Discovering Relations Between Named Entities from a Large Raw Corpus Using Tree Similarity-Based Clustering

Min Zhang[1], Jian Su[1], Danmei Wang[1,2], Guodong Zhou[1], and Chew Lim Tan[2]

[1] Institute for Infocomm Research,
21 Heng Mui Keng Terrace, Singapore 119613
{mzhang, sujian, stuwang, zhougd}@i2r.a-star.edu.sg
[2] Department of Computer Science,
National University of Singapore,
Singapore, 117543
tancl@comp.nus.edu.sg

Abstract. We propose a tree-similarity-based unsupervised learning method to extract relations between Named Entities from a large raw corpus. Our method regards relation extraction as a clustering problem on shallow parse trees. First, we modify previous tree kernels on relation extraction to estimate the similarity between parse trees more efficiently. Then, the similarity between parse trees is used in a hierarchical clustering algorithm to group entity pairs into different clusters. Finally, each cluster is labeled by an indicative word and unreliable clusters are pruned out. Evaluation on the New York Times (1995) corpus shows that our method outperforms the only previous work by 5 in F-measure. It also shows that our method performs well on both high-frequent and less-frequent entity pairs. To the best of our knowledge, this is the first work to use a tree similarity metric in relation clustering.

1 Introduction

The relation extraction task identifies various semantic relations such as location, affiliation, revival and so on between entities from text. For example, the sentence "George Bush is the president of the United States." conveys the semantic relation "President", between the entities "George Bush" (*PERSON*) and "the United States" (*GPE*[1]). The task of relation extraction was first introduced as part of the Template Element task in MUC6 and formulated as the Template Relation task in MUC7 [1]. Most work at MUC [1] was rule-based, which tried to use syntactic and semantic patterns to capture the corresponding relations by means of manually written linguistic rules. The major drawback of this method is the poor adaptability and the poor robustness in handling large-scale or new domain data due to two reasons. First, rules have to be rewritten for different tasks or when porting to different domains. Second, generating rules manually is quite labor- and time-consuming.

[1] GPE is an acronym introduced by the ACE (2004) program to represent a Geo-Political Entity --- an entity with land and a government.

Since then, various supervised learning approaches [2,3,4,5] have been explored extensively in relation extraction. These approaches automatically learn relation patterns or models from a large annotated corpus. To decrease the corpus annotation requirement, some researchers turned to weakly supervised learning approaches [6,7], which rely on a small set of initial seeds instead of a large annotated corpus. However, there is no systematic way in selecting initial seeds and deciding an "optimal" number of them.

Alternatively, Hasegawa et al. [8] proposed a cosine similarity-based unsupervised learning approach for extracting relations from a large raw corpus. The context words in between the same entity pairs in different sentences are used to form word vectors, which are then clustered according to the cosine similarity. This approach does not rely on any annotated corpus and works effectively on high-frequent entity pairs [8]. However, there are two problems in this approach:

- The assumption that the same entity pairs in different sentences have the same relation.
- The cosine similarity measure between the flat feature vectors, which only consider the words between entities.

In this paper, we propose a tree similarity-based unsupervised learning approach for relation extraction. In order to resolve the above two problems in Hasegawa et al. [8], we assume that the same entity pairs in different sentences can have different relation types. Moreover, rather than the cosine similarity measure, a similarity function over parse trees is proposed to capture much larger feature spaces instead of the simple word features.

The rest of the paper is organized as follows. In Section 2, we discuss the proposed tree-similarity-based clustering algorithm. Section 3 shows the experimental result. Section 4 compares our work with the previous work. We conclude our work with a summary and an outline of the future direction in Section 5.

2 Tree Similarity-Based Unsupervised Learning

We use the shallow parse tree as the representation of relation instances, and regard relation extraction as a clustering problem on shallow parse trees. Our method consists of three steps:

1) Calculating the similarity between two parse trees using a tree similarity function;
2) Clustering relation instances based on the similarities using a hierarchical clustering algorithm;
3) Labeling each cluster using indicative words as its relation type, and pruning out unreliable clusters.

In this section, we introduce the parse tree representation for a relation instance, define the tree similarity function, and describe the clustering algorithm.

2.1 Parse Tree Representation for Relation Instance

A parse tree T is a set of node $\{p_1 \ldots p_n\}$, which are connected hierarchically. Here, a node p_i includes a set of features $\{f_1, \ldots, f_4\}$ as follows:

- **Head Word (f_1):** for a leaf (or terminal) node, it is the word itself of the leaf node; for a non-terminal node, it is a "**Head Word**" propagated from a leaf node. This feature defines the main meaning of the phrase or the sub-tree rooted by the current node.
- **Node Tag (f_2):** for a leaf node, it is the part-of-speech of this node; for a non-terminal node, it is a phrase name, such as Noun Phrase (NP), Verb Phrase (VP). This feature defines the linguistic category of this node.
- **Entity Type (f_3)[2]:**it indicates the entity type which can be PER, COM or GPE if the current node refers to a Named Entity.
- **Relation Order (f_4):** it is used to differentiate asymmetric relations, e.g., "A belongs to B" or "B belongs to A".

These features are widely-adopted in Relation Extraction task. In the parse tree representation, we denote by $p_i.f_j$ the j^{th} feature of node p_i, by $p_i[j]$ the j^{th} child of node p_i, and by $p_i[\boldsymbol{C}]$ the set of all children of node p_i, *i.e.*, $p_i[j] \in p_i[\boldsymbol{C}]$.

2.2 Tree Similarity Function

Inspired by the special property of kernel-based methods[3], we extend the tree kernels in Zelenko et al. [3] to a novel tree similarity measure function, and apply the above tree similarity function to unsupervised learning for relation extraction. Mostly, in previous work, kernels are used in supervised learning algorithms such as SVM, Perceptron and PCA (Collins and Duffy, 2001). In our approach, the hierarchical clustering algorithm is adopted, this allows us to explore more robust and powerful similarity functions, other than a proper kernel function[4].

A similarity function returns a normalized, symmetric similarity score in the range [0, 1]. Especially, our tree similarity function $K(T_1, T_2)$ over two trees T_1 and T_2, with the root nodes r_1 and r_2, is defined as follows:

$$K(T_1, T_2) = m(r_1, r_2) * \{s(r_1, r_2) + K_C(r_1[\boldsymbol{c}], r_2[\boldsymbol{c}])\} \tag{1}$$

where,

[2] For the features of "**Entity Type**", please refer to the literature ACE [22] for details.

[3] As an alternative to the feature-based method [5], the advantage of kernels [9] is that they can replace any dot product between input points in a high dimensional feature space. Compared with the feature-based method, the kernel method displays several unique characteristics, such as implicitly mapping the feature space from low-dimension to high-dimension, and effectively modeling structure data. A few kernels over structured data have been proposed in NLP study [10-16]. Zelenko et al. [3] and Culotta et al. [4] explored tree kernels with SVM [9] for relation extraction. We study the tree kernels from similarity measure viewpoints.

[4] A function is a kernel function if and only if the function is *symmetric* and *positive semi-definite* [3, 9].

- $m(p_i, p_j)$ is a matching function over the features of two tree nodes p_i and p_j.

  In this paper, only the node tag feature (f_2) is considered:

$$m(p_i, p_j) = \begin{cases} 1 & \text{if } p_i.f_2 = p_j.f_2 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

  The binary function (1) means that two nodes are matched only if they share the same **Node Tag**.

- $s(p_1, p_2)$ is a similarity function between two nodes p_i and p_j:

$$s(p_i, p_j) = \begin{cases} 1 & \text{if } \begin{cases} p_i.f_1 = p_j.f_1 \\ \& \ p_i.f_3 = p_j.f_3 \end{cases} \\ 0.5 & \text{else if } p_i.f_1 = p_j.f_1 \\ 0.25 & \text{other features match} \\ 0 & \text{no match} \end{cases} \tag{3}$$

where the values of the weights are assigned empirically according to the discriminative ability of the feature types. Function (3) measures the similarity between two nodes according to the weights of matched features.

- K_C is the similarity function over the two children node sequences $p_1[\boldsymbol{c}]$ and $p_2[\boldsymbol{c}]$:

$$K_C(p_1[\boldsymbol{c}], p_2[\boldsymbol{c}]) = \underset{\boldsymbol{a},\boldsymbol{b},\, l(\boldsymbol{a})=l(\boldsymbol{b})}{\operatorname{argmax}} \{K(p_1[\boldsymbol{a}], p_2[\boldsymbol{b}])\} \tag{4}$$

$$K(p_1[\boldsymbol{a}], p_2[\boldsymbol{b}]) = \sum_{i=1}^{l(\boldsymbol{a})} K(p_1[\boldsymbol{a}_i], p_2[\boldsymbol{b}_i]) \tag{5}$$

where ***a*** and ***b*** are two index sequences, *i.e.*, ***a*** is a sequence $0<a_1...<a_m \leq |p_1[C]|$ and *l*(***a***) is the length of sequence ***a***, and likewise for ***b***. The node set $p_1[\boldsymbol{a}] = \{p_1[\boldsymbol{a}_1], ..., p_1[\boldsymbol{a}_m]\}$ is the subset of $p_1[\boldsymbol{c}]$, $p_1[\boldsymbol{a}] \subseteq p_1[\boldsymbol{c}]$, $p_1[a_i]$ is the i^{th} node of $p_1[\boldsymbol{a}]$, and likewise for p_2.

We define $K(T_1, T_2)$ in terms of the similarity function $s(r_1, r_2)$ between the parent nodes and the similarity function K_C over the two children node sequences $r_1[\boldsymbol{c}]$ and $r_2[\boldsymbol{c}]$. Formula (5) defines the similarity between two node sequences by summing up the similarity of each corresponding node pair. K_C in Formula (4) searches out such two children node subsequences $p_1[\boldsymbol{a}]$ and $p_2[\boldsymbol{b}]$, which has the maximum node sequence similarity among all the possible combining pairs of node subsequences. Given the similarity scores of all children node pairs, Formula (4) can be easily resolved by the dynamic programming (DP) algorithm[5]. By traversing the two trees from top to down and applying the DP algorithm layer by layer, the parse tree similarity $K(T_1, T_2)$ defined by Formula (1) is obtained. Due to the DP algorithm, the defined tree similarity function is computable in O(*mn*), where *m* and *n* are the number of nodes in the two trees, respectively. The matching function $m(p_i, p_j)$ in Formula (2) can narrow down the search space during similarity calculation, since the sub-trees with unmatched root nodes are unnecessary to be further explored.

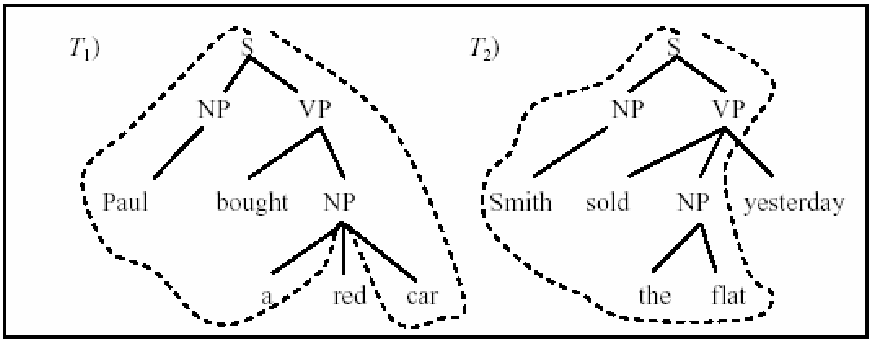

Fig. 1. Sub-structure with maximum similarity

From the above discussion, we can see that our defined tree similarity function is trying to detect the two trees' maximum isomorphic sub-structures. The similarity score between the maximum isomorphic sub-structures is returned as the value of the similarity function. Fig. 1 illustrates the sub-structures with the maximum similarity between two trees. Among the all matched sub-structures, only the sub-structures circled by the dashed lines are the isomorphic sub-structures with the maximum similarity. The similarity score between the sub-structures is obtained by summing up the similarity score between the corresponding matched nodes.

Finally, since the size of the input parse tree is not constant, the similarity score is normalized as follows:

$$\hat{K}(T_1, T_2) = \frac{K(T_1, T_2)}{\sqrt{K(T_1, T_1) * K(T_2, T_2)}} \tag{6}$$

The value of $\hat{K}(T_1, T_2)$ ranges from 0 to 1. In particular, $\hat{K}(T_1, T_2) = 1$ if and only if $T_1 = T_2$. For example, given two parse trees *A* and *B*, and *A* is a subtree of *B*, then under Formula (1), *K*(*A*, *B*) = *K*(*A*, *A*). However, after the normalization through

[5] A well-known application of Dynamic Programming is to compute the edit distance between two character strings. Let us regard a node as a character and a node sequence as a character string. Then given the similarity score between nodes, Formula (4) can be resolved using DP algorithm in the same way as that of strings. Due to space limitation, the implementation deatils are not discussed here.

Equation (6), we can get $\hat{K}(A,B) < \hat{K}(A,A) = 1$. In this way, we can differentiate such two cases.

According to the Formula (1) to (5), the similarity function $K(T_1,T_2)$ over the two trees in Fig. 1 is computed as follows:

$$\begin{aligned}
K(T_1,T_2) &= m(\mathrm{S},\mathrm{S}) * \{s(\mathrm{S},\mathrm{S}) + K_c([\mathrm{NP}, \mathrm{VP}],[\mathrm{NP}, \mathrm{VP}])\} \\
&= 0.25 + K(\mathrm{NP}, \mathrm{NP}) + K(\mathrm{VP}, \mathrm{VP}) \\
&= 0.25 + 0.25 + K(\mathrm{Paul}, \mathrm{Smith}) + 0.25 \\
&\quad + K_c([\mathrm{bought}, \mathrm{NP}],[\mathrm{sold}, \mathrm{NP}, \mathrm{yesterday}]) \\
&= 1 + K(\mathrm{bought}, \mathrm{sold}) + K(\mathrm{NP}, \mathrm{NP}) \\
&= 1{+}0.25{+}0.25{+}K_c([\mathrm{a}, \mathrm{red}, \mathrm{car}],[\mathrm{the}, \mathrm{flat}]) \\
&= 1.5 + K(\mathrm{a}, \mathrm{the}) + K(\mathrm{car}, \mathrm{flat}) \\
&= 2
\end{aligned}$$

The above similarity score is more than one. This is because we did not normalize the score using Formula (6).

2.3 Tree Similarity Based Unsupervised Learning

Our method consists of five steps:

1) Named Entity (NE) tagging and sentence parsing: Detailed and accurate NE types provide more effective information for relation discovery. Here we use Sekine's NE tagger [20], where 150 hierarchical types and subtypes of Named Entities are defined [21]. This NE tagger has also been adopted by Hasegawa et al. [8]. Besides, Collin's parser [18] is adopted to generate shallow parse trees.

2) Similarity calculation: The similarity between two relation instances is defined between two parse trees. However, the state-of-the-art of parser is always error-prone. Therefore, we only use the minimum span parse tree including the NE pairs when calculating the similarity function [4]. Please note that the two entities may not be the leftmost or rightmost node in the sub-tree.

3) NE pairs clustering: Clustering of NE pairs is based on the similarity score generated by the tree similarity function. Rather than k-means [17], we used a bottom-up hierarchical clustering method so that there is no need to determine the number of clusters in advance. This means that we are not restricted to the limited types of relations defined in MUC [1] or ACE [22]. Therefore, more substantial existing relations can be discovered. We adopt the group-average clustering algorithm [17] since it produces the best performance compared with the complete-link and single-link algorithms in our study.

4) Cluster labeling: In our study, we label each cluster by the most frequent "**Head Word**" in this cluster. As indicated in subsection 2.1, the "**Head Word**" of root node defines the main meaning of a parse tree. This way, the "**Head Word**" of the root node of the minimum span tree naturally characterizes the relation between this NE pair in this tree. Thus, we simply count the frequency of the "**Head Word**" of the root node in the cluster, and then chose the most frequent "**Head Word**" as the relation type of the cluster.

5) Cluster pruning: Unreliable clusters may be generated due to various reasons such as divergent relation type distributions and the fact that most of the entity pairs inside this cluster are totally unrelated. Therefore, pruning is necessary and done in our approach using two criteria. Firstly, if the most frequent "**Head Word**" occurs less than a predefined percentage in this cluster, which means that the relation type defined by this "**Head Word**" is not significant statistically, the cluster is pruned out. Secondly, we prune out the clusters whose NE pair number is below a predefined threshold because such clusters may not be representative enough for this relation.

3 Experiments

3.1 Experimental Setting

To verify our proposed method and establish proper comparison with Hasegawa et al. [8], we use the same corpus "The New York Times (1995)", and evaluate our work on the same two kinds of NE pairs: COMPANY-COMPANY (COM-COM) and PERSON-GPE (PER-GPE) as Hasegawa et al. in [8]. First, we iterate over all pairs of Named Entities occurring in the same sentence to generate potential relation instances. Then, according to the co-occurrence frequency of NE pairs, all the relation instances are grouped into three categories:

1) High frequent instances with the co-occurrence frequency not less than 30. In this category, only the relation instances, which satisfy the all criteria of Hasegawa et al. [8][6], are kept for final experiment. By doing so, this category data is the same as the entire experimental set used by Hasegawa et al. [8].
2) Intermediate frequent instances with the co-occurrence frequency between 5 and 30. In this category, only two distinct NE pairs are randomly picked at each frequency for final evaluation due to the large number of such NE pairs.
3) Less frequent instances with the co-occurrence frequency not more than 5. In this category, twenty distinct NE pairs are randomly picked at each frequency for final evaluation due to the similar reason as 2).

Table 1 reports the statistics of the entire evaluation corpus[7] which is manually tagged. Table 2 reports the percentage of the NE pairs which carry more than one relation types when occurring at different relation instances. The numbers inside parentheses in Table 1 and Table 2 correspond to the statistical values of the NE pair "PER-GPE", while the numbers outside parentheses are related to the NE pair "COM-COM". Table 2 shows that at least 9.88% of distinct NE pairs have more than one

[6] To discover reliable relations, Hasegawa et al. [8] sets five conditions to generate relation instance set. NE pair co-occurrence more than 30 times is one of the five conditions.

[7] Due to the parsing errors and NE tagging errors, the actual number of relation instances is less than the theory number that we should pick up.

relation types in the test corpus. Thus it is reasonable and necessary to assume that each occurrence of NE pairs forms one individual relation instance.

Table 1. Statistics on the manually annotated evaluation data

Category by frequency	# of instances	# of distinct NE pairs	# of relation types
High	8931 (13205)	65 (177)	10 (38)
Intermediate	672 (783)	38 (41)	6 (7)
Less	276 (215)	76 (81)	5 (8)

Table 2. % of distinct NE pairs with more than one relation types on the evaluation data

Category by frequency	% of NE pairs have more than one relations
High	15.4 (12.99)
Intermediate	28.9 (24.4)
Less	11.8 (9.88)

3.2 Evaluation Measures

All the experiments are carried out against the manually annotated evaluation corpus. We adopt the same criteria as Hasegawa et al. [8] to evaluate the performance of our method. Grouping and labeling are evaluated separately. For grouping evaluation, all the single NE pair clusters are labeled as non-relation while all the other clusters are labeled as the most frequent relation type counted in this cluster. For each individual relation instance, if the manually assigned relation type is the same as its cluster label, the grouping of this relation instance is counted as correct, otherwise, are counted as incorrect. Recall (**R**), Precision (**P**) and F-measure (**F**) are adopted as the main performance measure for grouping [8]. For labeling evaluation, a cluster is labeled correctly only if the labeling relation type, represented by most frequent "**Head Word**" of the root node of the minimal-span subtree, is the same as the cluster label gotten in the grouping process.

3.3 Experimental Results

Like other applications using clustering algorithms, the performance of the proposed method also depends on the threshold of the clustering similarity. Here this threshold is used to truncate the hierarchical tree, so that the different clusters are generated. When the threshold is set to 1, then each individual relation instance forms one unique group; when the threshold is set to 0, then the all relation instance form one big group. Table 3 reports the evaluation results of grouping, where the best F-measures and the corresponding similarity thresholds are listed. We can see that our method not only achieves good performance on the high-frequent data, but also performs well on the intermediate and less-frequent data. The higher frequency, the higher performance. Since the best thresholds of the two NE cases are the almost same, we just fix the universal threshold as the one used in "PER-GPE" case in each category.

Table 3. Performance evaluation of Grouping phase, the numbers inside parentheses correspond to the evaluation score of "PER-GPE" while the numbers outside parentheses are related to "COM-COM".

Category by frequency	Performance			Threshold
	F	**P** (%)	**R** (%)	
High	80 (87)	82 (90)	78 (84)	0.28 (0.29)
Intermediate	74 (76)	87 (84)	64 (69)	0.32 (0.30)
Less	62 (65)	75 (77)	53 (56)	0.36 (0.35)

Table 4. Best performance comparison in the high-frequent data (**F**)

	Our approach	Hasegawa et al. [8]
PER-GPE	87	82
COM-COM	80	77

Table 4 compares the performances of the proposed method and Hasegawa et al. [8], where the best F-measures on the same high-frequent data are reported. Table 4 shows that our method outperforms the previous approach by 5 and 3 F-measures in clustering NE pairs of "PER-GPE" and "COM-COM", respectively.

An interesting phenomenon is that the best threshold is set to be just above 0 for the cosine similarity in Hasegawa et al. [8]. This means that each word feature vector of each combination of NE pairs in the same cluster shares at least one word in common --- and most of these common words were pertinent to the relations [8]. This also prevents them from working well on less-frequent data [8]. In contrast, for the similarity function in our approach, the best threshold is much greater than 0. The difference between the two thresholds implies that the similarity function over the parse trees can capture more common structured features than the word feature vectors can. This is also the reason why our method is effective on both high and less-frequent data.

It is not surprising that we do have that a few identical NE pairs, occurring in different relation instances, are grouped into different relation sets. For example, the NE pairs "*General Electric Co.* and *NBC*", in one sentence "*General Electric Co.*, which bought *NBC* in 1986, will announce a new marketing plan.", is grouped into the relation set "*M&A*", but in another sentence "Prime Star Partners and *General Electric Co.*, parent of *NBC*, has signed up 430,000 subscribers.", is grouped into another relation set "*parent*". Among all the NE pairs that carry more than one relation types, 41.8% of them are grouped correctly using our tree similarity function.

The performance of grouping is the upper bound of the performances of labeling and pruning. In the final, there are 146 PER-GPE clusters and 95 COM-COM clusters are generated after grouping. Out of which, only 57 PER-GPE clusters and 42 COM-COM clusters are labeled correctly before pruning. This is because that a large portion of the non-relation clusters are labeled as one kind of true relations. After pruning, 117 PER-GPE clusters and 84 COM-COM clusters are labeled correctly. This is because lots of the non-relation clusters are labeled correctly by the pruning process, so we can say that pruning is a non-relation labeling process, which greatly improves the performance of labeling.

The experimental results discussed above suggest that our proposed method is an effective solution for discovering relation from a large raw corpus.

4 Discussions

It would be interesting to review and summarize how the proposed method deals with the relation extraction issue differently from other related works. Table 5 in the next page summarizes the differences between our method and Hasegawa et al. [8].

Table 5. The differences between our method and Hasegawa et al. [8]

	Our approach	Hasegawa et al. [8]
Similarity Measure	tree similarity over parse tree structures	cosine similarity between the context word feature vectors
Assumption	No	Yes (The same entity pairs in different sentences have the same relation)
Labeling	the most frequent "**Head Word**" of the root node of sub-tree	the most frequent context word
Pruning	Yes (We present two pruning criterion)	No
Data Frequency	effective on both high and less-frequent data	effective only on high-frequent data

In addition, since our tree similarity function has benefited from the relation tree kernels of Zelenko et al. [3], let us compare our similarity measure function with their relation kernel function [3] from the viewpoint of computational efficiency. Zelenko et al. [3] defined the first parse tree kernels for relation extraction, and then this relation tree kernels were extended to dependency tree kernels by Culotta et al. [4]. Their tree kernels sum up the similarity scores among all possible subsequences of children nodes with matching parents, and give a penalty to longer sequences. Their tree kernels are closely related to the convolution kernels [12]. But, by doing so, lots of sub-trees will be considered again and again. An extreme case occurs when two tree structures are identical. In that situation all the sub-trees will be considered exhaustedly, even if the sub-tree is a part of other bigger sub-trees. We use the maximum score in Formula (4) instead of the summation in our approach. With our approach, the entire tree is only considered once. The replacement of summation with maximization reduces the computational time greatly.

5 Conclusions and Future Directions

We modified the relation tree kernels [3] to be a tree similarity measure function by replacing the summation over all possible subsequences of children nodes with maximization, and used it in clustering for relation extraction. The experimental result showed much improvement over the previous best result [8] on the same test corpus. It also showed that our method is high effective on both high-frequent and less-frequent data. Our work demonstrated the effectiveness of combining the tree similarity measure with unsupervised learning for relation extraction.

Although our method shows good performance, there are still other aspects of the proposed method worth discussing here. Without additional knowledge, relation detecting and relation labeling are still not easy to be resolved, especially in less-frequent data. We expect that using additional easily-acquired knowledge can improve the performance of the proposed method. For example, we can introduce the WordNet [19] thesaurus information into Formula (3) to obtain more accurate node similarities and resolve data sparse problem. We can also use the same resource to improve the labeling scheme and find more abstract relation types like the definitions used in ACE program [22].

References

1. MUC. 1987-1998. The nist MUC website: http://www.itl.nist.gov/iaui/894.02/related_projects/muc/
2. Miller, S., Fox, H., Ramshaw, L. and Weischedel, R. 2000. A novel use of statistical parsing to extract information from text. Proceedings of NAACL-00
3. Zelenko, D., Aone, C. and Richardella, A. 2003. Kernel Methods for Relation Extraction. Journal of Machine Learning Research. 2003(2):1083-1106
4. Culotta, A. and Sorensen, J. 2004. Dependency Tree Kernel for Relation Extraction. Proceeding of ACL-04
5. Kambhatla, N. 2004. Combining Lexical, Syntactic, and Semantic Features with Maximum Entropy Models for Extracting Relations. Proceeding of ACL-04, Poster paper.
6. Agichtein, E. and Gravano, L. 2000. Snow-ball: Extracting Relations from Large Plain-text Collections. Proceedings of the Fifth ACM International Conference on Digital Libraries.
7. Stevenson, M. 2004. An Unsupervised WordNet-based Algorithm for Relation Extraction. Proceedings of the 4th LREC workshop "Beyond Named Entity: Semantic Labeling for NLP tasks"

8. Hasegawa, T., Sekine, S. and Grishman, R. 2004. Discovering Relations among Named Entities from Large Corpora. Proceeding of ACL-04
9. Vapnik, V. 1998. Statistical Learning Theory. John Wiley
10. Collins, M. and Duffy, N. 2001. Convolution Kernels for Natural Language. Proceeding of NIPS-01
11. Collins, M. and Duffy, N. 2002. New Ranking Algorithm for Parsing and Tagging: Kernel over Discrete Structure, and the Voted Perceptron. Proceeding of ACL-02.
12. Haussler, D. 1999. Convolution Kernels on Discrete Structures. Technical Report UCS-CRL-99-10, University of California
13. Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N. and Watkins, C. 2002. Text classification using string kernel. Journal of Machine Learning Research, 2002(2):419-444
14. Suzuki, J., Hirao, T., Sasaki Y. and Maeda, E. 2003. Hierarchical Directed Acyclic Graph Kernel: Methods for Structured Natural Language Data. Proceedings of ACL-03
15. Suzuki, J., Isozaki, H. and Maeda, E. 2003. Convolution Kernels with Feature Selection for Natural Language Processing Tasks. Proceedings of ACL-04
16. Moschitti, A. 2004. A study on Convolution Kernels for Shallow Semantic Parsing. Proceedings of ACL-04
17. Manning, C. and Schutze, H. 1999. Foundations of Statistical Natural Language Processing. The MIT Press: 500-527
18. Collins, M. 1999. Head-Driven Statistical Models for Natural Language Parsing. Ph.D. Thesis. University of Pennsylvania
19. Fellbaum, C. 1998. WordNet: An Electronic Lexical Database and some of its Applications. Cambridge, MA: MIT Press.
20. Sekine, S. 2001. OAK System (English Sentence Analysis). Http://nlp.cs.nyu.edu/oak
21. Sekine, S., Sudo, K. and Nobata, C. 2002. Extended named entity hierarchy. Proceedings of LREC-02
22. ACE. 2004. The Automatic Content Extraction (ACE) Projects. http://www.ldc.upenn.edu/Projects/ACE/