Fraunhofer Institut
Rechnerarchitektur
und Softwaretechnik

Konrad Rieck, Ulf Brefeld, Tammo Krueger

Approximate Kernels for Trees

FIRST Reports 5/2008

FIRST Reports
Herausgegeben von
Prof. Dr.-Ing. Stefan Jähnichen

ISSN 1613-5024

FIRST Reports 5/2008

Die FIRST Reports können bezogen werden über:

Fraunhofer-Institut für Rechnerarchitektur
und Softwaretechnik FIRST
Kekuléstraße 7
12489 Berlin

Tel.: ++49 (0)30 6392 18 00
Fax: ++49 (0)30 6392 18 05
E-Mail: first@first.fraunhofer.de
Internet: www.first.fraunhofer.de

Approximate Kernels for Trees

Konrad Rieck[1], Ulf Brefeld[2], and Tammo Krueger[1]

[1]Fraunhofer Institute FIRST
Intelligent Data Analysis
Berlin, Germany

[2] Technical University of Berlin
Dept. of Computer Science
Berlin, Germany

September 11, 2008

Abstract

Convolution kernels for trees provide effective means for learning with tree-structured data, such as parse trees of natural language sentences. Unfortunately, the computation time of tree kernels is quadratic in the size of the trees as all pairs of nodes need to be compared: large trees render convolution kernels inapplicable. In this paper, we propose a simple but efficient approximation technique for tree kernels. The approximate tree kernel (ATK) accelerates computation by selecting a sparse and discriminative subset of subtrees using a linear program. The kernel allows for incorporating domain knowledge and controlling the overall computation time through additional constraints. Experiments on applications of natural language processing and web spam detection demonstrate the efficiency of the approximate kernels. We observe run-time improvements of two orders of magnitude while preserving the discriminative expressiveness and classification rates of regular convolution kernels.

1 Introduction

Tree-structured data arises naturally in many application areas such as natural language processing, information retrieval, bioinformatics, and computational chemistry [e.g. 4, 5, 8, 17]. Examples for tree structures include parse trees of natural language, web documents of HTML or XML, and protein structures. These trees carry important hierarchical information which are often indispensable for learning accurate prediction models. Shallow representations of trees such as flat feature vectors generally fail to capture these hierarchical structures.

The prevalent tool for learning with structured data are kernel functions which assess the pairwise similarity of structured objects and provide an interface to kernel-based learning methods [13]. Kernels for structured data can be constructed using the convolution of local kernel functions [7]. A typical example for such a convolution is the parse tree kernel [5] which determines similarity of trees by counting the number of shared subtrees. Parse tree kernels are computed using dynamic programming, where a table of subtree counts for all pairs of tree nodes is maintained. While allocating and updating such a table is feasible for small tree sizes (say less than 200 nodes [e.g. 4, 5, 12, 15]), large trees involve computations that easily exhaust available resources in terms of memory and run-time. For example, computing a single kernel value for two HTML documents comprising 10,000 nodes each, requires about 1.6 gigabytes of memory and takes over 6 minutes on a state-of-the-art computer system. Given that kernel computations are performed millions of times in large-scale training processes, it is evident that known tree kernels are an inappropriate choice in many relevant learning tasks.

This paper contributes to the problem of learning with large tree data by proposing an approximation of tree kernels. Based on a given learning task the kernel computation is narrowed to a sparse set of subtrees rooted at discriminative grammar symbols, such that run-time and memory requirements are significantly improved. The corresponding optimization problem can be phrased as a linear program and solved with standard techniques. Our approach allows the inclusion of domain knowledge by incorporating appropriate constraints. Experiments conducted with tree data from applications of natural language processing and web spam detection demonstrate the discriminative expressiveness and efficiency of the proposed approximate kernels. For instance, when the trees are large web documents, the approximate tree kernels performs on par with regular convolution kernels and lead to speed-ups of two orders of magnitude.

The remainder of this paper is organized as follows. Kernels for parse trees are introduced in Section 2 and their approximation using linear programming is presented in Section 3. We report on our empirical results in Section 4. Section 5 concludes.

2 Kernels for Parse Trees

Before presenting the approximation of tree kernels, we first introduce basic notation and review related work. Let $G = (S, P, s)$ be a grammar with production rules P and a start symbol s defined over a set S of non-terminal and terminal symbols. A tree X is called a parse tree of G if every node $x \in X$ is labeled with a symbol $\ell(x) \in S$ and associated with a production rule. To navigate in a parse tree, we address the i-th child of a node x by x_i and denote the number of children by $|x|$. The number of nodes in X is indicated by $|X|$ and the set of all possible trees is given by $\mathcal{X}$.

A kernel $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is a symmetric and positive semi-definite function, which implicitly computes an inner product in a reproducing kernel Hilbert space [13]. A generic technique for defining kernels over structured data is convolution of local kernels defined over sub-structures [7].

Collins and Duffy [5] apply convolution kernels to parse trees by counting shared subtrees. Given two parse trees X and Z, their parse tree kernel is defined as

$$k(X, Z) = \sum_{x \in X} \sum_{z \in Z} c(x, z) \quad \text{with} \quad c(x, z) = \lambda \prod_{i=1}^{|x|} (1 + c(x_i, z_i)) . \tag{1}$$

where c determines the number of shared subtrees rooted in x and z recursively. The base cases of c correspond to $c(x, z) = 0$ if x and z are not derived from the same production and $c(x, z) = \lambda$ if x and z are leaf nodes of the same production . The parameter λ balances the respective contribution of small and large subtrees.

Several extensions of the parse tree kernel have been proposed. For example, setting the constant term in the product of Equation (1) to zero restricts the counting function to complete subtrees as proposed by Vishwanathan and Smola [12, 16]. Kashima et al. [8] extend the counting function to labeled ordered trees by considering ordered subsets of child nodes and point out relations to tree edit distances [9]. Finally, Suzuki and Isozaki [15] refine the parse tree kernel using statistical feature selection incorporated into the recursion of the counting function.

Computation of the parse tree kernel and all its extensions can be carried out using dynamic programming by maintaining a table of subtree counts for all possible node pairs [14]. The run-time and memory requirements for processing all pairs are prohibitive if the considered trees comprise hundreds or thousands of nodes. As a first step towards run-time improvement Moschitti [11] limits the computation to nodes with matching symbols, yet this extension only affects run-time and does not alleviate memory requirements. Moreover for larger trees with $|X| \gg |S|$ only a little speed-up is gained as only a minor portion of node pairs is discarded.

3 Approximate Tree Kernels

It is apparent that computation of standard tree kernels using dynamic programming is prohibitive for large tree structures. However, large trees often possess redundant substructures that slow-down the computation unnecessarily. For example, when learning to detect web spam, HTML elements for referencing other web documents play a salient role while the majority of formatting tags is irrelevant to the learning task. We exploit this observation by restricting the kernel computation to a sparse set of subtrees rooted at discriminative grammar symbols. The following section introduces the approximate tree kernels formally and Section 3.2 exemplifies the incorporation of prior domain knowledge.

3.1 Linear Programming for Approximate Tree Kernels

Given a set $(X_1, y_1), \dots, (X_n, y_n)$ of parse trees with noise-free labels $y_i \in \{-1, +1\}$, the aim of our approximation is determining a kernel $\hat{k}$, which enables good separation of the classes in y, while the number of considered subtrees in the computation of $\hat{k}$ is minimal. Since enumerating all possible subtrees is unfeasible, we introduce a selection function $\gamma : S \to \{0, 1\}$, which controls whether subtrees rooted at $s \in S$ are to be considered in the convolution or not. By means of γ, approximate tree kernels are defined as follows.

Definition 1. *The approximate tree kernel is defined as*

$$\hat{k}(X, Z) = \sum_{s \in S} \gamma(s) \sum_{\substack{x \in X \\ \ell(x)=s}} \sum_{\substack{z \in Z \\ \ell(z)=s}} c(x, z), \tag{2}$$

where $\gamma(s) \in \{0, 1\}$ for all symbols $s \in S$ and c is the counting function given in Equation (1).

Note that the exact parse tree kernel from Equation (1) is obtained as a special case of Equation (2) if $\gamma(s) = 1$ for all symbols $s \in S$.

We denote by $\hat{K}$ the approximate kernel matrix with elements $\hat{K}_{ij} = \hat{k}(X_i, X_j)$. Intuitively, we seek γ such that $\hat{K}$ discriminates well between the classes in y, a property that is realized by the outer product $yy^\top$. A simple way to adapt $\hat{K}$ to $yy^\top$ is obtained using the Frobenius inner product,

$$\langle yy^\top, \hat{K} \rangle_{\mathcal{F}} = \sum_{y_i = y_j} \hat{K}_{ij} - \sum_{y_i \neq y_j} \hat{K}_{ij}. \tag{3}$$

where $\langle \cdot, \cdot \rangle_{\mathcal{F}}$ denotes the Frobenius product between matrices and is defined as $\langle A, B \rangle_{\mathcal{F}} = \sum_{ij} a_{ij} b_{ij} = tr(AB)$. The right hand side of Equation (3) measures the within class (first term) and the between class (second term) similarity. Discriminative kernels $\hat{k}$ realize large values of $\langle yy^\top, \hat{K}\rangle_\mathcal{F}$, hence maximizing Equation (3) suffices for finding appropriate approximations. For the approximate tree kernel we arrive at the following integer linear program that has to be maximized with respect to γ to align $\hat{k}$ to the labels y,

$$\max_{\gamma\in\{0,1\}^{|S|}} \sum_{i,j=1}^{n} y_i y_j \hat{k}(X_i, X_j). \tag{4}$$

To solve Equation (4) efficiently and take care of the sparsity of γ — a crucial requirement for the efficient computation of $\hat{k}$ — we bound the number of selected symbols in γ to N and obtain the following linear program

$$\begin{aligned} \max_{\gamma\in[0,1]^{|S|}} \quad & \sum_{i,j=1}^{n} y_i y_j \sum_{s\in S} \gamma(s) \sum_{\substack{x\in X_i\\ \ell(x)=s}} \sum_{\substack{z\in X_j\\ \ell(z)=s}} c(x,z) \\ \text{s.t.} \quad & \sum_{s\in S} \gamma(s) = N. \end{aligned} \tag{5}$$

Finally, it remains to show that $\hat{k}$ is a valid kernel (Proposition 1) with restricted run-time and memory requirements (Proposition 2).

Proposition 1. *The approximate tree kernel is a kernel function.*

Proof. Let $\phi(X)$ be the vector of frequencies of all subtrees occurring in X as defined in [5]. By definition, $\hat{k}$ can be written as

$$\hat{k}(X,Z) = \langle P\phi(X), P\phi(Z)\rangle,$$

where P projects the dimensions of $\phi(X)$ on the subtrees rooted in symbols selected by γ^*. The projection P is independent of concrete X and Z, and hence $\hat{k}$ is a valid kernel. □

Proposition 2. *The approximate tree kernel $\hat{k}(X,Z)$ can be computed q times faster than $k(X,Z)$.*

Proof. Let $\tau(s,X)$ denote the occurrences of nodes $x \in X$ with $\ell(x) = s$. Then the speed-up q realized by $\hat{k}$ is lower bounded by

$$q \geq \frac{\sum_{s\in S} \tau(s,X)\tau(s,Z)}{\sum_{s\in S} \gamma(s)\tau(s,X)\tau(s,Z)} \tag{6}$$

as all nodes with identical symbols in X and Z are paired. For the trivial case where all elements $\gamma(s) = 1$, the factor q equals 1 and the run-time is identical to the parse tree kernel. In all other cases $q > 1$ holds as at least one symbol is discarded from the denominator in Equation (6). □

3.2 Extensions of the Approximate Tree Kernel

In practice it might be of interest to refine the approximation process, e.g. by fixing the amount of selected symbols or by arranging similar symbols in groups. We now provide extensions of the approximate kernel for trees.

Bounding the expected run-time. To upper bound the run-time of the kernel computation we introduce a function $\sigma(s)$ which measures the average frequency of the symbol s in the considered corpus of parse trees. Using σ we can bound the expected run-time by a constant b^2 using the following constraint

$$\sum_{s \in S} \gamma(s)\sigma(s) \leq b.$$

In the average-case b^2 node pairs will be considered during the computation of $\hat{k}$, since the expected amount of selected nodes is bounded by b.

Conjunction and disjunction of symbols. Dependencies in the symbols of a grammar can be modeled using conjunctions and disjunctions. In case the activation of symbol s_j requires the activation of s_{j+1}, the constraint $\gamma(s_j) - \gamma(s_{j+1}) = 0$ can be included in Equation (5). A conjunction of m symbols can be efficiently encoded by $m - 1$ additional constraints as

$$\forall_{j=1}^{m-1} \gamma(s_j) - \gamma(s_{j+1}) = 0.$$

For a disjunction of symbols $s_j, \ldots, s_{j+m}$ the following constraint guarantees that at least one representative of the group is active in the solution

$$\gamma(s_j) + \gamma(s_{j+1}) + \ldots + \gamma(s_{j+m}) \geq 1.$$

Alternatively, the above constraint can be modified to an exclusive disjunction of symbols by requiring equality to 1.

Extension to multi-class problems. The approximate tree kernel can be easily adapted to multi-class problems. Given labels y with $y_i \in \mathbb{N}$, the product $y_i y_j$ in Equation (5) is exchanged with the term $[\![y_i = y_j]\!]$, where the indicator $[\![E]\!]$ returns $+1$ if the expression E holds and -1 otherwise.

4 Experiments and Results

After defining approximate kernels for trees and their extensions, we present an empirical evaluation of their expressiveness and run-time performance. We conduct our experiments on tree data of two real-world applications, namely *question classification* and *web spam detection*.

Question Classification. Question classification is a preprocessing procedure in information retrieval. The task is to categorize a user-supplied question into predefined semantic categories. We employ the data collection by Li and Roth [10] consisting of 6,000 English questions assigned to six classes (abbreviation, entity, description, human, location, numeric value). Each question is transformed to a respective parse tree using the MEI Parser[1] [3], which extracts up to 70 grammatical symbols. For simplicity we restrict the setting to learning a discrimination between the category "entity" (1,339 instances) and all other categories.

Web Spam Detection. Web spam aims at obtaining high search ranks through massive amounts of bogus links. Detection of web spam is essential for providing proper search results. For our experiments we use the web spam data as described in [1]. The collection consists of HTML documents from normal and spam websites in the UK, totaling 15 gigabytes of compressed data. All sites are examined by humans and manually annotated. From the top 20 sites of both classes we sample 5,000 documents (974 spam, 4,026 normal). We use a fault-tolerant HTML parser[2] to obtain

[1]Maximum-Entropy-Inspired Parser, see `ftp://ftp.cs.brown.edu/pub/nlparser`

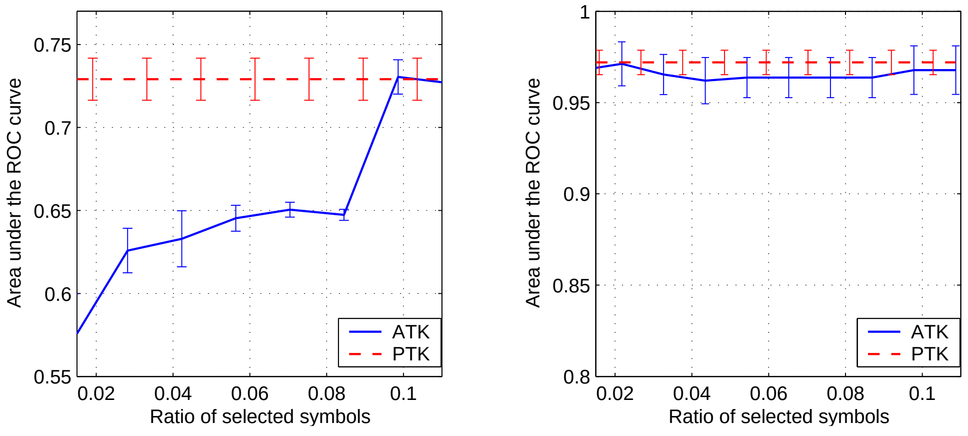
[2]Beautiful Soup Parser, see `http://www.crummy.com/software/BeautifulSoup`

parse trees from the HTML documents comprising up to 91 different symbols. To compute results using the exact parse tree kernel in reasonable time, we reduce the maximum tree size to 1,500 nodes.

For each data set we perform the following experimental procedure: 3,000 parse trees are randomly drawn and split into equally sized training, validation and testing partitions. An SVM classifier [2] is applied to the training data and its performance is determined using the area under the ROC curve (AUC). Model selection is performed on the validation set for the SVM regularization parameter and the tree kernel parameter. Reported results are obtained on the testing data using the best model on the validation set. The procedure is repeated 5 times and the results are averaged.

4.1 Classification Accuracy

For the first experiment we examine the expressiveness of the approximate tree kernel and the exact parse tree kernel on the considered learning tasks. We vary the number of selected symbols for the approximate kernel to observe their impact on the achieved classification accuracy.



(a) Question classification (b) Web spam detection

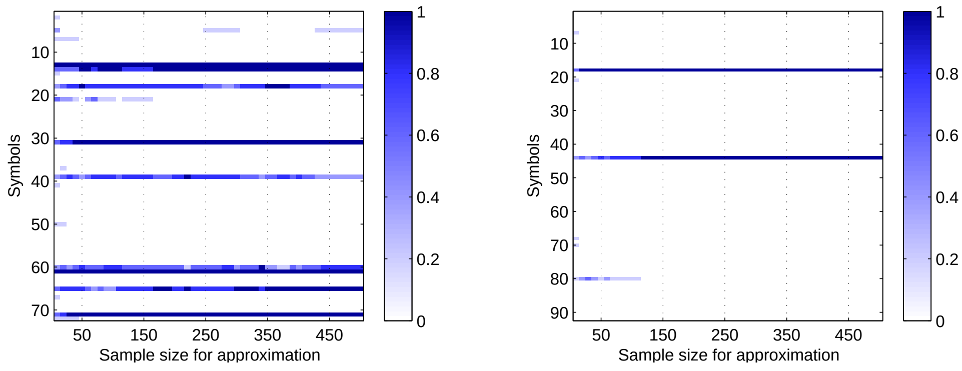
Figure 1: Classification performance of the approximate tree kernel (ATK) with varying number of selected symbols and the exact parse tree kernel (PTK).

Figure 1 presents the classification performance of the two tree kernels. The ratio of selected symbols for the approximate kernel is given on the x-axis and the measured AUC is shown on the y-axis. On both data sets the approximate kernel is on par with the exact kernel if more than 10% of the available symbols are selected. In particular for the web spam data less than 2% (2 symbols) suffice to achieve competitive results with the approximate kernel. Both, the approximate and exact tree kernel, yield an AUC of 73% on the question classification task and 97% AUC for web spam detection.

4.2 Approximation Stability

The previous experiment demonstrates the ability of our approximation to select discriminative symbols, yet it is not clear if such selection is stable for varying sample sizes. To examine this issue we repeat the previous experiment with a fixed number of selected symbols (8 for question classification and 2 for web spam detection) and varied the amount of data supplied for determining the selection function γ.

Figure 2 displays the assignments of the selection function γ, where the size of the provided data is given on the x-axis and the symbols are listed on the y-axis. The intensity of each point reflects how often the symbol has been chosen during 5 experimental runs.



(a) Question classification (b) Web spam detection

Figure 2: Stability of the approximate tree kernel on the question classification data set with 8 selected symbols and the web spam data set with 2 selected symbols.

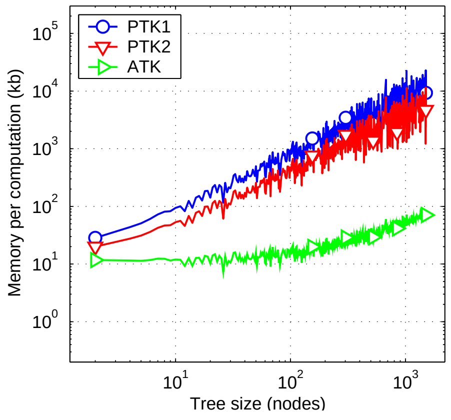
For the question classification given in Figure 2(a) the selection shows a stable pattern if more than 150 parse trees are provided for approximation. The following symbols are consistently chosen: NP, PP, VP, S1, SBARQ, SQ, TERM and DOT. The symbols NP, PP and VP capture the coarse semantic of the considered text, while SBARQ and SQ correspond to typical structure in questions. The symbols TERM (reflecting the class of all terminal symbols) and DOT (corresponding to punctuation) finally match the concrete text of questions. Note that some questions end using a full stop, e.g., "Define cosmology.".

Figure 2(b) reports on the selection stability for the web spam detection task. The selection stabilizes at 120 provided parse trees. To our surprise, the approximation picks the tags HTML and BODY. We credit this finding to the usage of templates in spam websites, which induce a strict order of high-level tags in the documents. Especially, header and meta tags beneath the HTML tag are effective for discriminating spam templates from normal web pages.

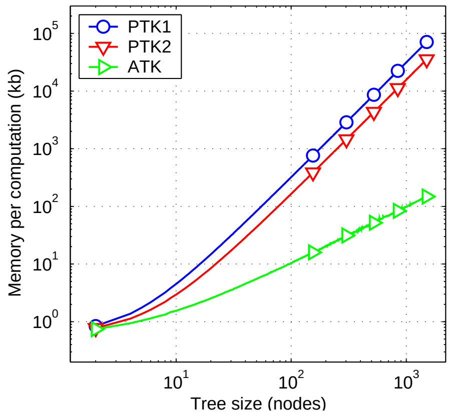
4.3 Performance Evaluation

To study the usage of run-time and memory resources we compare the approximate tree kernel to a standard implementation of the parse tree kernel (PTK1) [14] and an improved variant (PTK2) proposed by Moschitti [12]. We first estimate the average memory requirements by computing kernels between reference trees and 100 randomly drawn trees. For a worst-case scenario, the kernels are computed between identical parse trees, thus realizing the maximal number of matching node pairs. Figure 3 reports on the average and worst-case memory requirements for the web spam data set.

Figure 3(a) details the average memory consumption of the respective algorithms. The curve for the approximate kernel is significantly below the variants of the parse tree kernel. For the worst-case estimation in Figure 3(b), the memory consumption of the exact kernel scales quadratically in the number of involved nodes while the approximate tree kernel scales almost linearly in the number of nodes due to the fact



(a) Average-case memory

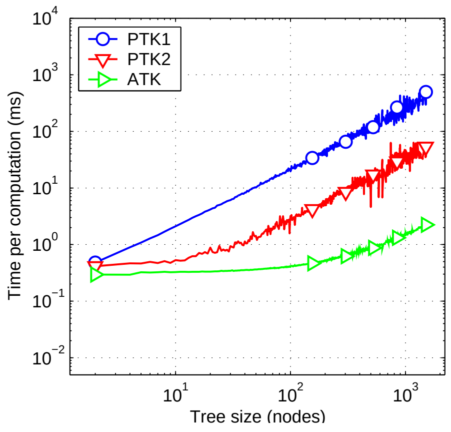


(b) Worst-case memory

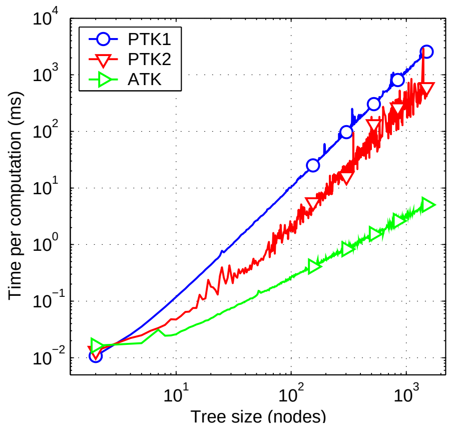
Figure 3: Memory requirements of the approximate tree kernel (ATK) and implementations of the exact parse tree kernel (PTK1, PTK2) on the web spam data set.

that only 2 symbols were selected by the approximate kernel.

For the last experiment we focus on the run-time of tree kernels. Figure 4 illustrates the results for the exact kernels and the approximate tree kernel obtained using the same setup as used for the memory experiments.



(a) Average-case run-time



(b) Worst-case run-time

Figure 4: Running time of the approximate tree kernel (ATK) and implementations of the exact parse tree kernel (PTK1, PTK2) on the web spam data set.

Figure 4(a) shows the average run-times in terms of the size of the trees. Although the improved variant proposed by Moschitti is significantly faster than a standard implementation, neither of the two show a reasonable run-time on the web spam detection data. By contrast, the approximate tree kernel computes similarities between trees almost two orders of magnitude faster. A similar picture is drawn by the worst-case analysis in Figure 4(b). The exact methods scale quadratically while the approximate tree kernel is computed in almost linear time in the number of tree nodes.

5 Conclusions

In this paper we studied an approximation of tree kernels. Classical parse tree kernels are not tractable for large tree structures, such as HTML or XML documents, as they may spend several minutes for a single kernel computation. We devised an approximation that accelerates computation by identifying a sparse set of discriminative subtrees. As a result, the run-time as well as memory requirements are significantly reduced. Moreover, our approach allows the incorporation of domain knowledge and fine-grained control of the approximation process.

Empirically, we did not find significant differences in terms of expressiveness between the approximate and exact tree kernels. However, we observed a substantial improvement in terms of memory requirements and run-time in favor of the approximate kernels. We monitored run-time improvements of two orders of magnitude for web spam detection. This efficiency denotes a valuable improvement in regard to large-scale application for tree kernels. Future research will exploit the use of our novel tree kernels for analysis of large DNA and protein structures in bioinformatics.

References

[1] C. Castillo, D. Donato, L. Becchetti, P. Boldi, S. Leonardi, M. Santini, and S. Vigna. A reference collection for web spam. *SIGIR Forum*, 40(2):11–24, December 2006.

[2] C.-C. Chang and C.-J. Lin. LIBSVM: Introduction and benchmarks. Technical report, National Taiwan University, 2000.

[3] E. Charniak. A maximum-entropy-inspired parser. Technical Report CS-99-12, Brown University, 1999.

[4] E. Cilia and A. Moschitti. Advanced tree-based kernels for protein classification. In *Artificial Intelligence and Human-Oriented Computing (AI*IA), 10th Congress*, LNCS, pages 218–229, 2007.

[5] M. Collins and N. Duffy. Convolution kernel for natural language. In *Advances in Neural Information Proccessing Systems (NIPS)*, volume 16, pages 625–632, 2002.

[6] N. Cristianini, J. Shawe-Taylor, A. Elisseeff, and J. S. Kandola. On kernel target alignment. In *Advances in Neural Information Proccessing Systems (NIPS)*, volume 14, pages 367–737, 2001.

[7] D. Haussler. Convolution kernels on discrete structures. Technical Report UCSC-CRL-99-10, UC Santa Cruz, July 1999.

[8] H. Kashima and T. Koyanagi. Kernels for semi-structured data. In *International Conference on Machine Learning (ICML)*, pages 291–298, 2002.

[9] T. Kuboyama, K. Shin, and H. Kashima. Flexible tree kernels based on counting the number of tree mappings. In *ECML/PKDD Workshop on Mining and Learning with Graphs*, 2006.

[10] X. Li and D. Roth. Learning question classifiers. In *International Conference on Computational Linguistics (ICCL)*, pages 1–7, 2002.

[11] A. Moschitti. Efficient convolution kernels for dependency and constituent syntactic trees. In *European Conference on Machine Learning (ECML)*, 2006.

[12] A. Moschitti. Making tree kernels practical for natural language processing. In *Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, 2006.

[13] B. Schölkopf and A. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.

[14] J. Shawe-Taylor and N. Cristianini. *Kernel methods for pattern analysis*. Cambridge University Press, 2004.

[15] J. Suzuki and H. Isozaki. Sequence and tree kernels with statistical feature mining. In *Advances in Neural Information Proccessing Systems (NIPS)*, volume 17, 2005.

[16] S. Vishwanathan and A. Smola. Fast kernels for string and tree matching. In *Advances in Neural Information Proccessing Systems (NIPS)*, pages 569–576, 2003.

[17] D. Zhang and W. S. Lee. Question classification using support vector machines. In *Annual International ACM SIGIR Conference*, pages 26–32, 2003.