Zhou GD, Zhu QM. Kernel-based semantic relation detection and classification via enriched parse tree structure. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 26(1): 45–56 Jan. 2011. DOI 10.1007/s11390-011-1110-2

Kernel-Based Semantic Relation Detection and Classification via Enriched Parse Tree Structure

Guo-Dong Zhou (周国栋), *Senior Member*, *CCF*, *Member*, *ACM*, *IEEE*
and Qiao-Ming Zhu (朱巧明), *Senior Member*, *CCF*

NLP Lab, School of Computer Science and Technology, Soochow University, Suzhou 215006, China

E-mail: gdzhou@suda.edu.cn; qmzhu@suda.edu.cn

Received December 28, 2009; revised October 28, 2010.

Abstract This paper proposes a tree kernel method of semantic relation detection and classification (RDC) between named entities. It resolves two critical problems in previous tree kernel methods of RDC. First, a new tree kernel is presented to better capture the inherent structural information in a parse tree by enabling the standard convolution tree kernel with context-sensitiveness and approximate matching of sub-trees. Second, an enriched parse tree structure is proposed to well derive necessary structural information, e.g., proper latent annotations, from a parse tree. Evaluation on the ACE RDC corpora shows that both the new tree kernel and the enriched parse tree structure contribute significantly to RDC and our tree kernel method much outperforms the state-of-the-art ones.

Keywords semantic relation detection and classification, convolution tree kernel, approximate matching, context sensitiveness, enriched parse tree structure

1 Introduction

Semantic Relation Detection and Classification (RDC) between named entities is to detect and classify various pre-defined semantic relations between pairs of entities in text. The research in RDC has been promoted by the Message Understanding Conferences (MUC, 1987∼1998) and the NIST Automatic Content Extraction (ACE, 2002∼2007) Program[1]. According to the ACE program, an entity is an object or a set of objects in the world and a relation is an explicitly or implicitly stated relationship between the entities. For example, the sentence "Bill Gates is the chairman and chief software architect of Microsoft Corporation." conveys the ACE-style relation "EMPLOYMENT.exec" between the entities "Bill Gates" (person name) and "Microsoft Corporation" (organization name). As a core research topic in information extraction, RDC can be very useful in many natural language processing (NLP) applications such as question answering, e.g., answering the query "Who is the president of the United States?", and information retrieval, e.g., expanding the query "Barack Obama" with "the president of the United States" via his relationship with "the United States".

In the literature, RDC is normally re-cast as a classification problem using a machine learning algorithm. During training, a classifier machine learning algorithm uses the annotated relation instances in the training data to learn a classifier while, during testing, the learned classifier is input instances to determine their relation classes (including non-relation) and thus extract possible relations. Most learning algorithms rely on feature-based representation of input instances. That is, an annotated instance is transformed into a collection of features $f_1, f_2, \ldots, f_N$, thereby producing an N-dimensional feature vector. However, in many NLP problems, it is computationally infeasible to generate features involving structural information or long-distance dependencies. For example, one cannot enumerate efficiently all the sub-tree features for a parse tree. As an alternative to the feature-based methods, kernel methods[2] can implicitly explore high-dimensional structural features efficiently using a kernel function. A kernel function is a similarity function satisfying the properties of being symmetric and positive-definite. More precisely, a kernel function K over the instance space X, $K : X \times X \rightarrow [0, \infty)$, maps a pair of instances x, $y \in X$ to their similarity score $K(x, y)$. It can be proven that a kernel function can measure the similarity between two input instances by implicitly computing the dot product of certain features in high

Regular Paper
Supported by the National Natural Science Foundation of China under Grant Nos. 60873150, 60970056 and 90920004.

(or even infinite)-dimensional feature spaces without explicitly enumerating all the features[3].

Much research has been done in RDC using machine learning approaches. In particular, feature-based methods[4-6] have achieved certain success by employing a large amount of diverse linguistic features, varying from lexical knowledge, entity-related information to syntactic parse trees, dependency trees and semantic information. However, it has been proven difficult for feature-based methods to effectively capture structural information in a parse tree[6], which is critical to further performance improvement in RDC.

An alternative to feature-based methods is to explore kernel methods[2], which can implicitly explore features in a high-dimensional space by employing a kernel to calculate the similarity between two objects directly as described above. In particular, the kernel methods can be effective in reducing the burden of feature engineering for structured objects in NLP, e.g., the parse tree structure in RDC, in that they can provide an elegant solution to implicitly explore structural features. Although earlier researches[7-9] fail to achieve the same level performance as feature-based methods, tree kernel methods achieve much progress recently. As two representatives, Zhang[10] applied the convolution tree kernel (CTK)[11] and achieved comparable performance with the state-of-the-art feature-based methods on the ACE RDC corpora while Zhou *et al.*[12] enabled the standard CTK with context-sensitiveness and achieved much better performance than the state-of-the-art feature-based methods on the ACE RDC corpora. This suggests the great potential of kernel methods in the RDC task.

This paper advances tree kernel methods in RDC from two aspects. First, a new CTK with context-sensitiveness and approximate matching is presented to effectively capture the inherent structural knowledge in a parse tree. Second, an enriched parse tree structure is proposed to well derive necessary structural information from a parse tree. In particular, the Berkeley parser[13] is applied to enrich the naïve grammar in the original parse tree with proper latent annotations. Evaluation on the ACE RDC corpora shows that both the new tree kernel and the enriched parse tree structure contribute significantly to RDC and our tree kernel method much outperforms the state-of-the-art ones by more than 4 units in F1-measure.

The layout of this paper is as follows. In Section 2, we review related work in more details. Then, our CTK is presented in Section 3 while the enriched parse tree structure is described in Section 4. Section 5 shows the experimental results. Finally, we conclude our work in Section 6.

2 Related Work

Kernel methods have the potential of better modeling structured objects due to its ability of directly measuring the similarity between two structured objects, without explicitly enumerating their substructures. In particular, they can be effective in reducing the burden of feature engineering for structured objects in NLP researches, e.g., the parse tree structure in RDC. Representative work in tree kernel-based RDC includes [7] Zelenko *et al.*, [8] by Culotta and Sorensen, [9] by Bunescu and Mooney, [10] by Zhang *et al.* and [12] by Zhou *et al.*

Zelenko *et al.*[7] proposed a kernel between two parse trees, which recursively matches nodes from roots to leaves in a top-down manner. For each pair of matched nodes, a subsequence kernel on their child nodes, which matches either contiguous or sparse subsequences of nodes, is invoked. They achieved certain success on two simple relation extraction tasks. Culotta and Sorensen[8] extended this kernel to estimate the similarity between augmented dependency trees. They achieved the F1-measure of 45.8 on the 5 relation types of the ACE RDC 2003 corpus. One problem with this kernel is that matched nodes must be at the same depth, counting from the root node. This is a strong constraint to the matching of syntax. Therefore, it is not surprising that this kernel achieved good precision but very low recall on the ACE RDC 2003 corpus. In addition, according to the top-down node matching mechanism of this kernel, once a node cannot be matched with any node at the same depth in another tree, all the sub-trees below this node are discarded even if some of them can be matched to their counterparts in another tree. Bunescu and Mooney[9] proposed a shortest path dependency tree kernel and achieved the F1-measure of 52.5 on the 5 relation types of the ACE RDC 2003 corpus. Their kernel is very straightforward. It just sums up the number of common word classes at each position in the two paths. They argued that the information to model a relationship between two entities could be typically captured by the shortest path between them in the dependency graph. One problem with their method is that the shortest path may not be able to well preserve necessary structural dependency tree information. Another problem with their method is that the two shortest paths must have the same length to get a non-zero similarity score in the kernel computation. Therefore, although their kernel shows non-trivial performance improvement compared to that of Culotta and Sorensen[8], such constraints make the two dependency tree kernels share similar behavior: high precision but much lower recall on the ACE RDC 2003 corpus.

As a state-of-the-art tree kernel method, [10] by Zhang *et al.* explored various parse tree structures and adopted the convolution tree kernel (CTK)[11] over parse trees to model structural information in RDC. They achieved the best F1-measure of 68.7 on the 5 relation types of the ACE RDC 2003 corpus, using the Shortest Path-enclosed Tree (SPT) as the parse tree structure. It is worth noting that this work benefited much from Moschitti[14], which originated convolution kernels for extracting relations (in shallow semantic parsing instead of RDC). Additionally, the SPT is actually an application of the PAF tree, as defined in [14] by Moschitti, where two arguments in the SPT are two named entities instead of a predicate verb and one of its arguments in the PAF tree. One problem with the standard CTK is that the sub-trees involved in the tree kernel computation are context-free, that is, they do not consider the information outside the sub-trees. This is quite different from the tree kernel in [8] by Culotta and Sorensen, where the sub-trees involved in the tree kernel computation are context-sensitive (that is, with the path from the tree root node to the sub-tree root node in consideration). For a systematic comparison of the CTK and other related kernels, please refer to [22] by Zhang *et al.* By integrating the advantages of both [10] by Zhang *et al.* and [8] by Culotta and Sorensen, [12] by Zhou *et al.* equipped the standard CTK with context sensitiveness. They achieved the best F1-measures of 71.0/73.2 on the 5/7 relation types of the ACE RDC 2003/2004 corpora respectively, using the SPT with the expanded predicate path as the parse tree structure. However, this kernel also has two drawbacks. First, it only allows exact matching (i.e., full matching) of sub-trees and thus may fail to effectively capture the commonality between similar sub-trees, although a CTK allows the matching of different sub-trees in two parse trees. Second, it does not distinguish different portions of the parse trees and thus may permit wrong matching between different portions of two parse trees.

3 CTK with Context Sensitiveness and Approximate Matching

Given two parse trees, we now study how to measure the similarity between them, using a convolution kernel. A convolution kernel[2] aims to capture structural information in terms of substructures. As a specialized convolution kernel, the convolution tree kernel[11] $K_{\mathrm{CTK}}(T_1, T_2)$ ("CTK" for convolution tree kernel) counts the number of common sub-trees as the syntactic structure similarity between two parse trees T_1 and T_2. Here, a parse tree T is implicitly represented by a vector of integer counts of each sub-tree type (regardless of its ancestors):

$$\boldsymbol{\phi}(T) = (\#subtree_1(T), \cdots, \#subtree_i(T), \ldots, \#subtree_n(T)) \quad (1)$$

where $\#subtree_i(T)$ is the occurrence number of the i-th sub-tree type ($subtree_i$) in T. Since the number of different sub-trees increases exponentially with the parse tree size, it is computationally infeasible to use the feature vector $\boldsymbol{\phi}(T)$ directly. To solve this computational issue, Collins and Duffy[11] proposed a convolution tree kernel to calculate the dot product between the above high-dimensional vectors implicitly as follows:

$$\begin{aligned} & K_{\mathrm{CTK}}(T_1, T_2) = \langle \boldsymbol{\phi}(T_1), \boldsymbol{\phi}(T_2) \rangle \\ &= \sum_i \#subtree_i(T_1) \cdot \#subtree_i(T_2) \\ &= \sum_i \Big(\sum_{n_1 \in N_1} I_{subtree_i}(n_1) \Big) \Big(\sum_{n_2 \in N_2} I_{subtree_i}(n_2) \Big) \\ &= \sum_{n_1 \in N_1, n_2 \in N_2} \Delta(n_1, n_2) \end{aligned} \quad (2)$$

where N_1 and N_2 are the sets of nodes in trees T_1 and T_2, respectively, and $I_{subtree_i}(n)$ 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 sub-trees rooted at n_1 and n_2, i.e.,

$$\Delta(n_1, n_2) = \sum_i I_{subtree_i}(n_1) \cdot I_{subtree_i}(n_2) \quad (3)$$

where N_j is the set of nodes in tree T_j, and $\Delta(n_1, n_2)$ evaluates the common sub-trees rooted at n_1 and n_2. Here, each node n encodes the identity of a sub-tree rooted at n and, if there are two nodes in the tree with the same label, the summation will go over both of them. Therefore, $\Delta(n_1, n_2)$ can be computed recursively as follows.

1) If the context-free productions (i.e., context-free grammar rules) at n_1 and n_2 do not match, then $\Delta(n_1, n_2) = 0$; otherwise go to 2).

2) If both n_1 and n_2 are POS tags, then $\Delta(n_1, n_2) = \lambda$; otherwise (i.e., if both n_1 and n_2 are constituent tags, such as NP and S) go to 3).

3) Calculate $\Delta(n_1, n_2)$ recursively as:

$$\Delta(n_1, n_2) = \lambda \prod_{k=1}^{\#ch(n_1)} (1 + \Delta(ch(n_1, k), ch(n_2, k))) \quad (4)$$

where $\#ch(n)$ is the number of children of node n, $ch(n, k)$ is the k-th child of node n and λ ($0 < \lambda < 1$) is the decay factor in order to make the kernel value less variable with respect to different sub-tree sizes[11]. Note that, in the above standard CTK, n_1 and n_2 in (4) should have the same number of children, i.e., $\#ch(n_1) = \#ch(n_2)$. This corresponds to a modified

kernel of

$$K(T_1, T_2) = \sum_i \lambda^{size_i} \cdot \#subtree_i(T_1) \cdot \#subtree_i(T_2) \quad (5)$$

where $size_i$ is the number of rules in the i-th sub-tree. Due to the exponential nature of the original kernel (when $\lambda = 1$), larger matchable sub-trees will have much more influence, then by analogy with a Gaussian kernel, we say that the original kernel is very peaked[11]. 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[11]. The experiments in [10-11, 14] empirically verify this point. To overcome this problem, the decay factor λ is introduced to downweight the contribution of sub-trees exponentially with their sizes. In addition, (4) holds because, given two nodes with the same children, one can construct common sub-trees using these children and common sub-trees of further offspring in a recursive way. The time complexity for computing this kernel is $O(|N1| \times |N2|)$. For details, please refer to [11]. Fig.1 illustrates a parse tree "NP→some/DT red/JJ cars/NNS" with all of its 11 sub-trees covered by the convolution tree kernel.

Although the above standard CTK has been successfully applied in many NLP applications, it has two shortcomings:

1) The sub-trees involved in the tree kernel computation are context-free (i.e., they do not consider the information outside the sub-trees).

2) It only allows exact matching (i.e., full matching) of sub-trees and thus may fail to effectively capture the commonality between similar sub-trees.

In order to resolve the above two shortcomings, this paper proposes a new CTK by equipping the standard CTK with context-sensitiveness and approximate matching.

3.1 Context Sensitiveness

The context sensitiveness is achieved by considering ancestral information (i.e., the root node path) of sub-trees, i.e., modifying (2) as follows:

$$K_{\mathrm{CTK}}(T_1, T_2) = \sum_{i=1}^{m} w_i \cdot \sum_{\substack{n_1[1,i] \in N_1[1,i] \\ n_2[1,i] \in N_2[1,i]}} \Delta(n_1[1,i], n_2[1,i]) \quad (2')$$

where

1) $n_j[1,i]$ is a root node path with length i in tree T_j, taking into account the n_j's $i-1$ ancestral nodes $n_j[2,i]$, such as the parent node and the grandparent node of n_j in T_j;

2) $\Delta(n_1[1,i], n_2[1,i])$ measures the common sub-trees with root node paths $n_1[1,i]$ and $n_2[1,i]$, m defines the maximal length of a root node path and w_i is the weight for a sub-tree with a root node path of length i. Obviously, the similarity formula in $(2')$ is a proper kernel since it is only a linear interpolation of several CTKs over different root node paths.

For example, Fig.2 illustrates a parse tree "NP→some/DT red/JJ cars/NNS" with all of its 33 sub-trees covered by the CTK with context sensitiveness, assuming that the NP node has the parent node VP and the grandparent node S. Obviously, the standard CTK is a special case of our tree kernel (with $m = 1$ and $w_1 = 1$ in $(2')$). In particular, our tree kernel not only counts the occurrence of each context-free sub-tree, which does not consider its ancestors, but also counts the occurrence of each context-sensitive sub-tree, which considers its ancestors.

3.2 Approximate Matching

The standard CTK only allows exact matching of sub-trees. For approximate matching, we can modify (2) in computing $\Delta(n_1, n_2)$ to allow insertion/deletion/substitution of tree nodes in the sub-trees in equipping the standard CTK with approximate matching:

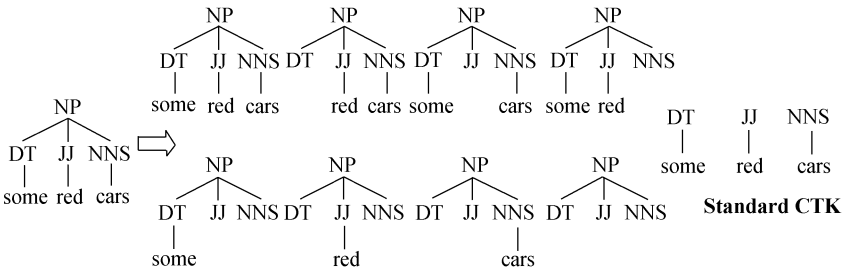

Fig.1. A syntactic parse tree and all of its 11 sub-trees covered by the standard CTK. (Note: 1) Part-of-speech tags used in the examples of this paper: DT (determiner), JJ (adjective), NNS (noun, plural), NN (noun, singular or mass), VBD (verb, past tense), VBN (verb, past participle or passive), NNP (proper noun, singular), CC (coordinating conjunction). 2) Constituent tags: VP (verb phrase), NP (noun phrase), S (sentence))

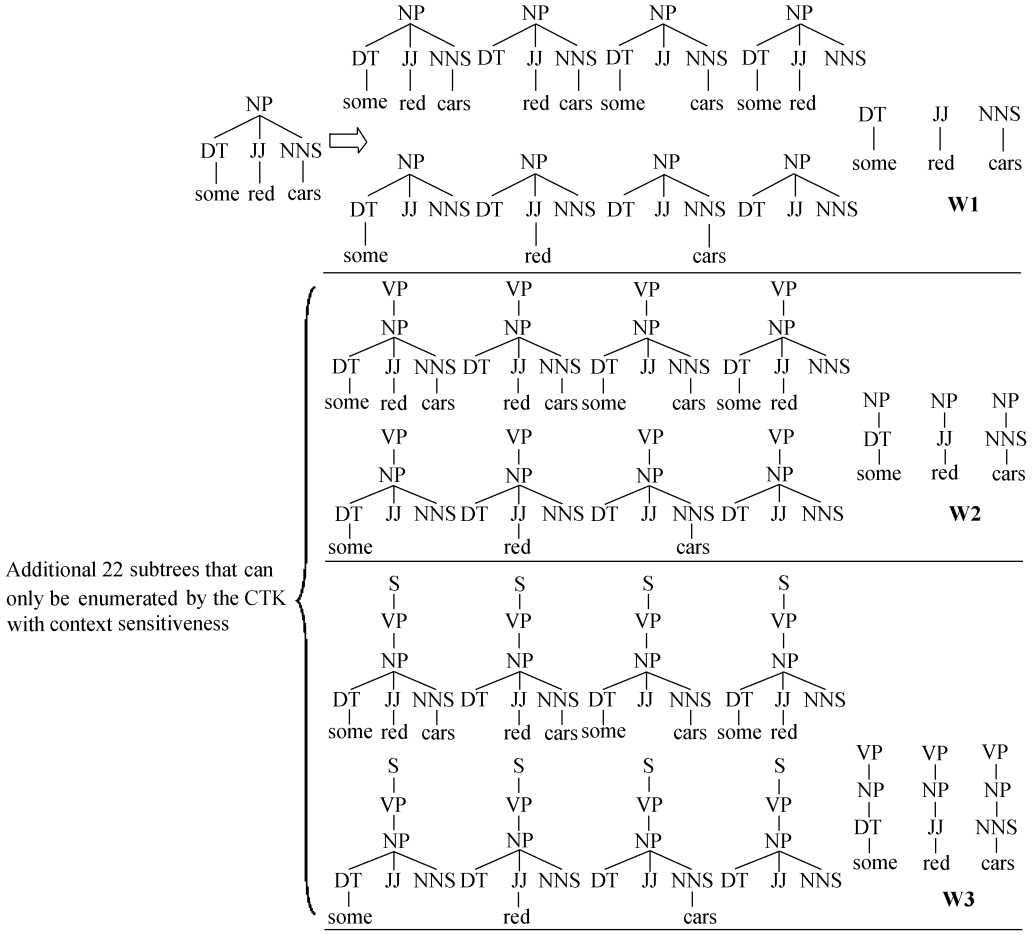

Fig.2. Syntactic parse tree and all of its 33 sub-trees covered by CTK with context-sensitiveness (assuming that the NP node has the parent node VP and the grandparent node S).

$$\Delta(n_1, n_2) = \lambda \cdot \lambda_1^{\#InsDels} \cdot \lambda_2^{\#Subs} \cdot \prod_{k=1}^{\#ch(x_1)} (1 + \Delta(ch(x_1, k), ch(x_2, k))) \quad (4')$$

where

1) *#InsDels* is the number of insertions/deletions of optional tree nodes and *#Subs* is the number of substitutions between similar tree nodes while λ_1 and λ_2 $(1 < \lambda_1, \lambda_2 < 1)$ are their weights;

2) x_j is the best matched child sequence between n_1 and n_2 of length $\#ch(x_j)$ with the approximate matching mechanism. Here, $ch(x_j, k)$ is the k-th child of x_j.

For example, Fig.3 illustrates a parse tree "NP→some/DT red/JJ cars/NNS" with all of its 28 sub-trees covered by the CTK with approximate matching, assuming that NN and NNS, the part-of-speech tags for singular noun and plural noun, are considered similar and allow approximate matching between them.

Similarly, we can modify $(2')$ in computing $\Delta(n_1[1,i], n_2[1,i])$ to further allow approximate matching in the CTK with context sensitiveness:

$$\Delta(n_1[1,i], n_2[1,i]) = \lambda \cdot \lambda_1^{\#InsDels} \cdot \lambda_2^{\#Subs} \cdot \prod_{k=1}^{\#ch(x_1[1,i])} (1 + \Delta(ch(x_1[1,i], k), ch(x_2[1,i], k))) \quad (4'')$$

where $x_j[1,i]$ is the best matched child sequence between $n_1[1,i]$ and $n_2[1,i]$ of length $\#ch(x_j[1,i])$ with the approximate matching mechanism.

Obviously, the number of insertions/deletions/substitutions in $(4')$ and $(4'')$ can be easily determined by computing the least edit distance between two productions by maximizing $\lambda_1^{\#InsDels} \cdot \lambda_2^{\#Subs}$ using dynamic programming. This is quite different from the complicated dynamic programming algorithm as proposed in [15] by Zhang *et al.*, which is motivated by the partial matching algorithm as proposed in [16] by Moschitti. Obviously, such maximization results in a proper kernel since it only attempts to achieve maximization at the child level and is not a recursive match.

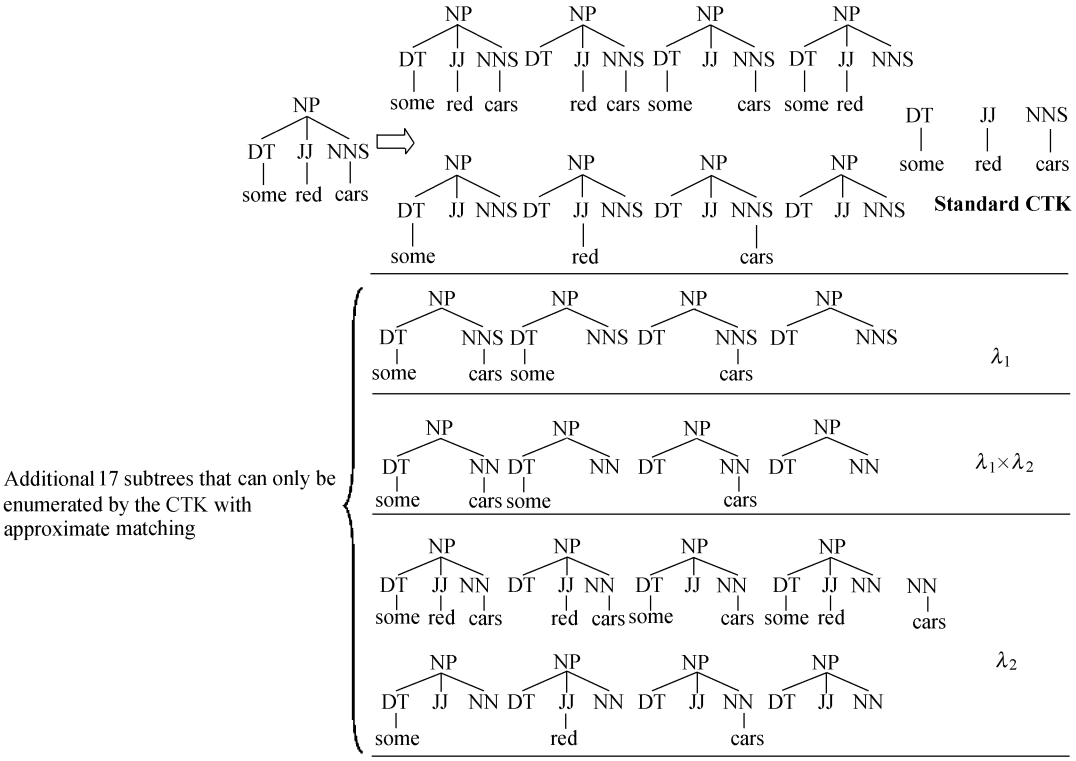

Fig.3. Syntactic parse tree and all of its 28 substructures covered by CTK with approximate matching (assuming that NN and NNS, the part-of-speech tags for singular noun and plural noun, are considered similar).

In particular, all the children of a node are optional tree nodes except its head child since head children normally contain critical information and should be similar for two productions to be matched (either exactly or approximately) while substitutions only apply between similar tree nodes, e.g., the part-of-speech tags (POSs) for mass nouns and plural nouns. This means insertions and deletions only apply to non-head children while substitutions only happen in the same group of similarly functioned tree nodes.

4 Parse Tree Structure

Since a relation instance between two entities is encapsulated in a parse tree and thus can be represented by (a portion of) the parse tree, it is critical to understand which portion of a parse tree is necessary and important for RDC in the tree kernel computation. For example, Zhang *et al.*[10] systematically explored five parse tree structures and found that the widely-used Shortest Path-enclosed Tree (SPT) performed best and that improper inclusion of more structural information does not improve the performance. In this paper, we base our parse tree structure on the SPT with the expanded predicate path (SPTX, in short), which has been proven to be superior over the SPT in tree kernel-based RDC[12], as our default parse tree structure.

4.1 SPTX

The parse tree structure, SPTX, as proposed in [12] by Zhou *et al.*, is motivated by two observations on 100 relation instances randomly sampled from the ACE RDC 2003 corpus.

1) The parse trees encapsulating semantic relations can be classified into 5 categories: a) embedded (37%), where one entity is embedded in another entity, e.g., "John" and "John's wife"; b) PP-linked (21%), where one entity is linked to another entity via PP attachment, e.g., "CEO" and "Microsoft" in the sentence "CEO of Microsoft announced ...'; c) semi-structured (15%), where the sentence consists of a sequence of noun phrases (including the two given entities), e.g., "Jane" and "ABC news" in the sentence "Jane, ABC news, California."; d) descriptive (7 instances), e.g., the relationship between "the US president" and "Barack Obama" in the sentence "the US president Barack Obama declared last Friday that ..."; e) predicate-linked and others (19%, including coordinated cases), where the predicate information is necessary to determine the relationship between two entities, e.g., "John" and "Mary" in the sentence "John and Mary got married last Friday."

2) The parse trees belonging to the "predicate-linked" category vary much syntactically and majority

(∼70%) of them need information outside SPT while it is safe (> 90%) to use SPT as the tree span for the other categories.

The idea behind SPTX is that the necessary parse tree span for a semantic relation should be determined dynamically according to its parse tree category and context. Given a parse tree and two entities in consideration, the parse tree category is first determined and then the parse tree structure is expanded accordingly. By default, SPT is adopted and the parse tree is only expanded when it belongs to the "predicate-linked" category, where the predicate information is necessary to determine the relationship between two entities, e.g., "John" and "Mary" in the sentence "John and Mary got married last Friday." In particular, the expansion is done by first moving up until a predicate-headed phrase is found and then moving down along the predicated-headed path to the predicate terminal node. Fig.4 shows an example for the predicate-linked category where the lines with arrows indicate the predicate expansion path.

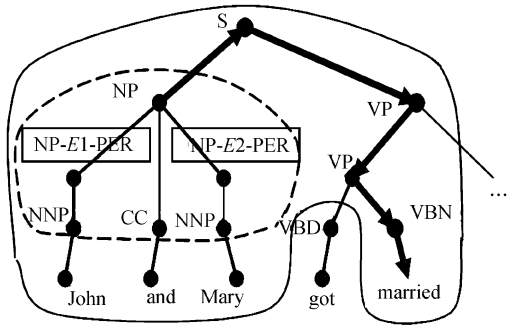

Fig.4. An example: SPTX (inside the solid line) vs. SPT (inside the dashed line). Note that $E1$ and $E2$ refer to the first and second mentions respectively with their entity classes (e.g., PER for person) attached.

4.2 Enriched SPTX

One problem with both SPT and SPTX is that they allow wrong matching between different portions of two parse trees when computing their similarity using a CTK, which counts the number of common sub-trees as the syntactic structure similarity between two parse trees. In order to avoid this problem, this paper proposes a simple but efficient distinguishing strategy to avoid wrong matching across different portions of two parse trees by explicitly distinguishing different portions using unique portion labels. In particular, we differentiate the SPTX into four portions: the first mention (A), the second mention (B), the shortest path between the two mentions (C), and the expanded predicate path for the predicate-linked category (D).

Fig.5 illustrates an enriched parse tree with unique portion labels for two mentions "John" and "Mary" in the sentence "John and Mary got married last Friday", where the additive labels "A", "B", "C" and "D" indicate the above four different portions, respectively. For example, "S-D" means that the "S" node occurs in the expanded path for the predicate-linked category.

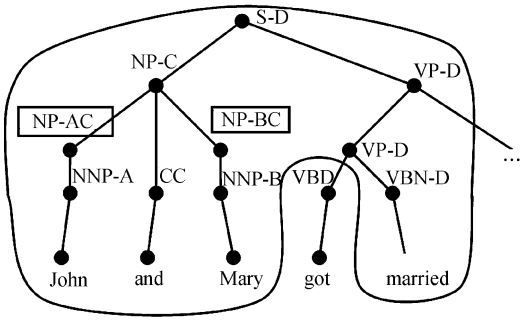

Fig.5. An example: enriched SPTX with unique portion labels (inside the solid line). Note that the entity class information is omitted.

In this way, a sub-tree in a particular portion (or a combined portion) of a parse tree will be only matched with a sub-tree in the same portion (or the same combined portion) of another parse tree. For example, a sub-tree in the first mention of a parse tree will be only matched with a sub-tree in the first mention of a parse tree while another sub-tree which covers the first mention and the shortest path portion of a parse tree will be only matched with another sub-tree which covers the first mention and the shortest path portion of another parse tree.

Another enrichment employed in this paper is motivated by recent progress in un-lexicalized parsing[13,17-18], which enriches basic non-terminal tags with some latent annotations. In the latent annotation scenario, the observed treebank is thought to be a coarse trace of a finer, unobserved grammar. For example, the single treebank non-terminal category NP may be better modeled by several finer categories representing subject NPs, object and so on. As a representative, the Berkeley parser (http://nlp.cs.berkeley.edu/Main.html#Parsing) adopted a hierarchical split-and-merge strategy[13] to enrich basic non-terminal tags with latent annotations and achieved better performance than the state-of-the-art fully lexicalized ones. Given an original Penn-style parse tree, we first employ the PCFGLA.TreeLabeler class in the Berkeley parser toolkit to get the most likely Berkeley-style parse tree (i.e., the binary X-bar parse tree with latent annotations) and then collapse it to a Penn-style parse tree with latent annotations by deleting the X-bar nodes. Such pruning of X-bar

nodes is error-free since the PCFGLA.TreeLabeler class in the Berkeley parser toolkit only transforms an original Penn-style parse tree into a binary X-bar parse tree by applying a left branching binarization scheme to introduce new cascaded X-bar nodes and annotates an additional numeric label to each parse tree node. Fig.6 shows an example of enriching a Penn-style parse tree with latent annotations using the Berkeley parser while Fig.7 shows the enriched SPTX with latent annotations.

5 Experimentation

This paper uses the official ACE RDC 2003 and 2004 corpora provided by LDC in all our experiments. The ACE RDC corpora are gathered from various newspapers, newswire and broadcasts. In this paper, we only measure the performance on "true" mentions (i.e., as annotated by LDC annotators) for fair comparison with related work in the literature. Moreover, we only model explicit relations because of poor inter-annotator agreement in the annotation of implicit relations and their limited number, as adopted in related work.

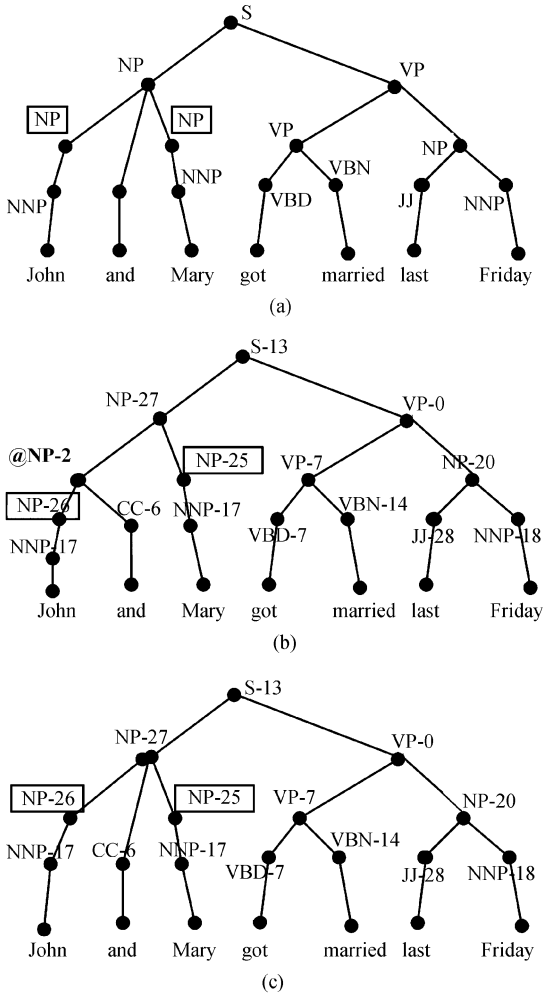

Fig.6. An example: the process of enriching the SPTX with latent annotations, using the Berkeley parser. Here @NP-2 indicates an X-bar node (with a latent label) and is thus pruned. (a) Original Penn-style parse tree without latent annotations. (b) Binary X-bar Berkeley-style tree with latent annotations, using the Berkeley parser. Here, @ indicates an X-bar node and the number after "-" indicates a latent label, e.g., @NP-2. (c) Final Penn-style tree with latent annotations.

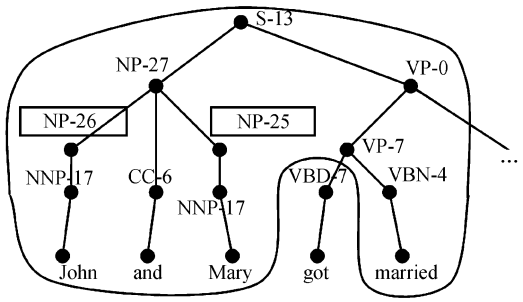

Fig.7. An example: enriched SPTX with latent annotations (inside the solid line). Here, the entity class information is omitted due to space limitation and the additive numbers denote the latent annotations of subcategories.

5.1 Experimental Setting

The 2003 corpus is divided into a training set and a test set. The training set consists of 674 annotated text documents (∼300 k words) and 9683 instances of relations. During development, 519 documents in the training set are used for training while the remaining 155 (674∼519) documents are set aside for fine-tuning all the free parameters in the systems. The test set is held out only for final evaluation. It consists of 97 documents (∼50 k words) and 1386 instances of relations. The 2003 corpus defines 5 relation types and 24 subtypes between 5 entity types, i.e., person, organization, location, facility and GPE. Table 1 lists the types and subtypes of relations in the ACE RDC 2003 corpus, along with their frequency of occurrence in the training data. It is observed that the 2003 corpus suffers from a small amount of annotated data for a few subtypes such as the subtype "Founder" under the type "ROLE". It is also observed that the 2003 task defines some difficult subtypes such as the subtypes "Based-In", "Located" and "Residence" under the type "AT", which are difficult even for human experts to differentiate.

Compared with the 2003 corpus, the 2004 corpus defines two more entity types (i.e., weapon and vehicle), much more entity subtypes, and different 7 relation types and 23 subtypes between 7 entity types. It contains 451 documents and 5702 relation instances. For fair comparison with related work in the literature, we

Table 1. Statistics of Relation Types and Subtypes in the ACE RDC 2003 Corpus with Their Occurring Frequency in the Training Data

Type	Subtype
AT (2781)	Based-In (347); Located (2126); Residence (308)
NEAR (201)	Relative-Location (201)
PART (1298)	Part-Of (947); Subsidiary (355); Other (6)
ROLE (4756)	Affiliate-Partner (204); Citizen-Of (328); Client (144); Founder (26); General-Staff (1331); Management (1242); Member (1091); Owner (232); Other (158)
SOCIAL (827)	Associate (91); Grandparent (12); Other-Personal (85); Other-Professional (339); Other-Relative (78); Parent (127); Sibling(18); Spouse(77)

only use the BNEWS and NWIRE portions of the corpus, which contain 348 documents (∼125 k words) and 4305 relation instances. Table 2 lists the types and subtypes of relations in the ACE RDC 2004 corpus, along with their occurring frequency in the BNEWS and NWIRE portions. Evaluation was done using 5-fold cross-validation. For efficiency, we use the same free parameters fine-tuned from the 2003 corpus.

In this paper, both corpora are parsed using Charniak's parser[19] and we iterate over all pairs of entity mentions occurring in the same sentence to generate potential relation instances. We also explicitly model the argument order of the two mentions involved. For example, when comparing mentions m_1 and m_2, we distinguish between m_1-ROLE.Citizen-Of-m_2 and m_2-ROLE.Citizen-Of-m_1. Note that, in the 2003 corpus, 6 of these 24 relation subtypes are symmetric: "NEAR.Relative-Location", "SOCIAL.Associate", "SOCIAL.Other-Relative", "SOCIAL.Other-Professional", "SOCIAL.Sibling", and "SOCIAL.Spouse". In this way, we model RDC in the 2003 corpus as a multi-class classification task with 43 $(24 \times 2 - 6 + 1)$ classes, two for each relation subtype (except the above 6 symmetric subtypes) and a "NONE" class for the case where the two mentions are not related. For the 2004 corpus, 6 of these 23 relation subtypes are symmetric: "PHYS.Near", "PER-SOC.Business", "PER-SOC.Family", "PER-SOC.Other", "EMP-ORG.Partner", and "EMP-ORG.Other". In this way, we model RDC in the 2004 corpus as a multi-class classification task with 41 $(23 \times 2 - 6 + 1)$ classes, two for each relation subtype (except the above 6 symmetric subtypes) and a "NONE" class for the case where the two mentions are not related.

Table 2. Statistics of Relation Types and Subtypes in the ACE RDC 2004 Corpus with Their Occurring Frequency in the BNEWS and NWIRE Portions

Type	Subtype
PHYS (1203)	Located (738); Near (87); Part-Whole (378)
PER-SOC (349)	Business (173); Family (121); Other (55)
EMP-ORG (1595)	Employ-Executive (489); Employ-Staff (539); Employ-Undetermine (78); Member-of-Group (191); Subsidiary (206); Partner (12); Other (80)
ART (211)	User-or-Owner (200); Inventor-or-Man (9); Other (2)
OTHER-AFF (141)	Ethic (39); Ideology (48); Other (54)
GPE-AFF (527)	Citizen-or-Residence (273); Based-In (215); Other (39)
DISC (279)	DISC (279)

Many classifiers, such as SVM, KNN and voted perceptrons, can be used with kernels by replacing the dot product with a kernel function. In this paper, we select SVM as the classifier since SVM represents the state-of-the-art method in the machine learning research community, and there are good implementations of the algorithm available. In our implementation, we use the binary-class SVM-Light developed by Joachims[20]. SVM is a supervised machine learning technique motivated by the statistical learning theory[3]. Based on the structural risk minimization of the statistical learning theory, SVM seeks an optimal separating hyper-plane to divide the training examples into two classes and make decisions based on support vectors that are selected as the only effective instances in the training set. Basically, SVM is a binary classifier. Therefore, we must extend SVM to multi-class (e.g., K) classification for the ACE relation extraction task. For efficiency, we apply the one vs. other strategy, which builds K classifiers so as to separate one class from all others, instead of the pairwise strategy, which builds $K \times (K-1)/2$ classifiers considering all pairs of classes. The final decision of an instance in the multiple binary classification is determined by the class which has the maximal SVM output.

For approximate matching, we only consider four groups of similar tree nodes: 1) adjective group: ADJP, JJ, JJR, JJS; 2) adverb group: ADVP, RB, RBR, RBS; 3) noun group: NP, NN, NNS, NNP, NNPS, NAC, NX; 4) verb group: VP, VB, VBD, VBG, VBP, VBZ, VBN (in the active voice only). Since voice information is very indicative in distinguishing subjects and objects, VBNs and VPs in the passive voice are determined heuristically and differentiated from other verbs. Moreover, latent annotations derived from the Berkeley parser are ignored in approximate matching only. That

is, any two tree nodes in the same group are considered similar, even with different latent annotations.

Due to space limitation, we only report our performance on the detection and classification of the relation types in both corpora, using Precision/Recall/F1-measure. For computational complexity, we compare the running time (including training and testing time) of different settings on a PC with a 2.13 GHz Intel Core 2 CPU and 2.0 GB memory. Since the experiments on the 2004 corpus are done by 5-fold cross-validation, the given running time of each experiment on the 2004 corpus is averaged per fold. To see whether an improvement is significant, we also conduct statistical significance testing using paired t-test. In this paper, "$\ggg$", "$\gg$" and "$>$" denote p-values of an improvement smaller than 0.01, in-between (0.01, 0,05] and bigger than 0.05, which mean significantly better, moderately better and slightly better, respectively.

5.2 Experimental Results

Table 3 compares various parse tree structures, using the standard CTK with the SVM-Light regularization parameter C and the CTK decay factor λ fine-tuned to 2.4 and 0.35 (thereafter in this paper) respectively. It shows the following.

1) SPTX (Shortest Path-enclosed Tree with the eXpanded predicate path) outperforms SPT (Shortest Path-enclosed Tree) by about 1.0 ($\gg$) in F1-measure on the 2003 and 2004 corpora respectively, at the expense of about 20% more running time. This is due to the special treatment of the predicate-linked category in SPTX by expanding the predicate-headed path into SPT.

2) Distinguishing different portions of SPTX with unique portion labels much improves the performance by 1.5 ($\gg$) and 1.6 ($\gg$) in F1-measure on the 2003 and 2004 corpora respectively (SPTX with Unique Portion Labels vs. SPTX). This suggests that the distinguishing strategy can better model the substructure similarity of two parse trees. This is due to the fact that the distinguishing strategy only not can avoid wrong matching between different portions but also capture the substructure similarity across different portions. It is also interesting to note that the distinguishing strategy can decrease the running time by about 10%. This is due to the fact that the distinguishing strategy can effectively avoid wrong matching between different portions of two parse trees.

3) Enriching SPTX with latent annotations improves the performance by 2.6 ($\ggg$) and 2.4 ($\ggg$) in F1-measure on the 2003 and 2004 corpora respectively (SPTX with latent annotations vs. SPTX). This is due to the fact that latent annotations convey much useful structural information in a parse tree. This validates the usefulness of learning more latent annotations in a parse tree in the future work.

4) Enriched SPTX with unique portion labels and latent annotations outperforms the SPTX by 3.7 ($\ggg$) and 3.5 ($\ggg$) in F1-measure on the 2003 and 2004 corpora respectively.

Table 4 compares various CTKs using the enriched SPTX with unique portion labels and latent annotations and, in particular, evaluates the contributions of equipping the standard CTK with context-sensitiveness and approximate matching. It shows the following.

Table 3. Comparison of Parse Tree Structures Using the Standard CTK on the ACE RDC 2003 (outside the parentheses) and 2004 (inside the parentheses) Corpora (Note that the running time includes both training and testing.)

Tree Kernel	Parse Tree Structure	Precision (%)	Recall (%)	F1	Time (min)
Standard CTK	SPT	75.1 (75.9)	64.1 (67.1)	68.9 (71.3)	98 (29)
	SPTX	75.3 (76.1)	65.4 (69.0)	69.8 (72.3)	119 (34)
	Enriched SPTX with unique portion labels	77.4 (78.4)	66.1 (69.7)	71.3 (73.9)	108 (31)
	Enriched SPTX with latent annotations	78.5 (79.0)	67.1 (70.8)	72.4 (74.7)	110 (32)
	Enriched SPTX with unique portion labels and latent annotations	80.6 (80.7)	67.5 (71.5)	73.5 (75.8)	115 (33)

Table 4. Comparison of Different CTKs Using Enriched SPTX with Unique Portion Labels and Latent Annotations on the ACE RDC 2003 (outside the parentheses) and 2004 (inside the parentheses) Corpora (Note that the running time includes both training and testing.)

Tree Kernel	Parse Tree Structure	Precision (%)	Recall (%)	F1	Time (min)
Standard CTK	Enriched SPTX with unique portion labels and latent annotations	80.6 (80.7)	67.5 (71.5)	73.5 (75.8)	115 (33)
CTK with Context-Sensitiveness		82.5 (82.9)	68.1 (71.8)	74.6 (77.0)	126 (35)
CTK with Approx. Matching		81.6 (82.1)	69.2 (72.7)	74.9 (77.1)	172 (46)
CTK with Context-Sensitiveness and Approx. Matching		82.7 (83.0)	69.8 (72.9)	75.7 (77.6)	194 (52)

1) Context sensitiveness alone (CTK with Context Sensitiveness vs. Standard CTK) increases the performance by 1.1 ($\gg$) and 1.2 ($\gg$) in F1-measure on the 2003 and 2004 corpora respectively at the expense of only 10% more running time. This is due to the fact that $\Delta(n_1[1], n_1[2]) = 0$ holds for the majority of context-free sub-tree pairs[11] and the computation for context-sensitive sub-tree pairs is necessary only when $\Delta(n_1[1], n_1[2]) \neq 0$. For an detailed explanation, please refer to [21] by Moschitti. Here, m is fine-tuned to 3 while w_1, w_2 and w_3 are fine-tuned to 1.0, 0.40 and 0.15 respectively. This suggests that consideration of context-sensitive sub-trees is useful in kernel-based RDC and does not much increase the computational complexity. This also suggests that the parent and grandparent nodes of a sub-tree contain much information for RDC while considering more ancestral nodes may not help. This may be due to the fact that, although our statistics on the 2003 training data indicate that more than 80% (on average) of sub-trees has a root node path longer than 3 (since most of the subtrees are deep from the root node and it is observed that more than 90% of the parsed trees in the training data are deeper than 6 levels), and sub-trees with a root node path longer than 3 may be prone to the full parsing errors and thus have negative impact. Further evaluation shows that different weighting of contex-sensitive sub-trees with different root node path lengths contributes about 0.5 ($>$) more in F1-measure, compared with equal weighting.

2) Approximate matching alone (CTK with Approximate Matching vs. Standard CTK) increases the performance by 1.4 ($\gg$) and 1.3 ($\gg$) in F1-measure on the 2003 and 2004 corpora respectively. This suggests that approximate matching in terms of tree node insertion/deletion/substitution is very useful in capturing the similarity between two sub-trees. However, such improvements are at the cost of about 50% more running time, largely due to dynamic programming in computing the least edit distance between two productions to achieve approximate matching. In this paper, λ_1 (for insertion and deletion) and λ_2 (for substitution) in ($2'$) are fine-tined to 0.6 and 0.4 respectively.

3) Integration of both context sensitiveness and approximate matching into the standard CTK (CTK with Context-Sensitiveness and Approx. Matching vs. Standard CTK) increases the performance by 2.2 ($\ggg$) and 1.8 ($\ggg$) in F1-measure on the 2003 and 2004 corpora respectively. It is also interesting to note that the enriched SPTX over the SPTX contributes much more than the new CTK with context-sensitiveness and approximate matching over the standard CTK.

Finally, Table 5 compares our system with the state-of-the-art ones. Please note that, a) all the systems adopt the same experimental setting; and b) the performance figures from the state-of-the-art ones are extracted from the referred papers except the ones for [6] by Zhou *et al.* on the 2004 corpus are re-produced. It shows that our tree kernel method much outperforms previous best tree kernel[12] by 4.7 and 4.4 in F1-measure on the 2003 and 2004 corpora respectively. It also shows that our tree kernel method much outperforms previous best feature-based method[6] by 7.7 and 7.5 in F1-measure on the 2003 and 2004 corpora respectively. This proves the great potential of syntactic structural information in RDC.

Table 5. Comparison of Difference Methods on the ACE RDC 2003 (outside the parentheses) and 2004 (inside the parentheses) Corpora (Note that our CTK refers to the last row of Table 4, i.e., the CTK with context-sensitiveness and approximate matching and enriched SPTX with unique portion labels and latent annotations.)

Systems	Precision (%)	Recall (%)	F1
Our CTK	82.7	69.8	75.7
	(83.0)	(72.9)	(77.6)
Zhou *et al.*[12]:	80.1	63.8	71.0
Tree Kernel	(81.1)	(66.7)	(73.2)
Zhang *et al.*[10]:	76.1	62.6	68.7
Tree Kernel	(72.5)	(56.7)	(63.6)
Zhou *et al.*[6]:	77.2	60.7	68.0
Feature-Based Linear Kernel	(80.8)	(61.9)	(70.1)

6 Conclusion

It is well known that structural information in a parse tree plays a critical role in many NLP applications (including RDC) and that kernel methods have the potential in effectively capturing structured knowledge. This paper significantly advances kernel-based RDC by a) integrating the advantages of the state-of-the-art tree kernels and equipping the standard CTK with context-sensitiveness and approximate matching, and b) proposing a new parse tree structure via a distinguishing strategy and a latent annotation scheme.

For the future work, we will explore more effective ways of representing the semantic relation structures in RDC and new kernels in better modeling such structures. Moreover, we will apply our new kernel method in other NLP applications, such as semantic role labeling. Finally, the coefficient in the kernel composition is chosen by cross-validation, which is only feasible for a small number of kernels. Therefore, it will be worthwhile to explore automatic coefficient learning, e.g., multiple kernel learning.

References

[1] Automatic Content Extraction. http://www.ldc.upenn.edu/Projects/ACE/, 2000-2007.

[2] Haussler D. Convolution kernels on discrete structures. Technical Report UCS-CRL-99-10, University of California, Santa Cruzca, USA, 1999.

[3] Vapnik V. Statistical Learning Theory. Chichester: Whiley, UK, 1998.

[4] Kambhatla N. Combining lexical, syntactic and semantic features with Maximum Entropy models for extracting relations. In *Proc. ACL 2004*, Barcelona, Spain, Jul. 21-26, 2004, pp.178-181.

[5] Zhao S B, Grishman R. Extracting relations with integrated information using kernel methods. In *Proc. ACL 2005*, Ann Arbor, USA, Jun. 25-30, 2005, pp.419-426.

[6] Zhou G D, Su J, Zhang J, Zhang M. Exploring various knowledge in relation extraction. In *Proc. ACL 2005*, Ann Arbor, USA, Jun. 25-30, 2005, pp.427-434.

[7] Zelenko D, Aone C, Richardella. Kernel methods for relation extraction. *Journal of Machine Learning Research*, 2003, 3: 1083-1106.

[8] Culotta A, Sorensen J. Dependency tree kernels for relation extraction. In *Proc. ACL 2004*, Barcelona, Spain, Jul. 21-26, 2004, pp.423-429.

[9] Bunescu R, Mooney R J. A shortest path dependency kernel for relation extraction. In *Proc. HLT/EMNLP 2005*, Vancouver, Canada, Oct. 6-8, 2005, pp.724-731.

[10] Zhang M, Zhang J, Su J, Zhou G D. A composite kernel to extract relations between entities with both flat and structured features. In *Proc. COLING-ACL 2006*, Sydney, Australia, Jul. 17-21, 2006, pp.825-832.

[11] Collins M, Duffy N. Convolution kernels for natural language. In *Proc. NIPS 2001*, Vancouver, Canada, Dec. 3-8, 2001, pp.625-632.

[12] Zhou G D, Zhang M, Ji D H, Zhu Q M. Tree kernel-based relation extraction with context-sensitive structured parse tree information. In *Proc. EMNLP-CoNLL 2007*, Prague, Czech, Jun. 28-30, 2007, pp.728-736.

[13] Petrov S, Barrett L, Thibaux R, Klein D. Learning accurate, compact and interpretable tree annotation. In *Proc. ACL 2006*, Sydney, Australia, Jul. 17-21, 2006, pp.433-440.

[14] Moschitti A. A study on convolution kernels for shallow parsing. In *Proc. ACL 2004*, Barcelona, Spain, Jul. 21-26, 2004, pp.335-342.

[15] Zhang M, Che W X, Aw A T, Tan C L, Zhou G D, Liu T, Li S. A grammar-driven convolution tree kernel for semantic role classification. In *Proc. ACL 2007*, Prague, Czech, Jun. 23-30, 2007, pp.200-207.

[16] Moschitti A. Efficient convolution kernels for dependency and constituent syntactic trees. In *Proc. ECML 2006*, Berlin, Germany, Sept. 18-22, 2006, pp.318-329.

[17] Klein D, Manning C D. Accurate unlexicalized parsing. In *Proc. ACL 2003*, Sapporo, Japan, Jul. 7-12, 2003, pp.423-430.

[18] Petrov S, Klein D. Discriminative log-linear grammars with latent variables. In *Proc. NIPS 2008*, Vancouver, Canada, Dec. 8-13, 2008, pp.1-8.

[19] Charniak E. Immediate-head parsing for language models. In *Proc. ACL 2001*, Tonlouse, France, Jul. 9-11, 2001, pp.129-137.

[20] Joachims T. Text categorization with support vector machine: Learning with many relevant features. In *Proc. ECML 1998*, Chemnitz, Germany, Apr. 21-23, 1998, pp.137-142.

[21] Moschitti A. Making tree kernels practical for natural language learning. In *Proc. EACL 2006*, Trento, Italy, Apr. 3-7, 2006, pp.113-120.

[22] Zhang M, Che W X, Zhou G D, Aw A T, Tan C L, Liu T, Li S. Semantic role labeling using a grammar-driven convolution tree kernel. *IEEE Transaction on Audio, Speech and Language Processing*, 2008, 16(7): 1315-1329.

Guo-Dong Zhou received 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 associate scientist, scientist and 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. He is a senior member of CCF and has been the member of ACM and IEEE since 1999. Currently, he serves as an editorial board member of Computational Linguistics and an associate editor of ACM Transaction on Asian Language Information Processing.

Qiao-Ming Zhu received his Ph.D. degree from Soochow University, Suzhou, China, in 2008. Currently, he is a professor at the university and acts as the deputy director of Department of Science, Technology and Industry. His research interests include natural language processing, information extraction and embedded systems. He is a senior member of CCF.