same as the PT kernel, the labeled order tree kernel is not grammar-driven. Thus, it may face the same issues (over-generation of nongrammatical substructures) as the PT kernel when used in NLP applications.

The LTAG-based tree kernel [21] was proposed to utilize LTAG-based features in parse tree reranking. The major characteristic with this kernel is that it needs to obtain a LTAG derivation tree for each parse tree before kernel calculation. In comparison, our kernel considers optional nodes using a set of empirical CFG rules.

Both the path-sensitive parse tree [11] and dependency tree [14] kernels match nodes from roots to leaf nodes recursively layer by layer in a top-down manner with the former working on parse trees and the latter on dependency trees. We note that these two kernels impose a strong constraint on kernel computation: matchable nodes should have an identical path of ascending nodes from the roots to the current nodes. For the shortest path

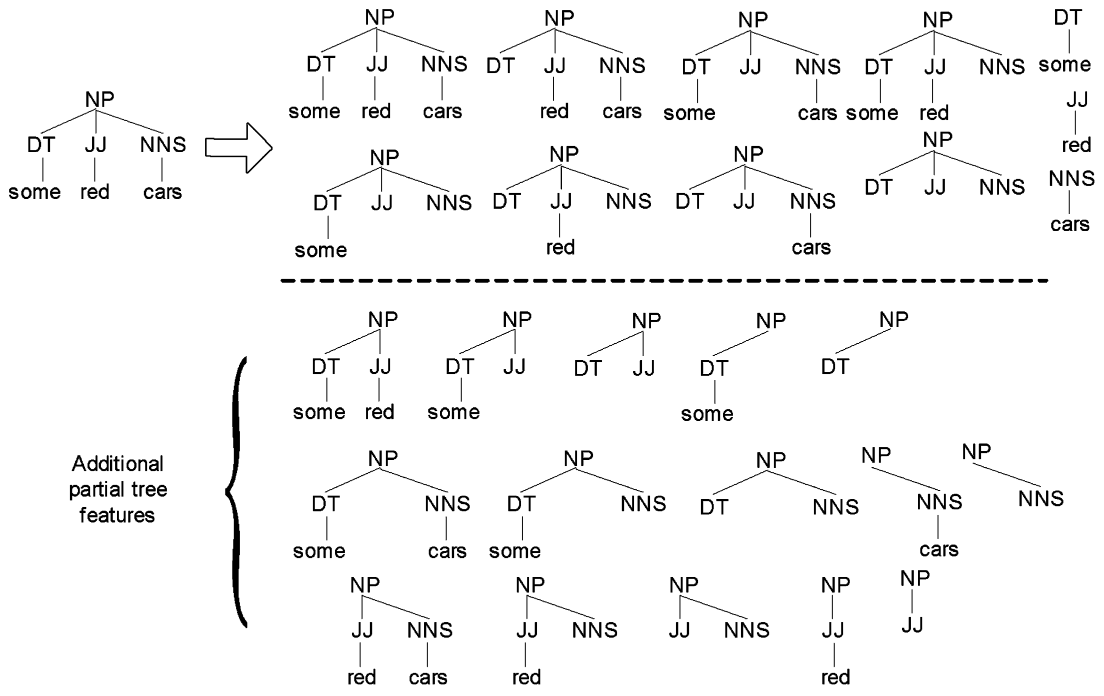

Fig. 5. Syntactic parse tree and all of its substructures covered by the PT kernel (compared with the standard convolution tree kernel).

tree kernel [19], [20] over dependency trees, it simply counts the number of common word classes at each position in the shortest paths between two constituents in dependency trees. This kernel is similar to the widely used path feature in SRL without containing any structured information. Compared with the above three kernels, the convolution tree kernel is more flexible and robust. Evaluation on the ACE corpus [47] of the relation extraction task [41] shows that the convolution tree kernel significantly outperforms the above three kernels. In particular, regarding the node similarity calculation, the path-sensitive dependency tree kernel defined a similarity function, returning the number of feature values in common between the two node feature vectors as the node similarity. Compared with our approximate node matching mechanism, their similarity function does not consider feature mutation and lack the approximate matching functionality. Furthermore, we distinguish the original feature from its mutation (refer to f_1 and f_1^i at (2) in this paper), and introduce a small weight λ_2 to penalize the approximate matching when involving feature mutation. Of course, we can introduce the approximate node matching mechanism into the node similarity function of the path-sensitive dependency tree kernel by modifying their node feature compatibility function. However, the problem is that, without feature mutation, we cannot guarantee the modified node similarity function is still a proper kernel due to its failure to include all possible variations of the original feature.

V. Experiments

In this section, we have systematically evaluated the grammar-driven convolution tree kernel on semantic role labeling.

A. Experimental Setting

We mainly use the CoNLL-2005 data on the SRL shared task [6] for our evaluation, which consists of the *Wall Street Journal* part of the Penn TreeBank [29], with annotated predicate-argument structures from the PropBank I corpus [4]. In total, there are 35 roles including seven Core (A0–A5, AA), 14 Adjunct (AM-), and 14 Reference (R-) arguments. As defined by the shared task, we use sections 02–21 for training, section 24 for development and section 23 for testing, respectively. Table I gives the statistics of the three data sets. For SRL, each sentence is preprocessed in pipeline, including POS tagging [48], named entity recognition [49] and syntactic parsing [50], [51]. Note that the predicates are already annotated in the corpus. Therefore, for fair comparison, we suppose that those predicates have been recognized perfectly in our experiments.

Furthermore, we use constituents as the labeling units to form the labeled arguments. Due to the errors from automatic syntactic parsing, it is impossible for all arguments to find their matching constituents in parse trees. Statistics on the CoNLL-2005 training data shows that 10.08% of arguments have no matching constituents using the Charniak parser [50], and the number increases to 11.89% using the Collins parser [51]. Therefore, we only adopt the Charniak parser in preprocessing. In order to speed up the learning process, we use a four-stage approach.

Stg 1: A pruning stage [33] is applied to filter out the constituents that are clearly not semantic arguments to the predicate.

Stg 2: The candidates derived from *Stg 1* are identified as either arguments or nonarguments using a binary classifier.

Stg 3: Identified arguments from *Stg 2* are classified into one of the semantic roles using a multicategory classifier.

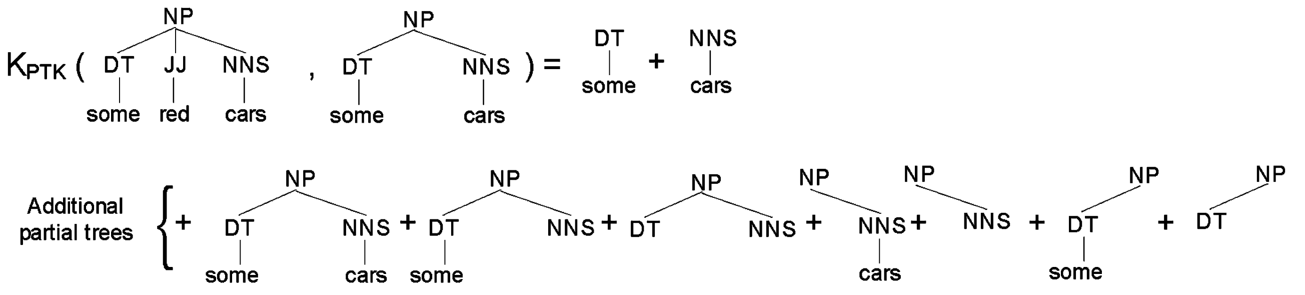

Fig. 6. Comparison of the PT kernel and the standard convolution tree kernel.

TABLE I
STATISTICS OF THE CoNLL-2005 SRL DATA

	Training	Development	Test
Sections	WSJ 02-21	WSJ 24	WSJ 23
Sentences	39,832	1,346	2,416
Arguments	239,858	8,346	14,077

Stg 4: A rule-based postprocessing stage [52] is used to handle unmatched arguments with constituents, such as AM-MOD and AM-NEG. In addition, for embedded arguments or arguments with the same labels, we only maintain the one with maximum probability or score.

Besides, the following are more details on experimental settings:

1) **Evaluation Method**: Accuracy is used to evaluate the performance of semantic role classification (*Stg 3*), while the whole SRL system is evaluated using *precision*, *recall*, and F_1 of the predicted arguments. Here, srl-eval.pl,[9] the official evaluation program of the CoNLL-2005 SRL shared task, is adopted to evaluate the system performance.
2) **Classifier**: SVM [38] is selected as our classifier for *Stgs 2* and *3*. Since SVM is a binary classifier, we adopt the *one versus others* strategy to handle the multiclassification problem in *Stg 3* and select the label with the largest margin as the final output. This also allows us to design a parallel training process which trains different binary classifiers at the same time. Finally, the Tree Kernel in the SVM-Light Tool (SVM-Light-TK) [16], [53] is modified to become a grammar-driven one.
3) **Kernel Setup**: For comparison, we also develop a state-of-the-art feature-based method, which uses the same Constituent, Predicate, and Predicate–Constituent related features as in [30], as the baseline feature-based method. In addition, the best-reported tree kernel $K_{\text{hybrid}} = \theta K_{\text{path}} + (1-\theta)K_{cs}$ $(0 \leq \theta \leq 1)$,[10] proposed by [23] in SRL, is adopted as our baseline kernel. For convenience, we call the original one in [23] a nongrammar-driven hybrid convolution tree kernel and the enhanced one which uses our grammar-driven tree kernel to compute K_{path} and K_{cs} a grammar-driven hybrid tree kernel $(K_{G-\text{hybrid}})$.

[9]http://www.lsi.upc.edu/~srlconll/srl-eval.pl.

[10]K_{path} and K_{cs} are two standard convolution tree kernels to describe predicate-argument path substructures and argument syntactic substructures, respectively. For details, please refer to [23].

For approximate structure matching, we extract about 4700 production rules with at least five occurrences, including the most frequent phrases, such as NP, VP, and ADJP form the entire CoNLL 2005 training data. Finally, we get 1404 rules with optional nodes by introducing three sets of optional nodes:

- NP: JJ, ADJP, ADVP, CC;
- VP: ADVP;
- ADJP: ADVP, CC, RB.

For approximate node matching, we define three equivalent node feature sets as follows:

- JJ, JJR, JJS;
- RB, RBR, RBS;
- NN, NNS, NNP, NNPS, NAC, NX.

Here, we ignore the verb-related equivalent node feature set "VB, VBD, VBG, VBN, VBP, VBZ" since voice information is very indicative in distinguishing A0 (Agent, operator) and A1 (Thing operated).

B. Experimental Results

To verify the effectiveness of the proposed grammar-driven kernel from different viewpoints, we report two kinds of experimental results on different experimental settings. In the first experiment, we use the gold parse trees extracted from the Penn TreeBank corpus and assume that the semantic role identification (*Stg 2*) has been done correctly. In this way, we can focus solely on the semantic role classification task (*Stg 3*). In the second experiment, we use the Charniak parser to automatically parse the entire CoNLL-2005 data and evaluate our kernel in the actual four-stage SRL system. The other reason to do so is that, although there are much research on SRL reported in the literature, they are evaluated using different experimental settings. In this way, we can compare our method with previous research in a more comprehensive way (please refer to Tables II–IV for details).

In order to speed up parameter tuning for the two experiments, only four WSJ sections (Sections 02–05) are used in training when fine-tuning the parameters. Of course, all the following performances are reported on the test data using the entire training data.

Experimental Results Using Gold Parse Trees on the Entire Data With Perfect Argument Identification: For comparison, we set the tree kernel decay factor $\lambda = 0.4$ [22], the SVM regularization parameter $c = 2.4$ [41], and the hybrid kernel parameter $\theta = 0.6$ [23]. In addition, the two penalty factors λ_1 (for approximate structure matching) and λ_2 (for approximate node matching) are fine-tuned to 0.6 and 0.3, respectively.

TABLE II
PERFORMANCE COMPARISON ON SEMANTIC ROLE CLASSIFICATION OF THE CONLL 2005 DATA

Classification Methods	Accuracy (%)
Baseline (Tree Kernel): non-grammar-driven hybrid tree kernel	85.21
Ours: with approximate node matching only	86.27
Ours: with approximate substructure matching only	87.12
Ours: with both approximate substructure and node matching	87.96
Baseline (feature-based): linear kernel	88.51
Baseline (feature-based): polynomial kernel ($d = 2$)	89.92
Ours: composite kernel integrating the grammar-driven hybrid convolution tree kernel and the feature-based polynomial kernel ($d = 2$)	91.02

TABLE III
PERFORMANCE COMPARISON WITH PREVIOUS WORK ON SEMANTIC ROLE CLASSIFICATION OF THE PROPBANK I CORPUS

Methods	Accuracy (%)
Ours: composite kernel with both approximate substructure and node matching mechanisms	**91.97**
Ours: composite kernel with the approximate substructure matching mechanism only	91.75
Ours: composite kernel with the approximate node matching mechanism only	91.43
Composite kernel without approximate matching	91.28
[54]: PAF kernel only	87.7
[34]: feature based	90.50
[30]: feature based	91.0

TABLE IV
PERFORMANCE COMPARISON ON SEMANTIC ROLE LABELING OF THE CONLL 2005 DATA USING SINGLE PARSE TREE

Methods	F_1 (%)
Our Composite Kernel: the grammar-driven hybrid convolution tree kernel + the feature-based polynomial kernel ($d = 2$)	**78.13**
Our Composite Kernel: with relaxing the constraint of at least **two** child nodes to at least **one** child node in defining a reduced rule	77.78
Our Composite Kernel: with the approximate substructure matching only	77.69
Our Composite Kernel: with the approximate node matching only	77.51
[23]'s Composite Kernel: the non-grammar-driven hybrid convolution tree kernel + the feature-based polynomial kernel ($d = 2$)	77.41
Baseline (feature-based): polynomial kernel ($d = 2$)	77.00
[55] (Feature-based): AdaBoost	76.46

Table II compares the performance of different methods on the test data. It shows that the following.

1) The grammar-driven hybrid convolution tree kernel significantly outperforms (χ^2 test with $p = 0.05$) the non-grammar one with an absolute improvement of 2.75% (87.96%–85.21%). This suggests that additional linguistically motivated substructures are very useful for semantic role classification and that the grammar-driven kernel is much more effective in capturing such useful substructures due to the consideration of linguistic knowledge in the kernel design.
2) Both the grammar-driven approximate node and substructure matching mechanisms are very useful for semantic role classification, with the performance improvements of 1.06% (86.27%–85.21%) and 1.91% (87.12%–85.21%), respectively. Moreover, it is interesting to find that their contributions are additive.
3) Our grammar-driven kernel with the approximate substructure and node matching mechanisms achieves only 0.55% (88.51%–87.96%) and 1.96% (89.92%–87.96%) worse than the state-of-the-art feature-based linear kernel and the feature-based polynomial kernel, respectively. In the literature, kernel methods perform much worse than feature-based methods in SRL [16], [22], [23]. This is the first research in kernel methods that achieves comparable performance with the state-of-the-art feature-based methods in SRL. While feature-based methods have been explored extensively and found difficult to further improve the performance, kernel methods have the potential for better performance. Our research justifies this potential and steps a big stride along the right direction since our composite kernel achieves significant (χ^2 test with $p = 0.05$) performance improvement over the previous best-reported both feature-based and kernel-based methods as discussed in the next paragraph.
4) The bigram combination of flat features by polynomial kernel ($d = 2$) is very useful, resulting in 1.41% (89.92%–88.51%) accuracy improvement.

In order to make proper use of syntactic structure information and diverse flat features, we also present a composite kernel to combine the grammar-driven hybrid tree kernel and the state-of-the-art feature-based polynomial kernel (with degree $d = 2$)

$$K_{\text{comp}} = \gamma K_{G-\text{hybrid}} + (1-\gamma)K_{\text{poly}} \quad (0 \le \gamma \le 1)$$

where γ is fine-tuned to 0.3. Table II shows that the composite kernel achieves 91.02% in accuracy, which is significantly better (χ^2 test with $p = 0.05$) than the polynomial kernel ($\gamma = 0$, Accuracy = 89.92%) and the grammar-driven hybrid convolution tree kernel ($\gamma = 1$, Accuracy = 87.96%). This suggests that these two kinds of kernels are quite complementary. This is not surprising since tree kernel methods focus on capturing syntactic structured information while feature-based methods cover different knowledge sources, such as Voice, Sub-Categorization, which are hard to be covered by the tree kernel.

For more comparison, evaluation is also done on the PropBank I corpus (LDC2004T14), with the training, development and test data following the conventional split of Sections 02–21, 00 and 23. Table III compares our method with previously best reported methods.[11] It shows that our method significantly (χ^2 test with $p = 0.05$) outperforms the previous best reported method with an accuracy improvement of 0.97% (91.97%–91.0%). This further verifies the effectiveness of the grammar-driven kernel method for semantic role classification.

[11]Since the two datasets are very similar (please refer to Section II), we simply use the same parameter settings on the dataset of the PropBank I corpus.

Finally, it shows that both the approximate matching mechanisms improve the performance but not significantly (χ^2 test with $p = 0.05$). This may be due to that the rich flat features used in the feature-based polynomial kernel compromise the contributions of the two approximate matching mechanisms.

Experimental Results With Automatic Parse Trees on the Entire Data With Imperfect Argument Identification: In order to examine the influence of the grammar-driven convolution tree kernel on a complete SRL system, we implement a four-stage SRL system, which adopts a state-of-the-art feature-based SVM classifier to identify semantic arguments (*Stg 2*) and our grammar-driven tree kernel to label the semantic roles (*Stg 3*). In particular, λ_1 and λ_2 are fine-tuned to 0.8 and 0.2, respectively, with other parameters remaining the same as previous experiments.

Table IV compares different methods working on a single parse tree. As the baseline system, [55] ranks the fifth among all the participating systems on the CoNLL 2005 SRL shared task and the best when using only one parse tree returned by the Charniak parser. It shows the following.

1) The feature-based polynomial kernel ($d = 2$) outperforms AdaBoost applied in [55].
2) The composite kernel in [23], which combines a nongrammar-driven hybrid convolution tree kernel and the feature-based polynomial kernel, outperforms the polynomial kernel and significantly (χ^2 test with $p = 0.05$) outperforms the best-reported system on the CoNLL 2005 SRL shard task when using only one parse tree.
3) Among all the methods, our composite kernel, which combines a grammar-driven hybrid convolution tree kernel and the feature-based polynomial kernel, achieves the best performance. It significantly (χ^2 test with $p = 0.05$) outperforms [23]'s composite kernel by 0.72% (78.13%–77.41%) in F_1 and significantly (χ^2 test with $p = 0.05$) outperforms the best reported system (based on one parse tree) on the CoNLL 2005 SRL shard task by 1.67% (78.13%–76.46%) in F_1. This further verifies the effectiveness of the grammar-driven convolution tree kernel for SRL.
4) The relaxation on the number of child nodes to at least one node in defining reduced rules leads to 0.35% (78.13%–77.78%) performance drop. This empirically verifies our assumption that the two-node constraint enables the reduced rules to well retain the semantics of the corresponding original rules. Statistics on the CONLL-2005 training set show that 73.59% rules have three to six children nodes while there are 18.59% rules have exact two children node and only 4.6% rules have exact one child node. This indicates that relaxation to at least one node may introduce two much flexibility to the approximate substructure matching mechanism and thus harm the performance.
5) Both the approximate matching mechanisms improve the performance (77.69% and 77.51% versus 77.41%), but not significantly (χ^2 test with $p = 0.05$). This may be due to that the rich flat features used in the feature-based polynomial kernel compromise the contributions of the two approximate matching mechanisms.

Figs. 7 and 8 exemplify the advantage of the grammar-driven convolution tree kernel over the nongrammar-driven one for

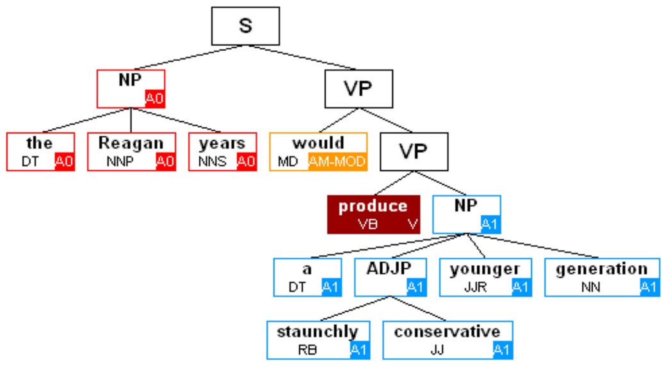

Fig. 7. SRL result using the grammar-driven hybrid convolution tree kernel.

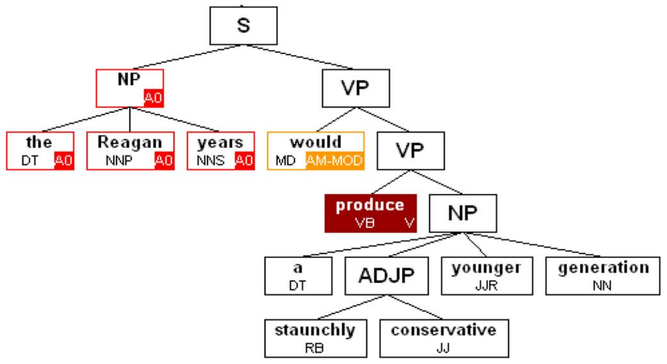

Fig. 8. SRL result using the nongrammar-driven hybrid convolution tree kernel.

SRL. The grammar-driven kernel can label argument A1 correctly while the nongrammar-driven cannot. This is mainly due to the following.

1) The production "NP → DT ADJP JJR NN" has much lower frequency (only five times) in the training data. This makes the nongrammar-driven one fail.
2) The original production "NP → DT ADJP JJR NN" bears much similarity with its frequently occurring counterpart "NP → DT JJ NN" (more than 28 000 times) due to both the approximate substructure and node matching mechanisms (with the mutation of JJ to JJR and the inclusion of ADJP as an optional node to the reduced rule "NP → DT [ADJP] JJR NN"). This makes the grammar-driven one work.

Finally, Table V compares the computational burden of the two composite kernels in training (CPU 2.66G $*$ 8 and Mem 8G). It shows that:

1) The grammar-driven convolution tree kernel only increases slightly the computational burden, although the grammar-driven kernel has the computational complexity of $O(p\rho^2|N_1| \cdot |N_2|)$ in its worse case. This verifies the efficiency of our dynamic programming algorithm.
2) It is very time-consuming to train an SVM classifier in a large dataset.

TABLE V
COMPARISON OF COMPUTATIONAL BURDEN IN TRAINING

Classification Methods	Training Time	
	4 Sections	20 Sections
Ours: the grammar-driven hybrid tree kernel	∼8 hours	∼7 days
[23]: the non-grammar-driven hybrid tree kernel	∼5 hours	∼6 days

VI. Conclusion and Future Work

Kernel methods have been widely studied in speech and language processing, and have the potential to systematically explore structured information for many NLP applications. In this paper, we propose a novel grammar-driven convolution tree kernel to explore more linguistic knowledge in the kernel design for SRL. The experimental results verify the effectiveness of the grammar-driven tree kernel in capturing grammatical substructures over the standard convolution tree kernel by allowing grammar-driven approximate matching of substructures and nodes.

Although tree kernels have achieved promising results in many NLP applications, they always lack linguistic consideration when generating the substructures. To the best of our knowledge, this is the first attempt to integrate linguistic knowledge in the tree kernel design, and our research brings a big step toward this direction.

A more broad motivation of this paper is that a measure of syntax-tree similarity should be informed by linguistics. We report our preliminary study on computing the linguistics-motivated syntax-tree similarity using tree kernel methods. In designing the grammar-driven convolution tree kernel, there are still many degrees of freedom worth exploring in the future.

- Different penalties according to what node is being deleted, and in what context. In this paper, a small penalty is uniformly applied to all removed optional nodes, and an infinite penalty is applied to all removed nonoptional nodes. In the PT kernel [54], there is no penalty for "exterior" deletions while the dependency tree kernel [14] got the best results when assigning infinite penalty to "interior" deletions. Therefore, in the future we will explore a more flexible penalizing mechanism by varying the deletion penalty according to what node is removed and in what context.
- Ability to align unequal terminals or nonterminals. In this paper, only preterminal and one nonterminal (NX) are considered in evaluating approximate node alignment matching mechanism. However, the mechanism also applies to other nonterminals and terminals (e.g., using WordNet-based similarity). We will continue to explore it in the future work.
- Allowing deletion of Chomsky-adjuncts, e.g., by flattening the tree.
- Other possible heuristics for determining which nonterminal nodes are obligatory to match. Currently, we only force all head nodes to match each other.

Another interesting research topic for future work is to integrate linguistic knowledge with tree kernels for effective feature selection in tree kernel-based NLP applications [56]. In particular, a linguistics and statistics-based theory is expected to be worked out, suggesting the effectiveness of different substructures and if they should be generated by the tree kernels. This will lead to better use of syntactic parse tree information instead of enumerating the substructures in an exhaustive way. In addition, the proposed kernel is worth further verifying in other NLP applications, such as relation extraction or parsing. We would also like to study whether the approximate matching mechanisms are applicable to other kinds of structured data, such as, protein structures in bioinformatics. Following this line, we would also like to investigate the way to generalize our proposed method for more structured data, with only minimal specialization for NLP applications.

Acknowledgment

The authors would like to thank Dr. A. Moschitti for his help in using his Tree Kernel Toolkit and fine-tuning the system. They would also like to thank the critical and insightful comments from the three anonymous reviewers.

References

[1] S. Narayanan and S. Harbabagiu, "Question answering based on semantic structures," in *Proc. COLING'04*, 2004.

[2] M. Surdeanu, S. Harabagiu, J. Williams, and P. Aarseth, "Using predicate–argument structures for information extraction," in *Proc. ACL'03*, 2003.

[3] S. P. Ponzetto and M. Strube, "Exploiting semantic role labeling, Wordnet and Wikipedia for coreference resolution," in *Proc. HLT-NAACL, Main Conf.*, New York, Jun. 2006.

[4] M. Palmer, D. Gildea, and P. Kingsbury, "The proposition bank: An annotated corpus of semantic roles," *Comput. Linguist.*, vol. 31, no. 1, pp. 71–106, 2005.

[5] X. Carreras and L. Màrquez, "Introduction to the CoNLL-2004 shared task: Semantic role labeling," in *Proc. HLT-NAACL 2004 Workshop: 8th Conf. Comput. Natural Lang. Learn. (CoNLL-2004)*, H. T. Ng and E. Riloff, Eds., Boston, MA, May 6–7 2004, pp. 89–97, Assoc. for Comput. Linguist..

[6] X. Carreras and L. Màrquez, "Introduction to the CoNLL-2005 shared task: Semantic role labeling," in *Proc. CoNLL-2005*, 2005, pp. 152–164.

[7] C. F. Baker, C. J. Fillmore, and J. B. Lowe, "The Berkeley FrameNet project," in *Proc. ACL-COLING'98*, 1998, pp. 86–90.

[8] D. Gildea and M. Palmer, "The necessity of parsing for predicate argument recognition," in *Proc. ACL'02*, 2002, pp. 239–246.

[9] V. Punyakanok, D. Roth, and W. T. Yih, "The necessity of syntactic parsing for semantic role labeling," in *Proc. IJCAI'05*, 2005, pp. 1117–1123.

[10] M. Collins and N. Duffy, "New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron," in *Proc. ACL'02*.

[11] D. Zelenko, C. Aone, and A. Richardella, "Kernel methods for relation extraction," *J. Mach. Learn. Res.*, vol. 3, pp. 1083–1106, 2003.

[12] M. Zhang, J. Zhang, and J. Su, "Exploring syntactic features for relation extraction using a convolution tree kernel," in *Proc. Human Lang. Technol. Conf. NAACL, Main Conf.*, New York, Jun. 2006.

[13] C. M. Cumby and D. Roth, "On kernel methods for relational learning," in *Proc. ICML*, 2003, pp. 107–114.

[14] A. Culotta and J. Sorensen, "Dependency tree kernels for relation extraction," in *Proc. ACL'04*, 2004, pp. 423–429.

[15] M. Collins and N. Duffy, "Convolution kernels for natural language," in *Proc. NIPS'01*, 2001.

[16] A. Moschitti, "Efficient convolution kernels for dependency and constituent syntactic trees," in *Proc. 17th Eur. Conf. Mach. Learn.*, Berlin, Germany, 2006.

[17] H. Kashima and T. Koyanagi, "Kernels for semi-structured data," in *Proc. 19th Int. Conf. Mach. Learn. ICML'02*, San Francisco, CA, USA, 2002, pp. 291–298, Morgan Kaufmann.

[18] H. Kashima, "Machine learning approaches for structured data," Ph.D. dissertation, Kyoto Univ., Kyoto, Japan, 2007.

[19] R. Bunescu and R. Mooney, "A shortest path dependency kernel for relation extraction," in *Proc. Human Lang. Technol. Conf. Conf. Empirical Methods in Natural Lang. Process.*, Vancouver, BC, Canada, Oct. 2005, pp. 724–731, Assoc. for Comput. Linguist..
[20] R. Bunescu, "Learning for information extraction," Ph.D. dissertation, Univ. of Texas, Austin, TX, 2007.
[21] L. Shen, A. Sarkar, and A. K. Joshi, "Using ltag based features in parse reranking," in *Proc. Conf. Empirical Methods in Natural Lang. Process.*, Morristown, NJ, 2003, pp. 89–96, Assoc. for Comput. Linguist..
[22] A. Moschitti, "A study on convolution kernels for shallow statistic parsing," in *Proc. ACL'04*, 2004, pp. 335–342.
[23] W. Che, M. Zhang, T. Liu, and S. Li, "A hybrid convolution tree kernel for semantic role labeling," in *Proc. COLING/ACL'06*, Sydney, Australia, Jul. 2006.
[24] D. Zhang and W. S. Lee, "Question classification using support vector machines," in *Proc. 26th Annual Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval SIGIR'03*, New York, 2003, pp. 26–32, ACM.
[25] P. Kingsbury, M. Palmer, and M. Marcus, "Adding semantic annotation to the Penn Treebank," in *Proc. Human Lang. Technol. Conf.*, San Diego, CA, 2002.
[26] C. J. Fillmore, "The case of case," in *Universals in Linguistic Theory*, E. Bach and R. T. Harms, Eds. New York: Holt, Rinehart, & Winston, 1968, pp. 1–210.
[27] B. Levin*, English Verb Classes and Alternations: A Preliminary Investigation*. Chicago, IL: Univ. of Chicago Press, 1993.
[28] D. P. A. Moschitti and R. Basili, "Tree kernels for semantic role labeling," *Comput. Linguist. J., Special Issue on Semantic Role Labeling*, accepted for publication.
[29] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of English: The Penn Treebank," *Comput. Linguist.*, vol. 19, no. 2, pp. 313–330, 1993.
[30] S. Pradhan, K. Hacioglu, V. Krugler, W. Ward, J. H. Martin, and D. Jurafsky, "Support vector learning for semantic argument classification," *Mach. Learn. J.*, 2005.
[31] C. Pollard and I. A. Sag*, Head-Driven Phrase Structure Grammar*. Chicago, IL: Univ. of Chicago Press, 1994.
[32] D. Gildea, "Probabilistic models of verb–argument structure," in *Proc. 19th Int. Conf. Comput. Linguist.*, 2002, pp. 1–7.
[33] N. Xue and M. Palmer, "Calibrating features for semantic role labeling," in *Proc. EMNLP'04*, 2004.
[34] Z. P. Jiang, J. Li, and H. T. Ng, "Semantic argument classification exploiting argument interdependence," in *Proc. IJCAI'05*, 2005.
[35] R. D. Nielsen and S. Pradhan, "Mixing weak learners in semantic parsing," in *Proc. EMNLP'04*, 2004.
[36] S. Pradhan, W. Ward, K. Hacioglu, J. Martin, and D. Jurafsky, "Semantic role labeling using different syntactic views," in *Proc. ACL'05*, 2005, pp. 581–588.
[37] V. Punyakanok, D. Roth, W.-T. Yih, and D. Zimak, "Semantic role labeling via integer linear programming inference," in *Proc. COLING'04*, 2004, pp. 1346–1352.
[38] V. N. Vapnik*, Statistical Learning Theory*. New York: Wiley, 1998.
[39] D. Haussler, "Convolution kernels on discrete structures," Tech. Rep. UCSC-CRL-99-10, Jul. 1999.
[40] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins, "Text classification using string kernels," *J. Mach. Learn. Res.*, vol. 2, pp. 419–444, 2002.
[41] M. Zhang, J. Zhang, J. Su, and G. Zhou, "A composite kernel to extract relations between entities with both flat and structured features," in *Proc. COLING/ACL'06*, Sydney, Australia, Jul. 2006.
[42] C. Saunders, H. Tschach, and J. Shawe-Taylor, "Syllables and other string kernel extensions," in *Proc. ICML'02: Proc. 19th Int. Conf. Mach. Learn.*, San Francisco, CA, 2002, pp. 530–537, Morgan Kaufmann.
[43] J. Suzuki, T. Hirao, Y. Sasaki, and E. Maeda, "Hierarchical directed acyclic graph kernel: Methods for structured natural language data," in *Proc. 41st Annu. Meeting Assoc. Comput. Linguist. ACL'03:*, Morristown, NJ, 2003, pp. 32–39, Assoc. for Comput. Linguist..
[44] J. Shawe-Taylor and N. Cristianini*, Kernel Methods for Pattern Analysis*. Cambridge, U.K.: Cambridge Univ. Press, 2004.
[45] S. V. N. Vishwanathan and A. Smola, "Fast kernels for string and tree matching," *Adv. Neural Inf. Process. Syst.*, vol. 15, 2003.
[46] F. C. N. Pereira and M. D. Riley, "Speech recognition by composition of weighted finite automata," AT&T Labs-Research, 1996.
[47] The automatic content extraction projects," Project-ACE [Online]. Available: http://www.ldc.upenn.edu/Projects/ACE/
[48] J. Giménez and L. Màrquez, "Fast and accurate part-of-speech tagging: The SVM approach revisited," in *Proc. RANLP'03*, 2003.
[49] H. L. Chieu and H. T. Ng, "Named entity recognition with a maximum entropy approach," in *Proc. CoNLL'03*, 2003, pp. 160–163.
[50] E. Charniak, "A maximum-entropy-inspired parser," in *Proc. NAACL'00*, 2000.
[51] M. Collins, "Head-driven statistical models for natural language parsing," Ph.D. dissertation, Pennsylvania Univ., Philadelphia, PA, 1999.
[52] T. Liu, W. Che, S. Li, Y. Hu, and H. Liu, "Semantic role labeling system using maximum entropy classifier," in *Proc. CoNLL'05*, 2005, pp. 189–192.
[53] T. Joachims*, Learning to Classify Text Using Support Vector Machines: Methods, Theory and Algorithms*. Norwell, MA: Kluwer, 2002.
[54] A. Moschitti, "Syntactic kernels for natural language learning: The semantic role labeling case," in *Proc. HHLT-NAACL, Companion Volume: Short Papers*, New York, Jun. 2006.
[55] M. Surdeanu and J. Turmo, "Semantic role labeling using complete syntactic analysis," in *Proc. CoNLL'05*, Ann Arbor, MI, Jun. 2005.
[56] J. Suzuki, H. Isozaki, and E. Maeda, "Convolution kernels with feature selection for natural language processing tasks," in *ACL '04: Proc. 42nd Annu. Meeting Assoc. Comput. Linguist.*, Morristown, NJ, 2004, p. 119, Assoc. for Comput. Linguist..

Min Zhang received the B.S. and Ph.D. degrees in computer science from the Harbin Institute of Technology (HIT), Harbin, China, in 1991 and 1997, respectively.

He joined the Institute for Infocomm Research, Singapore, in December 2003 and is currently a Senior Research Fellow at the institute. From 1997 to 1999, he worked as a Postdoctoral Research Fellow in the Korean Advanced Institute of Science and Technology (KAIST), Daejeon, Korea. He began his academic and industrial career as a Researcher at Lernout & Hauspie Asia Pacific, Singapore, in 1999. He joined Infotalk Technology, Singapore, as a Researcher in 2001 and became a Senior Research Manager in 2002. His current research interests include multilingual modeling, machine translation, information extraction, and machine learning.

Wanxiang Che is currently pursuing the Ph.D. degree at the School of Computer Science and Technology, Harbin Institute of Technology (HIT), Harbin, China. He received a Microsoft Scholarship in 2005.

His research interests are in the fields of natural language processing and semantic role labeling.

GuoDong Zhou received the B.Sc., degree from Xi'an Jiaotong University, Xi'an, China, in 1989, the M.Sc., degree from Shanghai Jiaotong University, Shanghai, China, in 1992, and the Ph.D. degree from the National University of Singapore in 1999.

He joined the Institute for Infocomm Research, Singapore, in 1999 and had been an Associate Lead Scientist at the institute until August 2006. Currently, he is a Professor at the School of Computer Science and Technology, Soochow University, Suzhou, China. His research interests include natural language processing, information extraction and machine learning.

Aiti Aw received the B.Sc. degree in computer science from the National University of Singapore in 1987.

She is currently a Senior Research Manager in the Human Language Technology Department at the Institute for Infocomm Research (I^2R), Singapore. She oversees the research and development of several multilingual processing and management projects. She cofounded a start-up within I^2R in 2000 called EWGate, focusing on machine translation. Her technical interests include machine translation and multilingual processing.

Chew Lim Tan (SM'03) received the B.Sc. (honors) degree in physics from the University of Singapore, in 1971, the M.Sc. degree in radiation studies from the University of Surrey, Surrey, U.K., in 1973, and the Ph.D. degree in computer science from the University of Virginia, Charlottesville, in 1986.

He is an Associate Professor in the Department of Computer Science, School of Computing, National University of Singapore. His research interests include pattern recognition, document image analysis, text and natural language processing, neural networks, and genetic programming. He has published more than 300 research publications in these areas. He is an Associate Editor of *Pattern Recognition*, Associate Editor of *Pattern Recognition Letters*, and an editorial member of the *International Journal on Document Analysis and Recognition*. He was a Guest Editor of the special issue on PAKDD 2006 of the *International Journal of Data Warehousing and Mining*, April–June 2007. He has served in the program committees of many international conferences.

Prof. Tan is the current President of the Pattern Recognition and Machine Intelligence Association (PREMIA) in Singapore. He is a member of the Governing Board of the International Association of Pattern Recognition (IAPR).

Ting Liu received the Ph.D degree in computer science from the Harbin Institute of Technology (HIT), Harbin, China, in 1998.

He is a professor at the School of Computer Science and Technology, HIT. From 1999 to 2001, he worked as an Associate Researcher at Microsoft Research Asia (MSRA), Beijing, China. Since 2001, he has held an associate professor position at HIT. His research areas include natural language processing and information retrieval.

Sheng Li received the B.S. degree in computer science in 1965.

He is a Professor at the School of Computer Science and Technology, Harbin Institute of Technology (HIT), Harbin, China. From 1998 to 2004, he was the chairman of HIT. His research areas include machine translation, natural language processing and information retrieval.