The University of Kansas

Technical Report

CPM: A Graph Pattern Matching Kernel with Diffusion for Accurate Graph Classification

Aaron Smalter, Jun Huan, and Gerald Lushington

ITTC-FY2009-TR-45910-01

August 2008

Project Sponsor:
National Institutes of Health

GPM: A Graph Pattern Matching Kernel with Diffusion for Accurate Graph Classification

Aaron Smalter, Jun Huan
Department of Electrical Engineering and Computer Science
University of Kansas
asmalter,jhuan@ku.com

Gerald Lushington
Molecular Graphics and Modeling Laboratory
University of Kansas
glushington@ku.edu

ABSTRACT

Graph data mining is an active research area. Graphs are general modeling tools to organize information from heterogenous sources and have been applied in many scientific, engineering, and business fields. With the fast accumulation of graph data, building highly accurate predictive models for graph data emerges as a new challenge that has not been fully explored in the data mining community.

In this paper, we demonstrate a novel technique called Graph Pattern Matching kernel (GPM). Our idea is to leverage existing frequent pattern discovery methods and to explore the application of kernel classifier (e.g. support vector machine) in building highly accurate graph classification. In our method, we first identify all frequent patterns from a graph database. We then map subgraphs to graphs in the graph database and use a process we call "pattern diffusion" to label nodes in the graphs. Finally we designed a novel graph matching algorithm to compute a graph kernel. We have performed a comprehensive testing of our algorithm using 16 chemical structure data sets and have compared our methods to all major graph kernel functions that we know. The experimental results demonstrate that our method outperforms state-of-the-art graph kernel methods with a large margin.

1. INTRODUCTION

With the rapid accumulation of annotated graph data (graphs with class labels), graph classification emerges as a important research topic in data mining. Different from unsupervised data mining methods such as graph pattern mining algorithms [11, 12, 34, 37, 41, 44] and graph databases search algorithms [17, 32, 40, 42], graph classification aims to construct accurate predictive models that link graphs to their class labels. Graph classification algorithms are desired in a wide range of applications such as:

- XML classification. Many XML documents are modeled as trees or graphs and it is important to build automated classifiers for XML data [43].
- Text mining. Natural language processing of sentences usually produces a tree (parsing tree) representation of a sentence. In many social science studies, building automated systems to classify sentences into several groups is an important task [20].
- Cheminformatics. Chemical structures have been studied using graphs for a long time [36]. Recently the National Institute of Health has started an ambitious project (the Molecular Library Initiative project) that aims to determine and publicize the biological activity of at least a million chemical compounds per year in the next 5 to 10 years [2]. Graph classification for cheminformatics helps guild experimental design, gain deeper understanding of the chemical structure-activity relationship and speed up drug discovery processes.

Additional applications of graph classification include protein function prediction based on structure [13], gene regulation networks analysis [14] and web mining [45].

Kernel functions are highly efficient tools to build accurate classification models for large volumes of data [9]. The advantage of kernel functions is due to their capability to map a set of data to a high dimensional Hilbert space without explicitly compute the coordinates of the structure. Specifically a binary function $K : X \times X \rightarrow \mathbb{R}$ is a *positive semi-definite* function if

$$\sum_{i,j=1}^{m} c_i c_j K(x_i, x_j) \geq 0 \quad (1)$$

for any $m \in \mathbb{N}$, any selection of samples $x_i \in X$ $(i \in [1, n])$, and any set of coefficients $c_i \in \mathbb{R}$ $(i \in [1, n])$. A binary function is symmetric if $K(x, y) = K(y, x)$ for all $x, y \in X$. A symmetric, positive semi-definite function ensures the existence of a Hillbert space $\mathcal{H}$ and a map $\Phi : X \rightarrow \mathcal{H}$ such that

$$k(x, x') = \langle \Phi(x), \Phi(x') \rangle \quad (2)$$

for all $x, x' \in X$. $\langle x, y \rangle$ denotes an inner product between two objects x and y. The result is known as the Mercer's theorem and a symmetric, positive semi-definite function is also known as a Mercer kernel function [30], or *kernel* function for simplicity.

By projecting the data space to a Hilbert space, kernel functions provide a uniformed analyzing environment for various data types including graphs [9], trees [1], and sequences [19], regardless the fact that the original data space

CIKM 2008 Napa Valley, California USA

may not look like a vector space at all. This strategy is known as the "kernel trick" and it has been applied to various data analysis tasks include classification [38], regression [27] and feature extraction through principle component analysis [31] among others.

Several graph kernel functions have been studied. The pioneer work was done by Haussler in his work of *R-convolution* kernel, providing a framework of which many current graph kernel function follow [9]. Recent progresses of graph kernel functions could be roughly divided into two categories. The first group of kernel functions consider the full adjacency matrix of graphs and hence measure the global similarity of two graphs. These include product graph kernels [8], random walk based kernels [16], and kernels based on shortest paths between pair of nodes [18]. The second group of kernel functions try to capture the local similarity of two graphs by counting the shared subcomponents of graphs. These include the subtree kernels [28], cyclic kernels [35], spectrum kernel [6], and recently subgraph kernels [33].

In this paper, we explore the second avenue and aim to leverage existing frequent pattern mining algorithms in building accurate graph kernel functions. Towards that end, we demonstrate a novel technique called graph pattern matching kernel (GPM). In our method, we first identify all frequent patterns from a graph database. We then map subgraphs to graphs in the graph data set and project nodes of graphs to a high dimensional space with a specially designed function. Finally we designed a novel graph alignment algorithm to compute the inner product of two graphs. We have tested our algorithm using 16 chemical structure data sets. The experimental results demonstrate that our method outperforms existing state-of-the-art with a large margin.

In summary we present the following contributions in this paper:

- We have designed a novel graph kernel function,
- Our kernel function offers a good measure of the local similarity between graphs and is insensitive to noises in graph structures,
- Our kernel function is non-parametric, i.e. we do not assume priori knowledge about the distribution of graphs, and
- We have implemented our kernel function and tested it with a series of cheminformatics data sets. Our experimental study demonstrates that our algorithm outperforms existing state-of-the-art with a large margin.

The rest of the paper is organized as follows. In the rest of this section, we give the notation that we are going to use and formalize the problem of graph classification. We also give a brief survey of research efforts that are closely related to our current effort. In section 2, we provide background information about graphs. In section 3, we present the details of our graph pattern matching kernels. In section 4 we use real-world data sets to evaluate our proposed methods and perform a comparison of ours to the current state-of-the-art. Finally we conclude and present our future plan in section 5.

1.1 Notations and Problem Statement

In this paper, we use capital letters, such as G, for a single graph and upper case calligraphic letters, such as $\mathcal{G} = G_1, G_2, \ldots, G_n$, for a set of n graphs. We assume each graph $G_i \in \mathcal{G}$ has an associated class label c_i from a label set C. C is the class label set of the graphs.

Problem Statement: Given a training set $D = \{(G_i, c_i,)\}_{i=1}^{n}$ of n graphs and their associated labels, the **graph classification problem** is to estimate a function $F : G^* \rightarrow C$ that accurately map graphs in a graph space G^* to their class labels in a label space C.

Below, we review several algorithms for graph classification that work within a common framework called a kernel function.

1.2 Related Work

We survey the work related to graph classification methods by dividing them into two categories. The first category of methods explicitly collect a set of *features* from the graphs. Possible choices are paths, cycles, trees, and general subgraphs [43]. Once a set of features is determined, a graph is described by a feature vector, and any existing classification methods such as Classification based on Association (CBA) [4] and decision tree [26] that work in an n-dimensional Euclidian space, may be applied for graph classification.

The second approach is to implicitly collect a (possibly infinite) set of features from graphs. Rather than computing the features, this approach computes the similarity of graphs, using the framework of "kernel functions" [38]. The advantage of a kernel method is that it has low chance of over fitting, which is a serious concern in high dimensional space with low sample size.

In what follows we review algorithms in the first category, which explicitly utilize identified features from graph to build classifiers. We then discuss graph kernel functions.

1.2.1 Graph Classification Based on Identified Features

Below we review two algorithms that use rule based methods for classifying graph data.

XRules [43] utilizes frequent tree-patterns to build a rule based classifier for XML data. Specifically, XRules first identifies a set of frequent tree-patterns. An association rule: $G \rightarrow c_i$ is then formed where G is a tree pattern and c_i is a class label. The *confidence* of the rule is the conditional probability $p(c_i|G)$ estimated from the training data. XRules carefully selects a subset of rules with high confidence values and uses those rules for classification.

Graph boosting [20] also utilizes substructures toward graph classification. Similar to XRules, graph boosting uses rules with the format of $G \rightarrow c_i$. Different from XRules, it uses the boosting technique to assign weights to different rules. The final classification result is computed as the weighted majority.

In the following discussion, we present the necessary background for a formal introduction to the graph classification problem, and introduce a suite of graph kernel functions for graph classification.

1.2.2 Kernel Functions for Graphs

In recent years a variety of graph kernel functions have been developed, with promising application results as described by Ralaviola *et al.* [29]. Among these methods, some kernel functions draw on graph features such as walks [16] or cycles [35], while others may use different approaches such

as genetic algorithms [3], frequent subgraphs [6], or graph alignment [7].

Below we review five categories of graph kernel functions: product graph kernels, marginalized kernels, spectrum kernel, optimal assignment kernels, and diffusion kernels.

Product Graph Kernels.

The feature space of this group of kernel functions is all possible node label sequences for walks in graphs. Since the number of possible walks are infinite, there is no way to enumerate all the features and then compute the kernel function [8].

In product graph kernels, a *product graph* is computed in order to make the kernel function computation feasible. A product graph of two graphs G, G' is a graph $G_\times$ where nodes are the cartesian production of the two related node sets from G and G', i.e. $V_\times = V[G] \times V[G']$. An edge $((u, v), (u', v'))$ in $G_\times$ is created if $(u, u') \in G$ and $(v, v') \in G$. It has been proved that a direct product graph kernel can be obtained by computing the matrix power series of the product graph.

Extending the feature spaces of walks, recent graph kernel functions use shortest paths [18], subtree patterns [28], and cyclic graphs [35].

Marginalized Kernels.

Rather than computing the shared paths exactly, which has prohibitive computational cost for large graphs, Kashima *et al.* [16] developed a Markov model to randomly generate walks of a labeled graph, using a transition probability matrix combined with a walk termination probability. These collections of random walks are then compared and the number of (expected) shared sequences is used to determine the overall similarity between two molecules.

Spectrum Kernel.

Spectrum kernels aims to simplify the aforementioned kernels by working in a finite dimensional feature space. In spectrum kernels, a feature space is a set of subgraphs (or as special cases, trees, cycles, and paths). The feature vector of a graph is derived by counting the occurrence of subgraphs in the graph. In the most straightforward way, the kernel function of two graphs is the inner product of their feature vectors [6]. Transformations of the inner product, such as min-max kernel [39] and Tanimoto kernel [21], are also widely used. The subtree kernel [24] is a variation on the spectrum kernel that uses subtrees instead of paths.

Optimal Assignment Kernel.

The optimal assignment kernel, proposed by Frölich et al [7], differs significantly from the marginalized graph kernel in that it attempts to align two graphs, rather than compare sets of linear substructures. This kernel function first computes the similarity between all nodes in one graph and all nodes in another. The similarity between the two graphs is then computed by finding the maximal weighted bipartite graph between the two sets of nodes, called the optimal assignment. The authors investigate an extension of this method whereby certain structure patterns defined *a priori* by expert knowledge, are collapsed into single nodes, and this reduced graph is used as input to the optimal assignment kernel.

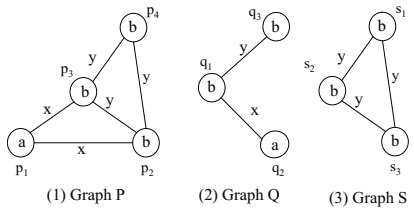

Figure 1: A Database of three labeled graphs.

2. BACKGROUND

In this section we discuss a few important definitions for graph database mining: labeled graphs, subgraph isomorphic relation, graph kernel function, and graph classification.

DEFINITION 2.1. *A **labeled graph** G is a quadruple $G = (V, E, \Sigma, \lambda)$ where V is a set of vertices or nodes and $E \subseteq V \times V$ is a set of undirected edges. Σ is a set of (disjoint) vertex and edge labels, and λ: $V \cup E \rightarrow \Sigma$ is a function that assigns labels to vertices and edges. We assume that a total ordering is defined on the labels in Σ.*

A *graph database* is a set of labeled graphs.

DEFINITION 2.2. *A graph $G' = (V', E', \Sigma', \lambda')$ is **subgraph isomorphic** to $G = (V, E, \Sigma, \lambda)$, denoted by $G' \subseteq G$, if there exists a 1-1 mapping $f : V' \rightarrow V$ such that*

- $\forall v \in V', \lambda'(v) = \lambda(f(v))$
- $\forall (u, v) \in E', (f(u), f(v)) \in E$, *and*
- $\forall (u, v) \in E', \lambda'(u, v) = \lambda(f(u), f(v))$

.

The function f is a *subgraph isomorphism* from graph G' to graph G. We say G' *occurs* in G if $G' \subseteq G$. Given a subgraph isomorphism f, the image of the domain V' ($f(V')$) is an *embedding* of G' in G.

EXAMPLE 2.1. *Figure 1 shows a graph database of three labeled graphs. The mapping (isomorphism) $q_1 \rightarrow p_3$, $q_2 \rightarrow p_1$, and $q_3 \rightarrow p_2$ demonstrates that graph Q is subgraph isomorphic to P and hence Q* occurs in *P. Set $\{p_1, p_2, p_3\}$ is an embedding of Q in P. Similarly, graph S occurs in graph P but not Q.*

3. GRAPH PATTERN MATCHING KERNELS

Here we present our design of a graph matching kernel with diffusion. We start the section by first presenting a general framework for graph matching. Then we present the pattern based graph matching kernel. Finally we show a technique we call "pattern diffusion" that significantly improves graph classification accuracy in practice.

3.1 Graph Matching Kernel

To derive an efficient algorithm scalable to large graphs, our idea is to use a function $\Gamma : V \rightarrow \mathbb{R}^n$ to map nodes in a graph to a n dimensional feature space that captures not only the node label information but also the neighborhood

topological information around the node. If we have such function Γ, we may design the following graph kernel:

$$K_m(G, G') = \sum_{(u,v)\in V[G]\times V[G']} K(\Gamma(u), \Gamma(v)) \quad (3)$$

K can be any kernel function defined in the co-domain of Γ. We call this function K_m a *graph matching kernel*. The following theorem indicates that K_m is symmetric and positive semi-definite and hence a real kernel function.

THEOREM 3.1. *The graph matching kernel is symmetric and positive semi-definite if the function K is symmetric and positive semi-definite.*

Proof sketch: the matching kernel is a special case of the R-convolution kernel and is hence positive semi-definite as proved in [23].

We visualize the kernel function by constructing a weighted complete bipartite graph: connecting every node pair (u,v) $\in V[G] \times V[G']$ with an edge. The weight of the edge (u,v) is $K(\Gamma(v), \Gamma(v))$. In Figure 2, we show a weighted complete bipartite graph for $V[G] = \{v_1, v_2, v_3\}$ and $V[G'] = \{u_1, u_2, u_3\}$.

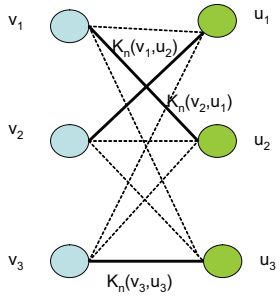

Figure 2: The maximum weighted bipartite graph for graph matching. Highlighted edges $(v1, u2)$, $(v2, u1)$, $(v3, u3)$ have larger weights than the rest of the edges (dashed).

From the figure we see that if two nodes are quite dissimilar, the weight of the related edge is small. Since dissimilar node pairs usually outnumber similar node pairs, if we use linear kernel for nodes, we may have a noisy kernel function and hence loose our signal. In our design, we use the RBF kernel function, as specified below, to penalize dissimilar node pairs.

$$K(X, Y) = e^{\frac{-||X-Y||_2^2}{2}} \quad (4)$$

where $||X||_2^2$ is the squared L_2 norm of a vector X.

3.2 Graph Pattern Matching Kernel

One way to design the function Γ is to take advantage of frequent patterns mined from a set of graphs. Intuitively if a node belongs to a subgraph F, we have some information about the local topology of the node. Following the intuition, given a node v in a graph G and a frequent subgraph F, we design a function Γ_F such that

$$\Gamma_F(v) = \begin{cases} 1 & \text{if } u \text{ belongs an embedding of } F \text{ in } G \\ 0 & \text{otherwise} \end{cases}$$

We call the function Γ_F as a "pattern membership function" since this function tests whether a node occurs in a specific subgraph feature ("membership to a subgraph").

Given a set of frequent subgraph $\mathcal{F} = F_1, F_2, \ldots, F_n$, we treat each membership function as a dimension and design the function $\Gamma_{\mathcal{F}}$ as below:

$$\Gamma_{\mathcal{F}}(v) = (\Gamma_{F_i}(v))_i^n \quad (5)$$

In other words, given n frequent subgraph, the function Γ maps a node v in G to a n-dimensional space, indexed by the n subgraphs, where values of the features indicate whether the node is part of the related subgraph in G.

EXAMPLE 3.1. *In Figure 3, we duplicated the figure Q in Figure 1. We show two subgraph features F_1 and F_2. F_1 has an embedding in Q at $\{q_1, q_2\}$ and F_2 occurs in Q at $\{q_1, q_3\}$. We depict the occurrences using shadings with different color and orientations. For node q_1, if we consider subgraph F_1 as a feature, we have $\Gamma_{F_1}(q_1) = 1$ since q_1 is part of an embedding of F_1 in Q. Also, we have $\Gamma_{F_1}(q_3) = 0$ since q_3 is not part of an embedding of F_1 in Q. Similarly we have $\Gamma_{F_2}(q_1) = 1$ and $\Gamma_{F_2}(q_3) = 1$. Hence $\Gamma_{F_1,F_2}(q_1) = (1, 1)$ and $\Gamma_{F_1,F_2}(q_3) = (0, 1)$. The values of the function Γ_{F_1,F_2} are also illustrated in the same figure using the annotated Q.*

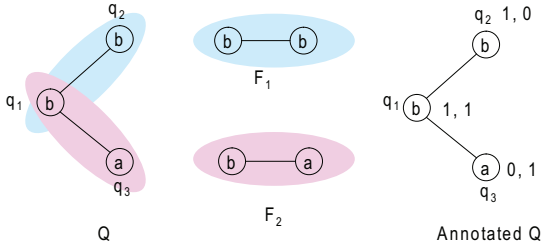

Figure 3: An example of pattern membership functions.

3.3 Graph Pattern Matching Kernel with Pattern Diffusion

Here we introduce a better technique than the pattern membership function to capture the local topology information of nodes. We call this technique "pattern diffusion". Our design has the following advantages:

- Our design is generic and does not assume any domain knowledge from a specific application. The diffusion process may be applied to graphs with dramatically different characteristics.
- The diffusion process is straightforward to implement and can be computed efficiently.
- We prove that the diffusion process is related to the probability distribution of a graph random walk. This explains why the simple process may be used to summarize local topological information.

Below, we outline the pattern diffusion kernel in three steps.

In the first step, we identify a seed as a starting point for the diffusion. In our design, a "seed" could be a single node,

or a set of connected nodes in the original graph. In our experimental study, we always use frequent subgraphs for seeds since we can easily compare a seed from one graph to a seed in another graph.

In the second step given a set of nodes S as seed, we recursively define a diffusion function f_t in the following way.

The base f_0 is defined as:

$$f_0(u) = \begin{cases} 1/|S| & \text{if } u \in S \\ 0 & \text{otherwise} \end{cases}$$

We define f_{t+1} $(t \geq 0)$ with f_t in the following way:

$$f_{t+1}(v) = f_t(v) \times (1 - \frac{\lambda}{d(v)}) + \sum_{u \in N(v)} f_t(u) \times \frac{\lambda}{d(u)} \quad (6)$$

In the notation, $N(v) = \{u|(u, v) \text{ is an edge } \}$ is the set of nodes that connects to v directly. $d(v) = |N(v)|$ is the node degree of v. λ is a parameter that controls the diffusion rate.

The formula 6 describes a process where each node distributes a λ fraction of its value to its neighbors evenly and in the same way receives some value from its neighbors. We call it "diffusion" because the process simulate the way a value is spreading in a network. Our intuition is that the distribution of such a value encodes information about the local topology of the network.

To constrain the diffusion process to a local region, we use one parameter called diffusion time, denoted by τ, to control the diffusion process. Specifically we limit the diffusion process to a local region of the original graph with nodes that are at most τ hops away from a node in the seed S. In this sense, the diffusion should be named "local diffusion".

Finally in the last step, for the seed S, we define the mapping function Γ_S^d as the limit function of f_t as t approaches to infinity, or

$$\Gamma_S^d = \lim_{t \to \infty} f_t \quad (7)$$

And given a set of frequent subgraph $\mathcal{F} = F_1, F_2, \ldots, F_n$ as seeds, we design the pattern diffusion function $\Gamma_{\mathcal{F}}^d$ as:

$$\Gamma_{\mathcal{F}}^d(v) = (\Gamma_{F_i}^d(v))_i^n \quad (8)$$

3.4 Connections of Other Graph Kernels

3.4.1 Connection to Marginalized Kernels

Here we show the connection of pattern matching kernel function to the marginalized graph kernel [16], which uses a Markov model to randomly generate walks of a labeled graph.

Given a graph G with nodes set $V[G] = \{v_1, v_2, \ldots, v_n\}$, and a seed $S \subseteq V[G]$, for each diffusion function f_t, we construct a vector $U_t = (f_t(v_1), f_t(v_2), \ldots, f_t(v_n))$. According to the definition of f_t, we have $U_{t+1} = M \times U_t$ where the matrix M is defined as:

$$M(i,j) = \begin{cases} \frac{\lambda}{d(v_j)} & \text{if } i \neq j \text{ and } i \in N(j) \\ 1 - \frac{\lambda}{d(v_i)} & i = j \\ 0 & \text{otherwise} \end{cases}$$

In this representation, we compute the stationary distribution ($f_S = \lim_{t \to \infty} f_t$) by computing $M^\infty \times U_0$.

We notice that the matrix M corresponds to a probability matrix corresponding to a Markov Chain since

- all entries are non-negative
- column sum is 1 for each column

Therefore the vector $M^\infty \times U_0$ corresponds to the stationary distribution of the local random walk as specified by M. In other words, rather than using random walk to retrieve information about the local topology of a graph, we use the stationary distribution to retrieve information about the local topology. Our experimental study shows that this in fact is an efficient way for graph classification.

3.4.2 Connection to Optimal Assignment Kernel

The optimal assignment (OA) kernel [7] carries the same spirit of our graph pattern matching kernel in that OA uses pairwise node kernel function to construct a graph kernel function. OA kernel has been utilized for cheminformatics applications and is found to deliver good results empirically.

There are two major differences between ours and the OA kernel. (1) OA kernel is not positive semi-definite and hence is not Mercer kernel in a strict sense. Non Mercer kernel functions are used to train SVM model and the problem is that the convex optimizer utilized in SVM will not converge to a global optimal and hence the performance of the SVM training may not be reliable. (2) OA utilizes a complicated recursive function to compute the similarity between nodes, which make the computation of the kernel function runs slowly for large graphs [33].

3.5 Pattern Diffusion Kernel and Graph Classification

We summarize the discussions we present so far and show how the kernel function is utilized to construct an efficient graph classification algorithm at both the training and testing phases.

3.5.1 Training Phase

In the training phase, we divide graphs of the training data set $D = \{(G_i, T_i,)\}_{i=1}^n$ into groups according to their class labels. For example in binary classification, we have two groups of graphs: positive or negative. For multi-class classification, we partition graphs according to their class label where graphs have the same class labels are grouped together. The training phase is composed of four steps:

- Obtain frequent subgraphs. We identify frequent subgraphs from each graph group and union the subgraph sets together as our seed set $\mathcal{F}$.
- For each graph G in the training data set, we use the node pattern diffusion function $\Gamma_{\mathcal{F}}^d$ to label nodes in G. Thus the feature vector of a node v is a vector $L_V = (\Gamma_{F_i}^d(v))_{i=1}^m$ with length $m = |\mathcal{F}|$.
- For two graphs G, G', we construct the complete weighted bipartite graph as described in section 3.1 and compute the kernel $K_m(G, G')$ using Equation 3 and Equation 4.
- Train a predictive model using a kernel classifier.

Table 1: Characteristics of our data sets. 'Background' column indicates protein target or organism type. # G: number of samples (chemical compounds) in the data set. # P: positive samples. # N: negative samples

Source	Dataset	Background	# G	# P	# N
BindingDB	AChE	Acetylcholinesterase	138	69	69
	ALF	Anthrax Lethal Factor	93	47	46
	EGF-TK	EGF-R Tyrosine Kinase	377	190	187
	HIV-P	HIV-1 Protease	202	101	101
	HIV-RT	HIV-1 Reverse Transcriptase	365	183	182
	HSP90	Heat Shock Protein 90	82	41	41
	MAPK	Map Kinase p38 alpha	255	126	129
Jorissen	CDK2	Cyclin-dependent Kinase 2	100	50	50
	COX2	Cyclooxygenase 2	100	50	50
	FXa	Factor Xa Protease	100	50	50
	PDE5	Phosphodiesterase type 5	100	50	50
	A1A	A1-adenosine	100	50	50
Predictive Toxicology Challenge	PTC-FM	female mice	344	152	192
	PTC-FR	female rats	336	129	207
	PTC-MM	male mice	351	121	230
	PTC-MR	male rats	349	143	206

3.5.2 *Testing Phase*

In the testing phase, we compute the kernel function for graphs in the testing and training data sets. We use the trained model to make predictions about graph in the testing set.

- For each graph G in the testing data set, we use $\Gamma^d_{\mathcal{F}}$ to label nodes in G and create feature vectors as we did in the training phase.
- We use Equation 3 and Equation 4 to compute the kernel function $K_m(G, G')$ for each graph G in the testing data set and for each graph G' in the training data set.
- Use kernel classifier and trained models to obtain prediction accuracy of the testing data set

Below we present our empirical study of different kernel functions including our pattern diffusion kernel.

4. EXPERIMENTAL STUDY

We conducted classification experiments using eight different graph kernel functions, including our Pattern Diffusion kernel, on sixteen different data sets. There are twelve chemical-protein binding data sets, and the rest are chemical toxicity data sets. We performed all of our experiments on a desktop computer with a 3Ghz Pertium 4 processor and 1 GB of RAM. In the following subsections, we describe the data sets and the classification methods in more detail along with the associated results.

In all classification experiments, we used the LibSVM [5] as our kernel classifier. We used nu-SVC with nu = 0.5. Our classification accuracy (TP+TN/S, TP: true positive, TN: true negative, S: total number of testing samples) is computed by averaging over a 10-fold cross-validation experiment. Standard deviation is computed similarly. To have a fair comparison, we simply used default SVM parameters in all cases, and did not tune any parameters to increase the accuracy of any method.

4.1 Data Sets

We have selected sixteen data sets covering prediction of chemical-protein binding activity and chemical toxicity. The first seven data sets are manually extracted from the BindindDB database [22]. The next five are established data sets taken from Jorissen et al. [15]. The last four are from the Predictive Toxicology Challenge[10] (PTC).

4.1.1 *BindingDB Sets*

The BindingDB database contains more than 450 proteins. For each protein, the database record chemicals that bind to the protein. Two types of activity measurements K_i and IC_{50} are provided. Both measurements measure inhibition/dissociation rates between a proteins and chemicals. From BindingDB, we manually selected 7 proteins with a wide range of known interacting chemicals (ranging from tens to several hundreds).

Since the binding activity measurements are real-valued we must convert them into binary class labels for classification. This is accomplished by finding the median activity value and splitting the data accordingly: that is all compounds with activity less then or equal to the cutoff is given one class label, and compounds with greater activity are given another. We also throw out the middle 25% of the data set, in order to impose some separation between the classes. This method of class label construction has the convenient property of producing data sets with equal proportion positive/negative classes, but may not accurately reflect true biological activity.

4.1.2 *Jorissen Sets*

The Jorissen data sets also contains information about chemical-protein binding activity. In this case the provider of the data set carefully selected positive and negative samples and hence are more reliable than the data sets we created from BindingDB. For more information about the creation of the data sets, see [15] in details.

4.1.3 *PTC Sets*

The Predictive Toxicology Challenge (PTC) data sets contain a series of chemical compounds classified according to their toxicity in male rats, female rats, male mice, and female mice. While chemical-protein binding activity is an important type of chemical activity, it is not the only type. Toxicity is another important, though different, kind of chemical activity we would like to predict in drug design. This data set is well curated and highly reliable.

The characteristics of the 16 data sets is provided in Table 1.

4.2 Kernel Functions

We have selected 6 different kernel functions for evaluation: Marginalized[16], spectrum[6], tanimoto[21], subtree[24], optimal assignment[7], together with our graph pattern matching kernel. All of these kernel functions depend on a decomposition of the graphs into pieces of some *size*. For the marginalized kernel, the walk size is controlled by the termination probability. For the spectrum and tanimoto kernels it is the path fragment length, and maximum depth for the subtree kernel. The optimal assignment kernel is controlled by the neighborhood size on which to match vertices. The pattern diffusion kernel uses subgraph patterns of a specific size, and also rate/time parameters to control the diffusion area. In order to keep these various functions efficient to compute we have chosen to use small parameters for all these functions; in most cases a value between 3 and 5 was chosen.

Four kernel functions (Marginalized, spectrum, tanimoto, subtree) are computed using the open source Chemcpp v1.0.2 package [25]. The optimal assignment kernel was computed using the JOELib2 package, and is not strictly a kernel function, but still provides good prediction accuracy. Our graph pattern matching kernel was computed using our own MATLAB code.

Whenever possible we used default parameters in kernel computation, which are specified below:

- Marginalized - termination probability = 0.1, tottering paths not filtered.
- Spectrum - path length = 4.
- Tanimoto - path length = 4.
- Subtree - maximum depth = 3, tottering patterns not filtered.
- Optimal assignment - neighbor-matching depth = 3.
- Pattern matching - diffusion rate = 0.2, diffusion time = 3 steps; subgraph support 25%, size <= 5.

4.3 Experimental Results

4.3.1 Comparison Between Kernel Functions

Here we present the results of our graph classification experiments with various kernel functions. The following Figure 4 shows the classification accuracy for different kernel functions and data sets, averaged over a 10-fold cross validation experiment. The precision and recall are given in Tables 5 and 6 contained in the appendix A. The standard deviations (omitted) of the accuracies are generally very high, from 5-10%, so statistically significant differences between kernel functions are generally not observed.

We can see from the data that our method is competitive for all sixteen data sets. If we examine the accuracy of each kernel function averaged over all data sets, we see that our GPM kernel performs the best overall. Again, the standard deviations are high so the differences between the top performing kernels are not statistically significant. Still, with 16 different data sets we can see some clear trends: GPM kernel delivers the highest classification accuracy in 8 out of the 16 data sets shown in Table 2.

Table 2: Comparison of best classification accuracy between kernel functions (ties included). B: the number of data sets in which a kernel function provides the best classification accuracy.

Kernel Function	# B	Data Sets
GPM	8	ALF, CDK2, COX2, A1A, HSP90, HIV-P, PTC-FM, PTC-MM
tanimoto	4	AChE, EGF-R, FXa, PDE5
marginalized	2	HIV-RT, PTC-FR
subtree	2	HIV-PT, PTC-MR
optimal assign.	1	MAPK
spectrum	0	n/a

The next Table 3 gives a comparison of our method to each of the other methods. Although GPM does not work well on a few data sets such as AChE, HIV-RT, MAPK, and PTC-FR/MR, overall it performs better when compared to any other kernel for a majority of data sets.

In general the GPM, spectrum and tanimoto kernels perform the best, with over all average accuracy of about 80%. The subtree, optimal assignment, and marginalized also perform very good, in mid to high 70%. The min/max tanimoto kernel performed much worse than the other methods, and hence it was not included in the figure. Note that the optimal assignment kernel is missing a prediction accuracy for the FXa data set, this was due to a terminal error in the JOELib2 software used to calculate this kernel on this data set.

Table 3: Comparison of number of 'better' data set accuracies between GPM and each other kernel function ('better' does not include ties here). N: # of data sets that GPM gives better classification accuracy.

Kernel Function	# N	Data Sets
tanimoto	11	All but AChE, EGF-R, MAPK, PTC-FR, PTC-MR
subtree	11	All but AChE, HIV-P, MAPK, PTC-FR, PTC-MR
optimal assign.	12	All but AChE, HIV-RT, MAPK, PTC-FR
marginalized	12	All but AChE, HIV-RT, PTC-FR, PTC-MR
spectrum	14	All but AChE, HIV-RT

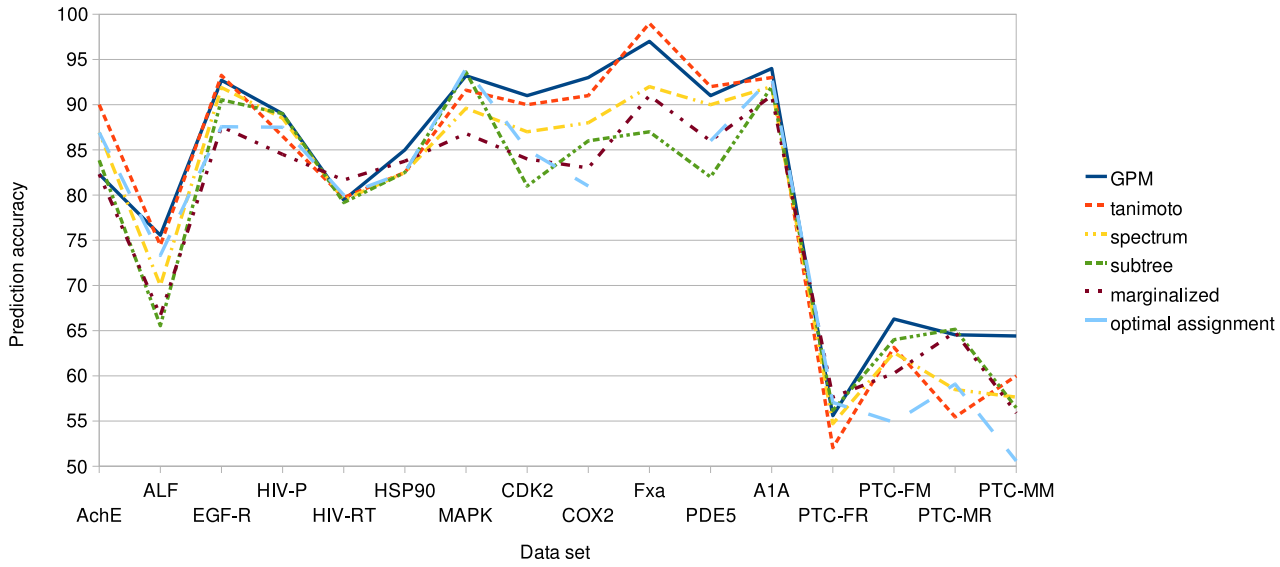

Figure 4: Average accuracy for all kernel functions and data sets.

4.3.2 *Comparison with Non-kernel Classifier*

In Table 4 we compare the performance of our GPM kernel to a non-kernel classifier: CBA or Classification Based on Association[4]. In CBA we treat mined frequent subgraphs as item sets. Despite the strengths of CBA, we can see that GPM method gives the best performance for all of the five data sets evaluated. We chose these five sets as they are curated and represent a several different protein targets.

Table 4: Comparison of GPM and CBA.

Data set	GPM	CBA
CDK2	91	80.46
COX2	93	77.86
Fxa	97	86.87
PDE5	91	87.14
A1A	94	87.76

4.3.3 *Using Different Feature Sets*

Once frequent subgraphs are mined, we generate three feature sets: (i) general subgraphs (all of mined subgraphs), (ii) tree subgraphs, and (iii) path subgraphs. We tried cycles as well, but did not include them in this study since typically less than two cyclic subgraphs were identified in a data set. These feature sets are then used for constructing kernel functions. We tested our GPM kernel using path, tree, and general subgraph patterns. From our preliminary experiments with chemical data we observed that in general using different feature sets negligible difference. Most general subgraphs in chemicals are either paths or trees with few branches, so there is little distinction between using these three features.

5. CONCLUSIONS AND FUTURE WORKS

With the rapid development of fast and sophisticated data collection methods, data has become complex, high-dimensional and noisy. Graphs have proven to be powerful tools for modeling complex, high-dimensional and noisy data; building highly accurate predictive models for graph data is a new challenge for the data mining community. In this paper we have demonstrated the utility of a novel graph kernel function, graph pattern matching kernel (GPM kernel). We showed that the GPM kernel can capture the intrinsic connection between a graph and its class label and has the lowest testing error in majority of the data sets we evaluated. Although we have developed a very efficient computational framework, computing a GPM kernel may be hard for large graphs. Our future work will concentrate on improving the computational efficiency of the GPM kernel for very large graphs.

Acknowledgments

This work has been supported by the Kansas IDeA Network for Biomedical Research Excellence (NIH/NCRR award #P20 RR016475), the KU Center of Excellence for Chemical Methodology and Library Development (NIH/NIGM award #P50 GM069663), and NIH grant #R01 GM868665.

6. REFERENCES

[1] F. Aiolli, G. D. S. Martino, A. Sperduti, and A. Moschitti. Fast on-line kernel learning for trees. *Proceedings of the International Conference on Data Mining*, pages 787 – 791, 2006.

[2] C. Austin, L. Brady, T. Insel, and F. Collins. Nih molecular libraries initiative. *Science*, 306(5699):1138–9, 2004.

[3] E. Barbu, R. Raveaux, H. Locteau, S. Adam, and P. Heroux. Graph classification using genetic algorithm and graph probing application to symbol recognition. *Proc. of the 18th International Conference on Pattern Recognition (ICPR)*, 2006.

[4] Y. M. Bing Liu, Wynne Hsu. Integrating classification and association rule mining. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, 1998.

[5] C. Chang and C. Lin. Libsvm: a library for support vector machines, 2001. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[6] M. Deshpande, M. Kuramochi, and G. Karypis. Frequent sub-structure-based approaches for classifying chemical compounds. *IEEE Transactions on Knowledge and Data Engineering*, 2005.

[7] Fröohlich, J. Wegner, F. Sieker, and A. Zell. Kernel functions for attributed molecular graphs - a new similarity-based approach to adme prediction in classification. *QSAR & Combinatorial Science*, 2006.

[8] T. Gärtner, P. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Sixteenth Annual Conference on Computational Learning Theory and Seventh Kernel Workshop*, 2003.

[9] D. Haussler. Convolution kernels on discrete structures. *Technical Report UCSC-CRL099-10, Computer Science Department, UC Santa Cruz*, 1999.

[10] C. Helma, R. King, and S. Kramer. The predictive toxicology challenge 2000-2001. *Bioinformatics*, 17(1):107–108, 2001.

[11] T. Horvath, J. Ramon, and S. Wrobel. Frequent subgraph mining in outerplanar graphs. In *SIGKDD*, 2006.

[12] J. Huan, W. Wang, and J. Prins. Efficient mining of frequent subgraph in the presence of isomorphism. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM)*, pages 549–552, 2003.

[13] J. Huan, W. Wang, A. Washington, J. Prins, R. Shah, and A. Tropsha. Accurate classification of protein structural families based on coherent subgraph analysis. In *Proceedings of the Pacific Symposium on Biocomputing (PSB)*, pages 411–422, 2004.

[14] Y. Huang, H. Li, H. Hu, X. Yan, M. S. Waterman, H. Huang, and X. J. Zhou. Systematic discovery of functional modules and context-specific functional annotation of human genome. *Bioinformatics*, pages ISMB/ECCB Supplement, 222–229, 2007.

[15] R. Jorissen and M. Gilson. Virtual screening of molecular databases using a support vector machine. *J. Chem. Inf. Model.*, 45(3):549–561, 2005.

[16] H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labeled graphs. In *Proc. of the Twentieth Int. Conf. on Machine Learning (ICML)*, 2003.

[17] Y. Ke, J. Cheng, , and W. Ng. Correlation search in graph databases. In *SIGKDD*, 2007.

[18] B. K.M. and K. H.-P. Shortest-path kernels on graphs. In *in Proc. of International Conference on Data Mining*, 2005.

[19] R. Kuang, E. Ie, K. Wang, K. Wang, M. Siddiqi, Y. Freund, and C. Leslie. Profile-based string kernels for remote homology detection and motif extraction. *Journal of Bioinformatics and Computational Biology*, 3 (3):527–550, 2005.

[20] T. Kudo, E. Maeda, and Y. Matsumoto. An application of boosting to graph classification. In *NIPS*, 2004.

[21] R. L, S. SJ, S. H, and B. P. Graph kernels for chemical informatics. *Neural Networks*, 18:1093–1110, 2005.

[22] T. Liu, Y. Lin, X. Wen, R. N. Jorrisen, and M. Gilson. Bindingdb: a web-accessible database of experimentally determined protein-ligand binding affinities. *Nucleic Acids Research*, 35:D198–D201, 2007.

[23] S. Lyu. Mercer kernels for object recognition with local features. In *IEEE Computer Vision and Pattern Recognition*, pages 223–229, 2005.

[24] P. Mahe and J. Vert. Graph kernels based on tree patterns for molecules. Technical Report HAL:ccsd-00095488, Ecoles des Mines de Paris, September 2006.

[25] J.-L. Perret, P. Mahe, and J.-P. Vert. Chemcpp: an open source c++ toolbox for kernel functions on chemical compounds, 2007. Software available at http://chemcpp.sourceforge.net.

[26] J. R. Quinlan. *C4.5 : Programs for Machine Learning*. Morgan Kaufmann, 1993.

[27] C. R and S. B. SVMTorch:. Support vector machines for large-scale regression problems. *Journal of Machine Learning Research*, 21, 2001.

[28] J. Ramon and T. Gärtner. Expressivity versus efficiency of graph kernels. In *Technical Report, First International Workshop on Mining Graphs, Trees and Sequences*, 2003.

[29] L. Ravaliola, S. J. Swamidass, and H. Saigo. Graph kernels for chemical informatics. *Neural Networks*, 2005.

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

[31] B. Schölkopf, A. J. Smola, and K.-R. Müller. Kernel principal component analysis. *Advances in kernel methods: support vector learning*, pages 327–352, 1999.

[32] D. Shasha, J. T. L. Wang, and R. Giugno. Algorithmics and applications of tree and graph searching. In *Proceeding of the ACM Symposium on Principles of Database Systems (PODS)*, 2002.

[33] A. Smalter, J. Huan, and G. Lushington. Structure-based pattern mining for chemical compound classification. *Proceedings of the 6th Asia Pacific Bioinformatics Conference*, 2008.

[34] J. Sun, S. Papadimitriou, P. S. Yu, and C. Faloutsos. Parameter-free mining of large time-evolving graphs. In *SIGKDD*, 2007.

[35] S. W. Tamas Horvath, Thomas Gartner. Cyclic pattern kernels for predictive graph mining. *SIGKDD*, 2004.

[36] N. Tolliday, P. A. Clemons, P. Ferraiolo, A. N. Koehler, T. A. Lewis, X. Li, S. L. Schreiber, D. S. Gerhard, and S. Eliasof. Small molecules, big players: the national cancer institute's initiative for chemical genetics. *Cancer Research*, 66:8935–42, 2006.

[37] H. Tong, Y. Koren, , and C. Faloutsos. Fast direction-aware proximity for graph mining. In *SIGKDD*, 2007.

[38] V. Vapnik. *Statistical Learning Theory*. John Wiley, 1998.

[39] N. Wale, I. Watson, , and G. Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 2007.

[40] D. Williams, J. Huan, and W. Wang. Graph database indexing using structured graph decomposition. In *in Proceedings of the 23rd IEEE International Conference on Data Engineering (ICDE)*, 2007.

[41] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *Proc. International Conference on Data Mining'02*, pages 721–724, 2002.

[42] X. Yan, P. S. Yu, and J. Han. Graph indexing based on discriminative frequent structure analysis. In *ACM Transactions on Database Systems (TODS)*, 2005.

[43] M. J. Zaki and C. C. Aggarwal. Xrules: An effective structural classifier for xml data. *Machine Learning Journal special issue on Statistical Relational Learning and Multi-Relational Data Mining*, 62, No. 1-2:137–170, 2006.

[44] Z. Zeng, J. Wang, L. Zhou, and G. Karypis. Coherent closed quasi-clique discovery from large dense graph databases. In *SIGKDD*, 2006.

[45] D. Zhou, J. Huang, and B. Schöolkopf. Learning from labeled and unlabeled data on a directed graph. *Proceedings of the 22nd International Conference on Machine Learning*, 2005.

APPENDIX

A. PRECISION AND RECALL FOR KERNEL EXPERIMENTS

For reference, here we provide a pair of tables listing the precision and recall for all graph kernel experiments with all data sets.

Table 5: Average precision for all kernel functions and data sets. Asterisk (∗) denotes the best precision for the data set among competing methods

Data Set	GPM	Tanimoto	Spectrum	Subtree	Marginalized	Optimal Assign.
AChE	82.34	90.9*	89.06	80.05	90.83	88.58
ALF	79.57*	77.5	71.5	72	68.74	73
EGF-R	91.12	91.91*	89.59	88.13	83.93	83.71
HIV-P	92.22*	86.01	91.06	91.65	83.87	91.67
HIV-RT	80.5	80.16	79.84	79.07	82.93*	81.31
HSP90	77.57	76.74	76.74	76.5	77.9	78.83*
MAPK	92.79	93.44	92.35	94.44*	87.48	91.97
CDK2	96*	89.07	86.48	79.24	81.32	81.32
COX2	96.07*	89.83	85.94	88.57	83.45	82.9
FXa	100*	97.5	90.83	87.5	87.38	79.82
PDE5	92.17	92.74*	86.81	75.95	83.48	85.31
A1A	88.33*	85.57	85.83	87.67	83.21	87.73
PTC-FR	46.8	42.56	45.62	46.95	51.85*	47.64
PTC-FM	51.13*	40.67	45.37	47.85	42.69	36.79
PTC-MR	56.24	44.82	47.35	56.01	57.63*	46.19
PTC-MM	60.19*	55.37	51.55	51.28	50.93	43.97

Table 6: Average recall for all kernel functions and data sets. Asterisk (∗) denotes the best recall for the data set among competing methods

Data Set	GPM	Tanimoto	Spectrum	Subtree	Marginalized	Optimal Assign.
AChE	83.13	88.45	85.95	92.74*	74.54	85.04
ALF	73.5*	70.67	70.17	66.5	69.83	72.67
EGF-R	95.36	95.39	95.43*	94.32	93.43	94
HIV-P	85.19	88.47*	85.91	86.11	86.19	83.56
HIV-RT	78.47	79.54	79.66	79.65	81.6*	79.37
HSP90	92.67*	91	91	91	88	84
MAPK	93.71	90.14	87.12	92.59	85.3	95.63*
CDK2	88.38	93.71*	90.05	84.71	92.05	87.9
COX2	91.57	93.14	93.48*	83.71	90.86	79.86
FXa	93.57	100*	95.24	84.29	96.9	93.48
PDE5	89.67	93.57	93.33	86.33	95*	89.57
A1A	96.67	100*	98	93.24	100*	100*
PTC-FR	42.69	54.24*	53.55	43.38	49.14	50.59
PTC-FM	39.75	27.36	32.26	34.26	41.96*	38.68
PTC-MR	47.06	50.78*	50.44	49.24	49.07	43.16
PTC-MM	56.12*	55.01	54.13	48.13	48.83	43.4