Cyclic Pattern Kernels for Predictive Graph Mining*

Tamás Horváth
Dept. of Computer Science III
University of Bonn and
Fraunhofer Institute AIS
D-53754 Sankt Augustin,
Germany
tamas.horvath@
ais.fraunhofer.de

Thomas Gärtner
Fraunhofer Institute AIS
D-53754 Sankt Augustin,
Germany
thomas.gaertner@
ais.fraunhofer.de

Stefan Wrobel
Fraunhofer Institute AIS and
Dept. of Computer Science III
University of Bonn
D-53754 Sankt Augustin,
Germany
stefan.wrobel@
ais.fraunhofer.de

ABSTRACT

With applications in biology, the world-wide web, and several other areas, mining of graph-structured objects has received significant interest recently. One of the major research directions in this field is concerned with predictive data mining in graph databases where each instance is represented by a graph. Some of the proposed approaches for this task rely on the excellent classification performance of support vector machines. To control the computational cost of these approaches, the underlying kernel functions are based on frequent patterns. In contrast to these approaches, we propose a kernel function based on a natural set of cyclic and tree patterns independent of their frequency, and discuss its computational aspects. To practically demonstrate the effectiveness of our approach, we use the popular NCI-HIV molecule dataset. Our experimental results show that cyclic pattern kernels can be computed quickly and offer predictive performance superior to recent graph kernels based on frequent patterns.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications - Data Mining; I.2.6 [**Artificial Intelligence**]: Learning; I.5.2 [**Pattern Recognition**]: Design Methodology - classifier design and evaluation

General Terms

Algorithms, Experimentation

Keywords

graph mining, kernel methods, computational chemistry

*This work was supported in part by the DFG project (WR 40/2-1) *Hybride Methoden und Systemarchitekturen für heterogene Informationsräume*.

KDD'04, August 22–25, 2004, Seattle, Washington, USA.

1. INTRODUCTION

In recent years, data mining has moved far beyond the original commercial applications into areas such as bioinformatics or web mining. While in most of the web mining applications instances are vertices of the single massive web graph [21]; in other application domains each instance can be a *graph*. This is perhaps most obvious in applications that deal with molecules, since each molecule consists of atoms (the vertices of the graph) that are connected by bonds (the edges of the graph). In such chemical domains there is usually also a label assigned to each vertex and edge, modelling for example atom and bond types.

Such graph structured instances have no natural representation as a single row of a single fixed-width table. Therefore, there has recently been an increased interest in methods that can accept graph-structured instances as input. In *predictive graph mining* dealing with graph instances, the learning algorithm is not only given a set of disjoint graphs of arbitrary size but also for each graph the value of some property is known. The task of the algorithm is then to produce a model which approximates well the unknown dependence between graphs and the value of this target property.

In this paper, we concentrate on predictive graph mining dealing with graph instances. In particular, we focus on a class of supervised learning algorithms, namely *support vector machines* [31] and other *kernel methods* [28]. Kernel methods have proven superior to other approaches in a large number of application areas and for several types of data including tabular and text data. For general graphs, however, it has proven challenging to design kernel functions that are both powerful enough to handle arbitrarily structured graphs while being efficient enough to be applied to large graph databases.

Many researchers have consequently resorted to represent each graph by its *frequent subgraphs* (see, e.g., [4, 9, 20]) and then to apply a kernel to the pattern sets corresponding to these frequent subgraphs. The first step of such approaches usually involves a levelwise algorithm similar to APriori [1] for association rules, that is able to find all subgraphs whose frequency in the graph database is beyond a user defined threshold. The efficiency of such an approach depends on the number of frequent patterns and thus on the frequency threshold that is chosen. A too low threshold hinders feasible computation of the frequent subgraphs; a too high threshold always risks losing interesting patterns that would have been necessary for optimal classification performance.

In this paper, we show that with a natural set of patterns, *cyclic and tree patterns*, it is possible to eliminate the restriction to frequent patterns. We map the graphs to these pattern sets, independent of the frequency of the patterns in the graph database. Similar to the frequent subgraph based approaches, we then apply a kernel to these pattern sets. The resulting kernel on graphs is called the *cyclic pattern kernel*. We draw on results from graph theory to arrive at an algorithm that is in practice capable of quickly identifying the set of cyclic and tree patterns even in large sets of example graphs. We present experiments on the popular NCI-HIV dataset containing 42687 molecules, to show that our kernel, compared to previous approaches, offers significant gains in accuracy, as measured by the area under the ROC curve [26]. We give a theoretical complexity analysis of the cyclic pattern kernel, indicating that its efficiency in practice results from a well-behavedness effect similar to the one observed in frequent set discovery for association rule mining.

The paper is organized as follows. In Section 2, we define all necessary notions of graphs and kernel methods. Before proceeding to the development of our own kernel, in Section 3 we first briefly review previous results on graph kernels. In Section 4, we define cyclic pattern kernels and discuss their computational aspects. Section 5 contains the empirical evaluation on the NCI-HIV dataset, and Section 6 concludes with pointers to future work.

2. PRELIMINARIES

In this section we define some necessary notions and notations related to graphs and kernel methods.

2.1 Graphs

We first recall some basic definitions from graph theory (see, e.g., Diestel's textbook [10] for more details). For a set S, $[S]^k$ denotes the family of k-subsets of S, i.e., $[S]^k = \{S' \subseteq S : |S'| = k\}$. A *labeled undirected graph* is a quadruple $G = (V, E, \Sigma, \lambda)$, where V is a finite set of *vertices*, $E \subseteq [V]^2$ is a set of *edges*, Σ is a finite linearly ordered set of *labels*, and $\lambda : V \cup E \to \Sigma$ is a function assigning a label to each element of $V \cup E$. The cardinalities of V and E are denoted by n and m, respectively. Unless otherwise stated, in this paper by graphs we always mean *labeled undirected* graphs. A *graph database* is a set of disjoint graphs. For a graph database $\mathcal{G}$, $|\mathcal{G}|$ denotes the number of graphs in $\mathcal{G}$. Note that graphs can be viewed as *relational structures* [11] and hence, graph databases can be considered as relational databases.

Walks, Paths, and Cycles A *walk* is a sequence

$$w = \{v_0, v_1\}, \{v_1, v_2\}, \ldots, \{v_{k-1}, v_k\}$$

of edges of a graph. The walk w is a *simple path* if the v_i's are all distinct. If $v_0 = v_k$ and $v_i \neq v_j$ for every i, j $(1 \leq i < j \leq k)$ then w forms a *simple cycle*. We denote by $\mathcal{S}(G)$ the set of simple cycles of a graph G. Two simple cycles C and C' in G are considered to be the same if and only if C or its reverse is a cyclic permutation of C'. We note that the number of simple cycles is exponential in the number n of vertices in worst case[1].

[1]In fact, the number of simple cycles in a graph can grow *faster* with n than 2^n, and remains exponential even for many restricted graph classes in worst case. For instance, in [2], Alt, Fuchs, and Kriegel investigate simple cycles of *planar* graphs, and show that there are planar graphs with lower bound 2.28^n on the number of simple cycles.

Biconnected Components Let $G = (V, E, \Sigma, \lambda)$ and $G' = (V', E', \Sigma, \lambda')$ be graphs. G' is a *subgraph* of G, if $V' \subseteq V$, $E' \subseteq E$, and $\lambda'(x) = \lambda(x)$ for every $x \in V' \cup E'$. A graph is *connected* if there is a (simple) path between any pair of its vertices. A *connected component* of a graph G is a maximal subgraph of G that is connected. A vertex v of a graph G is an *articulation* (also called *cut*) vertex, if its removal disconnects G (i.e., the subgraph obtained from G by removing v and all edges containing v has more connected components than G). A graph is *biconnected* if it contains no articulation vertex. A *biconnected component* (or *block*) of a graph is a maximal subgraph that is biconnected. It holds that biconnected components of a graph G are pairwise edge disjoint and form thus a partition on the set of G's edges. This partition, in turn, corresponds to the following equivalence relation on the set of edges: two edges are equivalent if and only if they belong to a common simple cycle. This property of biconnected components implies that an edge of a graph belongs to a simple cycle if and only if its biconnected component contains more than one edge. Edges not belonging to simple cycles are called *bridges*. The subgraph of a graph G formed by the bridges of G is denoted by $\mathcal{B}(G)$. Clearly, each bridge of a graph is a (singleton) biconnected component, and $\mathcal{B}(G)$ is a forest.

Isomorphism We will also use the notion of *isomorphism* between graphs. Let G_1 and G_2 be the graphs $(V_1, E_1, \Sigma, \lambda_1)$ and $(V_2, E_2, \Sigma, \lambda_2)$, respectively. G_1 and G_2 are *isomorphic* if there is a *bijection* $\varphi : V_1 \to V_2$ such that

- $\{u, v\} \in E_1$ if and only if $\{\varphi(u), \varphi(v)\} \in E_2$,
- $\lambda_1(u) = \lambda_2(\varphi(u))$, and
- $\lambda_1(\{u, v\}) = \lambda_2(\{\varphi(u), \varphi(v)\})$

hold for every $u, v \in V_1$.

Although by the definition of graphs we consider only *simple* graphs (i.e., graphs without loops and parallel edges), we finally note that the approach presented in this paper can be adapted to non-simple graphs as well.

2.2 Kernel Methods

Kernel methods [28] are a recent development within the machine learning and data mining communities. Being on one hand theoretically well founded in statistical learning theory, they have on the other hand shown good empirical results in many applications. One particular aspect of kernel methods such as the support vector machine is the formation of hypotheses by linear combination of positive definite kernel functions 'centred' at individual training examples. By the restriction to positive definite kernel functions, the underlying optimisation problem becomes convex and every locally optimal solution is globally optimal.

Kernel Functions Kernel methods can be applied to different kinds of (structured) data by using any positive definite kernel function defined on the data. Here then is the definition of a positive definite kernel ($\mathbb{Z}^+$ is the set of positive integers):

Let $\mathcal{X}$ be a set. A symmetric function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a *positive definite kernel* on $\mathcal{X}$ if, for all $n \in \mathbb{Z}^+$, $x_1, \ldots, x_n \in$

$\mathcal{X}$, and $c_1, \ldots, c_n \in \mathbb{R}$, it holds that

$$\sum_{i,j \in \{1,\ldots,n\}} c_i \; c_j \; k(x_i, x_j) \geq 0$$

Mercer's theorem guarantees that for every positive definite kernel function k, there is a map ϕ into an inner product space, such that for every $x, x' \in \mathcal{X}$ it holds that $k(x, x') = \langle \phi(x), \phi(x') \rangle$ where $\langle \cdot, \cdot \rangle$ denotes the inner product in that space. Although this inner product space may have infinite dimension, it is often possible to compute k in polynomial time. For a simple example, consider the map ϕ_δ that maps every positive integer x to a sequence of zeros and ones such that the x-th element of the sequence $\phi_\delta(x)$ is one and all other elements are equal to zero. Clearly, the inner product space in which the images reside (l_2) does not have a finite base. Still, for all $x, x' \in \mathbb{Z}^+$, $\langle \phi_\delta(x), \phi_\delta(x') \rangle$ can be computed in polynomial time.

Kernel Machines The usual supervised learning model [31] considers a set $\mathcal{X}$ of individuals and a set $\mathcal{Y}$ of labels, such that the relation between individuals and labels is a fixed but unknown probability measure on the set $\mathcal{X} \times \mathcal{Y}$. The common theme in many different kernel methods such as support vector machines, Gaussian processes, or regularised least squares regression is to find a hypothesis function that minimises not just the empirical risk (training error) but the *regularised risk*. This gives rise to the optimisation problem

$$\min_{f(\cdot) \in \mathcal{H}} \frac{C}{n} \sum_{i=1}^{n} V(y_i, f(x_i)) + \|f(\cdot)\|^2_{\mathcal{H}}$$

where $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ is a set of individuals with known label (the training set), C trades off between regularisations and empirical loss, $\mathcal{H}$ is a set of functions forming a Hilbert space (the hypothesis space), and V is a function that takes on small values whenever $f(x_i)$ is a good guess for y_i and large values whenever it is a bad guess (the loss function). The *representer theorem* shows that under rather general conditions on V, solutions of the above optimisation problem have the form

$$f(\cdot) = \sum_{i=1}^{n} c_i k(x_i, \cdot) \tag{1}$$

where k is the reproducing kernel of $\mathcal{H}$. Different kernel methods arise from using different loss functions.

Support Vector Machines Support vector machines [5, 28] are a kernel method that can be applied to binary supervised classification problems. They are derived from the above optimisation problem by choosing the so-called *hinge loss* $V(y, f(x)) = \max\{0, 1 - yf(x)\}$. The motivation for support vector machines often taken in literature is that the solution can be interpreted as a hyperplane that separates both classes and is maximally distant from the convex hulls of both classes. A different motivation is the computational attractiveness of sparse solutions of the function (1) used for classification.

Intersection Kernels An integral part of many kernels for structured data is the *decomposition* of an object into a set of its parts and the *intersection* of two sets of parts. The kernel on two objects is then defined as a measure of the intersection of the two corresponding sets of parts.

The general case of interest for set kernels is when the instances X_i are elements of a semiring of sets $\mathfrak{S}$ and there is a measure μ with $\mathfrak{S}$ as its domain of definition. Positive definiteness of the *intersection kernel*

$$k_\cap(X_i, X_j) = \mu(X_i \cap X_j) \tag{2}$$

holds under these general conditions.

As, however, the discrete case is the most common case and simpler than the general case, here we restrict our attention to this case. Let $\mathcal{X}$ be the set of possible 'parts' and let the objects be decomposed into sets of parts $X_i \subseteq \mathcal{X}$. For every X_i the characteristic function $\Gamma_i : \mathcal{X} \to \{0, 1\}$ is defined by $\Gamma_i(x) = 1 \Leftrightarrow x \in X_i$ and $\Gamma_i(x) = 0$ otherwise. For any measure μ on $\mathcal{X}$ and sets X_i with $\mu(X_i) < \infty$, the intersection kernel is a positive definite kernel on $2^{\mathcal{X}}$ as

$$\begin{aligned}\sum_{ij} c_i c_j \mu(X_i \cap X_j) &= \sum_{ij} c_i c_j \sum_{x \in \mathcal{X}} \Gamma_i(x) \Gamma_j(x) \mu(x) \\ &= \sum_{x \in \mathcal{X}} \left(\sum_i c_i \Gamma_i(x) \right)^2 \mu(x) \\ &\geq 0\end{aligned}$$

The above kernel function can easily be extended to the case of multisets, where the characteristic function $\Gamma_i : \mathcal{X} \to \mathbb{N}$ returns the number of times an element occurs in the multiset. This extension is of interest in the case that objects are not just decomposed into a set of its parts but into a multiset.

Note that in the discrete case considered here, with the set cardinality as the measure, the intersection kernel coincides with the inner product of the bitvector representations of the sets (or multiplicity vector representations of multisets). In the case that the sets X_i are finite or countable sets of vectors it is often beneficial to use set kernels other than the intersection kernel. For example the *crossproduct kernel*

$$k_\times(X_i, X_j) = \sum_{x_i \in X_i, x_j \in X_j} k(x_i, x_j)$$

In the case that the right hand side kernel is the matching kernel (defined as $k_\delta(x_i, x_j) = 1 \Leftrightarrow x_i = x_j$ and 0 otherwise), the crossproduct kernel coincides with the intersection kernel.

3. RELATED GRAPH KERNELS

The above described idea of decomposition and intersection kernels is reflected in most work on kernels for structured data, from the early and influential technical reports [17, 33] through other work on string kernels [23, 24] and tree kernels [6], to more recent work on graph kernels [15, 19].

We now briefly review previous results on graph kernels. While in this paper we are concerned with undirected graphs, prior work mostly considered directed graphs. However, we can regard an undirected graph as a directed graph with edges in either direction. Conceptually, the graph kernels presented in [13, 15, 19] are based on a measure of the walks in two graphs that have some or all labels in common. In [13] walks with equal initial and terminal label are counted, in [19] the probability of random walks with equal label sequences is computed, and in [15] walks with equal label sequences, possibly containing gaps, are counted. Note that even very simple graphs contain an infinite number of walks and thus even with a small number of different labels, the feature space corresponding to the kernel has in-

finite dimension. In [15] computation of these kernels is made possible in polynomial time by using the direct product graph and computing the limit of matrix power series involving its adjacency matrix. The work on rational graph kernels [7] generalises these graph kernels by using a general transducer between weighted automata instead of the direct graph product. However, only walks up to a given length are considered in the kernel computation.

Describing each vertex in a graph by the set of walks starting at this vertex can be seen as a colouring of the corresponding vertex. Such colourings are also used in isomorphism tests. There one would like two vertices to be coloured differently iff they do not lie on the same orbit of the automorphism group [12]. As no efficient algorithm for the ideal case is known, one often resorts to colourings such that two differently coloured vertices cannot lie on the same orbit. Using the walks as colours for each vertex satisfies the latter condition.

Indeed, it has been shown [15] that a graph kernel for which the kernels centred at two graphs are equivalent if and only if the two graphs are isomorphic, is at least as hard as deciding graph isomorphism (these kernels are called *complete graph kernels*). So far, the above mentioned graph kernels are the only known efficient alternatives to these complete graph kernels.

Now, consider the following kernel on graphs: Let $\mathcal{G}$ be the set of all graphs and $\Phi : \mathcal{G} \to 2^{\mathcal{G}}$ be a function mapping each graph G to the set of connected subgraphs of G. Using the intersection kernel (given in Equation (2)) on the images of each graph under Φ with the set cardinality as measure, the subgraph kernel[2] is defined as

$$k_{\mathrm{SG}}(G_i, G_j) = k_{\cap}(\Phi(G_i), \Phi(G_j)) = |\Phi(G_i) \cap \Phi(G_j)|$$

It has been shown in [15] that this kernel cannot be computed in polynomial time. In the next section we will consider the complexity of computing a related kernel function.

In literature, different approaches have been tried to overcome this problem. [16] restricts the decomposition to paths up to a given size, and [8] only considers the set of connected graphs that occur frequently as subgraphs in the graph database. The approach taken there to compute the decomposition of each graph is an iterative one [22]. The algorithm starts with a frequent set of subgraphs with one or two edges only. Then, in each step, from the set of frequent subgraphs of size l, a set of candidate graphs of size $l+1$ is generated by joining those graphs of size l that have a subgraph of size $l-1$ in common. Of the candidate graphs only those satisfying a frequency threshold are retained for the next step. The iteration stops when the set of frequent subgraphs of size l is empty.

4. CYCLIC PATTERN KERNELS

In this section, we first define *cyclic pattern kernels* (CPK) for graphs and then discuss their computational aspects. Our definition is based on the intersection kernel described in Section 2.2. To apply intersection kernels to graphs, we assign to each graph G the set of *cyclic* and *tree patterns* of G. These patterns, in turn, are induced by the sets of simple cycles and bridges of the graph, respectively.

[2]The intersection now means intersection with respect to isomorphism.

4.1 Kernel Definition

We start by defining the set of cyclic patterns induced by the set of simple cycles of a graph. Let $G = (V, E, \Sigma, \lambda)$ be a graph and

$$C = \{v_0, v_1\}, \{v_1, v_2\}, \ldots, \{v_{k-1}, v_0\}$$

be a sequence of edges that forms a simple cycle in G. The *canonical representation* of C is the lexicographically smallest string $\pi(C) \in \Sigma^*$ among the strings obtained by concatenating the labels along the vertices and edges of the cyclic permutations of C and its reverse. More precisely, denoting by $\rho(s)$ the set of cyclic permutations of a sequence s and its reverse, we define $\pi(C)$ by

$$\pi(C) = \min\{\sigma(w) : w \in \rho(v_0 v_1 \ldots v_{k-1})\},$$

where for $w = w_0 w_1 \ldots w_{k-1}$,

$$\sigma(w) = \lambda(w_0)\lambda(\{w_0, w_1\})\lambda(w_1)\ldots\lambda(w_{k-1})\lambda(\{w_{k-1}, w_0\}).$$

Clearly, π is unique up to isomorphism, and hence, it indeed provides a canonical string representation of simple cycles. The set of *cyclic patterns* of a graph G, denoted by $\mathcal{C}(G)$, is then defined by

$$\mathcal{C}(G) = \{\pi(C) : C \in \mathcal{S}(G)\} \ .$$

(We recall that $\mathcal{S}(G)$ denotes the set of simple cycles of G.)

To assign a set of cyclic patterns to a graph G, above we have used its set of simple cycles. To add more information to the kernel, we also consider the graph obtained by removing the edges of all simple cycles, or equivalently, by deleting every edge that belongs to some of G's biconnected components containing at least two edges. As discussed in Section 2.1, the resulting graph is a forest consisting of the set of bridges of the graph. To assign a set of tree patterns to G, we use this forest formed by the set $\mathcal{B}(G)$ of bridges of G. Similarly to simple cycles, we associate each tree T with a pattern $\pi(T) \in \Sigma^*$ that is unique up to isomorphism[3], and define the set of *tree patterns* $\mathcal{T}(G)$ assigned to G by

$$\mathcal{T}(G) = \{\pi(T) : \ T \text{ is a connected component of } \mathcal{B}(G)\} \ .$$

We are now ready to define cyclic pattern kernels for graphs. In the definition below, we assume without loss of generality that $\mathcal{C}(G)$ and $\mathcal{T}(G)$ are disjoint for every G in the database. Our kernel is an intersection kernel (2) on the sets defined by

$$\Phi_{\mathrm{CP}}(G) = \mathcal{C}(G) \cup \mathcal{T}(G) \tag{3}$$

for every G. More precisely, we define *cyclic pattern kernels* by

$$\begin{aligned} k_{\mathrm{CP}}(G_i, G_j) &= k_{\cap}(\Phi_{\mathrm{CP}}(G_i), \Phi_{\mathrm{CP}}(G_j)) \\ &= |\mathcal{C}(G_i) \cap \mathcal{C}(G_j)| + |\mathcal{T}(G_i) \cap \mathcal{T}(G_j)| \end{aligned} \tag{4}$$

for every G_i, G_j in a graph database $\mathcal{G}$, where the measure μ used in the intersection kernel is the cardinality.

[3]We first transform the *unordered free tree* T into a *rooted ordered tree* T' unique up to isomorphism, and define then $\pi(T)$ by the string representing T'. We omit the technical description and refer to e.g. [3, 34] for further details on canonical string representations of trees.

Algorithm 1 BASIC ALGORITHM

Require: graph G with n vertices and m edges
Ensure: $\Phi_{\mathrm{CP}}(G)$

1: **let** $S = B = \emptyset$
2: compute the biconnected components of G
3: **for all** biconnected component G' of G **do**
4: **if** G' contains more then one edge **then**
5: $S = S \cup \mathcal{C}(G')$
6: **else**
7: add the edge of G' to B
8: $S = S \cup \{\pi(t) :\ t$ is a connected component of $B\}$
9: **return** S

4.2 Computing the Pattern Set: General Case

As discussed in Section 2.2, even infinite dimension of the feature space associated with a kernel still does not imply its computational intractability. Another issue is, whether in general, the intractability of computing the value of a single feature implies the hardness of computing the kernel. We do not know the answer to this general problem which includes also the case of cyclic pattern kernels, as Φ_{CP} may contain patterns corresponding to Hamiltonian cycles as well. Using a similar argument as [15], in Proposition 1 below we show that computing cyclic pattern kernels is intractable.

PROPOSITION 1. *The problem of computing cyclic pattern kernels is NP-hard.*

PROOF. We shall use a reduction from the NP-complete Hamiltonian cycle problem. Let G and C_n be a graph and a simple cycle, respectively, such that both G and C_n consist of n vertices and are defined over the same singleton label set. Applying (4) to G and C_n, it holds that $k_{\mathrm{CP}}(G, C_n) = 1$ if and only if G has a Hamiltonian cycle. □

Although the cardinality of the set $\Phi_{\mathrm{CP}}(G)$ of cyclic and tree patterns of a graph G can be exponential in the number of vertices of G, we still turn to this problem restricting the approach to those well-behaved cases where $\Phi_{\mathrm{CP}}(G)$ can be computed in practically reasonable time. In Section 5 we will present a large real-world molecular graph database satisfying this property. Before focusing in later sections on well-behaved domains, below we first discuss some issues of computing $\Phi_{\mathrm{CP}}(G)$ in the general case.

The basic algorithm computing $\mathcal{C}(G)$ and $\mathcal{T}(G)$ for a graph G is sketched in Algorithm 1. From the arguments of Section 2.1 it follows that the algorithm computes the set of bridges in B, and returns finally $\Phi_{\mathrm{CP}}(G)$ in S. Regarding the complexity of Algorithm 1, we first note that Step 2 can be solved in linear time; in [29], Tarjan gives a depth-first search algorithm computing all biconnected components of a graph in time $O(n+m)$. Since the number of bridges of G is at most m, and the number of trees is bounded by the cardinality of the set of bridges of G, in Step 8 we compute the set $\mathcal{T}(G)$ of tree patterns in time polynomial in n. In Step 5, we compute the set of cyclic patterns of a biconnected component of G. From the point of view of efficiency, this step of the algorithm is critical, as even the number of cyclic patterns of a biconnected graph can be exponential in the number of vertices. In Example 2 below, we give a graph and show that $2^{O(|V|)}$ is a lower bound on the number of its cyclic patterns, where V is the set of vertices of the graph.

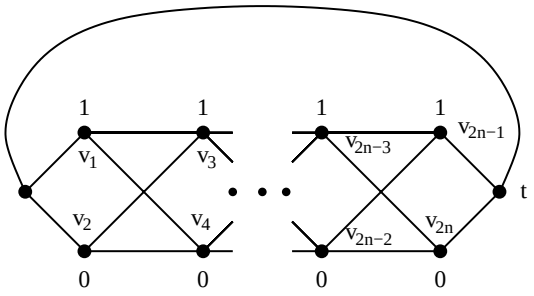

Figure 1: The graph used in Example 2

EXAMPLE 2. *Let $G = (V, E, \Sigma, \lambda)$ be a graph such that*

$$V = \{s, t, v_1, \ldots, v_{2n}\},$$
$$E = \{\{s, v_1\}, \{s, v_2\}, \{v_{2n-1}, t\}, \{v_{2n}, t\}, \{s, t\}\} \cup \{\{v_{2i-k}, v_{2i+l}\} : i = 1, \ldots, n-1,\ k = 0, 1,\ l = 1, 2\},$$
$$\Sigma = \{0, 1, 2\},$$

and

$$\lambda(x) = \begin{cases} k \bmod 2 & \text{if } x = v_k \in V \\ 2 & \text{otherwise} \end{cases}$$

for every $x \in V \cup E$. Note that by the above definition, λ assigns 2 to every edge in E, as well as to the vertices s and t. Omitting the labels of s, t, and the edges, the figure of G is given in Fig. 1. Let

$$L = \{\min\{w, w^{-1}\} : w \in \{0, 1\}^* \text{ and } |w| = n\}\ ,$$

where w^{-1} and $|w|$ denote the reverse and the length of w, respectively. (The strings in Σ^* are compared by the lexicographic order induced by the linear order on Σ.) It holds that for every $w \in L$, there is a simple cycle C in G such that we can obtain w by deleting all 2's from the canonical representation $\pi(C)$ of C. Hence, for the cardinality of the set $\mathcal{C}(G)$ of cyclic patterns we have

$$|\mathcal{C}(G)| \geq |L| > 2^{n-1} = 2^{O(|V|)}\ .$$

Since the cardinality of the set $\mathcal{C}(G)$ of cyclic patterns can also be exponential in the number n of G's vertices, we next consider the problem of computing $\mathcal{C}(G)$ in *polynomial output complexity*. That is, we ask whether there exists an algorithm that enumerates $\mathcal{C}(G)$ in time polynomial in n and $|\mathcal{C}(G)|$. Proposition 3 below states that even this problem is NP-hard.

PROPOSITION 3. *Let G be a graph with n vertices, and $N \leq |\mathcal{C}(G)|$ be an arbitrary non-negative integer. Then the problem of enumerating N elements from $\mathcal{C}(G)$ with parameters n and N is NP-hard.*

PROOF. We show that the NP-complete Hamiltonian cycle problem is polynomial-time reducible to the above enumeration problem. Let G be an ordinary undirected graph (i.e., a graph without labels) with n vertices. We assign a (labeled undirected) graph G' to G such that G' has the same sets of vertices and edges as G, and each vertex and edge of G' is labeled by the same symbol, say 0. Since simple cycles of the same length in G' are mapped to the same

pattern (i.e., simple cycles of length ℓ are associated with the pattern $0^{2\ell}$), $|\mathcal{C}(G')| < n$. Applying the enumeration algorithm with $N = n-1$, we obtain a set S containing at most $n-1$ elements of $\mathcal{C}(G')$. Clearly, 0^{2n} is in S if and only if G has a Hamiltonian cycle. □

In order to overcome the negative complexity result implied by Proposition 3 above, we consider a restriction that yields an effective *practical* problem class.

4.3 Graphs with Bounded Cyclicity

In contrast to the case of cyclic patterns, the set $\mathcal{S}(G)$ of simple cycles of a graph G can be listed in polynomial output complexity. A depth-first search algorithm computing $N \leq |\mathcal{S}(G)|$ simple cycles of a graph G in time $O((N+1)n+m)$ is given by Read and Tarjan in [27]. From their result it also follows that, for a given graph G and $k \geq 0$, one can decide efficiently, whether or not the number of simple cycles in G is bounded by k. Using these positive results, in this section we consider the case when the number of simple cycles is bounded by a constant for (almost) every graph in the database. In certain real-world graph databases (e.g., drug molecules), the assumption of such a bound seems reasonable. As an example, in the NCI dataset[4] there is a very natural and clear such bound; this molecular graph database consists of 250251 graphs containing altogether 2161831 simple cycles[5]. Note that effectively, we are assuming a certain kind of *well-behavedness* of our graph-structured input data which results in only a very small number of objects with extremely large numbers of simple cycles. We note that a similar assumption is made for example in association rule mining where one assumes that transaction data are well-behaved so as to not induce many frequent sets of overly large sizes.

Algorithm 2, a variant of Algorithm 1, computes the set Φ_{CP} of patterns (3) only for those graphs that have at most k simple cycles. In Step 6 of the algorithm, we call Read and Tarjan's algorithm [27] (subroutine RT) with parameters being the current biconnected component G' and $k-K+1$, where K is a variable counting the simple cycles that have been found so far. If G' contains more than $k-K$ simple cycles, the algorithm halts and returns the empty set (Step 7), as in this case G has more than k simple cycles. The efficiency of the algorithm follows from that of the subroutine RT.

Finding an Appropriate Bound We now turn to the problem of deciding whether there exists an upper bound on the number of simple cycles such that all but a small subset of the graphs in the database can be processed within a user defined time limit. Depending on the value of a threshold given by the user, a small subset of the database requiring potentially too much computation time can be disregarded. For instance, in the NCI-HIV graph database used in our experiments it makes sense to consider only those graphs that have less than 100 simple cycles, as even in this case, 99.76% of the whole dataset is still covered (see also Table 1 in Section 5.1).

More precisely, *given* an upper bound T on the time needed to compute the cyclic and tree patterns for the database $\mathcal{G}$ and a *threshold* $\tau \in [0,1]$, our goal is to *find* the smallest k such that (with high probability)

(i) there is a subset $\mathcal{G}' \subseteq \mathcal{G}$ with cardinality at least $\tau|\mathcal{G}|$ such that each graph in $\mathcal{G}'$ has at most k cycles, and

(ii) applying Algorithm 2 with parameter k to the graphs in $\mathcal{G}$, the total running time is bounded by T,

if such a k exists, and to print 'NO' otherwise. We first note that applying Algorithm 2 with parameter k to the graphs in $\mathcal{G}$, the running time is bounded by

$$O\left(|\mathcal{G}| \cdot ((k+2)n_{\max} + 2m_{\max})\right) \ ,$$

where $n_{\max}$ (resp. $m_{\max}$) is the maximum number of vertices (resp. edges) of any graph in $\mathcal{G}$. Thus, for a given time limit T, one can compute the maximum value of the upper bound K on the number of simple cycles for which the algorithm still runs in time T.

Given K, we would like to find the smallest $k' \leq K$ such that at least $\tau|\mathcal{G}|$ graphs in $\mathcal{G}$ have at most k' simple cycles. We note that this problem cannot be solved efficiently by *counting* the number of simple cycles in a graph. This follows from the negative result of Valiant [30] which states that counting the number of simple paths between two vertices in a graph is #P-complete. We therefore propose a technique based on sampling[6] and enumerating at most $K+1$ simple cycles of a graph. Using some sampling method, we first select a random sample S of the underlying graph database $\mathcal{G}$. Then, applying Read and Tarjan's algorithm, we assign the number $\kappa(G)$ defined by

$$\kappa(G) = \begin{cases} |\mathcal{S}(G)| & \text{if } |\mathcal{S}(G)| \leq K \\ K+1 & \text{otherwise} \end{cases}$$

to every $G \in S$. We then define k' by

$$k' = \min\{k : 0 \leq k \leq K+1 \text{ and } |S(k)| \geq \tau|S|\} \ ,$$

where $S(k) = \{G \in S : \kappa(G) \leq k\}$, and print k' if $k' \leq K$, and 'NO' otherwise (i.e., if $k' = K+1$).

Algorithm 2 BOUNDED CYCLICITY

Require: graph G with n vertices and m edges, and $k \in \mathbb{N}$
Ensure: $\Phi_{CP}(G)$ if $|\mathcal{S}(G)| \leq k$ and $\emptyset$ otherwise

```
1: let K = 0
2: let S = B = ∅
3: compute the biconnected components of G
4: for all biconnected component G' of G do
5:   if G' contains more then one edge then
6:     let X = RT(G', k − K + 1)
7:     if |X| = k − K + 1 then return ∅
8:     else
9:       S = S ∪ {p(C) : C ∈ X}
10:      K = K + |X|
11:  else
12:    add the edge of G' to B
13: S = S ∪ {π(t) : t is a connected component of B}
14: return S
```

[4] http://cactus.nci.nih.gov/

[5] The NCI-HIV dataset used in our experiments is an annotated subset of the NCI domain.

[6] Note that our problem is to estimate the unknown parameter of a Bernoulli distribution.

5. EMPIRICAL EVALUATION

To evaluate empirically the predictive performance of cyclic pattern kernels, in this section we use the HIV dataset of chemical compounds. This database is maintained by the National Cancer Institute (NCI)[7] and describes information of the compounds' capability to inhibit the HIV virus. It has been used frequently in the empirical evaluation of graph mining approaches (see, e.g., [4, 8, 9, 20]). So far, the only approaches to predictive graph mining on this dataset are described in [8, 9].

5.1 Data

In the NCI-HIV database, each compound is described by its chemical structure[8] and classified into three categories: confirmed inactive (CI), moderately active (CM), or active (CA). A compound is inactive if a test showed less than 50% protection of human CEM cells. All other compounds were retested. Compounds showing less than 50% protection (in the second test) are also classified inactive. The other compounds are classified active, if they provided 50% protection in both tests, and moderately active, otherwise.

The NCI-HIV dataset we used contains 42689 molecules, 423 of which are active, 1081 are moderately active, and 41185 are inactive. The total number of vertices and edges in this dataset is 1951154 and 2036712, respectively. Table 1 shows how the number of simple cycles is distributed over molecules. Clearly, the well-behavedness assumption made above holds for this dataset. Figure 2 illustrates these frequencies on a log-log scale.. On a PC with a Pentium III/850 MHz Processor, the set of cyclic and tree patterns for every graph in the database has been computed in less than 10 minutes.

Table 1: Distribution of simple cycles over compounds

simple cycles	compounds	fraction
0	1655	3.88%
$1-9$	36026	84.40%
$10-19$	4012	9.40%
$20-29$	514	1.20%
$30-59$	306	0.72%
$60-99$	75	0.18%
$100-199$	68	0.16%
$200-1000$	20	0.05%
>1000	11	0.03%

5.2 Comparative Analysis using ROC

Frequently, the predictive performance of algorithms is compared by means of a statistical test on the accuracies achieved by the learned model on given test sets. However, measuring accuracy is only meaningful if the class distribution and misclassification costs in the target environment are known and do not change over time [25]. Clearly, the cost of not finding an active compound should be higher than the cost of a false alarm. However, we do not know the cost of false alarms in this domain.

[7] `http://cactus.nci.nih.gov/`

[8] This database also describes other (propositional) features that we do not make use of, as we want to compare our results with those reported in [8]. Making use of the geometrical information will be considered in future work.

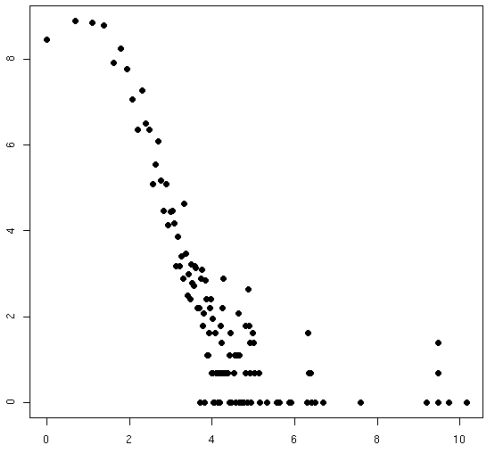

Figure 2: Log-log plot of the number of molecules (y) versus the number of simple cycles (x)

Recently, an alternative way of comparing the predictive performance on binary classification problems is becoming more and more popular in the machine learning and data mining communities. Receiver Operator Characteristic (ROC) is a two-dimensional tool (rather than one-dimensional error rates) that is able to overcome the above mentioned shortcomings of merely accuracy based comparisons. In the two-dimensional ROC space each classifier is represented by its true-positive rate (the fraction of positive examples correctly classified) and its false-positive rate (the fraction of negative examples incorrectly classified). The ideal classifier (ROC heaven) has true-positive rate 1 and false-positive rate 0. All classifiers with equal true-positive and false-positive rate (the diagonal in ROC space) correspond to randomly choosing a class for each query instance.

Any classifier with a continuous output, such as the support vector machine (compare Equation (1)), gives rise to a set of classifiers, each corresponding to a different threshold on the output. If the output for a query instance is above this threshold, it is considered positive (with respect to this threshold) and negative (with respect to this threshold) otherwise. The resulting ROC curve illustrates the possible tradeoffs between correctly classified positive examples and incorrectly classified negative examples.

Whenever a single number is preferred to compare different classifiers, ROC analysis offers the possibility of using the area under the ROC curve. To overcome the dependency of ROC analysis on a single test-set, it is desirable to combine ROC analysis with crossvalidation techniques. How to best average ROC curves, however, is still a matter of scientific dispute. In this paper we report the mean and variance of the area under the curve over different folds.

5.3 Empirical Results

To empirically evaluate the benefits of using all patterns rather than frequent patterns only, we compared our ap-

Table 2: Area under the ROC curve for different tasks and costs. Boldface numbers denote a significant win at a 1% level

CA vs CM			CA+CM vs CI			CA vs CI		
cost	CPK	FSG	cost	CPK	FSG	cost	CPK	FSG
1.0	**0.8132**(±0.014)	0.7740	1.0	**0.7745**(±0.017)	0.7420	1.0	**0.9187**(±0.011)	0.8683
1.5	**0.8222**(±0.010)	0.7802	1.5	**0.7818**(±0.016)	0.7504	1.5	**0.9264**(±0.006)	0.8676
2.0	**0.8248**(±0.013)	0.7860	15.0	0.8012(±0.018)	0.7864	15.0	**0.9340**(±0.011)	0.9023
2.5	**0.8273**(±0.013)	0.7816	35.0	0.8011(±0.017)	0.7783	35.0	**0.9321**(±0.010)	0.9097
3.0	**0.8290**(±0.012)	0.7841	50.0	0.7967(±0.017)	0.7731	50.0	**0.9318**(±0.009)	0.9122
15.0	**0.7970**(±0.019)	0.7566	100.0	0.7816(±0.018)	0.7486	100.0	0.9285(±0.010)	0.9138

Table 3: Area under the ROC curve for different tasks and costs (Gaussian $\gamma = 0.05$). Boldface numbers denote a significant win at a 1% level

CA vs CM			CA+CM vs CI			CA vs CI		
cost	γCPK	FSG	cost	γCPK	FSG	cost	γCPK	FSG
1.0	**0.8264**(±0.010)	0.7740	1.0	**0.8092**(±0.014)	0.7420	1.0	**0.9257**(±0.005)	0.8683
1.5	**0.8328**(±0.009)	0.7802	1.5	**0.8176**(±0.015)	0.7504	1.5	**0.9311**(±0.007)	0.8676
2.0	**0.8384**(±0.010)	0.7860	15.0	**0.8373**(±0.012)	0.7864	15.0	**0.9466**(±0.008)	0.9023
2.5	**0.8398**(±0.010)	0.7816	35.0	**0.8332**(±0.013)	0.7783	35.0	**0.9441**(±0.008)	0.9097
3.0	**0.8402**(±0.008)	0.7841	50.0	**0.8315**(±0.013)	0.7731	50.0	**0.9430**(±0.008)	0.9122
15.0	**0.8341**(±0.020)	0.7566	100.0	**0.8298**(±0.014)	0.7486	100.0	**0.9426**(±0.007)	0.9138

proach to the results presented in [8] and [9]. The classification problems considered there were: (1) distinguish between CA from CM, (2) distinguish CA and CM from CI, and (3) distinguish CA from CI. For each problem, the area under the ROC curve, averaged over a 5-fold crossvalidation, is given for different misclassification cost settings.

Note that, while the walk-based graph kernels considered for example in [15] (see Section 3) can be computed in polynomial time, the exponent of the polynomial appears to be too large for this application. For this reason, we have not included experiments with the walk-based graph kernel. We note, however, that successful applications using such graph kernels have been reported in [19] and [14] on smaller datasets. Computing the direct product graph takes time quadratic in the number of vertices of the graphs in $\mathcal{G}$. Inverting the adjacency matrix of the product graph or computing the eigen-decomposition of this matrix are both roughly of cubic time complexity in the number vertices of the product graph. For example, for a molecule of 214 atoms, we obtain a product graph with 34645 vertices (if we do not take the vertex labels into account we would have $214^2 = 45796$ vertices). Techniques for speeding up walk based graph kernels will be considered in future work.[9]

In our experiments we used a modified version of the SVM-light [18] support vector machine with the same set of misclassification cost parameters as used by [8]. The regularisation parameter was chosen by SVM-light. We used two different kernel functions. The first is the cyclic pattern kernel (CPK) given in Equation (4), the other is a Gaussian version of that kernel (γCPK), that is:

$$\exp[-\gamma(k_{\mathrm{CP}}(x_i,x_i) - 2k_{\mathrm{CP}}(x_i,x_j) + k_{\mathrm{CP}}(x_j,x_j))]$$

[9]For example, one could make use of the nice properties of eigen-decompositions under tensor product which is strongly related to the direct product graph. Alternatively, one could employ vertex colouring algorithms to increase the number of vertices with different colour in the original graphs – this would at the same time decrease the number of vertices in the product graph.

Tables 2 and 3 show our experimental results with cyclic pattern kernels (CPK) and those achieved in [8] (FSG) with a frequent subgraph based approach. As we did not know the variance of the area under the ROC curve for FSG, we assumed the same variance as for CPK[10]. Thus, to test the hypothesis that CPK significantly outperforms FSG, we used a pooled sample variance equal to the variance exhibited by CPK. As FSG and CPK were applied in a 5-fold crossvalidation, the estimated standard error of the average difference is the pooled sample variance times $\sqrt{\frac{2}{5}}$. The test statistic is then the average difference divided by its estimated standard error. This statistic follows a t distribution. The null hypothesis — CPK performs no better than FSG — can be rejected at the significance level α if the test statistic is greater than $t_8(\alpha)$, the corresponding percentile of the t distribution.

Table 2 reports the results achieved using the cyclic pattern kernel directly. In all experiments the mean result achieved by CPK is better than the result achieved by FSG. On a 1% significance level, CPK outperforms FSG in 13 out of 18 experimental settings performed in [8].

Table 3 reports the results achieved using the Gaussian version of the cyclic pattern kernel. For the parameter γ different values ($\gamma \in \{0.005, 0.05, 0.5\}$) were tried on the problem of distinguishing CA from CM with misclassification costs set to 1.0. These experiments showed that using the parameter 0.05 we are able to obtain slightly better areas under the ROC curve than with the other two parameters. For that, we kept the parameter $\gamma = 0.05$ throughout all experiments. In all experiments the mean result achieved by CPK is better than the results reported for FSG. Indeed, on a 1% significance level, γCPK outperforms FSG in all classification problems and cost settings that were reported in [8].

In [9] the authors of [8] describe improved results (FSG*).

[10]We could alternatively assume that the mean area of CPK is the 'true mean' we are comparing with. This would simply result in higher significance levels.

Table 4: Area under the ROC curve for different tasks and costs. Boldface numbers denote a significant win at a 5% level

CA vs CM			CA+CM vs CI			CA vs CI		
cost	CPK	FSG*	cost	CPK	FSG*	cost	CPK	FSG*
1.0	0.8132(±0.014)	0.810	1.0	0.7745(±0.017)	0.765	1.0	**0.9187**(±0.011)	0.839
2.5	**0.8273**(±0.013)	0.792	35.0	0.8011(±0.017)	0.794	100.0	**0.9285**(±0.010)	0.908

Table 5: Area under the ROC curve for different tasks and costs (Gaussian $\gamma = 0.05$). Boldface numbers denote a significant win at a 5% level

CA vs CM			CA+CM vs CI			CA vs CI		
cost	γCPK	FSG*	cost	γCPK	FSG*	cost	γCPK	FSG*
1.0	0.8264(±0.010)	0.810	1.0	**0.8092**(±0.014)	0.765	1.0	**0.9257**(±0.005)	0.839
2.5	**0.8398**(±0.010)	0.792	35.0	**0.8332**(±0.013)	0.794	100.0	**0.9426**(±0.007)	0.908

There the authors report results obtained by an optimised threshold on the frequency of patterns and by including additional, geometric features. Tables 4 and 5 compare our results to those reported in [9] for the optimised threshold. We do not compare our results to those obtained using the geometric features. We are considering to also include geometric features in our future work and expect similar improvements. Both CPK and γCPK perform better than FSG* in all learning problems and all misclassification cost settings reported in [9]. However, the improvement there is less significant. On a significance level[11] of 5% CPK performs significantly better than FSG* in 3 out of 6 cases; γCPK performs significantly better than FSG* in 5 out of 6 cases.

6. CONCLUSION AND FUTURE WORK

As an alternative to graph kernels based on frequent patterns, in this paper we have proposed a graph kernel based on cyclic and tree patterns independent of their frequency. To compute *cyclic pattern kernels*, we first extract all cyclic and tree patterns from each graph and then apply an intersection kernel to these pattern sets. Empirical results on the NCI-HIV domain indicate that this graph kernel is superior to frequent pattern based graph kernels.

Since computing cyclic pattern kernels is intractable in general, the approach presented in this paper is limited to well-behaved graph databases. That is, graph databases in which there exists a natural small upper bound on the number of simple cycles for almost every graph. As such a bound cannot be found by counting the number of simple cycles of each graph, we have proposed an algorithm based on sampling, that estimates whether the database meets this requirement. A small bound clearly exists for instance in the large NCI database including the NCI-HIV dataset that has frequently been used to evaluate graph mining approaches.

Despite the encouraging empirical results, there is still room for further work on graph kernels. It seems attractive to try to combine the ideas presented here with kernels based on walks in graphs [15, 19]. To compute these graph kernels the direct product graph has to be computed and an eigen-decomposition or inversion of its adjacency matrix has to be performed. Due to the large number of vertices of the product graphs in this domain, these approaches cannot directly be applied. Techniques for speeding up walk based graph kernels will be considered in future work.

Besides walk kernels, we are going to investigate graph kernels induced by shortest paths between vertices. In particular, we consider path (resp. walk) kernels based on the set of k shortest simple (resp. non-simple) paths between each pair of vertices.

In contrast to simple cycles, the number of *relevant cycles* of a graph can be computed in polynomial time without listing them [32]. In future work, we are going to investigate the predictive power of graph kernels based on relevant cycles. To overcome the other complexity limitation, that cyclic patterns cannot be enumerated with polynomial output complexity, we plan to investigate graph classes for which this problem can be solved polynomially.

In addition to the graph structure of molecules, most compound databases also contain information about the 3D coordinates of each atom in one of the molecule's low energy conformations. These coordinates can either be measured in experiments or computed with one of several software tools. The advantage of using software tools is that not only the coordinates of each atom in the molecule's lowest energy conformation can be computed, but the coordinates can be obtained for a set of low energy conformations. From a chemical point of view, these coordinates are important when deciding whether a molecule binds well to a target. Thus, from a machine learning perspective this information is likely to improve the predictive power of our classifier. We are working on advanced kernel functions for molecules that also take the 3D information into account.

[11] A 5% significance level is the usual level. Tables 2 and 3 showed stronger results than that, however, a significant win on a 5% significance level is usually considered sufficient.

7. REFERENCES

[1] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, Chapter 12, pages 307 – 328. AAAI/MIT Press, Cambridge, USA, 1996.

[2] H. Alt, U. Fuchs, and K. Kriegel. On the number of simple cycles in planar graphs. *Combinatorics, Probability & Computing*, 8(5):397–405, 1999.

[3] Asai, Arimura, Uno, and Nakano. Discovering frequent substructures in large unordered trees. In *Proc. of the 6th International Conference on Discovery Science*, volume 2843 of *LNAI*, pages 47–61. Springer Verlag, 2003.

[4] C. Borgelt and M. R. Berthold. Mining molecular fragments: Finding relevant substructures of molecules. In *Proc. of the 2002 IEEE International Conference on Data Mining*, pages 51–58. IEEE Computer Society, 2002.

[5] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, *Proc. of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152. ACM Press, 1992.

[6] M. Collins and N. Duffy. Convolution kernels for natural language. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2002.

[7] C. Cortes, P. Haffner, and M. Mohri. Positive definite rational kernels. In *Learning Theory and Kernel Machines, 16th Annual Conference on Learning Theory and 7th Kernel Workshop, Proceedings*. volume 2843 of *LNAI*, pages 41–56. Springer Verlag, 2003.

[8] M. Deshpande, M. Kuramochi, and G. Karypis. Automated approaches for classifying structures. In *Proc. of the 2nd ACM SIGKDD Workshop on Data Mining in Bioinformatics*, 2002.

[9] M. Deshpande, M. Kuramochi, and G. Karypis. Frequent sub-structure based approaches for classifying chemical compounds. In *Proc. of the 3rd IEEE International Conference on Data Mining*, pages 35–42. IEEE Computer Society, 2003.

[10] R. Diestel. *Graph theory*. 2nd edition, Springer Verlag, 2000.

[11] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Perspectives in Mathematical Logic. 2nd edition, Springer Verlag, 1999.

[12] M. Fürer. Graph isomorphism testing without numberics for graphs of bounded eigenvalue multiplicity. In *Proc. of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 624–631. ACM Press, 1995.

[13] T. Gärtner. Exponential and geometric kernels for graphs. In *NIPS Workshop on Unreal Data: Principles of Modeling Nonvectorial Data*, 2002.

[14] T. Gärtner, K. Driessens, and J. Ramon. Graph kernels and gaussian processes for relational reinforcement learning. In *Proc. of the 13th International Conference on Inductive Logic Programming*, volume 2835 of LNAI, pages 146–163. Springer Verlag, 2003.

[15] T. Gärtner, P. A. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Learning Theory and Kernel Machines, 16th Annual Conference on Learning Theory and 7th Kernel Workshop, Proceedings*. volume 2843 of *LNAI*, pages 129–143. Springer Verlag, 2003.

[16] T. Graepel. *PAC-Bayesian Pattern Classification with Kernels*. PhD thesis, TU Berlin, 2002.

[17] D. Haussler. Convolution kernels on discrete structures. Technical report, Department of Computer Science, University of California at Santa Cruz, 1999.

[18] T. Joachims. Making large–scale SVM learning practical. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 169–184. MIT Press, 1999.

[19] H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labeled graphs. In *Proc. of the 20th International Conference on Machine Learning*, pages 321–328. AAAI Press, 2003.

[20] S. Kramer, L. D. Raedt, and C. Helma. Molecular feature mining in HIV data. In *Proc. of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 136–143. ACM Press, 2001.

[21] S. R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal. The web as a graph. In *Proc. of the 19th ACM Symposium on Principles of Database Systems*, pages 1–10. ACM Press, 2000.

[22] M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *Proc. of the IEEE International Conference on Data Mining*, pages 313–320, IEEE Computer Society, 2001.

[23] C. Leslie and R. Kuang. Fast kernels for inexact string matching. In *Learning Theory and Kernel Machines, 16th Annual Conference on Learning Theory and 7th Kernel Workshop, Proceedings*. volume 2843 of *LNAI*, pages 114–128. Springer Verlag, 2003.

[24] H. Lodhi, J. Shawe-Taylor, N. Christianini, and C. Watkins. Text classification using string kernels. In T. Leen, T. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, volume 13. MIT Press, 2001.

[25] F. Provost, T. Fawcett, and R. Kohavi. The case against accuracy estimation for comparing induction algorithms. In *Proc. of the 15th International Conference on Machine Learning*, pages 445–453. Morgan Kaufmann, 1998.

[26] F. J. Provost and T. Fawcett. Robust classification for imprecise environments. *Machine Learning*, 42(3):203–231, 2001.

[27] R. C. Read and R. E. Tarjan. Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. *Networks*, 5(3):237–252, 1975.

[28] B. Schölkopf and A. J. Smola. *Learning with Kernels*. MIT Press, 2002.

[29] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[30] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[31] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer Verlag, 1995.

[32] P. Vismara. Union of all the minimum cycle bases of a graph. *The Electronic Journal of Combinatorics*, 4(1):73–87, 1997.

[33] C. Watkins. Kernels from matching operations. Technical report, Department of Computer Science, Royal Holloway, University of London, 1999.

[34] M. Zaki. Efficiently mining frequent trees in a forest. In *Proc. of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 71–80. ACM Press, 2002.