Semantic Relation Extraction With Kernels Over Typed Dependency Trees

Frank Reichartz Hannes Korte Gerhard Paass
Fraunhofer IAIS
Schloss Birlinghoven
St. Augustin, Germany
{frank.reichartz | hannes.korte | gerhard.paass}@iais.fraunhofer.de

ABSTRACT

An important step for understanding the semantic content of text is the extraction of semantic relations between entities in natural language documents. Automatic extraction techniques have to be able to identify different versions of the same relation which usually may be expressed in a great variety of ways. Therefore these techniques benefit from taking into account many syntactic and semantic features, especially parse trees generated by automatic sentence parsers. Typed dependency parse trees are edge and node labeled parse trees whose labels and topology contains valuable semantic clues. This information can be exploited for relation extraction by the use of kernels over structured data for classification. In this paper we present new tree kernels for relation extraction over typed dependency parse trees. On a public benchmark data set we are able to demonstrate a significant improvement in terms of relation extraction quality of our new kernels over other state-of-the-art kernels.

Categories and Subject Descriptors

I.2.7 [**Artificial Intelligence**]: Natural Language Processing—*text analysis*

General Terms

Algorithms, Design

Keywords

Relation Extraction, Information Extraction, Text Mining

1. INTRODUCTION

Often the semantic content of natural language sentences may be described by relations between entities. For example the sentence "Google buys AdMob for $750 Million in stock" can approximately be described by the relation ACQUIRE(Google,AdMob). Note, that this relation can be expressed in various other ways, e.g. as "Google plans to acquire AdMob" or "Google purchase of AdMob gets closer antitrust review". Classification methods are a way to detect such variant verbalizations. The automatic extraction of the semantic content of text is important for many practical problems like the automatic gathering of facts about entites from large text corpora for the contruction of knowledge bases. This problem is especially relevant when dealing with large amounts of textual data which contains interesting facts about entites e.g. news articles containing important facts about companies. For example, investment firms and banks are naturally interested in automatically acquired knowledge about companies from news ticker articles.

In a first step possible relation arguments are extracted by entity recognition methods performing mention detection, named entity recognition and co-reference resolution. In a second step a classifier decides for each pair of possible arguments whether the sentence expresses the target relation between the two arguments. It is obvious that many syntactic and semantic properties of a sentence have to be evaluated to detect relations. Parse trees provide extensive information on syntactic structure and can be automatically generated by parsers. Recently tree kernels have been developed which operate on parse trees for classification purposes in kernel based classifiers. Those kernels allow to compare large (often exponential, or in some cases, infinite) numbers of characteristics of trees in polynomial time. There are several variants of syntactic parse trees. In contrast to *phrase grammar* parse trees having nonterminal structural nodes, dependency parse trees only consist of word nodes, therefore allowing for a bijective mapping between words in the sentence and nodes in the tree. Their links characterize directly the syntactic dependency of words and may either be *untyped* or *typed* with edge labels providing additional information about the syntactic and semantic function of a link. While tree kernels for relation extraction have been proposed for phrase grammar parse trees [26] and untyped dependency parse trees [7, 19, 22] there are currently no kernels available considering subtree similarity in node and edge labeled trees (typed dependency parse trees).

In this paper we describe a new approach for relation extraction based on kernels for *typed dependency trees* which deeply integrates the edge labels in the kernel computations. On a public benchmark data set we show that on average the new kernels outperform other kernels by 2.4% F-Score on coarse grained relations which is an error reduction of 7.5%. On some fine grained relations our new kernels outperform other kernels by up to 9.4% F-Score and reduce the error

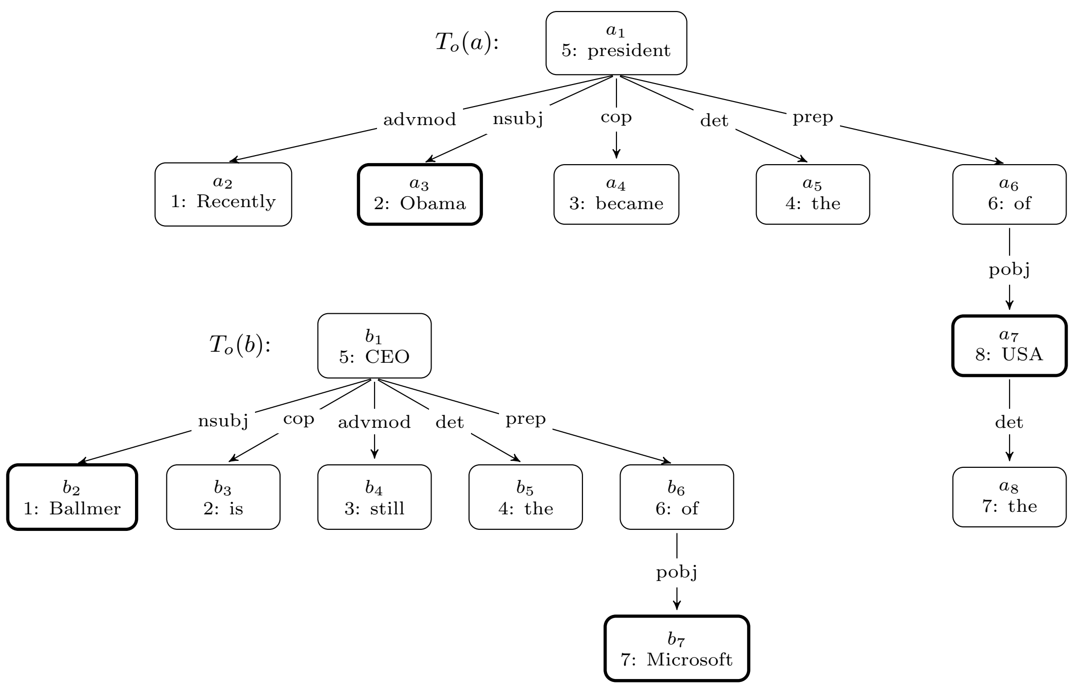

Figure 1: The typed dependency trees of two example sentences. The nodes are labeled with the position of the word in the sentence and the word itself. The edges are labeled with the type of the dependency. A thick border marks an entity mention.

by up to 27.9%. The results provide strong evidence that the usage of our typed dependency tree kernels for relation extraction significantly improves the extraction quality.

The remainder of the paper is organized as follows. We start by giving a short overview of the related work and outline typed dependency parse trees. After introducing our notation we define and elaborate our novel typed dependency tree kernel. Subsequently we evaluate the kernel and compare it to previous approaches using a public benchmark data set for training and testing. Finally we summarize the results and give a conclusion.

2. RELATED WORK

An early approach to relation extraction is based on patterns [11], usually expressed as regular expressions. The underlying hypothesis assumes that terms sharing similar linguistic contexts are connected by similar semantic relations. The manual identification of these patterns inspired other approaches in which the initial set of patterns is used to seed a bootstrapping process that automatically acquires additional patterns for relations [1]. However, the meaning of the extracted relations is often quite vague. Various authors follow related approaches, e.g. using frequent itemset mining to extract word patterns [3]. Others take into account the sequential structure of sentences, e.g. by conditional random fields [5]. These authors annotate words with rich features, e.g. part-of-speech information, and tag the words of the relation arguments by labels. The model is trained using manually annotated data and can, for instance, successfully predict semantic relations between diseases and treatments in the medical domain. A similar approach is used by [2].

Syntactic parse trees provide detailed information on syntactic structure and can, for instance, represent the relation between subject, verb and object in a sentence. For feature-based methods only a limited number of structural properties may be compared. Using parse trees [12] employ logic-based frequent structural patterns mining for relation extraction with promising results. Based on concepts of [25] and string kernels, [7] proposed the *Dependency Tree Kernel* to transform parse trees for the use in support vector machine classifiers. [6] investigated the *Shortest Path Kernel* that computes similarities between nodes on the shortest path of a dependency tree that connects the entities. [22] suggested the *Path Dependency Tree Kernel* and *All-pairs Dependency Tree Kernel* for untyped dependency parse trees yielding improved extraction accuracy. All those kernels accounts for sophisticated node similarity functions. [19] altered a node labeled tree to include new artificial nodes representing semantical dependencies and employed the partial tree kernel [18], which counts the common partial trees without considering a fine-grained node simililarity. Kernels for other types of parse tree paradigms e.g. phrase grammar parse trees have also been proposed [26]. All these kernels can be used inside a kernel classifier e.g. a Support Vector Machine (SVM) [24].

Besides work on tree kernels for relation extraction there have been tree kernels proposed for other tasks like question classification [18] which have shown improvements over bag-of-words approaches. Considering features associable with words, [10] proposed to use semantic background knowledge from various sources like WordNet, FrameNet and PropBank to enhance the Dependency Tree Kernel of [7] yielding good results.

- dep - dependent
  - aux - auxiliary
    - auxpass - passive auxiliary
    - cop - copula
  - conj - conjunct
  - cc - coordination
  - arg - argument
    - subj - subject
      - nsubj - nominal subject
        - nsubjpass - passive nominal subject
      - csubj - clausal subject
    - comp - complement
      - obj - object
        - dobj - direct object
        - iobj - indirect object
        - pobj - object of preposition
      - attr - attributive
      - ccomp - clausal complement with internal subject
      - ...
    - agent - agent
  - ref - referent
  - expl - expletive (expletive there)
  - mod - modifier
    - tmod - temporal modifier
    - advmod - adverbial modifier
    - det - determiner
    - prep - prepositional modifier
    - ...
  - sdep - semantic dependent
    - xsubj - controlling subject

Figure 2: Abbreviated list of the 48 different link labels for the Stanford Parser. The hierarchy is indicated by indentation.

Name	Values
word	*Balmer, CEO,...*
POS	NN, NNP,...
general-POS	noun, verb, adj,...
chunk tag	NP, VP, ADJP,...
entity type	person, location,...
mention type	name, nominal,...
hypernym	executive, city,...
relation argument	1, 2 or n/a

Table 1: List of features assigned to every node which are considered in the similarity function.

3. TYPED DEPENDENCY TREES

The grammatical dependency between the words of a sentence can be represented by a *typed dependency tree* which is a labeled directed tree where each node corresponds to a word [15]. The structure is determined by the relation between a word (a head) and its dependents. Dependent words in turn can be the head for other words yielding a tree structure. The edges of typed dependency trees are labeled by a number of attributes [8], e.g. as “attributive”, “marker” or “relative”. An overview of the edge types is given in figure 2. Dependency trees are well suited to represent the sentence structure for languages with free word-order e.g. German in a structured way. In this paper we consider typed and untyped dependency trees [9] as generated automatically by the *Stanford Parser* [14]. As an example consider the two sentences $a =$ “Recently Obama became the president of the USA” and $b =$ “Ballmer is still the CEO of Microsoft” which have the tree representations as shown in figure 1.

4. NOTATION

We denote the *typed* dependency parse tree of a sentence w as $T^*(w) = (V, E)$ which is a directed ordered tree with nodes V and edges E. Each node v_i is labeled with a word and each edge is labeled with a type as provided by a parser. For each node $v \in V$ the ordered sequence of its m children $ch(v) = (u_1, \dots, u_m)$ is given by the positions of the corresponding words in the sentence. We denote the set of all possible subsequence index sets of sequence s of length k as I_k. As in [23] the subsequence defined by the ordered sequence $o(i)$ of an index subset i is called $s(\mathbf{i})$. We can write the subsequence of children referenced by $\mathbf{i}$ from a node v of an ordered tree as $ch(v, \mathbf{i})$. In figure 1, for example, $ch(a_1) = (a_2, a_3, a_4, a_5, a_6)$ has the index set $\{1, 2, 3, 4, 5\}$ as it has 5 children. The subset $\{2, 3\}$ of this set in its representation as ordered sequence $(2, 3)$ defines the subsequence $ch(a_1, (2, 3)) = (a_3 = \text{“Obama”}, a_4 = \text{“became”})$ of child nodes of a_1.

In this paper we consider relations between two entities. Those entities can be automatically tagged and categorized by named entity recognition tools. We treat multiword entities as a single word w, e.g. the two words “Steve Ballmer” will be merged to a new word “Steve_Ballmer”. The *lowest common ancestor* $lca(v, w)$ of two nodes $v, w \in V$ in a directed tree $T = (V, E)$ is the lowest node in T which has both v, w as descendants. We define the set of nodes $sub(v, w)$ of two nodes v, w as the set containing all nodes of the complete subtree of a tree T rooted in $lca(v, w)$. In a directed tree T we define the implied path $p(u, v)$ between two nodes u, v as the sequence $(u, \dots, lca(u, v), \dots, v)$ of nodes where $u, \dots$ are the nodes on the path from the $lca(u, v)$ to u, analogous for v. As in a tree a node has always just one incoming edge we denote the label of this edge of a node u as $in(u)$.

5. TYPED DEPENDENCY TREE KERNELS

Our development is motivated by the Dependency Tree Kernel [7] which is defined over *untyped* dependency trees, and thus ignores types of dependencies (edge labels) between the tree nodes. We adapt the Dependency Tree Kernel by introducing the similarity of two edge labels as a decay factor for the whole underlying subtree. To weaken this strong influence of the edge labels we introduce a new parameter α, which defines the weight of the edge similarity in the computation of the subtree similarity. The resulting kernel over typed dependency trees is called Typed Dependency Kernel (TDK).

5.1 Typed Dependency Kernel

We propose the TDK which compares two dependency parse trees starting from the lowest common ancestors, i.e. the root nodes of the smallest subtrees enclosing both target entities of the respective instances. The tree nodes are characterized by different features like the word itself (“Obama”, “became”, ...), the part-of-speech tag (“NNP”, “VBD”, ...) or the general part-of-speech (“Noun”, “Verb”, ...), etc. An overview of the features is given in table 1.

To compare the relations in two instances $X = (x, (x_1, x_2))$, $Y = (y, (y_1, y_2))$ with trees x, y and relation argument nodes (x_1, x_2), (y_1, y_2) the TDK applies the node kernel Δ to the lowest common ancestors (LCA) of the relation argument nodes:

$$K_{\text{TDK}}(X,Y) = \Delta(lca(x_1,x_2), lca(y_1,y_2)) \tag{1}$$

The typed node kernel Δ is defined as

$$\Delta(u,v) = \begin{cases} 0 & \text{if } mat(u,v) = 0 \\ \gamma \cdot \big(sim(u,v) + \\ \quad C(ch(u),ch(v))\big) & \text{otherwise} \end{cases}$$

The matching function $mat(u,v) \to 0,1$ checks whether the nodes u and v are compatible, which is the case if general-POS, entity-type and relation argument match. The similarity function $sim(u,v) \to \mathcal{N}$ counts the number of matching features of two nodes u,v. To include the edge labels (the dependency types) into the kernel we define an additional decay factor

$$\gamma = 1 - \alpha + \alpha \cdot edgeSim(in(u), in(v)) \tag{2}$$

where $\alpha \in [0,1]$ is the amount of influence of the edge label similarity. The edge similarity of the incoming edges of two nodes u and v is given by $edgeSim(in(u), in(v)) \to [0,1]$. It is a symmetric function, whose concrete characteristics are described in section 6. For $\alpha = 0$ the edge label similarity is not considered, and thus the TDK is equal to the Dependency Tree Kernel [7]. The edge similarity therefore weights the similarity of the subtrees below two nodes.

The recursive computation of the underlying substructures is performed by $C(ch(u),ch(v))$, which is defined as

$$C(s,t) = \sum_{\substack{\mathbf{i} \in I_{|s|},\ \mathbf{j} \in I_{|t|}, \\ |\mathbf{i}|=|\mathbf{j}|}} \lambda^{d(\mathbf{i})+d(\mathbf{j})} \quad \Delta'(s(\mathbf{i}), t(\mathbf{j})) \quad M(s(\mathbf{i}), t(\mathbf{j}))$$

where s,t are sequences of nodes and $s(\mathbf{i})$ and $t(\mathbf{j})$ are the subsequences of nodes implied by the index sets $\mathbf{i},\mathbf{j}$. Furthermore the parameter $0 < \lambda \leq 1$ is a penalty factor for lengths and gaps of the subsequences, and $d(\mathbf{i})$ is the covered distance of the index sequence $\mathbf{i}$. With Δ' the node kernel Δ can be called on node sequences:

$$\Delta'\left((s_1,\ldots,s_n),(t_1,\ldots,t_n)\right) = \sum_{k=1}^{n} \Delta\left(s_k,t_k\right)$$

The function M checks if a sequence of nodes matches according to the matching function mat:

$$M((s_1,\ldots,s_n),(t_1,\ldots,t_n)) = \prod_{k=1}^{n} mat(s_k,t_k)$$

For the runtime critical kernel computations we employ the index caching strategy proposed by [22] for the Dependency Tree Kernel which allows for a general efficient computation of subsequence functions like $C(s,t)$. Because it has been proven that the sum and product of valid kernels are also valid kernels [23] and that the set of all valid kernels is closed under scalar multiplication, it is clear that K_{TDK} is a kernel because it is based on tree child node subsequence similarity functions which have been shown to be kernels [25].

Based on the TDK we can adapt strategies proposed by [22] for the Dependency Tree Kernel to the *typed* case which will allow for an efficient consideration of the structure of the subtrees in a more sophisticated way.

5.2 All-Pairs-TDK

Let V_x and V_y be the sets of nodes contained in the respective subtrees provided by the lowest common ancestors of the relation argument nodes of the two instances x and y. The All-Pairs-TDK applies the node kernel to every combination of nodes contained in these sets:

$$K_{\text{All-Pairs-TDK}}(X,Y) = \sum_{u \in V_x} \sum_{v \in V_y} \Delta(u,v)$$

where $\Delta(u,v)$ is the Typed Dependency Kernel. This kernel considers the similarity of every possible pair of nodes below and on the shortest path based on the similarities of the subtrees computed by the node kernel Δ of the Typed Dependency Kernel (TDK).

5.3 Path-TDK

The All-Pairs-TDK considers every node pair including substructures possibly irrelevant for the current relation and therefore introducing noise. This kernel follows the shortest path hypothesis [6] which states that the most valuable information is contained on the shortest path between the two relation argument nodes. However [22] showed that the substructure below the nodes on the path also holds valuable clues and proposed the Path Dependency Tree Kernel. We adapt this strategy to the typed case:

$$K_{\text{Path-TDK}}(X,Y) = \sum_{\substack{\mathbf{i} \in I_{|x|},\ \mathbf{j} \in I_{|y|}, \\ |\mathbf{i}|=|\mathbf{j}|,\ d(\mathbf{i}),d(\mathbf{j}) \leq q}} \mu^{d(\mathbf{i})+d(\mathbf{j})} \cdot \Delta'(x(\mathbf{i}), y(\mathbf{j})) \cdot M(x(\mathbf{i}), y(\mathbf{j}))$$

where $\Delta'(x(\mathbf{i}), y(\mathbf{j}))$ is the Typed Dependency Tree Kernel applied to all possible subsequences of nodes on the shortest path between the relation argument nodes. I_k is the set of all index sequences and $d(\mathbf{i}) = max(\mathbf{i}) - min(\mathbf{i}) + 1$ is the covered distance. The parameter $0 < \mu \leq 1$ is a regularization factor for gaps in the subsequence and $q \in \mathcal{N}$ is the limitation on the length of the subsequences of nodes on the shortest path. The function M is the sequence matching function defined in section 5.1. Therefore Path-TDK considers the similarity of the substructures from node pairs on the shortest path between two relation argument nodes. Because it has been proved that the sum and product of valid kernels is also a valid kernel [23] it is clear that the $K_{\text{All-Pairs-TDK}}$ and the $K_{\text{Path-TDK}}$ are kernels.

5.4 Efficient Computation

For the efficient computation of the TDK, which underlies the AP-TDK and Path-TDK, we looked into different computation approaches for $C(s,t)$, the child subsequence similarity, which counts the matching subsequences in an ordered labeled tree. For the computation of this function different approaches were suggested. [25] proposed an $O(mn^3)$ algorithm (*Zelenko*) for sequence lengths $n \leq m$. Another approach was later proposed by [22] yielding an $O(mn^2)$ solution based on dynamic programming (*DP*). However as those two approaches were motivated by string subsequence enumeration approaches [16, 23] they are most efficient for very large sequences. Especially their actual runtime for

Kernel	$\alpha = 0.0$	$\alpha = 0.1$	$\alpha = 0.2$	$\alpha = 0.3$	$\alpha = 0.4$	$\alpha = 0.5$	$\alpha = 0.6$	$\alpha = 0.7$	$\alpha = 0.8$	$\alpha = 0.9$	$\alpha = 1.0$
TDK	64.4 (.2)	66.2 (.2)	66.2 (.2)	66.3 (.2)	66.4 (.2)	66.4 (.3)	66.5 (.3)	66.5 (.3)	66.5 (.2)	66.4 (.2)	66.5 (.2)
All-Pairs-TDK	66.4 (.1)	68.3 (.2)	68.5 (.2)	68.7 (.2)	68.8 (.2)	68.9 (.2)	68.9 (.2)	68.9 (.2)	69.0 (.2)	69.0 (.2)	69.1 (.2)
Path-TDK	70.4 (.2)	72.2 (.2)	72.4 (.1)	72.5 (.2)	**72.7** (.1)	**72.7** (.1)	72.6 (.1)	72.5 (.1)	72.6 (.2)	72.5 (.2)	72.3 (.2)

Table 2: The F_{micro}-scores for different values of the edge similarity weight parameter α and different typed dependency tree kernels with 5-times repeated 5-fold cross validation. The values in parenthesis denote the standard error.

small values is dominated by the initialization of helper variables. For smaller sequence lengths the *index cache* strategy has been shown to be more efficient [22].

As the average outdegree of the nodes in the typed dependency trees is just 2.02 the application of the *index caching* strategy for the computation of $C(s,t)$ is promising. Let $SI_{p,q} = \{o(i) | i \in I_p \wedge |i| \leq q\}$ be the set of all ordered index subsequences of length q with highest index p. We can write the child subsequence kernel $C(s,t)$ as

$$C(s,t) = \sum_{q=1}^{min(m,n)} \sum_{\mathbf{i} \in SI_{m,q}} \sum_{\mathbf{j} \in SI_{n,q}} \lambda^{d(\mathbf{i})+d(\mathbf{j})} \; \Delta'(s(\mathbf{i}), t(\mathbf{j})) \; M(s(\mathbf{i}), t(\mathbf{j}))$$

with $m = |s|$ and $n = |t|$. This reformulation of the child subsequence formula allows an efficient computation by utilizing a cache for the Δ function. The computation procedure called *index caching* is described in the following pseudo-code. It is easy to see that the worst case runtime (if all nodes match) of this algorithm for $n \leq m$ is

$$\mathrm{O}\left(\sum_{q=1}^{n} q \cdot \binom{n}{q} \cdot \binom{m}{q}\right)$$

Algorithm *Index Cache:* For sequences of nodes s, t compute $C(s,t)$ with prepared index sequences SI.

```
 1: $k \leftarrow 0,\ m \leftarrow |s|,\ n \leftarrow |t|,\ p \leftarrow min(m,n)$
 2: for $q = 1 : p$ do
 3:   for $\mathbf{i} \in SI_{m,q}$ do
 4:     for $\mathbf{j} \in SI_{n,q}$ do
 5:       $x \leftarrow 0$
 6:       for $r = 1 : q$ do
 7:         if $mat(s(\mathbf{i}(r)), t(\mathbf{j}(r))) = 0$ then
 8:           goto 4 // jump to next index sequence $\mathbf{j}$
 9:         end if
10:         $x \leftarrow x + \Delta(s(\mathbf{i}(r)), t(\mathbf{j}(r)))$
11:       end for
12:       $k \leftarrow k + x\ \lambda^{d(\mathbf{i})+d(\mathbf{j})}$
13:     end for
14:   end for
15: end for
16: return $k$
```

We looked at the distribution of the different node degrees $m = |s|$, $n = |t|$ in typed dependency trees which governs the computation of $C(s,t)$. The distribution of the different combination during the computation of $C(s,t)$ is shown in figure 3. In addition the exact theoretical runtime (including cost for initialization of helper variables) of the three possible approaches for the computation of $C(s,t)$ is depicted. As can be seen the *index cache* strategy beats the other two computation approaches in exact runtime in 94.3% of the cases or is only slightly slower. Only in the infrequent cases where $m,n \geq 5$ the other solutions are theoretically faster. In order to show that this is useful in practice, we performed an empirical runtime analysis over all possible combinations of typed dependency trees of the whole corpus. We performed five consecutive runs of these experiments on the same machine. The results are depicted in table 3. The average absolute runtime for the *index cache* was 76.9 minutes. This provides strong evidences that the utilization of the *index cache* strategy is indeed reasonable for the computation of the TDK, AP-TDK, and Path-TDK.

Algorithm	Theoretical runtime	Rel. emp. runtime	Std. dev.
Index Cache [22]	$\sum_{q=1}^{n} q \binom{m}{q} \binom{n}{q}$	1	–
DP [22]	mn^2	2.2013	0.1057
Zelenko [25]	mn^3	2.1175	0.1055

Table 3: Empirical runtimes of the TDK with three different subsequence algorithms. The runtimes are averaged over five measurements and relative to the fastest algorithm.

6. EDGE SIMILARITY

The label of an edge in a *typed* dependency tree expresses a syntactic relation between the corresponding words. The label `nsubj` (nominal subject) for the edge $a_1 \rightarrow a_3$ in figure 1, for example, implies that a_3 = "Obama" is the syntactic subject of a_1 = "president". Some edge labels express very similar syntactic relations being quite dissimilar to other relations. For example the label type `nsubj` is syntactically far more similar to `csubj` (clausal subject) than it is to the `tmod` temporal modifier label.

The hierarchical edge label set of the Stanford Parser [8, 9] is shown in figure 2. We used it to define the similarity of edge labels for our typed dependency tree kernel. The hierarchy is a tree H in which each node is labeled with a type and has edges to its subtypes. The root `dep` of H is the most general type, whose children are the more specific subtypes like `aux` (auxiliary) and `arg` (argument). With $p(u,v)$ denoting the path between two nodes u,v in H, there are various ways of extracting a symmetric similarity score out of the dependency type hierarchy. Our preliminary experiments revealed that the difference between alternatives are minor. Therefore we only report the best performing choice which is the

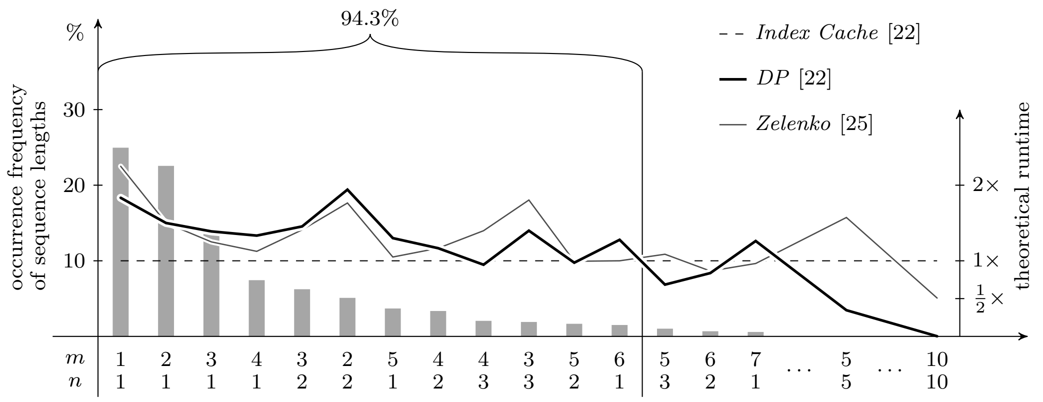

Figure 3: The algorithm runtimes and the occurrence frequencies. The vertical bars represent the occurrence frequency of sequence length combinations in the whole corpus. The lines denote the exact theoretical runtimes (including initializations) of the different algorithms relative to the *Index Cache* algorithm runtimes. In 94.3% of the cases the *Index Cache* algorithm is the fastest or only slightly slower than the other algorithms.

reciprocal of the path length and omitted the other ones for brevity. We define the similarity between edge labels as

$$edgeSim(u,v) = \frac{1}{|p(u,v)|}$$

which is higher if the path between the labels in figure 2 is short. Please note that $p(u,u) = (u)$. For example we get

$$edgeSim(\texttt{nsubj}, \texttt{csubj}) = 1/3$$
$$\text{as } p(\texttt{nsubj}, \texttt{csubj}) = (\texttt{nsubj}, \texttt{subj}, \texttt{csubj})$$
$$edgeSim(\texttt{tmod}, \texttt{csubj}) = 1/6$$
$$\text{as } p(\texttt{tmod}, \texttt{csubj}) = (\texttt{tmod}, \texttt{mod}, \texttt{dep}, \texttt{arg}, \texttt{subj}, \texttt{csubj})$$

This example outlines that two semantically more related types are assigned a higher similarity then two less related. The TDK has the same computational complexity as the Dependency Tree Kernel [7] as for all possible pairs of edge labels the *edgeSim* function can be calculated in advance and be retrieved in $O(1)$.

7. EXPERIMENTS

In this section we compare the performance of our new kernels for relation extraction with the performance of other dependency tree kernels. As benchmark data set we use the public ACE-2003 corpus [17].

7.1 Data Set

The Automatic Content Extraction (ACE) evaluation organized by the National Institute of Standards and Technology (NIST) covers the task of relation extraction. During this evaluation an annotated corpus was created suitable for supervised learning approaches. The ACE-2003 data set [17] consists of 519 documents which are all news related, e.g. newspaper articles or newswire texts. This data set comes with a pre-specified training and test set. The training set contains texts from the time period between January 1998 and July 1998 whereas the test contains document from October 1998 to December 1998. In all those documents entities and relations between the entites were annotated by human annotators. The annotation of entity mentions included their type, e.g. person or organization as well as to their mention type, i.e. named, nominal or pronominal. In total 24 different fine-grained relations like LOCATED and AFFILIATE-PARTNER were annotated, which were grouped into the following 5 top level relations: AT (587 Instances), NEAR (58 Instances), PART (329 Instances), ROLE (887 Instances) and SOCIAL (59 Instances).

7.2 Data Preparation

As our approach operates on parse trees we used the *Stanford Parser* [14], a freely available state-of-the-art parser producing parse trees with a high quality. We have run the parser on each document of the data-set and generated a *typed* and *untyped* parse tree for each sentence. Using the manually annotated entities each dependency tree was subsequently processed by our system, which first merged the multi-word entities and then associated each node with information like part-of-speech tags, entity type or mention type[1]. As currently neither nominal nor pronominal co-reference resolution can be done with sufficient quality, we restricted our experiments to *named–named* relations, where named entity recognition approaches may be used to extract the arguments in real-world scenarios without a labeled corpus. Nevertheless our kernels without any modification can also be applied to the all mention types setting as well. For each sentence we generated $\binom{n}{2} \cdot 2 = n \cdot (n-1)$ instances, where n is the number of named entity mentions in the sentence. Those instances together with the label of the relation type between the corresponding entity mention pairs are used as the input for the kernel based classifier. For this multi-class classification problem we used the one-vs-all classification approach and selected the class with the highest confidence.

7.3 Experimental setup

As the kernel classifier we use the Support Vector Machine implementation SVM^{light}[13] and implemented all reported tree kernels. Because the preprocessing and the experimental setup is crucial, this was the only way which allows a fair comparison of the different approaches.

[1]Also known as entity mention level

Kernel	5-times repeated 5-fold cross validation Prec	Rec	F_{micro}	F_{macro}	pre-specified test set Prec	Rec	F_{micro}	F_{macro}
Averaged over the five top-level relations								
SPK [Bunescu05]	**81.7**	47.1	59.8 (0.1)	35.1 (0.2)	72.7	34.9	47.2	25.4
DK [Culotta04]	72.7	56.4	63.5 (0.3)	46.1 (0.9)	77.5	46.3	57.9	39.3
TDK	75.3	59.5	66.5 (0.3)	50.8 (1.0)	78.5	46.9	58.7	40.1
All-Pairs-DK [Reichartz09]	73.1	60.2	66.0 (0.2)	59.7 (0.6)	77.8	51.3	61.9	41.9
All-Pairs-TDK	76.5	62.6	68.9 (0.2)	**62.6** (0.7)	78.8	51.0	62.0	42.2
Path-DK [Reichartz09]	79.2	63.1	70.3 (0.1)	59.0 (0.4)	76.1	54.0	63.2	42.3
Path-TDK	81.1	**65.9**	**72.7** (0.1)	62.1 (0.9)	**79.1**	**55.2**	**65.0**	**43.9**
Averaged over the 24 subrelations								
SPK [Bunescu05]	76.8	33.3	46.5 (0.2)	19.6 (0.2)	64.2	20.9	31.5	10.7
DK [Culotta04]	66.1	46.8	54.8 (0.2)	25.8 (0.2)	73.6	38.2	50.3	15.3
TDK	69.8	50.0	58.3 (0.2)	27.6 (0.1)	75.1	39.7	52.0	15.0
All-Pairs-DK [Reichartz09]	70.1	51.8	59.6 (0.2)	31.5 (0.1)	74.6	40.3	52.3	**18.6**
All-Pairs-TDK	73.5	53.8	62.1 (0.1)	32.8 (0.2)	**77.3**	39.7	52.5	15.1
Path-DK [Reichartz09]	76.3	53.7	63.1 (0.3)	32.2 (0.2)	69.6	39.7	50.6	17.3
Path-TDK	**77.5**	**56.3**	**65.2** (0.2)	**33.7** (0.1)	74.5	**42.7**	**54.3**	18.4

Table 4: Precision, recall, and F-scores in % for 5-times repeated 5-fold cross validation and on the pre-specified test set. The values in parenthesis denote the standard error. The upper and lower table gives results averaged over the five top-level relations and the 24 subrelations, respectively. Here SPK denotes the Shortest Path Kernel [6], DK denotes the Dependency Tree Kernel [7] and All-Pairs-, Path-DK denote the All-Pairs-, Path-Dependency Tree Kernel [22].

For the state-of-the-art kernels we used the same kernel and classifier parameters as in the cited papers. To avoid a fitting of free parameters to the test data and reporting biased results they were determined on data different from the test set. To estimate the variability and significance of differences between methods we have conducted a 5-times repeated 5-fold cross validation [4]. In addition we compared the performances on the pre-specified ACE test set.

We report the usual evaluation measure Precision and Recall, the F-score averaged over the classes (F_{macro}) and averaged over the instances (F_{micro}). For the cross validation runs we report the standard error of F-score as estimated according to [4].

We investigated the sensitivity of the F-score to the value of the parameter α which determines the influence of the edge labels on the similarity of parse trees. The results are shown in table 2. For all weights except 0 we get similar F-scores. For the Path-TDK, for example, we get scores ranging from 72.2% to 72.7% whereas ignoring the edge labels ($\alpha = 0$) leads to a significant decrease of the F-score to 70.4%. This indicates a low sensitivity of the results to the exact value of α.

7.4 Results

The upper part of table 4 describes the performance for top-level relations. It shows an overview of the results of different kernels on the typed dependency trees on the cross validation and the pre-specified test set. For the cross validation the Path-TDK outperforms all other kernels by at least 2.4% F_{micro} and 3.1% F_{macro}. In addition the typed kernels TDK, All-Pairs-TDK and Path-TDK all beat their untyped counterparts DK, All-Pairs-DK and Path-DK with an F_{micro} difference of at least 2.4% and an F_{macro} difference of at least 2.9%. The difference in F-values between the two best kernels Path-DK and Path-TDK is significant at a level of 99.9% according to the corrected resampled t-test [4]. Hence the types in typed dependency tree kernels provide essential information to improve relation extraction performance.

On the separate test set the Path-TDK achieved an F_{micro} of 65.0% which is an improvement of 2.8% over the second best kernel. The overall reduction in classification accuracy compared to the cross validation can be explained by the fact that the training documents are from a different time period then the test documents. Again the typed tree kernels show a better performance than their untyped variants, except for All-Pairs-DK and All-Pairs-TDK, which have nearly equal F-scores.

In the lower part of table 4 results for the more specific 24 subrelations are reported. This table gives a similar picture as for the top-level relations. Again the Path-TDK exhibits the best performance, yielding a 2.1% improved F_{micro} which is highly significant at a level of 99.9% according to the corrected resampled t-test. Again all typed dependency tree kernels do better than their untyped counterparts.

In table 5 results for the three top-level relations AT, PART and ROLE with the largest number of training examples are shown. Again the typed dependency kernels have the best results for the cross validation as well as for the prespecified test set. In most cases the Path-DTK has optimal per-

Relation	Kernel	5-times rep. 5-fold cross validation Prec	Rec	F-score.	pre-specified test set Prec	Rec	F-score
At	SPK [Bunescu05]	**79.7**	44.1	56.8 (0.4)	58.6	33.0	42.2
	DK [Culotta04]	64.7	47.1	54.5 (0.3)	65.6	40.8	50.3
	TDK	65.7	49.6	56.5 (0.4)	60.9	40.8	48.8
	All-Pairs-DK [Reichartz09]	64.3	53.9	58.6 (0.3)	63.2	46.6	53.6
	All-Pairs-TDK	68.7	57.4	62.6 (0.4)	**67.1**	47.6	55.7
	Path-DK [Reichartz09]	72.4	58.8	64.9 (0.3)	62.4	**51.5**	56.4
	Path-TDK	75.6	**62.3**	**68.3** (0.2)	65.0	50.5	**56.8**
Part	SPK [Bunescu05]	77.1	26.4	39.4 (0.5)	60.0	14.3	23.1
	DK [Culotta04]	69.5	55.9	62.0 (0.3)	87.1	42.9	57.4
	TDK	80.6	67.4	73.4 (0.3)	**90.6**	46.0	**61.1**
	All-Pairs-DK Reichartz09]	68.7	54.0	60.4 (0.6)	87.5	44.4	58.9
	All-Pairs-TDK	79.6	61.7	69.5 (0.4)	82.9	46.0	59.2
	Path-DK [Reichartz09]	80.2	59.1	68.1 (0.3)	87.9	46.0	60.4
	Path-TDK	**86.5**	**68.4**	**76.4** (0.2)	83.3	**47.6**	60.6
Role	SPK [Bunescu05]	**83.8**	62.4	71.5 (0.0)	84.1	49.0	61.9
	DK [Culotta04]	80.5	68.5	74.0 (0.3)	83.2	55.6	66.7
	TDK	80.8	68.9	74.4 (0.3)	83.2	55.6	66.7
	All-Pairs-DK [Reichartz09]	79.9	69.1	74.1 (0.3)	84.7	62.3	71.8
	All-Pairs-TDK	80.6	69.2	74.5 (0.3)	86.7	60.3	71.1
	Path-DK [Reichartz09]	83.3	**71.9**	**77.1** (0.1)	83.1	**64.9**	72.9
	Path-TDK	83.3	**71.9**	**77.1** (0.2)	**86.0**	**64.9**	**74.0**

Table 5: Precision, recall, and F-scores in % for 5-times repeated 5-fold cross validation and on the pre-specified test set for the three top-level relations with most training examples. The values in parenthesis denote the standard error. Here SPK denotes the Shortest Path Kernel [6], DK denotes the Dependency Tree Kernel [7] and All-Pairs-, Path-DK denote the All-Pairs-, Path-Dependency Tree Kernel [22].

formance except for the Part relation, where the TDK is slightly better for the prespecified test set.

Table 6 lists the performance figures for six fine-grained relations with the largest number of examples. The results exhibit a much higher fluctuation than the summary figures given before[2]. A possible explanation is the smaller number of observations for each relation. For the Part.Part-of relation and the At.Based-in relation the Path-TDK yields much better results than all other kernels. The Path-TDK shows an improvement of at least 9.4% F-scores over the state-of-the-art kernels on the At.Based-in relation. On the At.Based-in relation the Path-TDK achieves an improvement of at least 8.4% F-score. The two differences in F-values between the two best kernels Path-DK and Path-TDK is significant at a level of 99.9%. For the other relations, the difference is much smaller, and for two relations untyped approaches are better. Although the average figures indicate that typed dependency tree kernels generally achieve better results, there is a need to detect and utilize relation-specific features in order to improve learning.

8. CONCLUSION AND FUTURE WORK

In this paper we developed novel typed dependency tree kernels to detect semantic relations in natural language sentences. We were able to show that the additional information contained in the edge labels of the parse trees can be exploited by our typed tree kernels which on average do better than their untyped counterparts. This indicates that the type information in dependency trees contains essential clues for relation detection.

In summary the Path-TDK achieved significantly better F-values than the other approaches. This holds for coarse top-level relations as well as for more finegrained relations. The results for fine-grained relations often exhibit relatively large fluctuations. This suggests that a certain number of training examples is necessary to achieve a good performance. Further research should be going into the investigation of active learning strategies for a reduction of needed annotated data. The application of this techniques to other texts like product descriptions, where the goal is the extractions of product details is promising. As Typed dependency trees may be computed with high accuracy for natural language text from various sources like newspaper or social media. This opens a new way to reach better performance in practical relation extraction from text. As combining kernels for untyped dependency trees and phrase grammar trees is rewarding [19, 21], we plan to evaluate composite kernels for typed, untyped and phrase grammar tree kernels. Another promising research direction is the inclusion of other types of semantic information into the kernels. To this end word sense disambiguation approaches [20] might be employed or topic modeling algorithms may be used to assign broader semantic information to the words in a sentence.

[2]We omitted results for the test-set, as 37% of the different subrelations have no observation in the test-set.

9. ACKNOWLEDGEMENT

The work presented here was funded by the German Federal Ministry of Economy and Technology (BMWi) under the THESEUS project.

10. REFERENCES

[1] M. Banko, M. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open information extraction from the web. In *Proceedings IJCAI '07*, pages 2670–2676, 2007.

[2] M. Banko and O. Etzioni. The tradeoffs between open and traditional relation extraction. In *Proceedings of ACL-08: HLT*, pages 28–36, 2008.

[3] S. Blohm and P. Cimiano. Scaling up pattern induction for web relation extraction through frequent itemset mining. In *Proceedings KI 2008 WS on Ontology-Based IE Systems*, 2008.

[4] R. R. Bouckaert and E. Frank. Evaluating the replicability of significance tests for comparing learning algorithms. In *Proc. PAKDD 2004, Sydney, Australia*, pages 3–12, 2004.

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

[6] R. C. Bunescu and R. J. Mooney. A shortest path dependency kernel for relation extraction. In *Proceedings EMNLP '05*, 2005.

[7] A. Culotta and J. Sorensen. Dependency tree kernels for relation extraction. In *Proceedings ACL '04*, 2004.

[8] M.-C. de Marneffe, B. MacCartney, and C. D. Manning. Generating typed dependency parses from phrase structure parses. In *5th International Conference on Language Resources and Evaluation (LREC 2006)*, pages 449–454., 2006.

[9] M. C. de Marneffe and C. D. Manning. The Stanford typed dependencies representation. In *Coling 2008: Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation*, 2008.

[10] S. Harabagiu, C. A. Bejan, and P. Morarescu. Shallow semantics for relation extraction. In *Proceedings IJCAI '05*, 2005.

[11] M. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings COLING-92*, 1992.

[12] T. Horvath, G. Paass, F. Reichartz, and S. Wrobel. A logic-based approach to relation extraction from texts. In *ILP '09*, 2009.

[13] T. Joachims. Text categorization with support vector machines: learning with many relevant features. In *Proceedings ECML '98*, 1998.

[14] D. Klein and C. D. Manning. Accurate unlexicalized parsing. In *Proceedings ACL '03*, 2003.

[15] D. Klein and C. D. Manning. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Proceedings ACL '04*, 2004.

[16] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. Text classification using string kernels. *JMLR*, (2):419–444, 2002.

[17] A. Mitchell. ACE-2 Version 1.0; corpus LDC2003T11. Linguistic Data Consortium, Philadelphia. http://www.ldc.upenn.edu, 2003.

[18] A. Moschitti. Efficient convolution kernels for dependency and constituent syntactic trees. In *Proceedings ECML '06*, 2006.

[19] T.-V. T. Nguyen, A. Moschitti, and G. Riccardi. Convolution kernels on constituent, dependency and sequential structures for relation extraction. In *Proceedings EMNLP '09*, pages 1378–1387, 2009.

[20] G. Paaß and F. Reichartz. Exploiting semantic constraints for estimating supersenses with crfs. In *Proceedings SDM '09*, 2009.

[21] F. Reichartz, H. Korte, and G. Paass. Composite kernels for relation extraction. In *Proceedings ACL '09*, 2009.

[22] F. Reichartz, H. Korte, and G. Paass. Dependency tree kernels for relation extraction from natural language text. In *Proceedings ECML PKDD '09*, 2009.

[23] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[24] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, 1995.

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

[26] M. Zhang, G. Zhou, and A. Aw. Exploring syntactic structured features over parse trees for relation extraction using kernel methods. *Inf. Process. Manage.*, 44(2):687–701, 2008.

Relation / Kernel	5-times rep. 5-fold CV Prec	Rec	F-score
At.Located			
SPK [Bunescu05]	**77.6**	38.4	51.4 (0.4)
DK [Culotta04]	61.2	46.3	52.7 (0.5)
TDK	61.3	47.9	53.7 (0.5)
All-Pairs-DK [Reichartz09]	64.3	54.8	59.2 (0.8)
All-Pairs-TDK	66.9	55.3	60.5 (0.5)
Path-DK [Reichartz09]	71.9	**59.9**	**65.4** (0.6)
Path-TDK	71.0	**59.9**	65.0 (0.3)
Part.Part-of			
SPK [Bunescu05]	79.2	27.7	41.0 (1.4)
DK [Culotta04]	71.8	58.6	64.5 (0.1)
TDK	82.7	70.9	76.4 (0.2)
All-Pairs-DK [Reichartz09]	71.3	55.9	62.7 (0.4)
All-Pairs-TDK	82.1	65.2	72.7 (0.2)
Path-DK [Reichartz09]	80.5	62.5	70.3 (0.4)
Path-TDK	**87.6**	**72.3**	**79.2** (0.2)
Role.General-staff			
SPK [Bunescu05]	**85.3**	62.7	72.2 (0.5)
DK [Culotta04]	78.8	65.1	71.3 (0.3)
TDK	79.8	65.1	71.7 (0.1)
All-Pairs-DK [Reichartz09]	83.0	**68.3**	**74.9** (0.5)
All-Pairs-TDK	83.6	67.7	74.8 (0.4)
Path-DK [Reichartz09]	83.4	68.1	**74.9** (0.4)
Path-TDK	82.8	67.8	74.5 (0.2)
Role.Management			
SPK [Bunescu05]	72.2	40.9	52.2 (0.4)
DK [Culotta04]	71.3	62.0	66.3 (0.4)
TDK	73.8	61.1	66.9 (0.4)
All-Pairs-DK [Reichartz09]	71.4	58.9	64.5 (0.4)
All-Pairs-TDK	72.5	58.5	64.7 (0.5)
Path-DK [Reichartz09]	75.2	63.0	68.6 (0.3)
Path-TDK	**75.8**	**63.6**	**69.2** (0.2)
Role.Affiliate-partner			
SPK [Bunescu05]	75.5	13.3	22.6 (1.5)
DK [Culotta04]	71.4	48.5	57.7 (1.4)
TDK	78.5	55.0	64.6 (1.7)
All-Pairs-DK [Reichartz09]	86.2	64.4	73.7 (1.5)
All-Pairs-TDK	**88.8**	**65.4**	**75.3** (1.1)
Path-DK [Reichartz09]	81.7	55.4	65.9 (1.9)
Path-TDK	86.0	58.0	69.2 (1.7)
At.Based-in			
SPK [Bunescu05]	50.1	19.7	28.2 (2.5)
DK [Culotta04]	43.3	24.4	31.1 (1.7)
TDK	49.5	31.0	38.0 (1.6)
All-Pairs-DK [Reichartz09]	41.2	23.8	30.2 (1.5)
All-Pairs-TDK	50.1	30.5	37.8 (1.0)
Path-DK [Reichartz09]	51.0	28.2	36.2 (1.6)
Path-TDK	**59.5**	**37.2**	**45.6** (0.7)

Table 6: Precision, recall, and F-scores all in percent for experiments on the ACE 2003 data set with 5-times repeated 5-fold cross validation for the largest six subrelations. The values in parenthesis denote the standard error. SPK denotes the Shortest Path Kernel [6], DK denotes the Dependency Tree Kernel [7] and All-Pairs-, Path-DK denote the All-Pairs-, Path-Dependency Tree Kernel [22].