Descending-Path Convolution Kernel for Syntactic Structures

Chen Lin[1], **Timothy Miller**[1], **Alvin Kho**[1], **Steven Bethard**[2],
Dmitriy Dligach[1], **Sameer Pradhan**[1] and **Guergana Savova**[1],
[1] Children's Hospital Boston Informatics Program and Harvard Medical School
{firstname.lastname}@childrens.harvard.edu
[2] Department of Computer and Information Sciences, University of Alabama at Birmingham
bethard@cis.uab.edu

Abstract

Convolution tree kernels are an efficient and effective method for comparing syntactic structures in NLP methods. However, current kernel methods such as subset tree kernel and partial tree kernel understate the similarity of very similar tree structures. Although soft-matching approaches can improve the similarity scores, they are corpus-dependent and match relaxations may be task-specific. We propose an alternative approach called descending path kernel which gives intuitive similarity scores on comparable structures. This method is evaluated on two temporal relation extraction tasks and demonstrates its advantage over rich syntactic representations.

1 Introduction

Syntactic structure can provide useful features for many natural language processing (NLP) tasks such as semantic role labeling, coreference resolution, temporal relation discovery, and others. However, the choice of features to be extracted from a tree for a given task is not always clear. Convolution kernels over syntactic trees (tree kernels) offer a potential solution to this problem by providing relatively efficient algorithms for computing similarities between entire discrete structures. These kernels use tree fragments as features and count the number of common fragments as a measure of similarity between any two trees.

However, conventional tree kernels are sensitive to pattern variations. For example, two trees in Figure 1(a) sharing the same structure except for one terminal symbol are deemed at most 67% similar by the conventional tree kernel (PTK) (Moschitti, 2006). Yet one might expect a higher similarity given their structural correspondence.

The similarity is further attenuated by trivial structure changes such as the insertion of an adjective in one of the trees in Figure 1(a), which would reduce the similarity close to zero. Such an abrupt attenuation would potentially propel a model to memorize training instances rather than generalize from trends, leading towards overfitting.

In this paper, we describe a new kernel over syntactic trees that operates on descending paths through the tree rather than production rules as used in most existing methods. This representation is reminiscent of Sampson's (2000) leaf-ancestor paths for scoring parse similarities, but here it is generalized over all ancestor paths, not just those from the root to a leaf. This approach assigns more robust similarity scores (e.g., 78% similarity in the above example) than other soft matching tree kernels, is faster than the partial tree kernel (Moschitti, 2006), and is less *ad hoc* than the grammar-based convolution kernel (Zhang et al., 2007).

2 Background

2.1 Syntax-based Tree Kernels

Syntax-based tree kernels quantify the similarity between two constituent parses by counting their common sub-structures. They differ in their definition of the sub-structures.

Collins and Duffy (2001) use a subset tree (SST) representation for their sub-structures. In the SST representation, a subtree is defined as a subgraph with more than one node, in which only full production rules are expanded. While this approach is widely used and has been successful in many tasks, the production rule-matching constraint may be unnecessarily restrictive, giving zero credit to rules that have only minor structural differences. For example, the similarity score between the NPs in Figure 1(b) would be zero since the production rule is different (the overall similarity score is above-zero because of matching pre-terminals).

The partial tree kernel (PTK) relaxes the definition of subtrees to allow partial production rule

Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Short Papers), pages 81–86,
Baltimore, Maryland, USA, June 23-25 2014.

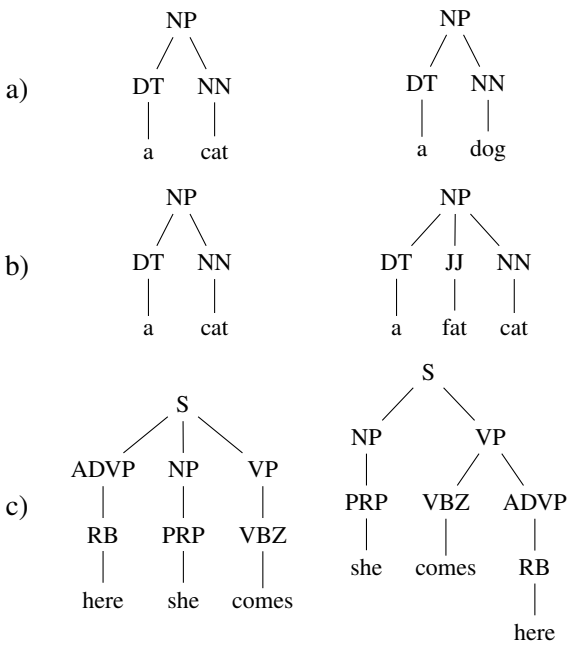

Figure 1: Three example tree pairs.

matching (Moschitti, 2006). In the PTK, a subtree may or may not expand any child in a production rule, while maintaining the ordering of the child nodes. Thus it generates a very large but sparse feature space. To Figure 1(b), the PTK generates fragments (i) [NP [DT a] [JJ fat]]; (ii) [NP [DT a] [NN cat]]; and (iii) [NP [JJ fat] [NN cat]], among others, for the second tree. This allows for partial matching – substructure (ii) – while also generating some fragments that violate grammatical intuitions.

Zhang et al. (2007) address the restrictiveness of SST by allowing soft matching of production rules. They allow partial matching of optional nodes based on the Treebank. For example, the rule $NP \rightarrow DT\ JJ\ NN$ indicates a noun phrase consisting of a determiner, adjective, and common noun. Zhang et al.'s method designates the JJ as optional, since the Treebank contains instances of a reduced version of the rule without the JJ node ($NP \rightarrow DT\ NN$). They also allow node matching among similar preterminals such as JJ, JJR, and JJS, mapping them to one equivalence class.

Other relevant approaches are the spectrum tree (SpT) (Kuboyama et al., 2007) and the route kernel (RtT) (Aiolli et al., 2009). SpT uses a q-gram – a sequence of connected vertices of length q – as their sub-structure. It observes grammar rules by recording the orientation of edges: a←b→c is different from a→b→c. RtT uses a set of routes as basic structures, which observes grammar rules by recording the index of a neighbor node.

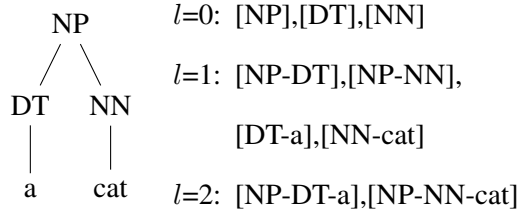

Figure 2: A parse tree (left) and its descending paths according to Definition 1 (l - length).

2.2 Temporal Relation Extraction

Among NLP tasks that use syntactic information, temporal relation extraction has been drawing growing attention because of its wide applications in multiple domains. As subtasks in TempEval 2007, 2010 and 2013, multiple systems were built to create labeled links from events to events/timestamps by using a variety of features (Bethard and Martin, 2007; Llorens et al., 2010; Chambers, 2013). Many methods exist for synthesizing syntactic information for temporal relation extraction, and most use traditional tree kernels with various feature representations. Mirroshandel et al. (2009) used the path-enclosed tree (PET) representation to represent syntactic information for temporal relation extraction on the TimeBank (Pustejovsky et al., 2003) and the AQUAINT TimeML corpus[1]. The PET is the smallest subtree that contains both proposed arguments of a relation. Hovy et al. (2012) used bag tree structures to represent the bag of words (BOW) and bag of part of speech tags (BOP) between the event and time in addition to a set of baseline features, and improved the temporal linking performance on the TempEval 2007 and Machine Reading corpora (Strassel et al., 2010). Miller at al. (2013) used PET tree, bag tree, and path tree (PT, which is similar to a PET tree with the internal nodes removed) to represent syntactic information and improved the temporal relation discovery performance on THYME data[2] (Styler et al., 2014). In this paper, we also use syntactic structure-enriched temporal relation discovery as a vehicle to test our proposed kernel.

3 Methods

Here we decribe the Descending Path Kernel (DPK).

[1] http://www.timeml.org
[2] http://thyme.healthnlp.org

Definition 1 (Descending Path): Let T be a parse tree, v any non-terminal node in T, dv a descendant of v, including terminals. A descending path is the sequence of indexes of edges connecting v and dv, denoted by $[v - \cdots - dv]$. The length l of a descending path is the number of connecting edges. When $l = 0$, a descending path is the non-terminal node itself, $[v]$. Figure 2 illustrates a parse tree and its descending paths of different lengths.

Suppose that all descending paths of a tree T are indexed $1, \cdots, n$, and $path_i(T)$ is the frequency of the i-th descending path in T. We represent T as a vector of frequencies of all its descending paths: $\Phi(T) = (path_1(T), \cdots, path_n(T))$.

The similarity between any two trees T_1 and T_2 can be assessed via the dot product of their respective descending path frequency vector representations: $K(T_1, T_2) = \langle \Phi(T_1), \Phi(T_2) \rangle$.

Compared with the previous tree kernels, our descending path kernel has the following advantages: 1) the sub-structures are simplified so that they are more likely to be shared among trees, and therefore the sparse feature issues of previous kernels could be alleviated by this representation; 2) soft matching between two similar structures (e.g., NP→DT JJ NN versus NP→DT NN) have high similarity without reference to any corpus or grammar rules;

Following Collins and Duffy (2001), we derive a recursive algorithm to compute the dot product of the descending path frequency vector representations of two trees T_1 and T_2:

$$\begin{aligned} K(T_1, T_2) &= \langle \Phi(T_1), \Phi(T_2) \rangle \\ &= \sum_i path_i(T_1) \cdot path_i(T_2) \\ &= \sum_{n_1 \in N_1} \sum_{n_2 \in N_2} \sum_i I_{path_i}(n_1) \cdot I_{path_i}(n_2) \\ &= \sum_{\substack{n_1 \in N_1 \\ n_2 \in N_2}} C(n_1, n_2) \end{aligned} \tag{1}$$

where N_1 and N_2 are the sets of nodes in T_1 and T_2 respectively, i indexes the set of possible paths, $I_{path_i}(n)$ is an indicator function that is 1 iff the descending $path_i$ is rooted at node n or 0 otherwise. $C(n_1, n_2)$ counts the number of common descending paths rooted at nodes n_1 and n_2:

$$C(n_1, n_2) = \sum_i I_{path_i}(n_1) \cdot I_{path_i}(n_2)$$

$C(n_1, n_2)$ can be computed in polynomial time by the following recursive rules:

Rule 1: If n_1 and n_2 have different labels (e.g., "DT" versus "NN"), then $C(n1, n2) = 0$;

Rule 2: Else if n_1 and n_2 have the same labels and are both pre-terminals (POS tags), then

$$C(n_1, n_2) = 1 + \begin{cases} 1 & \text{if } term(n_1) = term(n_2) \\ 0 & \text{otherwise.} \end{cases}$$

where $term(n)$ is the terminal symbol under n;

Rule 3: Else if n_1 and n_2 have the same labels and they are not both pre-terminals, then:

$$C(n_1, n_2) = 1 + \sum_{\substack{n_i \in children(n_1) \\ n_j \in children(n_2)}} C(n_i, n_j)$$

where $children(m)$ are the child nodes of m.

As in other tree kernel approaches (Collins and Duffy, 2001; Moschitti, 2006), we use a discount parameter λ to control for the disproportionately large similarity values of large tree structures. Therefore, Rule 2 becomes:

$$C(n_1, n_2) = 1 + \begin{cases} \lambda & \text{if } term(n_1) = term(n_2) \\ 0 & \text{otherwise.} \end{cases}$$

and Rule 3 becomes:

$$C(n_1, n_2) = 1 + \lambda \sum_{\substack{n_i \in children(n_1) \\ n_j \in children(n_2)}} C(n_i, n_j)$$

Note that Eq. (1) is a convolution kernel under the kernel closure properties described in Haussler (1999). Rules 1-3 show the equivalence between the number of common descending paths rooted at nodes n_1 and n_2, and the number of matching nodes below n_1 and n_2.

In practice, there are many non-matching nodes, and most matching nodes will have only a few matching children, so the running time, as in SST, will be approximated by the number of matching nodes between trees.

3.1 Relationship with other kernels

For a given tree, DPK will generate significantly fewer sub-structures than PTK, since it does not consider all ordered permutations of a production rule. Moreover, the fragments generated by DPK are more likely to be shared among different trees. For the number of corpus-wide fragments, it is

Kernel	ID	#Frag	Sim	N(Sim)
SST $O(\rho\|N_1\|\|N_2\|)$	a	9	3	0.50
	b	15	2	0.25
	c	63	7	0.20
DPK $O(\rho^2\|N_1\|\|N_2\|)$	a	11	7	**0.78**
	b	13	9	**0.83**
	c	31	22	**0.83**
PTK $O(\rho^3\|N_1\|\|N_2\|)$	a	20	10	0.67
	b	36	15	0.65
	c	127	34	0.42

Table 1: Comparison of the worst case computational complexicity (ρ - the maximum branching factor) and kernel performance on the 3 examples from Figure 1. $\#Frag$ is the number of fragments, $N(Sim)$ is the normalized similarity. Please see the online supplementary note for detailed fragments of example (a).

possible that DPK $\leq$ SST $\leq$ PTK. In Table 1, given $\lambda = 1$, we compare the performance of 3 kernels on the three examples in Figure 1. Note that for more complicated structures, i.e., examples b and c, DPK generates fewer fragments than SST and PTK, with more shared fragments among trees.

The complexity for all three kernels are at least $O(|N_1||N_2|)$ since they share the pairwise summation at the end of Equation 1. SST, due to its requirement of exact production rule matching, only takes one pass in the inner loop which adds a factor of ρ (the maximum branching factor of any production rule). DPK does a pairwise summation of children, which adds a factor of ρ^2 to the complexity. Finally, the efficient algorithm for PTK is proved by Moschitti (2006) to contain a constant factor of ρ^3. Table 1 orders the tree kernels according by their listed complexity.

It may seem that the value of DPK is strictly in its ability to evaluate all paths, which is not explicitly accounted for by other kernels. However, another view of the DPK is possible by thinking of it as cheaply calculating rule production similarity by taking advantage of relatively strict English word ordering. Like SST and PTK, the DPK requires the root category of two subtrees to be the same for the similarity to be greater than zero. Unlike SST and PTK, once the root category comparison is successfully completed, DPK looks at all paths that go through it and accumulates their similarity scores independent of ordering – in other words, it will ignore the ordering of the children in its production rule. This means, for example, that if the rule production NP $\rightarrow$ NN JJ DT were ever found in a tree, to DPK it would be indistinguishable from the common production NP $\rightarrow$ DT JJ NN, despite having inverted word order, and thus would have a maximal similarity score. SST and PTK would assign this pair a much lower score for having completely different ordering, but we suggest that cases such as these are very rare due to the relatively strict word ordering of English. In most cases, the determiner of a noun phrase will be at the front, the nouns will be at the end, and the adjectives in the middle. So with small differences in production rules (one or two adjectives, extra nominal modifier, etc.) the PTK will capture similarity by comparing every possible partial rule completion, but the DPK can obtain higher and faster scores by just comparing one child at a time because the ordering is constrained by the language. This analysis does lead to a hypothesis for the general viability of the DPK, suggesting that in languages with freer word order it may give inflated scores to structures that are syntactically dissimilar if they have the same constituent components in different order.

Formally, Moschitti (2006) showed that SST is a special case of PTK when only the longest child sequence from each tree is considered. On the other end of the spectrum, DPK is a special case of PTK where the similarity between rules only considers child subsequences of length one.

4 Evaluation

We applied DPK to two published temporal relation extraction systems: (Miller et al., 2013) in the clinical domain and Cleartk-TimeML (Bethard, 2013) in the general domain respectively.

4.1 Narrative Container Discovery

The task here as described by Miller et al. (2013) is to identify the CONTAINS relation between a time expression and a same-sentence event from clinical notes in the THYME corpus, which has 78 notes of 26 patients. We obtained this corpus from the authors and followed their linear composite kernel setting:

$$K_C(s_1, s_2) = \tau \sum_{p=1}^{P} K_T(t_1^p, t_2^p) + K_F(f_1, f_2) \quad (2)$$

where s_i is an instance object composed of flat features f_i and a syntactic tree t_i. A syntactic tree t_i

can have multiple representations, as in Bag Tree (BT), Path-enclosed Tree (PET), and Path Tree (PT). For the tree kernel K_T, subset tree (SST) kernel was applied on each tree representation p. The final similarity score between two instances is the τ-weighted sum of the similarities of all representations, combined with the flat feature (FF) similarity as measured by a feature kernel K_F (linear or polynomial). Here we replaced the SST kernel with DPK and tested two feature combinations FF+PET and FF+BT+PET+PT. To fine tune parameters, we used grid search by testing on the default development data. Once the parameters were tuned, we tested the system performance on the testing data, which was set up by the original system split.

4.2 Cleartk-TimeML

We tested one sub-task from TempEval-2013 – the extraction of temporal relations between an event and time expression within the same sentence. We obtained the training corpus (TimeBank + AQUAINT) and testing data from the authors (Bethard, 2013). Since the original features didn't contain syntactic features, we created a PET tree extractor for this system. The kernel setting was similar to equation (2), while there was only one tree representation, PET tree, P=1. A linear kernel was used as K_F to evaluate the exact same flat features as used by the original system. We used the built-in cross validation to do grid search for tuning the parameters. The final system was tested on the testing data for reporting results.

4.3 Results and Discussion

Results are shown in Table 2. The top section shows THYME results. For these experiments, the DPK is superior when a syntactically-rich PET representation is used. Using the full feature set of Miller et al. (2013), SST is superior to DPK and obtains the best overall performance. The bottom section shows results on TempEval-2013 data, for which there is little benefit from either tree kernel.

Our experiments with THYME data show that DPK can capture something in the linguistically richer PET representation that the SST kernel cannot, but adding BT and PT representations decrease the DPK performance. As a shallow representation, BT does not have much in the way of descending paths for DPK to use. PT already ignores the production grammar by removing the inner tree nodes. DPK therefore cannot get useful information and may even get misleading cues from these two representations. These results show that, while DPK should not always replace SST, there are representations in which it is superior to existing methods. This suggests an approach in which tree representations are matched to different convolution kernels, for example by tuning on held-out data.

Features	K_T	P	R	F
	THYME			
FF+PET	DPK	0.756	0.667	**0.708**
	SST	0.698	0.630	0.662
FF+BT+	DPK	0.759	0.626	0.686
PET+PT	SST	0.754	0.711	**0.732**
	TempEval			
FF+PET	DPK	0.328	0.263	**0.292**
	SST	0.325	0.263	0.290
FF	-	0.309	0.266	0.286

Table 2: Comparison of tree kernel performance for temporal relation extraction on THYME and TempEval-2013 data.

For TempEval-2013 data, adding syntactic features did not improve the performance significantly (comparing F-score of 0.290 with 0.286 in Table 3). Probably, syntactic information is not a strong feature for all types of temporal relations on TempEval-2013 data.

5 Conclusion

In this paper, we developed a novel convolution tree kernel (DPK) for measuring syntactic similarity. This kernel uses a descending path representation in trees to allow higher similarity scores on partially matching structures, while being simpler and faster than other methods for doing the same. Future work will explore 1) a composite kernel which uses DPK for PET trees, SST for BT and PT, and feature kernel for flat features, so that different tree kernels can work with their ideal syntactic representations; 2) incorporate dependency structures for tree kernel analysis 3) applying DPK to other relation extraction tasks on various corpora.

6 Acknowledgements

Thanks to Sean Finan for technically supporting the experiments. The project described was supported by R01LM010090 (THYME) from the National Library Of Medicine.

References

Fabio Aiolli, Giovanni Da San Martino, and Alessandro Sperduti. 2009. Route kernels for trees. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 17–24. ACM.

Steven Bethard and James H Martin. 2007. Cu-tmp: temporal relation classification using syntactic and semantic features. In *Proceedings of the 4th International Workshop on Semantic Evaluations*, pages 129–132. Association for Computational Linguistics.

Steven Bethard. 2013. Cleartk-timeml: A minimalist approach to TempEval 2013. In *Second Joint Conference on Lexical and Computational Semantics (* SEM)*, volume 2, pages 10–14.

Nate Chambers. 2013. Navytime: Event and time ordering from raw text. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013)*, pages 73–77, Atlanta, Georgia, USA, June. Association for Computational Linguistics.

Michael Collins and Nigel Duffy. 2001. Convolution kernels for natural language. In *Neural Information Processing Systems*.

David Haussler. 1999. Convolution kernels on discrete structures. Technical report, University of California in Santa Cruz.

Dirk Hovy, James Fan, Alfio Gliozzo, Siddharth Patwardhan, and Chris Welty. 2012. When did that happen?: linking events and relations to timestamps. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 185–193. Association for Computational Linguistics.

Tetsuji Kuboyama, Kouichi Hirata, Hisashi Kashima, Kiyoko F Aoki-Kinoshita, and Hiroshi Yasuda. 2007. A spectrum tree kernel. *Information and Media Technologies*, 2(1):292–299.

Hector Llorens, Estela Saquete, and Borja Navarro. 2010. Tipsem (english and spanish): Evaluating CRFs and semantic roles in TempEval-2. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, pages 284–291. Association for Computational Linguistics.

Timothy Miller, Steven Bethard, Dmitriy Dligach, Sameer Pradhan, Chen Lin, and Guergana Savova. 2013. Discovering temporal narrative containers in clinical text. In *Proceedings of the 2013 Workshop on Biomedical Natural Language Processing*, pages 18–26, Sofia, Bulgaria, August. Association for Computational Linguistics.

Seyed Abolghasem Mirroshandel, M Khayyamian, and GR Ghassem-Sani. 2009. Using tree kernels for classifying temporal relations between events. *Proc. of the PACLIC23*, pages 355–364.

Alessandro Moschitti. 2006. Efficient convolution kernels for dependency and constituent syntactic trees. In *Machine Learning: ECML 2006*, pages 318–329. Springer.

James Pustejovsky, Patrick Hanks, Roser Sauri, Andrew See, Robert Gaizauskas, Andrea Setzer, Dragomir Radev, Beth Sundheim, David Day, Lisa Ferro, et al. 2003. The TimeBank corpus. In *Corpus linguistics*, volume 2003, page 40.

Geoffrey Sampson. 2000. A proposal for improving the measurement of parse accuracy. *International Journal of Corpus Linguistics*, 5(1):53–68.

Stephanie Strassel, Dan Adams, Henry Goldberg, Jonathan Herr, Ron Keesing, Daniel Oblinger, Heather Simpson, Robert Schrag, and Jonathan Wright. 2010. The DARPA machine reading program-encouraging linguistic and reasoning research with a series of reading tasks. In *LREC*.

William Styler, Steven Bethard, Sean Finan, Martha Palmer, Sameer Pradhan, Piet de Groen, Brad Erickson, Timothy Miller, Lin Chen, Guergana K. Savova, and James Pustejovsky. 2014. Temporal annotations in the clinical domain. *Transactions of the Association for Computational Linguistics*, 2(2):143−154.

Min Zhang, Wanxiang Che, Ai Ti Aw, Chew Lim Tan, Guodong Zhou, Ting Liu, and Sheng Li. 2007. A grammar-driven convolution tree kernel for semantic role classification. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 200–207.