Appears in *Proceedings of the Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*
Vancouver, B.C., pp. 724--731, October 2005.

A Shortest Path Dependency Kernel for Relation Extraction

Razvan C. Bunescu and **Raymond J. Mooney**
Department of Computer Sciences
University of Texas at Austin
1 University Station C0500
Austin, TX 78712
razvan,mooney@cs.utexas.edu

Abstract

We present a novel approach to relation extraction, based on the observation that the information required to assert a relationship between two named entities in the same sentence is typically captured by the shortest path between the two entities in the dependency graph. Experiments on extracting top-level relations from the ACE (Automated Content Extraction) newspaper corpus show that the new shortest path dependency kernel outperforms a recent approach based on dependency tree kernels.

1 Introduction

One of the key tasks in natural language processing is that of Information Extraction (IE), which is traditionally divided into three subproblems: coreference resolution, named entity recognition, and relation extraction. Consequently, IE corpora are typically annotated with information corresponding to these subtasks (MUC (Grishman, 1995), ACE (NIST, 2000)), facilitating the development of systems that target only one or a subset of the three problems. In this paper we focus exclusively on extracting relations between predefined types of entities in the ACE corpus. Reliably extracting relations between entities in natural-language documents is still a difficult, unsolved problem, whose inherent difficulty is compounded by the emergence of new application domains, with new types of narrative that challenge systems developed for previous well-studied domains. The accuracy level of current syntactic and semantic parsers on natural language text from different domains limit the extent to which syntactic and semantic information can be used in real IE systems. Nevertheless, various lines of work on relation extraction have shown experimentally that the use of automatically derived syntactic information can lead to significant improvements in extraction accuracy. The amount of syntactic knowledge used in IE systems varies from part-of-speech only (Ray and Craven, 2001) to chunking (Ray and Craven, 2001) to shallow parse trees (Zelenko et al., 2003) to dependency trees derived from full parse trees (Culotta and Sorensen, 2004). Even though exhaustive experiments comparing the performance of a relation extraction system based on these four levels of syntactic information are yet to be conducted, a reasonable assumption is that the extraction accuracy increases with the amount of syntactic information used. The performance however depends not only on the amount of syntactic information, but also on the details of the exact models using this information. Training a machine learning system in a setting where the information used for representing the examples is only partially relevant to the actual task often leads to overfitting. It is therefore important to design the IE system so that the input data is stripped of unnecessary features as much as possible. In the case of the tree kernels from (Zelenko et al., 2003; Culotta and Sorensen, 2004), the authors reduce each relation example to the smallest subtree in the parse or dependency tree that includes both entities. We will show in this paper that increased extraction performance can be

obtained by designing a kernel method that uses an even smaller part of the dependency structure – the shortest path between the two entities in the undirected version of the dependency graph.

2 Dependency Graphs

Let e_1 and e_2 be two entities mentioned in the same sentence such that they are observed to be in a relationship R, i.e $R(e_1, e_2) = 1$. For example, R can specify that entity e_1 is LOCATED (AT) entity e_2. Figure 1 shows two sample sentences from ACE, with entity mentions in bold. Correspondingly, the first column in Table 1 lists the four relations of type LOCATED that need to be extracted by the IE system. We assume that a relation is to be extracted only between entities mentioned in the same sentence, and that the presence or absence of a relation is independent of the text preceding or following the sentence. This means that only information derived from the sentence including the two entities will be relevant for relation extraction. Furthermore, with each sentence we associate its dependency graph, with words figured as nodes and word-word dependencies figured as directed edges, as shown in Figure 1. A subset of these word-word dependencies capture the predicate-argument relations present in the sentence. Arguments are connected to their target predicates either directly through an arc pointing to the predicate ('troops $\rightarrow$ raided'), or indirectly through a preposition or infinitive particle ('warning $\leftarrow$ to $\leftarrow$ stop'). Other types of word-word dependencies account for modifier-head relationships present in adjective-noun compounds ('several $\rightarrow$ stations'), noun-noun compounds ('pumping $\rightarrow$ stations'), or adverb-verb constructions ('recently $\rightarrow$ raided'). In Figure 1 we show the full dependency graphs for two sentences from the ACE newspaper corpus.

Word-word dependencies are typically categorized in two classes as follows:

- **[Local Dependencies]** These correspond to local predicate-argument (or head-modifier) constructions such as 'troops $\rightarrow$ raided', or 'pumping $\rightarrow$ stations' in Figure 1.

- **[Non-local Dependencies]** Long-distance dependencies arise due to various linguistic constructions such as coordination, extraction, raising and control. In Figure 1, among non-local dependencies are 'troops $\rightarrow$ warning', or 'ministers $\rightarrow$ preaching'.

A Context Free Grammar (CFG) parser can be used to extract local dependencies, which for each sentence form a dependency tree. Mildly context sensitive formalisms such as Combinatory Categorial Grammar (CCG) (Steedman, 2000) model word-word dependencies more directly and can be used to extract both local and long-distance dependencies, giving rise to a directed acyclic graph, as illustrated in Figure 1.

3 The Shortest Path Hypothesis

If e_1 and e_2 are two entities mentioned in the same sentence such that they are observed to be in a relationship R, our hypothesis stipulates that the contribution of the sentence dependency graph to establishing the relationship $R(e_1, e_2)$ is almost exclusively concentrated in the shortest path between e_1 and e_2 in the undirected version of the dependency graph.

If entities e_1 and e_2 are arguments of the same predicate, then the shortest path between them will pass through the predicate, which may be connected directly to the two entities, or indirectly through prepositions. If e_1 and e_2 belong to different predicate-argument structures that share a common argument, then the shortest path will pass through this argument. This is the case with the shortest path between 'stations' and 'workers' in Figure 1, passing through 'protesters', which is an argument common to both predicates 'holding' and 'seized'. In Table 1 we show the paths corresponding to the four relation instances encoded in the ACE corpus for the two sentences from Figure 1. All these paths support the LOCATED relationship. For the first path, it is reasonable to infer that if a PERSON entity (e.g. 'protesters') is doing some action (e.g. 'seized') to a FACILITY entity (e.g. 'station'), then the PERSON entity is LOCATED at that FACILITY entity. The second path captures the fact that the same PERSON entity (e.g. 'protesters') is doing two actions (e.g. 'holding' and 'seized') , one action to a PERSON entity (e.g. 'workers'), and the other action to a FACILITY entity (e.g. 'station'). A reasonable inference in this case is that the 'workers' are LOCATED at the

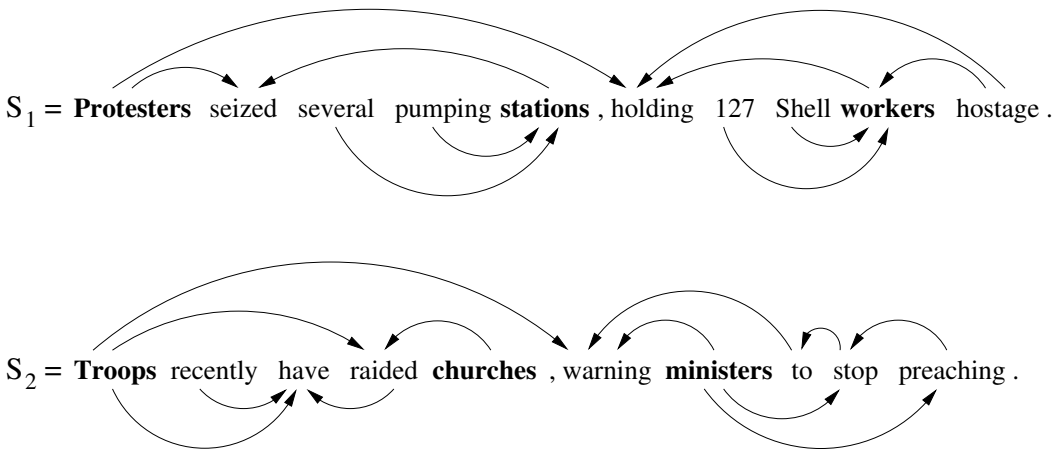

Figure 1: Sentences as dependency graphs.

Relation Instance	Shortest Path in Undirected Dependency Graph
S_1: protesters AT stations	**protesters** $\longrightarrow$ seized $\longleftarrow$ **stations**
S_1: workers AT stations	**workers** $\longrightarrow$ holding $\longleftarrow$ protesters $\longrightarrow$ seized $\longleftarrow$ **stations**
S_2: troops AT churches	**troops** $\longrightarrow$ raided $\longleftarrow$ **churches**
S_2: ministers AT churches	**ministers** $\longrightarrow$ warning $\longleftarrow$ troops $\longrightarrow$ raided $\longleftarrow$ **churches**

Table 1: Shortest Path representation of relations.

'station'.

In Figure 2 we show three more examples of the LOCATED (AT) relationship as dependency paths created from one or two predicate-argument structures. The second example is an interesting case, as it illustrates how annotation decisions are accommodated in our approach. Using a reasoning similar with that from the previous paragraph, it is reasonable to infer that 'troops' are LOCATED in 'vans', and that 'vans' are LOCATED in 'city'. However, because 'vans' is not an ACE markable, it cannot participate in an annotated relationship. Therefore, 'troops' is annotated as being LOCATED in 'city', which makes sense due to the transitivity of the relation LOCATED. In our approach, this leads to shortest paths that pass through two or more predicate-argument structures.

The last relation example is a case where there exist multiple shortest paths in the dependency graph between the same two entities – there are actually two different paths, with each path replicated into three similar paths due to coordination. Our current approach considers only one of the shortest paths, nevertheless it seems reasonable to investigate using all of them as multiple sources of evidence for relation extraction.

There may be cases where e_1 and e_2 belong to predicate-argument structures that have no argument in common. However, because the dependency graph is always connected, we are guaranteed to find a shortest path between the two entities. In general, we shall find a shortest sequence of predicate-argument structures with target predicates $P_1, P_2, ..., P_n$ such that e_1 is an argument of P_1, e_2 is an argument of P_n, and any two consecutive predicates P_i and P_{i+1} share a common argument (where by "argument" we mean both arguments and complements).

4 Learning with Dependency Paths

The shortest path between two entities in a dependency graph offers a very condensed representation of the information needed to assess their relationship. A dependency path is represented as a sequence of words interspersed with arrows that in-

(1) He had no regrets for **his** actions in **Brcko**.

his → actions ← in ← **Brcko**

(2) U.S. **troops** today acted for the first time to capture an alleged Bosnian war criminal, rushing from unmarked vans parked in the northern Serb-dominated **city** of Bijeljina.

troops → rushing ← from ← vans → parked ← in ← **city**

(3) Jelisic created an atmosphere of terror at the **camp** by killing, abusing and threatening the **detainees**.

detainees → killing ← Jelisic → created ← at ← **camp**

detainees → abusing ← Jelisic → created ← at ← **camp**

detainees → threatning ← Jelisic → created ← at ← **camp**

detainees → killing → by → created ← at ← **camp**

detainees → abusing → by → created ← at ← **camp**

detainees → threatening → by → created ← at ← **camp**

Figure 2: Relation examples.

dicate the orientation of each dependency, as illustrated in Table 1. These paths however are completely lexicalized and consequently their performance will be limited by data sparsity. We can alleviate this by categorizing words into classes with varying degrees of generality, and then allowing paths to use both words and their classes. Examples of word classes are part-of-speech (POS) tags and generalizations over POS tags such as Noun, Active Verb or Passive Verb. The entity type is also used for the two ends of the dependency path. Other potentially useful classes might be created by associating with each noun or verb a set of hypernyms corresponding to their synsets in WordNet.

The set of features can then be defined as a Cartesian product over these word classes, as illustrated in Figure 3 for the dependency path between 'protesters' and 'station' in sentence S_1. In this representation, sparse or contiguous subsequences of nodes along the lexicalized dependency path (i.e. path fragments) are included as features simply by replacing the rest of the nodes with their corresponding generalizations.

The total number of features generated by this dependency path is $4 \times 1 \times 3 \times 1 \times 4$, and some of them are listed in Table 2.

$$\begin{bmatrix} \text{protesters} \\ \text{NNS} \\ \text{Noun} \\ \text{PERSON} \end{bmatrix} \times [\rightarrow] \times \begin{bmatrix} \text{seized} \\ \text{VBD} \\ \text{Verb} \end{bmatrix} \times [\leftarrow] \times \begin{bmatrix} \text{stations} \\ \text{NNS} \\ \text{Noun} \\ \text{FACILITY} \end{bmatrix}$$

Figure 3: Feature generation from dependency path.

protesters	→	seized	←	stations
Noun	→	Verb	←	Noun
PERSON	→	seized	←	FACILITY
PERSON	→	Verb	←	FACILITY
... (48 features)				

Table 2: Sample Features.

For verbs and nouns (and their respective word classes) occurring along a dependency path we also use an additional suffix '(-)' to indicate a negative polarity item. In the case of verbs, this suffix is used when the verb (or an attached auxiliary) is modified by a negative polarity adverb such as 'not' or 'never'. Nouns get the negative suffix whenever they are modified by negative determiners such as 'no', 'neither' or 'nor'. For example, the phrase "**He** never went to **Paris**" is associated with the dependency path '**He** → went(-) ← to ← **Paris**'.

Explicitly creating for each relation example a vector with a position for each dependency path feature is infeasible, due to the high dimensionality of the feature space. Here we can exploit *dual* learning algorithms that process examples only via computing their dot-products, such as the Support Vector Machines (SVMs) (Vapnik, 1998; Cristianini and Shawe-Taylor, 2000). These dot-products between feature vectors can be efficiently computed through a *kernel* function, without iterating over all the corresponding features. Given the kernel function, the SVM learner tries to find a hyperplane that separates positive from negative examples and at the same time maximizes the separation (margin) between them. This type of max-margin separator has been shown both theoretically and empirically to resist overfitting and to provide good generalization performance on unseen examples.

Computing the dot-product (i.e. kernel) between two relation examples amounts to calculating the

number of common features of the type illustrated in Table 2. If $\boldsymbol{x}$ = $x_1x_2...x_m$ and $\boldsymbol{y}$ = $y_1y_2...y_n$ are two relation examples, where x_i denotes the set of word classes corresponding to position i (as in Figure 3), then the number of common features between $\boldsymbol{x}$ and $\boldsymbol{y}$ is computed as in Equation 1.

$$K(\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} 0, & m \neq n \\ \prod_{i=1}^{n} c(x_i, y_i), & m = n \end{cases} \quad (1)$$

where $c(x_i, y_i) = |x_i \cap y_i|$ is the number of common word classes between x_i and y_i.

This is a simple kernel, whose computation takes $O(n)$ time. If the two paths have different lengths, they correspond to different ways of expressing a relationship – for instance, they may pass through a different number of predicate argument structures. Consequently, the kernel is defined to be 0 in this case. Otherwise, it is the product of the number of common word classes at each position in the two paths. As an example, let us consider two instances of the LOCATED relationship:

1. '**his** actions in **Brcko**', and
2. '**his** arrival in **Beijing**'.

Their corresponding dependency paths are:

1. '**his** $\rightarrow$ actions $\leftarrow$ in $\leftarrow$ **Brcko**', and
2. '**his** $\rightarrow$ arrival $\leftarrow$ in $\leftarrow$ **Beijing**'.

Their representation as a sequence of sets of word classes is given by:

1. $\boldsymbol{x}$ = [x_1 x_2 x_3 x_4 x_5 x_6 x_7], where x_1 = {his, PRP, PERSON}, x_2 = {$\rightarrow$}, x_3 = {actions, NNS, Noun}, x_4 = {$\leftarrow$}, x_5 = {in, IN}, x_6 = {$\leftarrow$}, x_7 = {Brcko, NNP, Noun, LOCATION}
2. $\boldsymbol{y}$ = [y_1 y_2 y_3 y_4 y_5 y_6 y_7], where y_1 = {his, PRP, PERSON}, y_2 = {$\rightarrow$}, y_3 = {arrival, NN, Noun}, y_4 = {$\leftarrow$}, y_5 = {in, IN}, y_6 = {$\leftarrow$}, y_7 = {Beijing, NNP, Noun, LOCATION}

Based on the formula from Equation 1, the kernel is computed as $K(\boldsymbol{x}, \boldsymbol{y}) = 3{\times}1{\times}1{\times}1{\times}2{\times}1{\times}3 = 18$.

We use this relation kernel in conjunction with SVMs in order to find decision hyperplanes that best separate positive examples from negative examples. We modified the LibSVM[1] package for SVM learning by plugging in the kernel described above, and used its default one-against-one implementation for multiclass classification.

5 Experimental Evaluation

We applied the shortest path dependency kernel to the problem of extracting top-level relations from the ACE corpus (NIST, 2000), the version used for the September 2002 evaluation. The training part of this dataset consists of 422 documents, with a separate set of 97 documents allocated for testing. This version of the ACE corpus contains three types of annotations: coreference, named entities and relations. Entities can be of the type PERSON, ORGANIZATION, FACILITY, LOCATION, and GEO-POLITICAL ENTITY. There are 5 general, top-level relations: ROLE, PART, LOCATED, NEAR, and SOCIAL. The ROLE relation links people to an organization to which they belong, own, founded, or provide some service. The PART relation indicates subset relationships, such as a state to a nation, or a subsidiary to its parent company. The AT relation indicates the location of a person or organization at some location. The NEAR relation indicates the proximity of one location to another. The SOCIAL relation links two people in personal, familial or professional relationships. Each top-level relation type is further subdivided into more fine-grained subtypes, resulting in a total of 24 relation types. For example, the LOCATED relation includes subtypes such as LOCATED-AT, BASED-IN, and RESIDENCE. In total, there are 7,646 intra-sentential relations, of which 6,156 are in the training data and 1,490 in the test data.

We assume that the entities and their labels are known. All preprocessing steps – sentence segmentation, tokenization, and POS tagging – were performed using the OpenNLP[2] package.

5.1 Extracting dependencies using a CCG parser

CCG (Steedman, 2000) is a type-driven theory of grammar where most language-specific aspects of the grammar are specified into lexicon. To each lex-

[1] URL:http://www.csie.ntu.edu.tw/~cjlin/libsvm/

[2] URL: http://opennlp.sourceforge.net

ical item corresponds a set of syntactic categories specifying its valency and the directionality of its arguments. For example, the words from the sentence "protesters seized several stations" are mapped in the lexicon to the following categories:

$$\begin{aligned} \text{protesters} &: NP \\ \text{seized} &: (S\backslash NP)/NP \\ \text{several} &: NP/NP \\ \text{stations} &: NP \end{aligned}$$

The transitive verb 'seized' expects two arguments: a noun phrase to the right (the object) and another noun phrase to the left (the subject). Similarly, the adjective 'several' expects a noun phrase to its right. Depending on whether its valency is greater than zero or not, a syntactic category is called a *functor* or an *argument*. In the example above, 'seized' and 'several' are functors, while 'protesters' and 'stations' are arguments.

Syntactic categories are combined using a small set of typed combinatory rules such as functional application, composition and type raising. In Table 3 we show a sample derivation based on three functional applications.

<table>
<tr><td>protesters</td><td>seized</td><td>several</td><td>stations</td></tr>
<tr><td>NP</td><td>$(S\backslash NP)/NP$</td><td>NP/NP</td><td>NP</td></tr>
<tr><td>NP</td><td>$(S\backslash NP)/NP$</td><td colspan="2">NP</td></tr>
<tr><td>NP</td><td colspan="3">$S\backslash NP$</td></tr>
<tr><td colspan="4">S</td></tr>
</table>

Table 3: Sample derivation.

In order to obtain CCG derivations for all sentences in the ACE corpus, we used the CCG parser introduced in (Hockenmaier and Steedman, 2002)[3]. This parser also outputs a list of dependencies, with each dependency represented as a 4-tuple $\langle f, a, w_f, w_a \rangle$, where f is the syntactic category of the functor, a is the argument number, w_f is the head word of the functor, and w_a is the head word of the argument. For example, the three functional applications from Table 3 result in the functor-argument dependencies enumerated below in Table 4.

[3]URL:http://www.ircs.upenn.edu/~juliahr/Parser/

f	a	w_f	w_a
NP/NP	1	'several'	'stations'
$(S\backslash NP)/NP$	2	'seized'	'stations'
$(S\backslash NP)/NP$	1	'seized'	'protesters'

Table 4: Sample dependencies.

Because predicates (e.g. 'seized') and adjuncts (e.g. 'several') are always represented as functors, while complements (e.g. 'protesters' and 'stations') are always represented as arguments, it is straightforward to transform a functor-argument dependency into a head-modifier dependency. The head-modifier dependencies corresponding to the three functor-argument dependencies in Table 4 are: 'protesters → seized', 'stations → seized', and 'several → stations'.

Special syntactic categories are assigned in CCG to lexical items that project unbounded dependencies, such as the relative pronouns 'who', 'which' and 'that'. Coupled with a head-passing mechanism, these categories allow the extraction of long-range dependencies. Together with the local word-word dependencies, they create a directed acyclic dependency graph for each parsed sentence, as shown in Figure 1.

5.2 Extracting dependencies using a CFG parser

Local dependencies can be extracted from a CFG parse tree using simple heuristic rules for finding the head child for each type of constituent. Alternatively, head-modifier dependencies can be directly output by a parser whose model is based on lexical dependencies. In our experiments, we used the full parse output from Collins' parser (Collins, 1997), in which every non-terminal node is already annotated with head information. Because local dependencies assemble into a tree for each sentence, there is only one (shortest) path between any two entities in a dependency tree.

5.3 Experimental Results

A recent approach to extracting relations is described in (Culotta and Sorensen, 2004). The authors use a generalized version of the tree kernel from (Zelenko et al., 2003) to compute a kernel over

relation examples, where a relation example consists of the smallest dependency tree containing the two entities of the relation. Precision and recall values are reported for the task of extracting the 5 top-level relations in the ACE corpus under two different scenarios:

– **[S1]** This is the classic setting: one multi-class SVM is learned to discriminate among the 5 top-level classes, plus one more class for the no-relation cases.

– **[S2]** Because of the highly skewed data distribution, the recall of the SVM approach in the first scenario is very low. In (Culotta and Sorensen, 2004) the authors propose doing relation extraction in two steps: first, one binary SVM is trained for *relation detection*, which means that all positive relation instances are combined into one class. Then the thresholded output of this binary classifier is used as training data for a second multi-class SVM, which is trained for *relation classification*. The same kernel is used in both stages.

We present in Table 5 the performance of our shortest path (SP) dependency kernel on the task of relation extraction from ACE, where the dependencies are extracted using either a CCG parser (SP-CCG), or a CFG parser (SP-CFG). We also show the results presented in (Culotta and Sorensen, 2004) for their best performing kernel K4 (a sum between a bag-of-words kernel and their dependency kernel) under both scenarios.

Method	Precision	Recall	F-measure
(S1) SP-CCG	67.5	37.2	48.0
(S1) SP-CFG	**71.1**	**39.2**	**50.5**
(S1) K4	70.3	26.3	38.0
(S2) SP-CCG	63.7	41.4	50.2
(S2) SP-CFG	**65.5**	**43.8**	**52.5**
(S2) K4	67.1	35.0	45.8

Table 5: Extraction Performance on ACE.

The shortest-path dependency kernels outperform the dependency kernel from (Culotta and Sorensen, 2004) in both scenarios, with a more significant difference for SP-CFG. An error analysis revealed that Collins' parser was better at capturing local dependencies, hence the increased accuracy of SP-CFG. Another advantage of our shortest-path dependency kernels is that their training and testing are very fast – this is due to representing the sentence as a chain of dependencies on which a fast kernel can be computed. All the four SP kernels from Table 5 take between 2 and 3 hours to train and test on a 2.6GHz Pentium IV machine.

To avoid numerical problems, we constrained the dependency paths to pass through at most 10 words (as observed in the training data) by setting the kernel to 0 for longer paths. We also tried the alternative solution of normalizing the kernel, however this led to a slight decrease in accuracy. Having longer paths give larger kernel scores in the unnormalized version does not pose a problem because, by definition, paths of different lengths correspond to disjoint sets of features. Consequently, the SVM algorithm will induce lower weights for features occurring in longer paths, resulting in a linear separator that works irrespective of the size of the dependency paths.

6 Related Work

In (Zelenko et al., 2003), the authors do relation extraction using a tree kernel defined over shallow parse tree representations of sentences. The same tree kernel is slightly generalized in (Culotta and Sorensen, 2004) and used in conjunction with dependency trees. In both approaches, a relation instance is defined to be the smallest subtree in the parse or dependency tree that includes both entities. In this paper we argued that the information relevant to relation extraction is almost entirely concentrated in the shortest path in the dependency tree, leading to an even smaller representation. Another difference between the tree kernels above and our new kernel is that the tree kernels used for relation extraction are *opaque* i.e. the semantics of the dimensions in the corresponding Hilbert space is not obvious. For the shortest-path kernels, the semantics is known by definition: each path feature corresponds to a dimension in the Hilbert space. This transparency allows us to easily restrict the types of patterns counted by the kernel to types that we deem relevant for relation extraction. The tree kernels are also more time consuming, especially in the sparse setting, where they count sparse subsequences of children common to nodes in the two trees. In (Zelenko et al., 2003), the

tree kernel is computed in $O(mn)$ time, where m and n are the number of nodes in the two trees. This changes to $O(mn^3)$ in the sparse setting.

Our shortest-path intuition bears some similarity with the underlying assumption of the relational pathfinding algorithm from (Richards and Mooney, 1992) : "in most relational domains, important concepts will be represented by a small number of fixed paths among the constants defining a positive instance – for example, the grandparent relation is defined by a single fixed path consisting of two parent relations." We can see this happening also in the task of relation extraction from ACE, where "important concepts" are the 5 types of relations, and the "constants" defining a positive instance are the 5 types of entities.

7 Future Work

Local and non-local (deep) dependencies are equally important for finding relations. In this paper we tried extracting both types of dependencies using a CCG parser, however another approach is to recover deep dependencies from syntactic parses, as in (Campbell, 2004; Levy and Manning, 2004). This may have the advantage of preserving the quality of local dependencies while completing the representation with non-local dependencies.

Currently, the method assumes that the named entities are known. A natural extension is to automatically extract both the entities and their relationships. Recent research (Roth and Yih, 2004) indicates that integrating entity recognition with relation extraction in a global model that captures the mutual influences between the two tasks can lead to significant improvements in accuracy.

8 Conclusion

We have presented a new kernel for relation extraction based on the shortest-path between the two relation entities in the dependency graph. Comparative experiments on extracting top-level relations from the ACE corpus show significant improvements over a recent dependency tree kernel.

9 Acknowledgements

This work was supported by grants IIS-0117308 and IIS-0325116 from the NSF.

References

Richard Campbell. 2004. Using linguistic principles to recover empty categories. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*, pages 645–652, Barcelona, Spain, July.

Michael J. Collins. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL-97)*, pages 16–23.

Nello Cristianini and John Shawe-Taylor. 2000. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press.

Aron Culotta and Jeffrey Sorensen. 2004. Dependency tree kernels for relation extraction. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*, Barcelona, Spain, July.

Ralph Grishman. 1995. Message Understanding Conference 6. http://cs.nyu.edu/cs/faculty/grishman/muc6.html.

Julia Hockenmaier and Mark Steedman. 2002. Generative models for statistical parsing with combinatory categorial grammar. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL-2002)*, pages 335–342, Philadelphia, PA.

Roger Levy and Christopher Manning. 2004. Deep dependencies from context-free statistical parsers: Correcting the surface dependency approximation. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*, pages 327–334, Barcelona, Spain, July.

NIST. 2000. ACE – Automatic Content Extraction. http://www.nist.gov/speech/tests/ace.

Soumya Ray and Mark Craven. 2001. Representing sentence structure in hidden Markov models for information extraction. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001)*, pages 1273–1279, Seattle, WA.

Bradley L. Richards and Raymond J. Mooney. 1992. Learning relations by pathfinding. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 50–55, San Jose, CA, July.

D. Roth and W. Yih. 2004. A linear programming formulation for global inference in natural language tasks. In *Proceedings of the Annual Conference on Computational Natural Language Learning (CoNLL)*, pages 1–8, Boston, MA.

Mark Steedman. 2000. *The Syntactic Process*. The MIT Press, Cambridge, MA.

Vladimir N. Vapnik. 1998. *Statistical Learning Theory*. John Wiley & Sons.

D. Zelenko, C. Aone, and A. Richardella. 2003. Kernel methods for relation extraction. *Journal of Machine Learning Research*, 3:1083–1106.