Graph Classification via Topological and Label Attributes

Geng Li, Murat Semerci†, Bülent Yener, and Mohammed J. Zaki
Rensselaer Polytechnic Institute, Troy, NY
†Bogazici University, Istanbul, Turkey
{lig2,yener,zaki}@cs.rpi.edu, semercim@gmail.com

ABSTRACT

Graph classification is an important data mining task, and various graph kernel methods have been proposed recently for this task. These methods have proven to be effective, but they tend to have high computational overhead. In this paper, we propose an alternative approach to graph classification that is based on feature-vectors constructed from different global topological attributes, as well as global label features. The main idea here is that the graphs from the same class should have similar topological and label attributes. Our method is simple and easy to implement, and via a detailed comparison on real benchmark datasets, we show that our topological and label feature-based approach delivers better or competitive classification accuracy, and is also substantially faster than other graph kernels. It is the most effective method for large unlabeled graphs.

1. INTRODUCTION

With the proliferation of graph data, there has been a lot of interest in recent years to develop effective methods for classifying graph objects [13]. Applications range from chem-informatics [21, 19] (e.g., compounds that are active or inactive for some target) and bioinformatics [5, 2] (e.g., classifying proteins into different families, classifying tissue samples), to telecommunication networks (e.g., classifying customers based on their calling behavior) and social networks (e.g., classifying users based on their feeds on Twitter, Facebook, etc.).

The graph classification problem can be stated as follows: There is a dataset of graphs $G_i \in \mathcal{D}$, with $i = 1, \ldots, N$. Each graph $G_i = (V_i, E_i)$ is given as a collection of vertices $V_i = \{v_{i1}, \ldots, v_{in}\}$ and edges $E_i = \{(v_a, v_b) | v_a, v_b \in V_i\}$. The graph G_i may have labels on the nodes and edges, drawn from some common set of labels Σ for the entire dataset $\mathcal{D}$. Finally, each graph G_i has a corresponding class $y_i \in C$, where C is the set of k categorical class labels, given as $C = \{1, \ldots, k\}$. The goal of graph classification is to learn a model $f : \mathcal{D} \rightarrow C$ that predicts the class label for any graph. Typically the model is learned from a *training set* of graphs with known class labels. The model is then evaluated on a *testing set* of graphs. The accuracy of the classification model can be tested by comparing the predicted output $\hat{y}_i = f(G_i)$ with the true class label y_i (provided it is known).

The main challenge in classifying graphs is how to convert the discrete graph objects into numeric features or similarities for effective classification. Graph kernel methods have attracted a lot of attention due to their ability to represent the graph data as a $N \times N$ symmetric, positive semi-definite *kernel matrix* $\mathbf{K} = \{\kappa(G_i, G_j)\}_{i,j=1}^{N}$ that records the pairwise similarities between graphs in $\mathcal{D}$. Conceptually, the kernel function $\kappa(G_i, G_j)$ represents an inner-product between the vectors corresponding to the two graphs G_i and G_j in some N-dimensional *feature space*; see [23] for more details on kernel methods. Once the kernel matrix has been constructed, it is possible to classify the graphs with a Support Vector Machine (SVM) [27], using the supplied kernel matrix $\mathbf{K}$. There has been a lot of research activity in trying to develop more effective and efficient graph kernel functions κ. These methods can broadly be classified into methods based on random walks [10, 15], shortest paths [4], cycles [12] subtrees [22, 21, 24], and subgraphs [25, 17, 26]. Despite the research above, it is fair to say that efficient and effective graph classification still remains a challenge, especially for large graphs.

In this paper we propose an alternative approach to constructing a feature-vector for graph classification. Instead of relying on "patterns" like path, cycles, subtrees and subgraphs, we compute several global topological and label attributes from each graph $G_i \in \mathcal{D}$. The values for these attributes yield a numeric feature-vector $F_i = (f_{i1}, \ldots, f_{ip})$. The set of feature vectors F_i and the corresponding class labels y_i are then used to construct an SVM classifier. We show that our approach is both effective and scalable compared to state-of-the-art graph kernel methods. We conduct an extensive set of experiments over several real graphs, representing chemical compounds, proteins, and cell-graph datasets. We demonstrate that our approach yields better or competitive accuracy in a fraction of the time taken by other kernels. Our method is particularly effective in classifying large unlabeled graphs, since it is able to effectively capture the structural differences among the classes.

2. RELATED WORK

Graph kernels compute the similarity between pairs of graphs in $\mathcal{D}$, based on the common patterns they share. The patterns can range from the simple to the complex. Specif-

MLG '11 San Diego, CA, USA

ically the kernels are designed to exploit random walks [10, 15, 5, 28], shortest paths [4], cyclic patterns [12], subtrees [22, 21, 19, 24] and subgraphs [25, 17, 26]. Another class of graph kernels, e.g., the diffusion kernel [16], deal with the similarity between nodes of a single graph. However, our focus in this paper is on kernels between different graphs, which we discuss in more detail below.

Random Walk Kernels: The similarity of two graphs $G_i, G_j \in \mathcal{D}$ can be quantified by counting labeled walks that are common to both of them. The random walk kernel [10], one of the first graph kernels, is based on this idea. The kernels in [15, 5] are also based on random walks over labeled graphs. Computing the pair-wise kernel values has worst case $O(n^6)$ complexity, where n denotes the number of nodes in G_i and G_j. A more efficient version of the random walk kernel was proposed in [28], reducing the complexity to $O(n^3)$ per pair of graphs. One potential problem with these kernels is that artificially high kernel values may be obtained by repeatedly visiting same nodes and edges multiple times [18]. We refer to [29] for a recent overview of random walk based graph kernels.

Shortest Path Kernels: The shortest-path graph kernel [4] first computes the shortest-path graph $S = (V_S, E_S)$ for each graph $G = (V, E) \in \mathcal{D}$. Here $V_S = V$, and a weighted edge (v_a, v_b) exists in E_S if v_a and v_b are connected by a path in G, with the edge weight representing the shortest path length between v_a and v_b (infinity if they are not reachable). Given the shortest-path graphs S_i and S_j for two input graph G_i and G_j the kernel is defined as the sum over all pairs of edges from S_i and S_j, using any suitable positive definite kernel on the edges. The all-pairs shortest-path graphs can be computed in $O(n^3)$ time, and the kernel can then be computed in $O(n^4)$ time, since S_i and S_j each have $O(n^2)$ edges. Other variants of the shortest path kernel include equal length shortest paths, k shortest paths, k shortest disjoint paths, and so on [4].

Cyclic Pattern Kernels: The cyclic pattern kernel [12] is based on counting the number of common cycles that occur in both graphs. Since there is no known polynomial time algorithm to find all the cycles in a graph, sampling and time-bounded enumeration of cycles are used to measure the similarity of the graphs.

Subtree Kernels: Subtree kernels are based on common subtrees in the graphs [22]. The main idea is to consider pairs of nodes from G_i and G_j and see if they share common tree-like neighborhoods, i.e., to count the pairs of identical subtrees of height h rooted at vertex $v_a \in G_i$ and $v_b \in G_j$. The kernel is defined as the sum over all pairs of vertices of a suitably defined vertex pair kernel. The complexity of this approach is $O(n^2 h 4^d)$, where d denotes the maximum degree. Another subtree kernel was proposed in [21], based on a path representation of the trees obtained via a depth-first search on the input graphs. The kernel function is computed on these paths (e.g., the ratio of the longest common path).

The recently proposed Weisfeiler-Lehman Kernel [24], is a fast subtree kernel that scales up to large, labeled graphs. It uses the Weisfeiler-Lehman isomorphism test, which uses iterative multiset-label determination, label compression, and relabeling steps. The isomorphism test terminates after a pre-specified number of iterations h. If the sets of labels for nodes are not identical, then two graphs are considered as non-isomorphic, otherwise, they are isomorphic. The WL graph kernel counts the matching multiset labels for the two graphs G_i and G_j in each iteration of the WL isomorphism test. The WL kernel has $O(mh)$ complexity, where m is the number of edges in the graphs.

Graphlet and Subgraph Kernels: Similar graphs should have similar subgraphs. Graphlet kernels measure the similarity of two graphs by the dot product of count vectors of all possible connected subgraphs of order k (i.e., the graphlets, also called as k-minors) [25, 17]. For any k (usually set to 3, 4, or 5), there are $2^{\binom{k}{2}}$ possible graphlets of size k, but many of them are isomorphic. Usually, to avoid the dependence on the size, the count vector is normalized into a probability vector, and the graphlet kernel is re-defined as the dot product of the normalized count vectors for two graphs. Exhaustive enumeration of all graphlets has complexity $O(n^k)$. For a graph with bounded degree d, the connected graphlets can be enumerated in $O(nd^{(k-1)})$ [25].

Frequent subgraph mining can also be used to define a kernel between two graphs [26]. Let $\mathcal{F} = \{s_1, \ldots, s_p\}$ denote the set of p frequent and discriminative subgraph patterns mined from $\mathcal{D}$. Each graph $G_i \in \mathcal{D}$ is then represented as a binary feature vector $\{0,1\}^p$ where feature j is set to 1 if and only if s_j is isomorphic to a subgraph in G_i. The kernel between G_i and G_j can be defined over their binary feature vectors. CORK [26] implements this approach; it uses gSpan [31] to mine the subgraphs, and selects near-optimal features (subgraphs) from that set, that are most discriminative for classification.

In our experiments in Section 4, we compare with the following graph kernel methods: fast geometric Random-walk (RW) kernel [28], Shortest-path (SP) kernel [4], Graphlet (GK) kernel [25], Ramon-Gärtner (RG) subtree kernel [22], and Weisfeiler-Lehman (WL) subtree kernel [24]. We also compare with CORK [26].

3. GRAPH ATTRIBUTES FOR CLASSIFICATION

As we have seen above, while many sophisticated graph kernels have been proposed, efficiency and scalability remain as challenges, for large graph datasets. Our basic idea is to compute several topological and label attributes for each graph in the dataset, and to use the derived feature-vector attributes for classification. Like most of the graph kernel work, we use a Support Vector Machine (SVM) as the classifier of choice. The graph attributes we use are listed below.

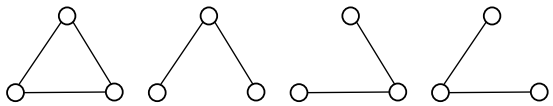

Figure 1: A triangle with its three triples

1. **Average degree**: The degree of a node is defined as the number of its neighboring edges. Average degree is the average value of the degree of all nodes in the graph, i.e., $\bar{d}(G) = \sum_i^n d(u_i)/n$, where $d(u_i)$ denotes the degree of node u_i.
2. **Average clustering coefficient**: For a node u, the clustering coefficient $c(u)$ represents the likelihood that any two neighbors of u are connected. More formally, the clustering coefficient of a node u is defined as:

$c(u) = \frac{\lambda(u)}{\tau(u)}$, where $\lambda(u)$ is the number of triangles (complete graph with three nodes) of a node u and $\tau(u) = \frac{d(u)^2 - d(u)}{2}$, the number of triples a node u has. Figure 1 shows a triangle and its three triples. Alternatively, the clustering coefficient for node u can be defined as the ratio of the number of actual edges between the neighbors of u to the number of possible edges between them. The clustering coefficient $C(G)$ of a graph is the average of $c(u)$ taken over all the nodes in the graph, i.e., $C(G) = \frac{1}{n}\sum_{i=1}^{n} c(u_i)$. Here we use $C(G)$ as one of our global graph features. Generally, average clustering coefficient is a very popular metric in network analysis, but in some specific graph datasets, such as chemical compounds, there do not exist many triangles in any graph instance, which results in the clustering coefficient taking on value close to 0.

3. **Average effective eccentricity**: The eccentricity of a node u is defined as $e(u) = max\{d(u,v) : v \in V\}$, where the distance $d(u,v)$ is the length of the shortest path from u to v. For *effective eccentricity* we take the maximum length of the shortest path from u, so that u can reach at least 90 percent of nodes in the graph. Effectiveness is a more robust measure if we take noise into consideration. The average effective eccentricity is the average of effective eccentricities of all nodes in the graph.
4. **Maximum effective eccentricity (effective diameter)**: Maximum effective eccentricity is defined as the maximum value of effective eccentricity over all nodes in the graph. Note that the maximum eccentricity is the graph diameter, i.e., $diam(G) = max\{e(u)|u \in V\} = max\{d(u,v)|u,v \in V\}$. Maximum effective eccentricity is thus the same as effective diameter.
5. **Minimum effective eccentricity (effective radius)**: Minimum effective eccentricity is defined as the minimum value of effective eccentricity over all nodes in the graph. Note that minimum eccentricity is called the graph radius, i.e., $rad(G) = min\{e(u)|u \in V\}$, thus minimum effective eccentricity is the effective radius.
6. **Average path length (closeness centrality)**: The closeness centrality of a node u is defined as the reciprocal of the averaged total path length between node u and every other node that is reachable from node u, where $u \in V$, i.e., $close(u) = \frac{n-1}{\sum_{v \in V, v \neq u} d(u,v)}$. We take the average of closeness centrality of all nodes as a global metric for a graph.
7. **Percentage of central points**: We define a point u to be a central point if it has an eccentricity equal to the effective radius of the graph, i.e., it satisfies: $\{u \in V : \textit{effective-rad}(G) = e(u)\}$. The ratio of the number of central points to the total number of points in the graph is selected as a feature.
8. **Giant connected ratio**: A giant component is a subgraph that is connected and has the maximum number of nodes. We take the ratio of the number of nodes of the giant connected component to the total number of nodes in the entire graph as a global metric for a graph. Note that if the entire graph is connected, the ratio is 1, thus in those datasets that are comprised of connected graphs, this attribute will not be a good graph descriptor. However, not all graphs in our experimental study are connected, thus this ratio may be a meaningful attribute to use.
9. **Percentage of isolated points**: We define a isolated point in a graph to be a node with degree zero. The ratio of isolated points to the total number of nodes in the entire graph is considered as a feature. For graphs that are connected, this feature will not be meaningful, but there are datasets we used in our study that do have isolated points.
10. **Percentage of end points**: A node which has a degree of one is defined as an end point. The ratio of the number of end points to the total number of nodes in the entire graph is selected as a feature.
11. **Number of nodes**: Total number of nodes in the graph.
12. **Number of edges**: Total number of edges in the graph.
13. **Spectral radius**: The spectral radius is defined as the largest magnitude eigenvalue of the adjacency matrix of the graph. More formally, let $|\lambda_1| > |\lambda_2| > \ldots > |\lambda_s|$ be the distinct eigenvalues of the adjacency matrix A of the graph, sorted by their magnitude. The spectral radius of the graph, $\rho(G)$, is defined as: $\rho(G) = |\lambda_1|$.
14. **Second largest eigenvalue**: The value of the second largest eigenvalue of the adjacency matrix A, i.e., $|\lambda_2|$.
15. **Trace**: Sum of the eigenvalues of the adjacency matrix, i.e., $\sum_i^n \lambda_i$. This is in fact equivalent to the trace of the adjacency matrix A, i.e., $Tr(A) = \sum_{i=1}^{n} a_{ii}$. This feature is useful only if the graph has several loops, i.e., an edge joining a vertex to itself. For a simple graph, which is loop-free, the trace equals 0 because the elements on the main diagonal of A are all zeros.
16. **Energy**: Squared sum of the eigenvalues of the adjacency matrix A. More formally, the energy of a graph G is: $E(G) = \sum_i^n \lambda_i^2$.
17. **Number of eigenvalues**: Number of distinct eigenvalues, $s \leq n$, of the adjacency matrix A of the graph. The adjacency matrix A of an undirected graph has n eigenvalues, however, they are not necessarily distinct.
18. **Label Entropy**: We employ label entropy to measure the uncertainty of labels. Suppose a graph G has q different labels: l_1,..., l_q, then the label entropy is given as: $H(G) = -\sum_{i=1}^{q} p(l_i) \log p(l_i)$.
19. **Neighborhood Impurity**: We define the impurity degree of a node u as:

$$ImpurityDeg(u) = |L(v) : v \in N(u), L(u) \neq L(v)|$$

where $L(u)$ is the label, and $N(u)$ is the neighborhood of (the nodes adjacent to) node u. If all nodes in the neighborhood of u have the same node label, the impurity degree is 0. For the whole graph, we are only interested in the nodes that have impurity degree larger than 0, i.e., the nodes which have at least one neighbor whose label is different. The neighborhood impurity of a graph G is defined as the average impurity degree over nodes with positive impurity.
20. **Link Impurity**: An edge (u,v) is defined to be impure if $L(u) \neq L(v)$. The link impurity of a graph G is defined as: $\frac{|(u,v) \in E : L(u) \neq L(v)|}{m}$, where m is the number of total edges in graph G.

F	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	f_{16}	f_{17}	f_{18}	f_{19}	f_{20}
V	2.10	0.00	5.75	8	4	0.29	0.15	1.00	0.00	0.45	20	21	2.56	2.15	0.0	42.00	20	1.09	1.11	0.48

Table 1: Global graph feature vector for the example. F: Feature, V: Value.

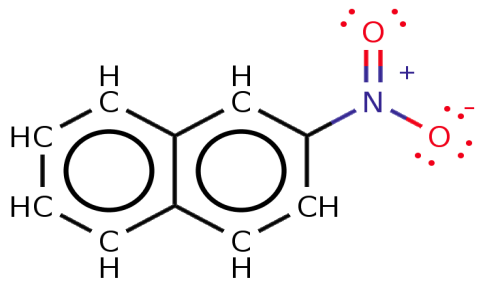

Figure 2: A chemical compound from PTC dataset (with implicit hydrogens)

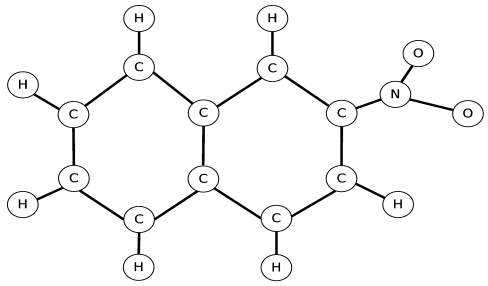

Figure 3: The graph representation for Figure 2. Labels on node/atoms: O:Oxygen, H:Hydrogens, N:Nitrogen, C:Carbon. Labels on edges/bonds: A:Aromatic, S:Single, D:Double.

Example: Figure 2 illustrates an example from PTC chemical compound dataset (see Section 4.2 for description). Figure 3 is a graph representation for the molecule in Figure 2. Nodes/edges are assigned a label based on their types, properties, etc. Table 1 gives the graph feature vector based on the 20 global graph attributes that are listed in the order given above. For instance, there are 20 nodes in the graph, with 9 nodes having degree 1, and 11 nodes having degree 3. The average degree (f_1) is therefore $\bar{d} = \frac{9\times1+11\times3}{20} = \frac{42}{20} = 2.1$. Since there are no triangles in the graph the clustering coefficient (f_2) is 0. As another example, the graph has $m = 21$ edges, but there are 11 impure links ($L(u) \neq L(v)$), thus the link impurity is given as $f_{20} = 10/21 = 0.48$. The other features can be computed based on their definition. □

Graph Classification: In computing the feature values, if a certain graph in the dataset is disconnected and contains several components, we compute the average value for a given graph metric over all the components. Each graph G_i in the database is finally represented by its corresponding feature vector F_i over the 20 topological and label attributes. However, using the raw or unnormalized feature values does not perform well. This is mainly because the original feature have different range of values (see Table 1 for example), which would give more importance to features with larger values than those with smaller values. Instead we normalize the feature values via range and z-normalization.

In range normalization each value x of a feature f_i is transformed into $r(x) = \frac{x-\min}{\max-\min}$, where min and max denote the minimum and maximum value for f_i. In z-normalization x is replaced by $z\text{-}score(x) = \frac{x-\mu}{\sigma}$, where μ and σ are the mean and standard deviation for f_i. By normalizing the values all features are considered on equal footing, which helps improve the classification accuracy. Once each graph G_i in the dataset $\mathcal{D}$ has been transformed into its corresponding normalized feature-vector of length 20, $F_i = (f_{i1}, \ldots, f_{i20})$, we use a SVM classifier over the new feature-vector dataset, using the Gaussian or radial basis (RBF) kernel (we tried a linear kernel too, but RBF gave better results).

Computational Complexity: The various graph attributes range from the simple to the complex, with higher computational times for the more complex features. In the analysis below, we use n to denote the number of nodes $|V|$ (also called the graph order), and m to denote the number of edges $|E|$ (also called the graph size).

For each graph in the database, the number of nodes (f_{11}) and edges (f_{12}) are already known, so their cost is $O(1)$. If they are not known before-hand, we can compute them in one pass over the entire graph in time $O(n+m)$. The degree based attributes like the percentage of isolated or end nodes (f_9, f_{10}), and average degree (f_1) can be computed in time linear in the graph order and size, i.e., in $O(n+m)$ time. The giant connected component (f_8) can also be found via breadth-/depth-first search in $O(n+m)$ time.

The clustering coefficient (f_2) can be computed in time $O(nd_{\max}^2)$, where $d_{\max}$ is the maximum degree for the graph. A better approximation is to use the average degree $d = \frac{2m}{n}$. To compute the clustering coefficient for each node takes on average $O(d^2) = \frac{2m^2}{n^2}$. The time to compute the average clustering coefficient over all nodes is then $O(\frac{m^2}{n})$.

The eccentricity based attributes (f_3, f_4, f_5, f_7) and the average path length (f_6) can be easily computed from the all-pairs shortest path matrix. The all-pairs matrix can be computed in time $O(n^2 + nm)$ via n calls of single-source shortest paths, each of which can be computed in breadth-/depth-first search in time $O(n+m)$, since we assume that each edge has weight one. From the shortest path matrix, the attributes can be computed in $O(n^2)$ time.

The spectral attributes (f_{13} to f_{17}) depend on the eigen-decomposition of G, which can be computed in $O(n^3)$ time in the worst case. However, typically real-world graphs are sparse, which can be exploited to reduce the complexity to $O(n^2)$ [20]. Also note that when the input graphs are very large, we compute only the top $k \geq 2$ eigenvalues. For sparse graphs, the top k eigenvalues can be computed in $O(mkt+nk^2t+k^3t)$ time (e.g., using the Implicitly Restarted Lanczos Method [1]), where t is the number of iterations until convergence. For sparse graphs, with $m = O(n)$, if $k \ll n$, the time reduces to $O(nt)$. The trace (f_{15}) and energy (f_{16}) are computed only over these k eigenvalues. The number of eigenvalues (f_{17}) is not very informative in this case.

Finally, label entropy (f_{18}) can be computed in $O(n)$, neighborhood impurity (f_{19}) can be computed in $O(nd_{\max})$ and link impurity (f_{20}) can be computed in $O(n+m)$ time.

4. EXPERIMENTS

4.1 Experimental Setup

We compare our graph feature based classification approach with state-of-the-art graph kernel classifiers. More specifically, we compare with the following graph kernel methods: fast geometric Random-walk (RW) kernel [28], Shortest-path (SP) kernel [4], Graphlet (GK) kernel [25], Ramon-Gärtner (RG) subtree kernel [22], and Weisfeiler-Lehman (WL) subtree kernel [24]. We relied on a Matlab implementation of all of these kernels[1]. As suggested in [24], we used the tuned parameter settings for each of the kernels as follows. For RW, the decay weight is chosen in the range $\lambda \in \{10^{-6}, \ldots, 10^{-2}\}$. For RG we set $\lambda_r = \lambda_s = 1$. For the WL kernel, we choose $h = \{1, \ldots, 10\}$, which means that 10 different kernel matrices are computed. For SP, we use equal length shortest paths, and for GK we use connected 3-minors. We also compare with the subgraph-feature based approach of CORK [26][2], which is implemented in C++. For CORK, we use 10% minimum support to mine the frequent discriminative subgraphs.

In the discussion below, our graph feature approach is denoted as GF. GF was written in Python, with NumPy [14] and Networkx [11] modules for linear algebra and graph support. Note that both NumPy and Matlab use low-level C implementations for most matrix operations, therefore, the timing results are comparable[3], though there might be slight differences. We present results for three variants of the GF approach: GF(no) denotes the method using a raw or unnormalized feature vector, whereas GF(r) and GF(z) use range and z-score normalized feature vectors, respectively.

We use the libsvm [6] (Support Vector Machine library) for all of the kernels and our method. The graph kernels use the kernel matrix computed via the particular graph kernel, whereas we use the default Gaussian or radial basis (RBF) kernel in libsvm: $\kappa(G_i, G_j) = \exp\left\{-\gamma \|F_i - F_j\|^2\right\}$, where $\gamma = \frac{1}{p}$, where p equals the number of features, which is 20 for all datasets, except the cell-graphs and CATH (w/o L). For these latter two datasets, the number of features is $n' = 17$, since they do not have any labels on the nodes or edges, and thus we do not use the label-based features. We also use a RBF kernel for CORK, since that gave us the best results.

For each method, we perform 10 runs of 10-fold cross-validation, and we tune the C parameter, for C-SVM, using only the training folds. Note that all times reported below are only for the graph kernel matrix computation (for other methods), or for the graph feature computation (for GF). The time for SVM is not included, since all methods use the same libsvm package, albeit with a different kernel. The SVM training and testing times are mostly comparable. All the experiments were performed on MAC OS X 10.5 with two 2.66GHz Dual Core Intel Xeon processors, with 4GB 667MHz DDR2 memory.

For performance assessment, we report the average accuracy and standard deviation over the 10-fold cross-validation run 10 times. We also assess whether the accuracy of our method is significantly better or worse than the accuracy of the best previous method. For this we use the paired t-test for the difference between the accuracies in each fold. We report the value of the t-statistic, given as: $t = \frac{\sqrt{N}\mu}{\sigma}$, where μ and σ^2 denote the mean and variance of the difference in accuracy between the two methods, and $N = 100$ is the total sample size (10 runs times 10 folds). We fail to reject the null hypothesis, that there is no significant difference, if $t \in (-t_{\alpha/2,N-1}, t_{\alpha/2,N-1})$, where α is the significance level for a two-tailed t-test with $N - 1$ degrees of freedom. We use $\alpha = 0.05$ for which the interval is $(-1.98, 1.98)$. If the value of t-statistic is outside this interval, the two methods are statistically significantly different in performance.

4.2 Datasets

We used three different types of graph datasets: chemical compounds, proteins, and cell graphs. See Table 2 for statistics on the different graphs.

Chemical Compounds: The chemical compound datasets include MUTAG [7], NCI1 and NCI109 [30], and PTC[4], which have been employed as benchmark datasets in previous graph kernel papers. MUTAG is a dataset of mutagenic aromatic and heteroaromatic nitro compounds assayed for mutagenicity on bacterium *Salmonella typhimurium*. We used two balanced subsets of the NCI (National Cancer Institute) datasets. The class labels are based on an anti-cancer screen, as active or inactive. The PTC (The Predictive Toxicology Challenge) datasets record the carcinogenicity of several hundred chemical compounds for Male Rats (MR), Female Rats (FR), Male Mice (MM) and Female Mice (FM). As one can in Table 2(a), these graphs are very small (20-30 nodes, and 25-40 edges) and sparse, with average degree around 2.

Proteins: The D&D dataset [9], which has also been used by previous studies, consists of 1178 proteins, with 691 enzymes and 487 non-enzymes. In addition, we created two new datasets from CATH[5], a manually curated database of protein domain structures. CATH1 consists of proteins in the same class (Mixed Alpha-Beta), but having different architectures (Alpha-Beta Barrel vs. 2-layer Sandwich). CATH2 has proteins in the same class (Mixed Alpha-Beta), architecture (Alpha-Beta Barrel), and topology (TIM Barrel), but in different homology classes (Aldolase vs. Glycosidases). CATH2 is harder to classify, since proteins in the same topology class are structurally similar. The protein graphs are 10 times larger in size than chemical compounds, with 200-300 nodes and 700-1250 edges (Table 2(b)), and average degree is 5-8. We use another variant of the CATH datasets, without the node labels (which correspond to the amino acids). We denote these as CATH1 (w/o L) and CATH2 (w/o L).

Cell-graphs: We also performed experiments on cancer Cell-Graph datasets [2, 3, 8]. These graphs were constructed from the histo-pathological samples from three different types of tissues: breast, bone and brain. Within each type of tissue we consider three classes: healthy (Normal/Benign), cancerous (Invasive, Osteo-sarcoma:Ost, Glioma), and damaged (Non-invasive, Fracture:frac, Inflammation). We perform binary classification for each pair of classes within a tissue type. Usually, it is much easier to distinguish between normal and cancerous classes, but harder to classify cancerous versus damaged classes, for each tissue type. For

[1] obtained from Prof. K. Borgwardt and N. Shervashide

[2] obtained from Marisa Thoma

[3] For performance comparison of NumPy and Matlab, see `http://www.scipy.org/PerformancePython` for instance.

[4] `www.predictive-toxicology.org/ptc`

[5] `www.cathdb.info`

dataset	size (N)	classes	positive	negative	avg. $\|V\|$	avg.$\|E\|$	Max. $\|V\|$	Max. $\|E\|$	avg. deg
MUTAG	188	2	125	63	17.7	38.9	28	33	2.19
NCI1	4110	2	2057	2053	29.9	32.3	111	119	2.16
NCI109	4127	2	2079	2048	29.7	32.1	111	119	2.16
PTC(MM)	336	2	129	207	25.0	25.4	109	108	1.98
PTC(FM)	349	2	143	206	25.2	25.6	109	108	1.99
PTC(MR)	344	2	152	192	25.6	26.0	109	108	1.99
PTC(FR)	351	2	121	230	26.1	26.5	109	108	1.99

(a) Chemical Compound Datasets

class	size (N)	classes	positive	negative	avg. $\|V\|$	avg.$\|E\|$	Max. $\|V\|$	Max. $\|E\|$	avg. deg
D & D	1178	2	691	487	284.3	715.7	5748	14267	4.98
CATH1	712	2	384	328	205.7	819.8	568	2356	7.79
CATH2	190	2	109	81	308.0	1254.8	568	2220	8.14

(b) Protein Datasets

tissue	class	size (N)	avg. $\|V\|$	avg.$\|E\|$	Max. $\|V\|$	Max. $\|E\|$	avg. deg
Breast	Invasive (EI)	202	966.9	12503.6	1956	51454	22.07
	Non-invasive (EN)	93	889.7	13459.4	1940	48750	25.47
	Benign (EB)	151	829.4	15677.7	1885	49165	34.72
Bone	Ost(OO)	49	532.2	2324.7	2855	18790	5.42
	Frac(OF)	39	497.7	1599.2	1913	13564	4.25
	Normal(ON)	20	174.2	1174.0	612	7309	8.14
Brain	Glioma(AG)	329	4550.2	43400.5	7311	98572	18.04
	Inflamation(AI)	107	4244.1	39457.7	7113	90029	17.06
	Benign(AB)	210	789.0	3988.9	1755	9309	9.65

(c) Cell-Graph Datasets: E: Breast; O: Bone; A: Brain

Table 2: **Benchmark Datasets**

example, for the breast tissue, classifying EB vs. EN and EB vs. EI is easier than classifying EI vs. EB. In contrast to the chemical compounds and proteins, the cell graphs are even larger and denser (see Table 2(c)). Average graph size is 5-10 times larger than proteins, with 200-4500 nodes and 100-43000 edges. Average degree is 4-8 for bone, 10-18 for brain, and 22-35 for breast tissue. The cell-graphs are unlabeled (i.e., no labels on nodes or edges).

4.3 Graph Kernel Comparison

Chemical Compound Datasets: Table 3 shows the accuracy comparison for our GF approach versus other graph kernels on the chemical compound datasets. Each cell records the average classification accuracy, as well as the standard deviation, over the 10 fold cross-validation over 10 different runs. Table 4 reports the wall clock running times for each method on the different datasets. A '–' in any cell means that the computation of the kernel matrix did not finish in one day, and thus the run was aborted.

We can observe from the results that graph features with unnormalized/raw values, denoted GF(no), do not perform well in terms of accuracy. Furthermore, for these graphs, the z-normalized GF(z) approach delivers the best results, except for PTC(FM) and PTC(FR).

On the MUTAG dataset, GF(z), is the best overall method. Looking at the t-statistic, it is significantly better than the next best method, SP kernel, at a significance level of $\alpha = 0.05$, since $t \notin (-1.98, 1.98)$. Considering the time, we can see that GF takes only a fraction of the time compared to other methods. It is 6 times faster than SP.

On the NCI1 and NCI109 datasets, the WL subtree kernel has the best accuracy. The NCI datasets are quite tree-like; we can see in Table 2 that for both the average and maximum number of edges, we have $|E| \approx |V|$. Given that the WL kernel is a subtree kernel, it is well suited for such tree-like datasets. However, in terms of time, GF is over 25 times faster.

The PTC datasets are also tree-like, and thus the WL kernel performs well. However, while WL is the best method for MM, GF(z) is the best method for MR. These differences are statistically significant. On FM, the WL kernel has a slight advantage, whereas on FR data, GF(r) is slightly better, though there is no significant difference between them. In terms of computational time, GF method is vastly superior, being 6 times faster than WL. In terms of time, GK is the second fastest method, but its accuracy is not very high. Compared to RW, SP, RG, and CORK, our GF approach is one to three orders of magnitude faster.

Protein Datasets: The protein graphs are much larger compared to the chemical compound datasets. The accuracy and timing results are shown in Tables 5 and 6.

In terms of accuracy, among the GF variants, the unnormalized version is the worst, whereas GF(r) gives the best results, except on CATH2(w/o L).

For the D&D enzyme dataset, the only kernel methods that finished within 24 hours, were GF, GK, WL and CORK. Here WL has the best accuracy, but GF is about 3 times faster. Even though CORK is 4 times faster, since it uses C++ and GF uses python, the timing comparison is not entirely fair.

On the CATH datasets (with labels), RW, and RG were not able to finish in the alloted time (1 day). For CATH1, we find that GF(r) has a slight (though not significant) advantage over other approaches in terms of accuracy. CATH1 is an easier dataset to classify, since the proteins in the two classes differ at the architecture level, and thus are structurally different. All the methods do well on this dataset. On CATH2 data, CORK gave the best overall results, and GF was significantly worse. CATH2 is a much harder dataset to classify from a structural viewpoint. This is because the two classes have significant structural similarity. On the other hand, there are possibly significant differences in the protein sequences between the two classes. GF mainly relies

	MUTAG	NCI1	NCI109	PTC(MM)	PTC(FM)	PTC(MR)	PTC(FR)
GF(no.)	86.75 ± 6.38	67.18 ± 2.99	66.30 ± 1.67	60.99 ± 4.05	55.90 ± 6.40	51.15 ± 5.87	58.12 ± 6.52
GF(r)	87.11 ± 8.59	69.64 ± 1.69	69.44 ± 2.26	62.78 ± 6.83	63.90 ± 3.97	57.52 ± 6.63	**66.71 ± 6.47**
GF(z)	**91.37 ± 4.77**	75.62 ± 2.05	74.22 ± 1.66	63.38 ± 5.43	59.87 ± 6.13	**63.94 ± 7.05**	62.96 ± 6.65
RW	84.01 ± 6.61	–	–	60.58 ± 8.92	58.98 ± 9.72	51.40 ± 5.77	64.63 ± 8.74
SP	88.13 ± 7.15	73.82 ± 1.61	72.89 ± 2.17	57.52 ± 9.98	52.41 ± 9.79	58.46 ± 6.08	63.67 ± 5.27
GK	83.93 ± 6.48	69.18 ± 2.62	69.82 ± 1.89	58.04 ± 8.22	55.86 ± 8.95	55.19 ± 5.66	59.41 ± 5.36
RG	86.23 ± 4.41	–	–	64.30 ± 7.89	58.45 ± 7.01	57.61 ± 8.32	63.52 ±6.40
WL	86.89 ± 6.33	**84.11 ± 1.61**	**83.50 ± 2.34**	**67.23 ± 5.87**	**64.37 ± 6.57**	58.10 ± 7.18	65.22 ± 5.34
CORK	86.19 ± 7.82	78.12 ± 1.61	77.76 ± 1.48	61.85 ± 8.04	57.90 ± 6.53	60.75 ± 7.31	65.51 ± 9.82
t-statistic	**4.02**	**-28.61**	**-29.27**	**-3.91**	-0.88	**3.26**	1.40

Table 3: Accuracy (± Standard Deviation): Chemical Compound Datasets. (bold t-statistic means statistically significant.)

	MUTAG	NCI1	NCI109	PTC(MM)	PTC(FM)	PTC(MR)	PTC(FR)
GF	**0.78s**	**36.48s**	**36.77s**	**2.30s**	**2.56s**	**2.66s**	**2.50s**
RW	5m3s	–	–	2h3m42s	2h16m11s	2h12m7s	2h17m28s
SP	4.61s	16m56s	21m2s	35.82s	35.79s	36.14s	37.72s
GK	1.42s	3m21s	3m25s	4.88s	5.05s	5.04s	5.22s
RG	42m54s	–	–	2h11m1s	2h16m54s	2h14m17s	2h20m6s
WL	5.88s	15m30s	16m1s	14.86s	16.22s	15.51s	16.10s
CORK	1m1s	33m19s	35m44s	19.92s	23.90s	23.03s	27.04s

Table 4: Running Times on Chemical Compound Datasets (h:hours, m:minutes, s:seconds)

	D&D	CATH1	CATH2	CATH1(w/o L)	CATH2(w/o L)
GF(no)	62.99 ± 4.49	83.29 ± 4.98	61.58 ± 10.27	82.86 ± 5.43	61.05 ± 8.87
GF(r)	76.32 ± 2.72	**99.02 ± 0.90**	81.57 ± 5.39	**98.60 ± 1.54**	77.89± 7.74
GF(z)	75.95 ± 2.66	98.46 ± 1.32	79.27 ± 9.46	97.90 ± 1.57	**81.05 ± 3.49**
RW	–	–	–	–	–
SP	–	98.88 ± 1.37	96.32 ± 3.37	97.89 ± 1.70	76.32 ± 8.57
GK	75.13 ± 2.71	98.32 ± 0.84	94.74 ± 4.71	97.61 ± 2.52	66.84 ± 11.05
RG	–	–	–	–	–
WL	**78.29 ± 3.05**	98.59 ± 1.09	94.21 ± 4.97	98.59 ± 1.26	76.84 ± 8.22
CORK	71.22 ± 4.56	94.24 ± 2.77	**97.89 ± 2.58**	–	–
t-statistic	**-3.75**	0.07	**-26.75**	0.15	**4.67**

Table 5: Accuracy (± Standard Deviation): Protein Datasets. (bold t-statistic means statistically significant.)

	D&D	CATH1	CATH2	CATH1(w/o L)	CATH2(w/o L)
GF	52m35s	**4m36s**	**2m15s**	**4m33s**	**2m15s**
RW	–	–	–	–	–
SP	–	1h42m12s	42m26s	1h2m9s	30m32s
GK	23h14m53s	37m8s	15m54s	18m37s	7m49s
RG	–	–	–	–	–
WL	2h12m57s	22m33s	4m45s	20m20s	4m19s
CORK	**14m10s**	41m28s	43m6s	–	–

Table 6: Running Times on Protein Datasets (h:hours, m:minutes, s:seconds)

on topological graph features, and the three label features (f_{18} to f_{20}) are not able to capture the subsequence similarity (since they are local and are not designed to look at subsequences). GF is thus not able to distinguish between the two classes as well as the other kernels that can effectively utilize label information.

To verify this hypothesis, we removed the node (amino acid) labels from the proteins. On CATH1(w/o L) and CATH2(w/o L), all methods have to rely only on topological information. CORK aborted on the unlabeled CATH data, since the (sub)-graph isomorphism checks in this case become too expensive. We now find that GF(z) is significantly superior to other methods on CATH2(w/o L). While the accuracy of GF(z) remains close to the labeled case, the other methods suffer a significant drop in accuracy. For example, WL has an accuracy of 94.21 on CATH2, but only 76.85 on CATH2(w/o L). This confirms two things: i) there is significant sequence similarity between sequences in the same class, exploiting which helps the other methods, and ii) even though the classes are topologically similar, there are enough structural differences that GF is still able to exploit.

In terms of time, we can see that GF is 2-5 times faster than WL, about 20 times faster than SP, 4-8 times faster than GK, and 10-20 times faster than CORK.

Cell-Graph Datasets: Table 7 shows the accuracy comparison for GF versus other graph kernels on the cell-graph datasets. The corresponding timing results are shown in Table 8. One of the differences between cell-graph and the other datasets is that cell-graphs are much larger (e.g., 4550 nodes and 43400 edges for brain graphs). Furthermore, while the other datasets are labeled, the cell-graph data does not have any labels.

We show the results separately for each tissue type, and we show results for binary classification. Among the GF variants, GF(z) is usually better than GF(r), or is close to it. The unnormalized version is significantly worse. For GF, we use only the top $k = 2$ eigenvalues for Bone(O), and $k = 100$ for Brain(A). This is because computing all the eigenvalues for these large graphs is expensive.

For the cell-graph datasets, GF significantly outperforms all other methods in terms of accuracy, and also has an advantage in terms of time. For the largest graphs, from brain

	EB vs EI	EN vs EB	EN vs EI	OF vs ON	ON vs OO	OF vs OO
GF(no)	57.78 ± 7.25	61.92 ± 5.21	64.48 ± 4.96	65.67 ± 20.76	71.19 ± 14.06	58.83 ± 16.59
GF(r)	87.70 ± 5.30	**86.35 ± 7.02**	82.76 ± 5.35	**98.33 ± 5.00**	**94.29 ± 7.00**	53.47 ± 22.08
GF(z)	**88.05 ± 4.27**	84.58 ± 6.96	**83.84 ± 4.41**	97.67 ± 6.67	92.86 ± 7.14	**63.75 ± 14.09**
RW	–	–	–	90.00 ± 11.06	76.43 ± 14.30	49.72 ± 14.28
SP	76.27 ± 5.16	77.61 ± 6.29	73.22 ± 6.14	94.67 ± 8.19	78.33 ± 14.57	60.67 ± 12.19
GK	66.01 ± 9.57	75.19 ± 8.13	62.38 ± 6.91	56.00 ± 21,12	60.71 ± 13.27	51.39 ± 13.57
RG	–	–	–	72.33 ± 12.52	67.04 ± 15.54	48.87 ± 14.07
WL	87.23 ± 4.57	74.81 ± 6.09	71.22 ± 8.15	**98.33 ± 5.00**	63.57 ± 17.63	61.25 ± 13.60
t-statistic	0.23	**9.31**	**12.74**	0	**12.28**	1.29

	AG vs AI	AG vs AB	AB vs AI
GF(no)	75.36 ± 5.06	61.04 ± 3.98	66.20 ± 8.36
GF(r)	**88.28 ± 3.40**	98.70 ± 1.45	99.06 ± 2.81
GF(z)	87.91 ± 2.86	**99.26 ± 0.91**	98.74 ± 2.87
RW	–	–	–
SP	–	–	–
GK	–	–	97.79 ± 2.01
RG	–	–	–
WL	–	–	**99.38 ± 1.88**
t-statistic	–	–	-0.14

Table 7: Accuracy (± Standard Deviation) on Cell-Graph Datasets – E: Breast, O: Bone, and A: Brain. (bold t-statistic means statistically significant.)

	EB vs EI	EN vs EB	EN vs EI	OF vs ON	ON vs OO	OF vs OO	AG vs AI	AG vs AB	AB vs AI
GF	**1h11m13s**	47m11s	**1h15m40s**	**23.7s**	1m26s	1m43s	**17h50m5s**	**13h53m38s**	**5h15m44s**
RW	–	–	–	11m16s	18m14s	40m8s	–	–	–
SP	8h21m36s	5h44m23s	9h40m4s	19m8s	1h11m58s	1h27m2s	–	–	–
GK	14h12m4s	9h51m37s	11h46m28s	5m33s	9h47s	10m47s	–	–	9h11m12s
RG	–	–	–	24.15s	**43.70s**	**1m29s**	–	–	–
WL	1h55m42s	**44m29s**	1h23m35s	55.30s	2m56s	2m32s	–	–	7h24m29s

Table 8: Running Times on Cell-Graph Datasets (h:hours, m:minutes, s:seconds).

tissues, even GK and WL were not able to complete within a day of computation time. We also do not report the accuracies for CORK, since it aborted on the cell-graphs datasets. Due to the large graph size, and the lack of labels, the graph mining step in CORK fails to enumerate any discriminative subgraphs.

The accuracy of the competing methods depends on the two classes being compared. WL has comparable accuracy only on the easier to classify pairs, namely benign and cancerous breast graphs (EB vs. EI), normal and fractured bone graphs (OF vs. ON), and benign and inflamed brain graphs (AB vs. AI). On the other hand, on the other hard pairs, such as inflamed/noninvasive/damaged versus cancer (EN vs. EI, OF vs. OO, and AI vs. AG), our graph feature approach is significantly superior. For example, on EN vs. EI, GF(z) has an accuracy of 83.84, whereas SP has an accuracy of 73.22, WL has an accuracy of 71.22 and GK has an even lower value (62.38).

It is interesting to note that since cell-graph datasets do not have labels, all the kernels can only use structural information for computing the kernel matrix. The fact that GF has the best accuracies implies that the topological attributes we compute are well-suited to extract discriminating features among the graphs from different classes. In fact, these attributes are much better at capturing the topological differences than the corresponding kernels based on subtrees, graphlets, shortest paths, and random walks, especially on large, unlabeled graphs, such as the cell graphs.

5. CONCLUSIONS

We propose a simple yet effective and efficient graph classification approach that is based on topological and label graph attributes. The graph dataset is converted into a feature-vector dataset, which can be classified easily using any classifier. Our main idea is that graphs from the same class should have similar attribute values. Based on an extensive comparison with state-of-the-art graph kernel classifiers, we show that our approach yields competitive or better accuracies, and has typically much lower computational times. Our conclusion is that graph attributes are effective in capturing discriminating structural information from different classes. While no method is uniformly the best, our approach is particularly effective for unlabeled graphs. Combining our graph features, with the best features from other approaches, such as the WL kernel, has the potential to yield even better methods, especially for labeled graphs.

This work opens up fruitful directions for future research. First, we would like to consider many of the other graph attributes such as betweenness, efficiency, entropy, various centralities, and even features spanning local attributes. Second, we would like to construct better label attributes. Third, we would like to exploit additional features from the other graph kernels (e.g., WL kernel). Finally, we would like to understand which graph and kernel features are the most informative in terms of classification, and eventually, even for clustering.

Acknowledgment

We thank Prof. Karsten Borgwardt and Nino Shervashidze for providing the Matlab implementation for the various graph kernel methods, and Marisa Thoma for the CORK implementation. This work was supported in part by NSF grant EMT-0829835, and NIH grant 1R01EB0080161-01A1.

6. REFERENCES

[1] Z. Bai, J. Demmel, J.Dongarra, A.Ruhe, and H.Vorst. *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*. SIAM, Philadelphia, 2000.

[2] C. Bilgin, C. Demir, C. Nagi, and B. Yener. Cell-graph mining for breast tissue modeling and classification. In *29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 5311 –5314, Aug. 2007.

[3] C. C. Bilgin, P. Bullough, G. E. Plopper, and B. Yener. Ecm-aware cell-graph mining for bone tissue modeling and classification. *Data Mining and Knowledge Discovery*, 20:416–438, 2010.

[4] K. M. Borgwardt and H.-P. Kriegel. Shortest-path kernels on graphs. In *5th IEEE International Conference on Data Mining*, pages 74–81, Washington, DC, USA, 2005.

[5] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. V. N. Vishwanathan, A. J. Smola, and H.-P. Kriegel. Protein function prediction via graph kernels. In *ISMB (Supplement of Bioinformatics)*, pages 47–56, 2005.

[6] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[7] A. Debnath, R. L. de Compadre, G. Debnath, A. Shusterman, and C. Hansch. The structure-activity relationship of mutagenic aromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry*, 34:786–797, 1991.

[8] C. Demir, S. H. Gultekin, and B. Yener. Learning the topological properties of brain tumors. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2(3):262–270, 2005.

[9] P. Dobson and A. J. Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of Molecular Biology*, 330(4):771–783, July 2003.

[10] T. Gärtner, P. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In *16th Annual Conference on Computational Learning Theory*, pages 129–143, 2003.

[11] A. A. Hagberg, D. A. Schult, and P. J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, Pasadena, CA USA, Aug. 2008.

[12] T. Horvàth, T. Gärtner, and S. Wrobel. Cyclic pattern kernels for predictive graph mining. In *10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 158–167, 2004.

[13] T. Joachims, T. Hofmann, Y. Yue, and C.-N. Yu. Predicting structured objects with support vector machines. *Communications of the ACM*, 52(11):97–104, 2009.

[14] E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python, `www.scipy.org`, 2001.

[15] H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labeled graphs. In *International Conference on Machine Learning*, pages 321–328, 2003.

[16] I. R. Kondor and J. Lafferty. Diffusion kernels on graphs and other discrete structures. In *International Conference on Machine Learning*, pages 315–322, 2002.

[17] I. R. Kondor, N. Shervashidze, and K. M. Borgwardt. The graphlet spectrum. In A. P. Danyluk, L. Bottou, and M. L. Littman, editors, *International Conference on Machine Learning*, volume 382, page 67, 2009.

[18] P. Mahè, N. Ueda, T. Akutsu, J.-L. Perret, and J.-P. Vert. Extensions of marginalized graph kernels. In *International Conference on Machine Learning*, 2004.

[19] P. Mahè and J.-P. Vert. Graph kernels based on tree patterns for molecules. *Machine Learning*, 75(1):3–35, 2009.

[20] C. Meyer. *Matrix Analysis and Applied Linear Algebra*. SIAM, 2000.

[21] L. Ralaivola, S. J. Swamidass, H. Saigo, and P. Baldi. Graph kernels for chemical informatics. *Neural Networks*, 18:1093–1110, 2005.

[22] J. Ramon and T.Gärtner. Expressivity versus efficiency of graphs kernels. In *1st International Workshop on Mining Graphs, Trees and Sequences*, 2003.

[23] B. Scholkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001.

[24] N. Shervashidze and K. M. Borgwardt. Fast subtree kernels on graphs. In *Advances in Neural Information Processing Systems*, pages 1660–1668, 2009.

[25] N. Shervashidze, S. V. Vishwanathan, T. H. Petri, K. Mehlhorn, and K. M. Borgwardt. Efficient graphlet kernels for large graph comparison. In *12th International Conference on Artificial Intelligence and Statistics*, pages 488–495, 2009.

[26] M. Thoma, H. Cheng, A. Gretton, J. Han, H.-P. Kriegel, A. Smola, L. Song, P. S. Yu, X. Yan, and K. M. Borgwardt. Discriminative frequent subgraph mining with optimality guarantees. *Statistical Analysis and Data Mining*, 3(5):302âĂŞ–318, Oct. 2010.

[27] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, 1995.

[28] S. V. N. Vishwanathan, K. M. Borgwardt, and N. N. Schraudolph. Fast computation of graph kernels. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Neural Information Processing Systems*, pages 1449–1456, 2006.

[29] S. V. N. Vishwanathan, N. N. Schraudolph, I. R. Kondor, and K. M. Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11:1201–1242, Apr. 2010.

[30] N. Wale and G. Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. In *IEEE International Conference on Data Mining*, 2006.

[31] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *IEEE Int'l Conference on Data Mining*, 2002.