Discovery of Dependency Tree Patterns for Relation Extraction

Hongzhi Xu, Changjian Hu, and Guoyang Shen

NEC Laboratories China
Beijing, 100084, China,
{xuhongzhi, huchangjian, shenguoyang}@research.nec.com.cn

Abstract. Relation extraction is to identify the relations between pairs of named entities. In this paper, we try to solve the problem of relation extraction by discovering dependency tree patterns (a pattern is an embedded sub dependency tree indicating a relation instance). Our approach is to find an optimal rule (pattern) set automatically based on the proposed dependency tree pattern mining algorithm. The experimental results show that the extracted patterns can achieve a high precision and a reasonable recall rate when used as rules to extract relation instances. Furthermore, an additional experiment shows that other machine learning based relation extraction methods can also benefit from the extracted patterns by using them as features.

1 Introduction

Relation extraction is the task that aims to identify relations between pairs of named entities from free text. From the perspective of methodologies employed to conduct relation extraction, current existing methods can be classified into three categories. The first category is rule-based methods, which identifies relations by utilizing inference rules. The rules could be extended regular expressions incorporating with pre-defined constraints. The second is to treat relation extraction as a classification problem with a kernel machine such as SVM. The problem is then how to define a kernel function between two relation instances. The third category of relation extraction method is regarding relation extraction as sequence labeling problem. Then the existing methods for sequence labeling, such as Maximum Entropy (ME) and Conditional Random Fields (CRFs), could be used to solve the relation extraction problem directly.

Our algorithm proposed in this paper belongs to the rules learning framework. The reason we choose the rule-based method is that the rules could contain more semantic information than features used in statistical methods. A pattern is a sub dependency tree that indicates a relation instance. In other words, a pattern corresponds to a trimmed sentence, e.g. a sentence with all unrelated components removed. This is certainly an advantage to the flat patterns which will face significant problems if the sentences are very complex. Experiments show that our patterns are more effective used for relation extraction.

The rest of the paper is organized as follows. Section 2 gives a brief introduction to the related work. Section 3 describes our new framework for relation extraction. In Section 4, we present our algorithm for automatic extraction of the dependency tree patterns. In Section 5, experiments are conducted to evaluate our new algorithm. Section 6 is the concluding remarks and future work.

2 Related Work

Generally, the current methods could be classified into three categories based on the methodologies they took, namely 1) rule-based methods, 2) kernel methods, 3) sequence labeling methods.

For rule-based methods, Brin (1999) proposed an algorithm DIPRE. Given several relation instances as the seeds, DIPRE can learn rules (regular expressions) to extract 'author-book' relations

23rd Pacific Asia Conference on Language, Information and Computation, pages 851–858

from the web pages. Agichtein (2000) proposed Snowball which improved DIPRE by introducing the notion of match degree, where only sentences, whose match degrees are higher than a pre-defined threshold, are used to generate new rules. Lin (2001) proposed DIRT which learns inference rules from the paths connecting pairs of arguments of the relations in dependency trees for question answering systems.

For kernel methods, Zelenko (2002) propose the continuous and sparse tree kernels based on the syntactic parse tree for relation extraction. Culotta (2004) used this kernel on dependency trees to train a SVM(Cortes and Vapnik, 1995) classifier for relation extraction. Similar works include the shortest path dependency tree kernel(Bunescu and Mooney, 2005) and subsequence kernel(Bunescu and Mooney, 2006).

For the sequence labeling methods, Kambhatla (2004) proposed an algorithm that makes use of features extracted from the dependency trees and syntactic parse trees, including the path from the first argument to the second argument, the parents of the first and second arguments of a relation in a dependency tree, the context words with their POS tags. All the features are used to train a ME model, which is then used to extract new relation instances. Culotta (2006) used a data mining algorithm to find implicit relations between different relation types. For example, a *father-of* relation and a *husband-wife* relation may imply a *mother-of* relation. These global information, used in a CRFs framework, can effectively improve the performance of the relation extraction task. Bundschus (2008) adopted rich features except syntactic features to train a CRFs model for two different biological relations extraction: *gene-disease* and *disease-treatment*.

The most related work is proposed by Dat in (2007). However, there are several differences between his algorithm and ours. First, Dat only considers subtrees whose leaf nodes are either arguments or the keyword that determines the relation type i.e., the subtrees are heuristically selected. But our algorithm treats the dependency trees as general subtrees with some extra constraints. Second, the subtrees in Dat's method are only used as features to train SVM models. Our algorithm uses the subtrees as patterns to directly extract new relation instances. The patterns themselves could perform well, although they are also able to be used as features. Third, the tree nodes are represented by original words in Dat's work, but our tree nodes are represented by several attributes including POS tags, high-level POS tags and dependency types. With our node representation, we can define any match function between two nodes. For example, we could define that if the POS tags of two nodes are same, then the two nodes are same.

3 A New Framework For Relation Extraction

3.1 Dependency Tree Pattern Definition

Tree Nodes Representation. We use different features/attributes to represent the tree nodes, including Tokens, POS tags, high-level POS tags, dependency types and roles. The roles including *ARG-1*, *ARG-2*, *KEY* and *OTHR* denote the identifications of tree nodes in relation instances. *ARG-1* and *ARG-2* are the arguments of a relation instance, *KEY* is the keyword that determines the relation type and *OTHR* stands for *other*. Based on these attributes, we could define any match functions in $\{0, 1\}$ between two nodes.

Embedded Subtrees. To make the dependency tree patterns to have higher matching abilities, we use embedded subtrees that allow to skip nodes when matching a target tree. This is useful to skip some useless nodes when we use patterns to extract relation instances.

Closed Subtrees Actually, for the relation extraction task, we only need the closed patterns, because in a given tree collection, any tree that satisfies an unclosed pattern p' must satisfies a certain closed pattern p that contains p' and has the same support as p'. In other words, p contains all useful information that p' contains.

Proof. Suppose there is a tree t that satisfies an unclosed pattern p' but doesn't satisfy any closed pattern. Then there is no other pattern could contain pattern p' and have the same support as p', so based on the definition of closed pattern, p' is closed which is contradictory to the known condition that p' is not closed.

Preserve the Cross Nodes Based on a linguistic consideration that two nodes in a dependency tree build their relationship through their cross node, we constrain that the subtrees should preserve all cross nodes when matching dependency trees. The cross node of two nodes v_1 and v_2, denoted by $crn(v_1, v_2)$, is the first common ancestor of v_1 and v_2. A cross-node-preserving subtree p of T is a subtree that the cross node $crn(v_1, v_2)$ of each pair of nodes v_1 and v_2 in p is also the cross node of v_1 and v_2 in T. For example, in Figure 1, (2) is a cross-node-preserving subtree of (1), while (3) is not because the cross node of F and G is C in (1), but the cross node of F and G is A in (3).

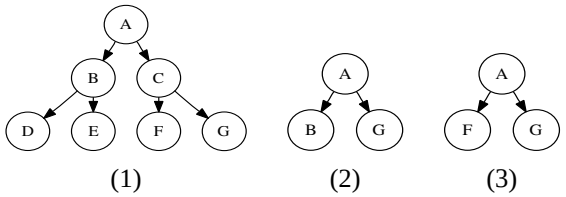


Figure 1: An example to show a cross node preserved subtree. (2) is a cross-node-preserving subtree of (1), while (3) is not.

Based on all the above analysis, we finally define dependency tree pattern as in Definition 1. In all, A pattern contains at least three nodes with roles *ARG-1*, *ARG-2* and *KEY*. The relation type of a pattern is the same as that of *KEY*.

Definition 1. A Dependency Tree Pattern is a rooted, ordered, embedded and cross-node-preserving subtree that contains at least three nodes with roles: *ARG-1*, *ARG-2* and *KEY*, whose support is at least min_supp.

It is true that there may be more than one relation instances in one dependency tree. In this situation, we only assign one relation instance with its arguments and keyword the roles *ARG-1*, *ARG-2* and *KEY*, and assign all other nodes as role *OTHR* at one time. In other words, a single sentence may generate several dependency trees with different roles assigning.

4 Mining Dependency Tree Patterns

4.1 Pattern Matching

Pattern matching is the basic problem for counting the support of a subtree in frequent subtree mining that should be solved efficiently. Actually, the only difference of our defined pattern is that we preserve the cross nodes. So, we design a new matching algorithm that could directly judge whether a subtree is a cross-node-preserving one. We modify a dynamic programming algorithm proposed in (Kilpelainen, 1992), and the modified algorithm is shown in Figure 2.

Here all the nodes q are represented by their post traversal order number, $label(q)$ is the label of node q. The algorithm maintains a table $e(v, c, w)$, where v is a node in a pattern p; c, w are nodes in a tree t; $e(v, c, w)$ records the minimal post traversal order number of the node in tree t, that matches node v in pattern p with the cross node c in $rr(w)$ of tree t, $rr(w)$ represents the right relatives of the node w; $lr(w)$ represents the left relatives of the node w, including the nodes $1, ..., \min(desc(w)) - 1$. If a node x is the right relative of y, then y is the left relative of x. There is also a virtual node 0 is the left relative of all nodes. $desc(w)$ denotes the descendants of the node

Input: A Pattern p, a Tree t.
Output: Table $e(v, c, w)$.

1. For each v in p; c, w in t.
2. $e(v, c, w) = n + 1$;
3. For $v = 1, 2, ..., m$.
4. $q = 0$;
5. Let $v_1, ..., v_k$ be the children of v;
6. For $w = 1, 2, ..., n$.
7. If $label(v) == label(w)$
8. $p = \min(desc(w) \cup \{n + 1\}) - 1$;
9. $i = 0$;
10. While $i < k$ and $p < w$.
11. $p = e(v_{i+1}, w, p)$;
12. If $p \in desc(w)$.
13. $i = i + 1$;
14. If $i == k$.
15. While $q \in lr(w)$.
16. Get cross node $c = crn(q, w)$;
17. $e(v, c, q) = w$;
18. $q = q + 1$;

Figure 2: A matching algorithm of our defined dependency tree patterns.

w. Specially, the cross node of the virtual node 0 and a node w, $crn(0, w) = -1$ which is also a virtual node. Suppose the root of pattern p is r, and if the value of e(r, -1, 0) is less than $n+1$, then there must be a cross node preserved matching of p in t at the node e(r, -1, 0) of t. Kilpelainen (1992) has proved that the matching algorithm requires $O(mn)$ time and space complexity, where m is the pattern size and n is the tree size.

4.2 Pattern Mining Algorithm

We modify a traditional subtree mining algorithm TRIP(Tatikonda and Parthasarathy, 2006) for our dependency tree patterns extraction. A parameter min_prec is used to select only those accurate patterns, i.e. only those patterns whose precisions are larger than or equal to min_prec will be used. Here, precision is defined as $\#t\_rel/(\#t\_rel + \#f\_rel)$, where $\#t\_rel$ and $\#f\_rel$ are the number of true and false relation instances a pattern identifies respectively.

4.3 Pruning

Traditional subtree mining algorithms try to extract all possible subtrees. Due to the combinational explosion problem, the number of subtrees grows exponentially with the size of the patterns. If min_supp is set to a small value, there will be a large number of patterns, which will cause the failure of the mining process.

Considering that we only need closed patterns, we adopt a pruning process from an existing algorithm CMTreeMiner which is designed specially for mining closed patterns(Chi and Xia, 2005).

5 Experimental Evaluation

In this section, we use our algorithm to extract business relations between companies, including $Cooperation$ ($COOP$), $Competition$ ($COMP$) and $Acquisition$ ($ACQU$).

We first test the performance of the extracted patterns when directly used to extraction new relation instances, then we use the patterns as features in a CRFs framework.

5.1 Data Set

To construct a data set, we first build a company list. Then we use company pairs as queries to submit them to a search engine and find the news articles that contain the company pairs, where some pairs have actual relations and others don't. The sentences are labeled manually with all relation instances. Figure 3 is the statistical information organized by the number of entities

and number of relations contained in sentences. Figure 3(Left) shows the distribution of relation instances in sentences regardless of the relation type. Most sentences contain only one relation. We also add 1,165 sentences that contain no relations. Figure 3(Middle) shows the distribution of the entities in sentences. In this data set, all the entities are companies. Figure 3(Right) shows the distribution of different types of relations.

Figure 3: Statistical information of the corpus.

5.2 Experimental Setting

First, we use Stanford parser(Klein and Manning, 2003) to parse our sentences into dependency trees. For the tree node representation, i.e. matching function between two nodes, we compare four different representations: POS, High-POS, POS+DEP, High-POS+DEP. For example, POS+DEP means two nodes are same if their POS tags and the dependency types to their parents are both same. High-POS stands for the generalized POS tags, e.g. noun, verb, etc.

We split the corpus by 1:1, i.e. 50% is used to extract dependency tree patterns, the other 50% to test the extracted patterns. Since our work is not to deal with the NER problem, we assume that all the companies are known. In the experiment, we set $min\_supp = 5$ and use different values for the parameter min_prec to filter the poor patterns. Note that all results are the average value of ten times of running.

5.3 Mining Result

Figure 4 shows F1 measure with the change of min_prec. We can see that High-POS gives the best performance in terms of F-Measure. We have found that the more special the nodes, the precision tends to become higher, while recall and F-Measure tend to become lower; recall and F-Measure also become worse with the increasing of the parameter min_prec.

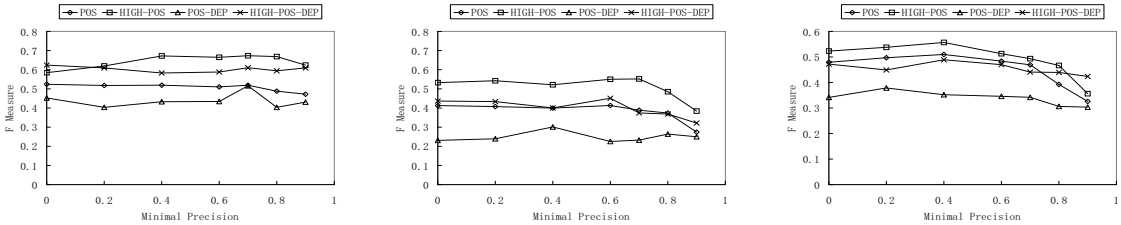


Figure 4: Performance(F-measure) of the extracted patterns with change of parameter min_prec(minimal precision). (COOP(left), COMP(middle), ACQU(right))

Figure 5 shows three typical dependency tree patterns, which show the tokens that correspond to the nodes in patterns are really key subcomponents of the sentences that contain relation instances. Take the first pattern(COOP) for example, in the sentence, the unrelated constituents "us-based", "CDMA Intellectual property rights framework" that modifies "agreement" could be filtered. The case is similar for the second pattern(COMP). For the third pattern(ACQU), it is interesting that it identifies only a part of a sentence that contains a relation instance, e.g. "of/IN ..." and "before/IN ...".

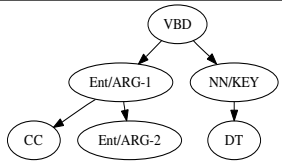
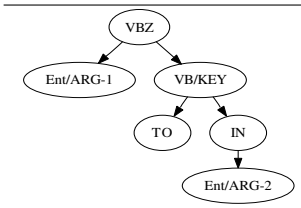
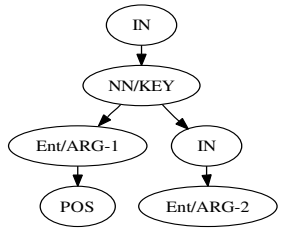
	[China/NNP Unicom/NNP]/ARG-1 [and/CC] us-based/JJ [Qualcomm/NNP Incorporated/NNP]/ARG-2 [signed/VBD] [a/DT] CDMA/JJ Intellectual/NNP property/NN rights/NNS framework/NN [agreement/NN]/KEY ... (Trimmed Sentence: ARG-1 and ARG-2 signed a agreement.)
	[Marvell/NNP Technology/NNP Group/NNP]/ARG-1 [plans/VBZ] to/TO launch/VB a/DT new/JJ version/NN of/IN its/PRP$ "/" monahans/JJ "/" processor/NN by/IN December/NNP [to/TO] [compete/VB]/KEY better/JJR [against/IN] [Texas/NNP Instruments/NNPS]/ARG-2 and/CC [Freescale/NNP Semiconductor/NNP]/ARG-2 in/IN ... (Trimmed Sentence: ARG-1 plans to compete against ARG-2.)
	1. Customers/NNS will/MD also/RB see/VB the/DT fruits/NNS [of/IN] [Intel/NNP]/ARG-1 ['s/POS] [acquisition/NN]/KEY [of/IN] [Dayna/NNP Communications/NNPS]/ARG-2 ,/, part/NN of/IN ... (Trimmed Sentence: .. of ARG-1's acquisition of ARG-2.) 2. Sears/NNP Holdings/NNPS ,/, which/WDT was/VBD formed/VBN [through/IN] [Kmart/NNP Holding/NNP]/ARG-1 ['s/POS] [purchase/NN]/KEY [of/IN] [Sears/NNP]/ARG-2 ,/, [Roebuck/NNP]/ARG-2 on/IN march/NN 24/CD ,/, ... (Trimmed Sentence: .. through ARG-1's purchase of ARG-2.)

Figure 5: Selected Patterns of COOP(top), COMP(middle), ACQU(bottom) and some corresponding examples from the corpus.

5.4 Relation Extraction with CRFs

In this subsection, we design experiments to show that the extracted patterns could be used as features in other machine learning algorithms to improve the overall performance of relation extraction. Specially, we use High-POS to represent the nodes and use the dependency tree patterns as binary features, denoted by $f(p_i)$. If a dependency tree of a relation instance satisfies pattern p_i, then $f(p_i) = 1$, otherwise, $f(p_i) = 0$. We use CRFs to train our relation extraction model.

5.4.1 Problem Formalization We fixed one entity as the first argument *ARG-1*, and guess the labels of other tokens as *OTHR* or *ARG-2*. The label *KEY* is given through a dictionary containing a list of keywords with their relation types. For example, in the sentence "A could compete with B and C.", 'A' is fixed as the first argument, i.e. the label of 'A' is known as *ARG-1*. Then the algorithm will guess the labels of other tokens in the sentence. The correct labeling is "A/*ARG-1* could/*OTHR* compete/*KEY* with/*OTHR* B/*ARG-2* and/*OTHR* C/*ARG-2*."

5.4.2 Experiment We conduct experiments with the CRFs framework on the same data set. Here, we split the data set into three parts by 5:3:2. The 50% is used to extract dependency tree patterns, the 30% is used to train a CRFs model and the 20% is used for testing. Note that we use different data for pattern extraction and CRFs training in order to avoid over fitting problem. We use CRFPP[1] as the CRFs toolkit.

In this experiment, we try several types of features that have been used in the current systems as follows: (1) Context words with their POS tags, and bag of words between two arguments and the path features extract from dependency trees and parse trees used in a ME framework(Kambhatla, 2004), denoted by $Baseline(B)$; (2) Window features judging whether two entities are located in a same window as used in (Bundschus and Dejori, 2008). We extend the window feature by observing whether a keyword is also contained in the window. For example, we use A to denote the first argument, K as the keywords and B as the current entity. Then a window feature value

[1] http://crfpp.sourceforge.net/

Table 1: Experimental result of relation extraction with CRFs. Pat1 stands for patterns with Token Tree Nodes, Pat2 for POS, Pat3 for High-POS.

		Baseline	B+Win	B+Pat1	B+Pat2	B+Pat3	B+W+Pat1	B+W+Pat3
COOP	P	0.7630	0.7355	0.7746	0.7452	0.7745	**0.7909**	0.7554
	R	0.5593	**0.7542**	0.5678	0.6568	0.6695	0.7373	0.7458
	F1	0.6455	0.7448	0.6553	0.6982	0.7182	**0.7632**	0.7505
COMP	P	0.6928	0.6931	**0.7396**	0.7233	0.7341	0.7083	0.7114
	R	0.4291	0.5668	0.2874	0.4656	0.5142	0.5506	**0.5789**
	F1	0.5300	0.6236	0.4140	0.5665	0.6048	0.6196	**0.6384**
ACQU	P	0.7597	0.7679	**0.7895**	0.7310	0.7396	0.7799	0.7579
	R	0.5087	0.6043	0.3913	0.5435	0.6174	0.5391	**0.6261**
	F1	0.6094	0.6763	0.5233	0.6235	0.6730	0.6375	**0.6857**
MacroAvg F1		0.5950	0.6816	0.5309	0.6294	0.6653	0.6734	**0.6915**

could be AKB, $A\_compete\_B$, KAB, etc. The extended window features are denoted by Win. We set window size to 10 for Win features. We use Pat to denote our dependency tree pattern features. Specially, we compare three different tree node representations: tokens/words (Pat1) as used in (Dat and Matsuo, 2007), POS (Pat2) and High-POS (Pat3).

Table 1 is the experimental result. We can see that, when the features Win are added, the performance are significantly improved. This is because most relation instances are located in same windows. For the pattern features, the patterns whose tree nodes are represented with tokens (Pat1) will harm the performance because of the data sparseness problem which will make the model over fit the training data and thus give wrong weights to the pattern features. The patterns with POS (Pat2) tree node representation perform better than tokens, and High-POS (Pat3) perform the best among all kinds of pattern features. The performance of $Pat3$ is similar to the Win features, this is because both of them try to estimate whether two arguments are related although in different ways.

It is worthy to note that, in our data set, most sentences contain real relations, which has made the Win features especially effective. This is why the deep syntactic analysis of the sentence structure is not better than the simple Win features. However, the window features don't work well when the sentences are relatively complex, e.g. with a lot of modifications and parentheses. This is where Pat features could work better. So, when the two features are both added, the performance could be further improved. Furthermore, in real application, the real relations will be more rare than in our data set, we can hope that Pat features will perform better then.

6 Conclusion and Future Work

In this paper, we proposed a new algorithm to automatically discover dependency tree patterns for entity relation extraction. Our algorithm can be considered as a more general version of (Dat and Matsuo, 2007) as we have discussed above. The experiments with CRFs show that the generalization of the tree nodes from tokens to POS and High-POS is very important which could significantly relieve the data sparseness problem since sparse features will do harm to the relation extraction model as shown in Table 1.

In this work, we only represent tree nodes with fixed POS levels and their roles. There are cases when words have more discriminative ability than POS tags, or some other nodes could be further generalized into higher level to get a pattern with more generalization ability without decreasing the precision. In sum, by assigning each node attribute with different levels (e.g. words, POS, high-level-POS, etc.), we could generate patterns with more discriminative ability. How to construct adaptive match functions between two nodes will be our future research topic.

References

Agichtein, E. and L. Gravano. 2000. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the fifth ACM conference on Digital libraries*, pages 85–94, New York, NY, USA.

Brin, S. 1999. Extracting patterns and relations from the world wide web. In *The World Wide Web and Databases*, Lecture Notes in Computer Science, pages 172–183. Springer Berlin / Heidelberg.

Bundschus, M., M. Dejori, M. Stetter, V. Tresp, and H.-P. Kriegel. 2008. Extraction of semantic biomedical relations from text using conditional random fields. *BMC Bioinformatics*, 9(207).

Bunescu, R. and R. Mooney. 2005. A shortest path dependency kernel for relation extraction. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP)*, pages 724–731, Vancouver, October.

Bunescu, R. and R. Mooney. 2006. Subsequence kernels for relation extraction. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 171–178. MIT Press, Cambridge, MA.

Chi, Y., S. Nijssen, R. R. Muntz, and J. N. Kok. 2005. Frequent subtree mining - an overview. *Fundamenta Informaticae Special Issue on Graph and Tree Mining*, 66(1-2/2005):161–198.

Chi, Y., Y. Xia, Y. Yang, and R. R. Muntz. 2005. Mining closed and maximal frequent subtrees from databases of labeled rooted trees. *IEEE Transactions on Knowledge and Data Engineering*, 17(2):190–202.

Cortes, C. and V. Vapnik. 1995. Support vector networks. *Machine Learning*, 20(3):273–297.

Culotta, A., A. McCallum, and J. Betz. 2006. Integrating probabilistic extraction models and data mining to discover relations and patterns in text. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the ACL*, pages 296–303, June.

Culotta, A. and J. Sorensen. 2004. Dependency tree kernels for relation extraction. In *21th International Conference on Computational Linguistics.*

Dat, N., Y. Matsuo, and M. Ishizuka. 2007. Relation extraction from wikipedia using subtree mining. In *Proceedings of 22nd Conference on Artificial Intelligence (AAAI)*, pages 1414–1420.

Kambhatla, N. 2004. Combining lexical, syntactic and semantic features with maximum entropy models for extracting relations. In *21th International Conference on Computational Linguistics.*

Kilpelainen, P. 1992. *Tree Matching Problems with Applications to Structured Text Databases*. PhD thesis, Department of Computer Science, University of Helsinki, Helsinki, Finland.

Klein, D. and C. D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Meeting of the Association for Computational Linguistics*, pages 423–430.

Lin, D. and P. Pantel. 2001. Discovery of inference rules for question answering. In *Natural Language Engineering*, volume 7, pages 343–360. Cambridge University Press.

Tatikonda, S., S. Parthasarathy, and T. Kurc. 2006. Trips and tides: New algorithms for tree mining. In *Proceedings of the Conference on Information and Knowledge Management(CIKM)*, volume 5, pages 455–464, Arlington, Virginia, USA.

Zelenko, D., C. Aone, and A. Richardella. 2002. Kernel methods for relation extraction. In *Proceedings of the Conference on Empirical Methods on Natural Language Processing(EMNLP)*, pages 71–78, Philadelphia.