1 Kernels for Graphs

Hisashi Kashima
IBM Research, Tokyo Research Laboratory
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502, Japan
hkashima@jp.ibm.com

Koji Tsuda
Max Planck Institute for Biological Cybernetics
Spemannstr. 38
72070, Tübingen, Germany
koji.tsuda@tuebingen.mpg.de

Akihiro Inokuchi
IBM Research, Tokyo Research Laboratory
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502, Japan
inokuchi@jp.ibm.com

This chapter discusses the construction of kernel functions between labeled graphs. We provide a unified account of a family of kernels called *label sequence kernels* that are defined via label sequences generated by graph traversal. For cyclic graphs, dynamic programming techniques cannot simply be applied, because the kernel is based on an infinite dimensional feature space. We show that the kernel computation boils down to obtaining the stationary state of a discrete-time linear system, which is efficiently performed by solving simultaneous linear equations. Promising empirical results are presented in classification of chemical compounds.

1.1 Introduction

Many real world data are represented not as vectors, but as *graphs* including sequences and trees as special cases. Examples of such data include biological sequences, phylogenetic trees, RNA structures (Durbin et al., 1998), natural lan-

guage texts (Manning and Schütze, 1999), semi-structured data such as HTML and XML (Abiteboul et al., 2000), and so on. In computational biology, graph data are attracting considerable attention in drug design. Chemical compounds can be represented as labeled graphs and their automatic classification to predict the effectiveness or toxicity of drugs is of crucial importance in the rationalization of drug discovery processes (Kramer and De Raedt, 2001; Inokuchi et al., 2000). In protein engineering, three dimensional structures of proteins are often represented as distance graphs (Holm and Sander, 1993).

Kernel methods such as support vector machines are becoming increasingly popular for their high performance (Schölkopf and Smola, 2002). In kernel methods, all computations are done via a *kernel function*, which is defined as the inner product of two vectors in a feature space. A kernel function needs to be designed to capture the characteristics of the objects appropriately, and at the same time, to be computed efficiently. Furthermore it must satisfy the mathematical property called *positive semidefiniteness*. A kernel function should deliberately be designed to satisfy this property, because ad hoc similarity functions are not always positive semidefinite, e.g. Shimodaira et al. (2002) and Bahlmann et al. (2002).

Haussler (1999) introduced 'convolution kernels', a general framework for handling discrete data structures by kernel methods. In convolution kernels, objects are decomposed into parts, and kernels are defined in terms of the (sub)kernels between parts. After this seminal paper, a number of kernels for structured data were proposed, for example, Watkins (2000), Jaakkola et al. (2000), Leslie et al. (2003), Lodhi et al. (2002), and Tsuda et al. (2002) for sequences, Vishwanathan and Smola (2003), Collins and Duffy (2002), and Kashima and Koyanagi (2002) for trees. Most of them are basically based on the same idea. An object such as a sequence or a tree is decomposed into substructures, e.g. substrings, subsequences, subtrees, and a feature vector is composed of the counts of the substructures. Since the dimensionality of feature vectors is typically very high, the explicit computations of feature values should be avoided. So most of the kernels adopt efficient procedures such as dynamic programming or suffix trees.

In this chapter, we discuss the construction of kernel functions between labeled graphs[1]. We try to give a unified overview on the recent researches for graph kernels (Kashima and Inokuchi, 2002; Kashima et al., 2003; Gärtner, 2002; Gärtner et al., 2003). A common point of these works is that features are composed of the counts of *label sequences* produced by graph traversal. For the labeled graph shown in Figure 1.1, a label sequence is produced by traversing the vertices, and looks like

$$(A, c, C, b, A, a, B), \tag{1.1}$$

1. Note that they should be distinguished from kernels in graph structured input spaces such as kernels between two vertices in a graph, e.g., diffusion kernels (Srinivasan et al., 1996; Kandola et al., 2003; Lafferty and Lebanon, 2003) or kernels between two paths in a graph, e.g., path kernels (Takimoto and Warmuth, 2002).

where the vertex labels A, B, C, D and the edge labels a, b, c, d appear alternately. The essential difference among the kernels lies in how graphs are traversed and how weights are involved in computing a kernel. We call this family of kernel 'label sequence kernels'. This family of kernels can be computed efficiently and capture essential features of labeled graphs. As we will see below, label sequence kernels are closely related to the kernels between probability distributions (Jebara and Kondor, 2003), especially the kernels between hidden Markov models (Lyngsø et al., 1999). Mathematically it is possible to consider a kernel based on more complicated substructures such as subgraphs. However the practical computation becomes considerably more difficult, because counting the number of all possible subgraphs turns out to be NP-hard (Gärtner et al., 2003).

When the graphs are acyclic, the label sequence kernels are computed simply by dynamic programming as shown in Section 1.2.3.1. However, when the graphs are cyclic, label sequences of infinite length can be produced because the traversal may never end. In that case, the number of features becomes infinite. For computing a kernel based on infinite dimensional vectors, we exploit recursive structures in features, and it will be shown that the kernel computation is reduced to finding the stationary state of a discrete-time linear system (Rugh, 1995), which can be done efficiently by solving simultaneous linear equations with a sparse coefficient matrix.

In the following, we describe the kernel function proposed by Kashima et al. (2003).[2] This kernel is defined as the expectation of a string kernel over all possible label sequences, which is regarded as a special case of *marginalized kernels* (Tsuda et al., 2002). The relations to other label sequence kernels are discussed, and several extensions are proposed as well. Finally, in order to investigate how well our kernel performs on the real data, we show promising results on classifying chemical compounds.

1.2 Label Sequence Kernel Between Labeled Graphs

In this section, we introduce a kernel between labeled graphs. First of all, let us define a labeled graph. Let Σ_V, Σ_E be the sets of vertex labels and edge labels, respectively. Let $\mathcal{X}$ be a finite nonempty set of vertices, v be a total function $v : \mathcal{X} \to \Sigma_V$, $\mathcal{L}$ be a set of ordered pairs of vertices called edges, and e be a total function $e : \mathcal{L} \to \Sigma_E$. Then $G = (\mathcal{X}, v, \mathcal{L}, e)$ is a labeled graph with directed edges. Figure 1.1 shows such a graph. For the time being, we assume that there are no multiple edges from one vertex to another. Our task is to construct a kernel function $K(G, G')$ between two labeled graphs.

2. Notice that the notations here are in part changed from those in Kashima et al. (2003) for better presentation.

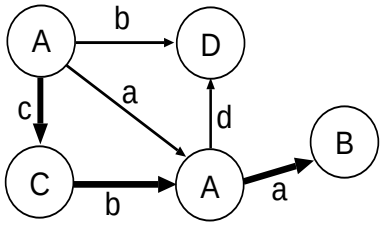

Figure 1.1 An example of labeled graphs. Vertices and edges are labeled by upper and lower case letters, respectively. By traversing along the bold edges, the label sequence (1.1) is produced.

1.2.1 Random Walks on Graphs

For extracting features from graph G, a set of label sequences is produced by random walking. At the first step, $x_1 \in \mathcal{X}$ is sampled from an initial probability distribution $p_s(x_1)$. Subsequently, at the i-th step, the next vertex $x_i \in \mathcal{X}$ is sampled subject to a transition probability $p_t(x_i|x_{i-1})$, or the random walk ends with probability $p_q(x_{i-1})$:

$$\sum_{x_i=1}^{|\mathcal{X}|} p_t(x_i|x_{i-1}) + p_q(x_{i-1}) = 1. \tag{1.2}$$

When we do not have any prior knowledge, we can set p_s to be the uniform distribution, the transition probability p_t to be an uniform distribution over the vertices adjacent to the current vertex, and the termination probability p_q to be a small constant probability.

From the random walk, we obtain a sequence of vertices called *path*:

$$\mathbf{x} = (x_1, x_2, \ldots, x_\ell), \tag{1.3}$$

where ℓ is the length of $\mathbf{x}$ (possibly infinite). The probability for the path $\mathbf{x}$ is described as

$$p(\mathbf{x}|G) = p_s(x_1) \prod_{i=2}^{\ell} p_t(x_i|x_{i-1}) p_q(x_\ell).$$

Let us define a *label sequence* as an alternating sequence of vertex labels and edge labels:

$$\mathbf{h} = (h_1, h_2, \ldots, h_{2\ell-1}) \in (\Sigma_V \Sigma_E)^{\ell-1} \Sigma_V.$$

Associated with a path $\mathbf{x}$, we obtain a label sequence

$$\mathbf{h}_\mathbf{x} = (v_{x_1}, e_{x_1,x_2}, v_{x_2}, \ldots, v_{x_\ell}).$$

The probability for the label sequence $\mathbf{h}$ is equal to the sum of the probabilities of the paths emitting $\mathbf{h}$,

$$p(\mathbf{h}|G) = \sum_{\mathbf{x}} \delta(\mathbf{h} = \mathbf{h}_{\mathbf{x}}) \cdot \left(p_s(x_1) \prod_{i=2}^{\ell} p_t(x_i|x_{i-1}) p_q(x_\ell) \right)$$

where δ is a function that returns 1 if its argument holds, 0 otherwise.

1.2.2 Label Sequence Kernel

Next we define a kernel K_z between two label sequences $\mathbf{h}$ and $\mathbf{h}'$. We assume that two kernel functions, $K_v(v, v')$ and $K_e(e, e')$, are readily defined between vertex labels and edge labels, respectively. We constrain both kernels $K_v(v, v'), K_e(e, e') \geq 0$ to be non-negative[3]. An example of vertex label kernel is the identity kernel,

$$K_v(v, v') = \delta(v = v'). \tag{1.4}$$

If the labels are defined in $\mathbb{R}$, the Gaussian kernel,

$$K_v(v, v') = \exp(- \parallel v - v' \parallel^2 / 2\sigma^2) \tag{1.5}$$

could be a natural choice (Schölkopf and Smola, 2002). Edge kernels are defined similarly. The kernel for label sequences is defined as the product of label kernels when the lengths of two sequences are equal ($\ell = \ell'$):

$$K_z(\mathbf{h}, \mathbf{h}') = K_v(h_1, h'_1) \prod_{i=2}^{\ell} K_e(h_{2i-2}, h'_{2i-2}) K_v(h_{2i-1}, h'_{2i-1}). \tag{1.6}$$

If the lengths are different ($\ell \neq \ell'$), then K_z is simply zero ($K_z(\mathbf{h}, \mathbf{h}') = 0$).

The function K_z is proven to be a valid kernel function as follows: The set of all possible label sequences $\mathcal{H}$ can be divided into subsets according to their lengths as $\mathcal{H}_1, \mathcal{H}_2, \ldots$. Let us define $K_z^{(j)}$ as K_z whose domain is limited to the subset $\mathcal{H}_j \times \mathcal{H}_j$, then $K_z^{(j)}$ is a valid kernel as it is described as the tensor product of kernels (Schölkopf and Smola, 2002). Now let us expand the domain of $K_z^{(j)}$ to the whole set $\mathcal{H} \times \mathcal{H}$ by assigning zero when one of the inputs is not included in $\mathcal{H}_j$, and call it $\bar{K}_z^{(j)}$. This operation is called zero extension (Haussler, 1999), which preserves positive semidefiniteness. Since K_z is the sum of all $\bar{K}_z^{(j)}$'s, it turns out to be a valid kernel.

Finally, our label sequence kernel is defined as the expectation of K_z over all possible $\mathbf{h}$ and $\mathbf{h}'$.

$$K(G, G') = \sum_{\mathbf{h}} \sum_{\mathbf{h}'} K_z(\mathbf{h}, \mathbf{h}') p(\mathbf{h}|G) p(\mathbf{h}'|G'). \tag{1.7}$$

3. This constraint will play an important role in proving the convergence of our kernel in Section 1.2.5.

This kernel is valid, because it is described as an inner product of two vectors $p(\mathbf{h}|G)$ and $p(\mathbf{h}'|G')$.

1.2.3 Efficient Computation of Label Sequence Kernels

The label sequence kernel (1.7) can be expanded as

$$K(G,G') = \sum_{\ell=1}^{\infty} \sum_{\mathbf{h}} \sum_{\mathbf{h}'} \left(K_v(h_1,h'_1) \prod_{i=2}^{\ell} K_e(h_{2i-2},h'_{2i-2}) K_v(h_{2i-1},h'_{2i-1}) \right) \times \left(\sum_{\mathbf{x}} \delta(\mathbf{h}=\mathbf{h}_{\mathbf{x}}) \cdot \left(p_s(x_1) \prod_{i=2}^{\ell} p_t(x_i|x_{i-1}) p_q(x_\ell) \right) \right) \times \left(\sum_{\mathbf{x}'} \delta(\mathbf{h}=\mathbf{h}_{\mathbf{x}'}) \cdot \left(p_s(x'_1) \prod_{i=2}^{\ell} p_t(x'_i|x'_{i-1}) p_q(x'_\ell) \right) \right),$$

where $\sum_{\mathbf{h}} := \sum_{h_1 \in \Sigma_V} \sum_{h_2 \in \Sigma_E} \cdots \sum_{h_{2\ell-1} \in \Sigma_V}$ and $\sum_{\mathbf{x}} := \sum_{x_1=1}^{|\mathcal{X}|} \cdots \sum_{x_\ell=1}^{|\mathcal{X}|}$. The straightforward enumeration of all terms to compute the sum takes probibitive computational cost. For cyclic graphs, it is simply impossible because ℓ spans from 1 to infinity. Nevertheless there is an efficient method to compute this kernel as shown below. The method are based on the observation that the kernel has the following nested structure.

$$K(G,G') = \lim_{L\to\infty} \sum_{\ell=1}^{L} \sum_{x_1,x'_1} s(x_1,x'_1) \left(\sum_{x_2,x'_2} t(x_2,x'_2,x_1,x'_1) \left(\sum_{x_3,x'_3} t(x_3,x'_3,x_2,x'_2) \times \left(\cdots \left(\sum_{x_\ell,x'_\ell} t(x_\ell,x'_\ell,x_{\ell-1},x'_{\ell-1}) q(x_\ell,x'_\ell) \right) \right) \cdots \right) \right), \tag{1.8}$$

where

$$\begin{aligned} s(x_1,x'_1) &:= p_s(x_1) p'_s(x'_1) K_v(v_{x_1}, v'_{x'_1}) \\ t(x_i,x'_i,x_{i-1},x'_{i-1}) &:= p_t(x_i|x_{i-1}) p'_t(x'_i|x'_{i-1}) K_v(v_{x_i}, v'_{x'_i}) K_e(e_{x_{i-1}x_i}, e_{x'_{i-1}x'_i}) \\ q(x_\ell,x'_\ell) &:= p_q(x_\ell) p'_q(x'_\ell). \end{aligned} \tag{1.9}$$

1.2.3.1 Acyclic Graphs

Let us first consider acyclic graphs, that is, directed graphs without cycles. Precisely, it means that if there is a directed path from vertex x_1 to x_2 then there is no directed paths from vertex x_2 to x_1. When a directed graph is acyclic, the vertices can be numbered in a topological order[4] such that every edge from a vertex numbered i to a vertex numbered j satisfies $i < j$ (see Figure 1.2).

Since there are no directed paths from vertex j to vertex i if $i < j$, we can employ

4. Topological sorting of graph G can be done in $O(|\mathcal{X}| + |\mathcal{L}|)$ (Cormen et al., 1990).

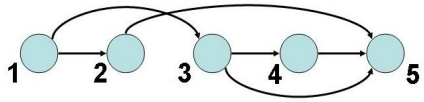

Figure 1.2 A topologically sorted directed acyclic graph. The label sequence kernel can be efficiently computed by dynamic programming running from right to left.

dynamic programming. When G and G' are directed acyclic graphs, (1.8) can be written as

$$K(G,G') = \sum_{x_1.x_1'} s(x_1,x_1')q(x_1,x_1') + \lim_{L\to\infty} \sum_{\ell=2}^{L} \sum_{x_1,x_1'} s(x_1,x_1') \times \left(\sum_{x_2>x_1,x_2'>x_1'} t(x_2,x_2',x_1,x_1') \left(\sum_{x_3>x_2,x_3'>x_2'} t(x_3,x_3',x_2,x_2') \times \left(\cdots \left(\sum_{x_\ell>x_{\ell-1},x_\ell'>x_{\ell-1}'} t(x_\ell,x_\ell',x_{\ell-1},x_{\ell-1}')q(x_\ell,x_\ell') \right) \right) \cdots \right) \right). \tag{1.10}$$

The first and second terms correspond to the label sequences of length one and those longer than one, respectively. By defining

$$r(x_1,x_1') := q(x_1,x_1') + \lim_{L\to\infty} \sum_{\ell=2}^{L} \left(\sum_{x_2>x_1,x_2'>x_1'} t(x_2,x_2',x_1,x_1') \times \left(\cdots \left(\sum_{x_\ell>x_{\ell-1},x_\ell'>x_{\ell-1}'} t(x_\ell,x_\ell',x_{\ell-1},x_{\ell-1}')q(x_\ell,x_\ell') \right) \right) \cdots \right), \tag{1.11}$$

we can rewrite (1.10) as the following:

$$K(G,G') = \sum_{x_1,x_1'} s(x_1,x_1')r(x_1,x_1').$$

The merit of defining (1.11) is that we can exploit the following recursive equation.

$$r(x_1,x_1') = q(x_1,x_1') + \sum_{j>x_1,j'>x_1'} t(j,j',x_1,x_1')r(j,j'). \tag{1.12}$$

Since all vertices are topologically ordered, $r(x_1,x_1')$ for all x_1 and x_1' can be efficiently computed by dynamic programming (Figure 1.3). The worst case time complexity of computing $K(G,G')$ is $O(c\cdot c'\cdot|\mathcal{X}|\cdot|\mathcal{X}'|)$ where c and c' are the maximum out degree of G and G', respectively.

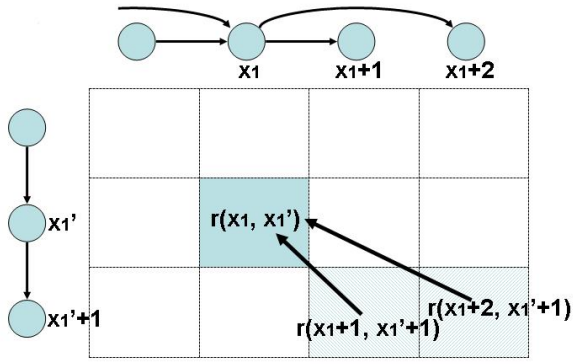

Figure 1.3 Recursion for computing $r(x_1, x_1')$ using recursive equation (1.12). $r(x_1, x_1')$ can be computed based on the precomputed values of $r(x_2, x_2')$, $x_2 > x_1$, $x_2' > x_1'$.

1.2.3.2 General Directed Graphs

In the case of cyclic graphs, we do not have topologically sorted graphs anymore. This means that we can not employ the one-pass dynamic programming algorithm for acyclic graphs. However, we can obtain a recursive form of the kernel like (1.12), and reduce the problem to solving a system of simultaneous linear equations. Let us rewrite (1.8) as

$$K(G, G') = \lim_{L \to \infty} \sum_{\ell=1}^{L} \sum_{x_1, x_1'} s(x_1, x_1') r_\ell(x_1, x_1'),$$

where for $\ell \geq 2$,

$$r_\ell(x_1, x_1') := \left(\sum_{x_2, x_2'} t(x_2, x_2', x_1, x_1') \left(\sum_{x_3, x_3'} t(x_3, x_3', x_2, x_2') \times \left(\cdots \left(\sum_{x_\ell, x_\ell'} t(x_\ell, x_\ell', x_{\ell-1}, x_{\ell-1}') q(x_\ell, x_\ell') \right) \right) \cdots \right) \right),$$

and $r_1(x_1, x_1') := q(x_1, x_1')$. Replacing the order of summation, we have the following:

$$\begin{aligned} K(G, G') &= \sum_{x_1, x_1'} s(x_1, x_1') \lim_{L \to \infty} \sum_{\ell=1}^{L} r_\ell(x_1, x_1') \\ &= \sum_{x_1, x_1'} s(x_1, x_1') \lim_{L \to \infty} R_L(x_1, x_1'), \end{aligned} \tag{1.13}$$

where

$$R_L(x_1, x_1') := \sum_{\ell=1}^{L} r_\ell(x_1, x_1').$$

Thus we need to compute $R_\infty(x_1, x_1')$ to obtain $K(G, G')$.

Now let us restate this problem in terms of linear system theory (Rugh, 1995). The following recursive relationship holds between r_k and r_{k-1} ($k \geq 2$):

$$r_k(x_1, x_1') = \sum_{i,j} t(i, j, x_1, x_1') r_{k-1}(i, j). \tag{1.14}$$

Using (1.14), the recursive relationship for R_L also holds as follows:

$$\begin{aligned} R_L(x_1, x_1') &= r_1(x_1, x_1') + \sum_{k=2}^{L} r_k(x_1, x_1') \\ &= r_1(x_1, x_1') + \sum_{k=2}^{L} \sum_{i,j} t(i, j, x_1, x_1') r_{k-1}(i, j) \\ &= r_1(x_1, x_1') + \sum_{i,j} t(i, j, x_1, x_1') R_{L-1}(i, j). \end{aligned} \tag{1.15}$$

Thus R_L can be perceived as a discrete-time linear system (Rugh, 1995) evolving as the time L increases. Assuming that R_L converges, (see Section 1.2.5 for the convergence condition), we have the following equilibrium equation:

$$R_\infty(x_1, x_1') = r_1(x_1, x_1') + \sum_{i,j} t(i, j, x_1, x_1') R_\infty(i, j). \tag{1.16}$$

Therefore, the computation of our kernel finally boils down to solving simultaneous linear equations (1.16) and substituting the solutions into (1.13).

Now let us restate the above discussion in the language of matrices. Let $\mathbf{s}$, $\mathbf{r}_1$ and $\mathbf{r}_\infty$ be $|\mathcal{X}| \cdot |\mathcal{X}'|$ dimensional vectors such that

$$\mathbf{s} = (\cdots, s(i, j), \cdots)^\top, \quad \mathbf{r}_1 = (\cdots, r_1(i, j), \cdots)^\top, \quad \mathbf{r}_\infty = (\cdots, R_\infty(i, j), \cdots)^\top, \tag{1.17}$$

respectively. Let the transition probability matrix T be a $|\mathcal{X}||\mathcal{X}'| \times |\mathcal{X}||\mathcal{X}'|$ matrix,

$$[T]_{(i,j),(k,l)} = t(i, j, k, l).$$

Equation (1.13) can be rewritten as

$$K(G, G') = \mathbf{r}_\infty^T \mathbf{s} \tag{1.18}$$

Similarly, the recursive equation (1.16) is rewritten as

$$\mathbf{r}_\infty = \mathbf{r}_1 + T\mathbf{r}_\infty.$$

The solution of this equation is

$$\mathbf{r}_\infty = (I - T)^{-1}\mathbf{r}_1.$$

Finally, the matrix form of the kernel is

$$K(G, G') = (I - T)^{-1}\mathbf{r}_1\mathbf{s}. \tag{1.19}$$

Computing the kernel requires solving a linear equation or inverting a matrix with $|\mathcal{X}||\mathcal{X}'| \times |\mathcal{X}||\mathcal{X}'|$ coefficients. However, the matrix $I - T$ is actually sparse because the number of non-zero elements of T is less than $c \cdot c' \cdot |\mathcal{X}| \cdot |\mathcal{X}'|$ where c and c' are the maximum out degree of G and G', respectively (see (1.9) for the definition of T). Therefore, we can employ efficient numerical algorithms that exploit sparsity (Barrett et al., 1994). In our implementation, we employed a simple iterative method that updates R_L by using (1.15) until convergence starting from $R_1(x_1, x'_1) = r_1(x_1, x'_1)$.

1.2.4 Allowing Multiple Edges Between Vertices

Up to this point, we assumed that there are no multiple edges from one vertex to another. However, a slight modification allows to incorporate multiple edges. Suppose that there are $M_{x_{i-1}x_i}$ directed edges from vertex x'_{i-1} to vertex x'_i with labels $e^m_{x_{i-1}x_i}$, and transition probabilities $p^m_t(x_i|x_{i-1})$ $(m = 1, 2, \ldots, M_{x_{i-1}x_i})$. Instead of (1.9), by considering all pair of $e^m_{x_{i-1}x_i}$ and $e^m_{x'_{i-1}x'_i}$, we have only to redefine $t(x_i, x'_i, x_{i-1}, x'_{i-1})$ as the following.

$$t(x_i, x'_i, x_{i-1}, x'_{i-1}) := K(v_{x_i}, v'_{x'_i}) \times \sum_{m=1}^{M_{x_{i-1}x_i}} \sum_{m'=1}^{M'_{x_{i-1}x_i}} p^m_t(x_i|x_{i-1}) p'^{m'}_t(x'_i|x'_{i-1}) K(e^m_{x_{i-1}x_i}, e^{m'}_{x'_{i-1}x'_i})$$

1.2.5 Convergence Condition

Since loops are allowed in general directed graphs, infinite number of paths can be generated. Therefore some convergence condition is needed to justify (1.16). The following theorem holds:

Theorem 1.1
The infinite sequence $\lim_{L\to\infty} R_L(x_1, x'_1)$ converges for any $x_1 \in \{1, \cdots, |\mathcal{X}|\}$ and $x'_1 \in \{1, \cdots, |\mathcal{X}'|\}$, if the following inequality holds for $i_0 \in \{1, \cdots, |\mathcal{X}|\}$ and $j_0 \in \{1, \cdots, |\mathcal{X}'|\}$,

$$\sum_{i=1}^{|\mathcal{X}|} \sum_{j=1}^{|\mathcal{X}'|} t(i, j, i_0, j_0) q(i, j) < q(i_0, j_0). \tag{1.20}$$

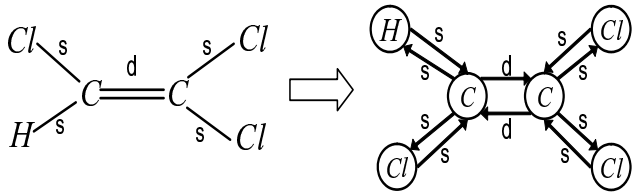

Figure 1.4 A chemical compound is conventionally represented as an undirected graph (left). Atom types and bond types correspond to vertex labels and edge labels, respectively. The edge labels 's' and 'd' denote single and double bonds, respectively. As our kernel assumes a directed graph, undirected edges are replaced by directed edges (right).

For the proof, see Kashima et al. (2003). The condition (1.20) seems rather complicated, but we can have a simpler condition, if the termination probabilities are constant over all vertices.

Corollary 1.2
If $p_q(i) = p'_q(j) = \gamma$ for any i and j, the infinite sequence $\lim_{L\to\infty} R_L(x_1, x'_1)$ converges if

$$K_v(v, v')K_e(e, e') < \frac{1}{(1-\gamma)^2}. \tag{1.21}$$

Apparently, the above lemma holds if $0 \leq K_v, K_e \leq 1$. Standard label kernels such as (1.4) and (1.5) satisfy this condition.

1.3 Experiments

We applied our kernel to the prediction of properties of chemical compounds. A chemical compound can naturally be represented as an undirected graph by considering the atom types as the vertex labels, for example C, Cl and H, and the bond types as the edge labels, for example s (single bond) and d (double bond). For our graph kernel, we replaced an undirected edge by two directed edges (Figure 1.4) since the kernel assumes directed graphs.

1.3.1 Pattern Discovery Algorithm

We compare our graph kernel with the pattern-discovery (PD) method by Kramer and De Raedt (2001) which is one of the best state-of-the-art methods in predictive toxicology. Like our graph kernel, the PD method counts the number of label

sequences appearing in the graph[5]. There are other methods which count more complicated substructures such as subgraphs (Inokuchi et al., 2000), but we focus on Kramer and De Raedt (2001) whose features are similar to ours.

Assume that we have n graphs $G_1, \ldots, G_n$. Let us define $\#(\mathbf{h}, G)$ as the number of appearances of a label sequence $\mathbf{h}$ in G. The PD method identifies the set of all label sequences $\mathcal{H}$ which appear in more than m graphs:

$$\mathcal{H} = \{\mathbf{h} \mid \sum_{i=1}^{n} \delta\left(\#\left(\mathbf{h}, G_i\right) > 0\right) \geq m\},$$

where the parameter m is called the minimum support parameter. Furthermore, it is possible to add extra conditions, for example selecting only the paths frequent in a certain class and scarce in the other classes. Each graph G is represented by a vector as

$$G \rightarrow (\#(\mathbf{x}_1, G), \ldots, \#(\mathbf{x}_{|\mathcal{H}|}, G)), \tag{1.22}$$

whose dimensionality is the cardinality of $\mathcal{H}$. The PD method is useful for extracting comprehensive features. However, as the minimum support parameter gets smaller, the dimensionality of the feature vectors becomes so large that a prohibitive amount of computation is required. Therefore, the user has to control the minimum support parameter m, such that the feature space does not lose necessary information and, at the same time, computation stays feasible.

The PD method contrasts markedly with our method. Our kernel method puts emphasis on dealing with infinite, but less interpretable features, while the PD method tries to extract a relatively small number of meaningful features. Looking at the algorithms, our method is described by just one equation (1.16), while the PD method's algorithm is rather complicated (De Raedt and Kramer, 2001).

1.3.2 Datasets

We used two datasets, the PTC dataset (Helma et al., 2001) and the Mutag dataset (Srinivasan et al., 1996). The PTC dataset is the results of the following pharmaceutical experiments. 417 compounds were given to four types of test animals: Male Mouse (MM), Female Mouse (FM), Male Rat (MR) and Female Rat (FR). According to their carcinogenicity, each compound is assigned one of the following labels: $\{\mathrm{EE}, \mathrm{IS}, \mathrm{E}, \mathrm{CE}, \mathrm{SE}, \mathrm{P}, \mathrm{NE}, \mathrm{N}\}$ where CE, SE and P indicate "relatively active", and NE and N indicate "relatively inactive", and EE, IS and E indicate "can not be decided". In order to simplify the problem, we relabeled CE, SE and P as "positive", and NE and N as "negative". The task is to predict whether a given compound is positive or negative for each type of test animals.

5. Notice that the definition of label sequences is different from ours in several points, for example a vertex will not be visited twice in a path. See Kramer and De Raedt (2001) for details.

Table 1.1 Several statistics of the datasets such as numbers of positive examples (#positive) and negative examples (#negative), maximum degree (max. degree), maximum size of graphs (max. $|\mathcal{X}|$), average size of graphs (avg. $|\mathcal{X}|$), and numbers of vertex labels ($|\Sigma_V|$) and edge labels ($|\Sigma_E|$).

	MM	FM	MR	FR	MUTAG
#POSITIVE	129	143	152	121	125
#NEGATIVE	207	206	192	230	63
MAX. $\lvert\mathcal{X}\rvert$	109	109	109	109	40
AVG. $\lvert\mathcal{X}\rvert$	25.0	25.2	25.6	26.1	31.4
MAX. DEGREE	4	4	4	4	4
$\lvert\Sigma_V\rvert$	21	19	19	20	8
$\lvert\Sigma_E\rvert$	4	4	4	4	4

Thus we eventually had four two-class problems. In the Mutag dataset, the task is a two-class classification problem to predict whether each of the 188 compounds has mutagenicity or not. Each statistics of the datasets are summarized in Table 1.1.

1.3.3 Experimental Settings and Results

Assuming no prior knowledge, we defined the probability distributions for random walks as follows. The initial probabilities were simply uniform, i.e. $p_s(x) = 1/|\mathcal{X}|$. The termination probabilities were determined as a constant γ over all vertices. The transition probabilities $p_t(x|x_0)$ were set as uniform over adjacent vertices. We used (1.4) as the label kernels. In solving the simultanous equations, we employed a simple iterative method (1.15). In our observation, 20-30 iterations were enough for convergence in all cases. For the classification algorithm, we used the voted kernel perceptron (Freund and Shapire, 1999), whose performance is known to be comparable to SVMs. In the pattern discovery method, the minimum support parameter was determined as $0.5, 1, 3, 5, 10, 20\%$ of the number of compounds, and the simple dot product in the feature space (1.22) was used as a kernel. In our graph kernel, the termination probability γ was changed from 0.1 to 0.9.

Table 1.2 and Table 1.3 show the classification accuracies in the five two-class problems measured by leave-one-out cross validation. No general tendencies were found to conclude which method is better (the PD was better in MR, FR and Mutag, but our method was better in MM and FM). Thus it would be fair to say that the performances were comparable in this small set of experiments. Even though we could not show that our method is constantly better, this result is still appealing, because the advantage of our method lies in its simplicity both in concepts and in computational procedures.

1.4 Related Works

Table 1.2 Classification accuracies (%) of the pattern discovery method. 'MinSup' shows the ratio of the minimum support parameter to the number of compounds m/n.

MinSup	MM	FM	MR	FR	Mutag
0.5%	60.1	57.6	61.3	**66.7**	88.3
1 %	**61.0**	**61.0**	**62.8**	63.2	87.8
3 %	58.3	55.9	60.2	63.2	**89.9**
5 %	60.7	55.6	57.3	63.0	86.2
10 %	58.9	58.7	57.8	60.1	84.6
20%	**61.0**	55.3	56.1	61.3	83.5

Table 1.3 Classification accuracies (%) of our graph kernel. The parameter γ is the termination probability of random walks, which controls the effect of the length of label sequences.

γ	MM	FM	MR	FR	Mutag
0.1	62.2	59.3	57.0	62.1	84.3
0.2	62.2	61.0	57.0	62.4	83.5
0.3	64.0	61.3	56.7	62.1	**85.1**
0.4	**64.3**	61.9	56.1	63.0	**85.1**
0.5	64.0	61.3	56.1	64.4	83.5
0.6	62.8	61.9	54.4	65.8	83.0
0.7	63.1	62.5	54.1	63.2	81.9
0.8	63.4	**63.4**	54.9	64.1	79.8
0.9	62.8	61.6	**58.4**	**66.1**	78.7

We have presented one kernel for graphs based on label sequences, but variants can be obtained by changing the following two points:

- Removing probabilistic constraints: In our setting, the random walk parameters are determined such that the probabilities of all label sequences sum to one. One can remove these constraints and simply consider transition "weights", not probabilities.
- Changing the rate of weight decay: The probabilities (or weights) associated with a label sequence could decay as the length of the sequence increases. Variants can be obtained by introducing an extra decaying factor depending on the sequence length.

Recently, Gärtner et al. (2003) proposed two graph kernels called *geometric* and *exponential kernels*. Let $w_t(x_i|x_{i-1})$ denote a weight of for transition from x_{i-1} to x_i. Their kernels can be recovered in our framework by setting $p_s(\cdot) = 1$, $p_q(\cdot) = 1$

and replacing the transition probability $p_t(x_i|x_{i-1})$ with

$$\sqrt{\lambda_k} w_t(x_i|x_{i-1})$$

where λ_k is the decaying factor depending on the current sequence length k. In our setting, when the random walk passes through an edge, the probability is multiplied by the same factor regardless of the current sequence length. However, in their setting, the decay rate may get larger when the edge is visited *later*, i.e., after traversing many vertices.

In the geometric kernel, λ_k does not depend on k, i.e., $\lambda_k = \lambda$. This kernel is quite similar to our kernel and is computed by means of matrix inversion as in (1.19). An interesting kernel called the exponential kernel is derived when

$$\lambda_k = \frac{\beta}{k}.$$

It turns out that this kernel is computed efficiently by the matrix exponential:

$$K(G, G') = \sum_i \sum_j \left[\lim_{L\to\infty} \sum_{\ell=1}^{L} \frac{(\beta T)^\ell}{\ell!} \right]_{ij} = \sum_i \sum_j \left[e^{\beta T} \right]_{ij}.$$

Obviously possible variants are not limited to these two cases, so there remains a lot to explore.

Label sequence kernels have an intrinsic relationship to the kernels between probability distributions called *probability product kernels* (Jebara and Kondor, 2003). Here the kernel between two probability distributions $\mathbf{p}$ and $\mathbf{p}'$ is defined as

$$K(p, p') = \int_\Omega p(\mathbf{x})^\rho p'(\mathbf{x})^\rho d\mathbf{x} \tag{1.23}$$

When $\rho = 1$, the kernel is called *expected likelihood kernel*, and probability with which both two probability distributions generate $\mathbf{x}$ independently. Also, when $\rho = 1/2$, the kernel is called Bhattacharrya kernel, which is related to the Hellinger distance. In fact, when edges are not labeled and the vertex kernel is determined as the identity kernel, our kernel can be regarded as the expected likelihood kernel between two Markov models. In such cases. the graph G is perceived as a transition graph of a Markov model and random walking amounts to the emission of symbols. The same idea can be extended to define a kernel for hidden markov models(Lyngsø et al., 1999). An HMM can be regarded as a labeled graph where edges are not labeled and vertices are *probabilistically* labeled, that is, a vertex randomly emits one of the symbols according to some probability distribution.

If we regard the kernel $K_z(\mathbf{h}, \mathbf{h}')$ as a joint distribution $p_z(\mathbf{h}, \mathbf{h}')$ that emits a pair of sequences $\mathbf{h}$ and $\mathbf{h}'$, it can be an instance of *rational kernels* (Cortes et al., 2003),

$$K(\mathbf{x}, \mathbf{x}') = \sum_{\mathbf{h}} \sum_{\mathbf{h}'} p_z(\mathbf{h}, \mathbf{h}') p(\mathbf{h}|\mathbf{x}) p'(\mathbf{h}'|\mathbf{x}'),$$

that define a kernel between two probabilistic automata $p(\mathbf{h}|\mathbf{x})$ and $p'(\mathbf{h}'|\mathbf{x}')$ via probabilistic transducer $p_z(\mathbf{h}, \mathbf{h}')$. The rational kernels are not limited to the probabilistic setting, and provide an unified framework for designing kernels via weighted transducers. Cortes et al. (2003) provided no algorithms for acyclic cases. The techniques we introduced in this chapter can be easily applied to the rational kernels for cyclic cases.

1.5 Conclusion

This chapter discussed the design of kernel functions between directed graphs with vertex labels and edge labels. We defined the label sequence kernel by using random walks on graphs, and reduced the computation of the kernel to solving a system of simultaneous linear equations. In contrast to the pattern-discovery method, our kernel takes into account all possible label sequences without computing feature values explicitly. The structure we dealt with in this paper is fairly general, and promising in a wide variety of problems in bioinformatics. Potential targets would be DNA and RNA sequences with remote correlations, HTML and XML documents in MEDLINE, topology graphs and distance graphs of 3-D protein structures, just to mention some.

References

S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: from relations to semistructured data and XML.* Morgan Kaufmann, 2000.

C. Bahlmann, B. Haasdonk, and H. Burkhardt. On-line handwriting recognition with support vector machines – a kernel approach. In *Proc. of the 8th Int. Workshop on Frontiers in Handwriting Recognition (IWFHR)*, pages 49–54, 2002.

R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition.* SIAM, Philadelphia, PA, 1994.

M. Collins and N. Duffy. Convolution kernel for natural language. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, Cambridge, MA, 2002. MIT Press.

T. C. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, MA, 1990.

Corinna Cortes, Patrick Haffner, and Mehryer Mohri. Rational kernels. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15.* MIT Press, 2003. In press.

L. De Raedt and S. Kramer. The levelwise version space algorithm and its application to molecular fragment finding. In B. Nebel, editor, *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 853–862. Morgan Kaufmann, 2001.

R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis - Probabilistic models of proteins and nucleic acids.* Cambridge University Press, Cambridge, UK, 1998.

Y. Freund and R. Shapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296, 1999.

T. Gärtner. Exponential and geometric kernels for graphs. In *NIPS*02 Workshop on Unreal Data: Principles of Modeling Nonvectorial Data*, 2002. Available from http://mlg.anu.edu.au/unrealdata/.

T. Gärtner, P. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Proc. of the Sixteenth Annual Conference on Computational Learning Theory*, 2003.

D. Haussler. Convolution kernels on discrete structures. Technical Report UCSC-CRL-99-10, University of California, Santa Cruz, Santa Cruz, CA, July 1999.

C. Helma, R.D. King, S. Kramer, and A. Srinivasan. The Predictive Toxicology Challenge 2000-2001. *Bioinformatics*, 17(1):107–108, 2001.

L. Holm and C. Sander. Protein structure comparison by alignment of distance matrices. *Journal of Molecular Biology*, 233(1):123–38, 1993.

A. Inokuchi, T. Washio, and H. Motoda. An Apriori-based algorithm for mining frequent substructures from graph data. In *Proc. of the 4th PKDD*, pages 13–23, 2000. URL `citeseer.nj.nec.com/inokuchi00aprioribased.html`.

T. Jaakkola, M. Diekhans, and D. Haussler. A discriminative framework for detecting remote protein homologies. *Journal of Computational Biology*, 7(1-2): 95–114, 2000.

T. Jebara and R. Kondor. Bhattacharyya and expected likelihood kernels. In *Proc. of the Sixteenth Annual Conference on Computational Learning Theory*, 2003.

Jaz Kandola, John Shawe-Taylor, and Nello Cristianini. Learning semantic similarity. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*. MIT Press, 2003. In press.

H. Kashima and A. Inokuchi. Kernels for graph classification. In *IEEE ICDM Workshop on Active Mining*. Maebashi, Japan, 2002.

H. Kashima and T. Koyanagi. Kernels for semi-structured data. In *Proceedings of the Ninteenth International Conference on Machine Learning*, pages 291–298, 2002.

H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labeled graphs. In *Proceedings of the Twentieth International Conference on Machine Learning*, 2003.

S. Kramer and L. De Raedt. Feature construction with version spaces for biochemical application. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 258–265, 2001.

John Lafferty and Guy Lebanon. Information diffusion kernels. In Suzanna Becker, Sebastian Thrun, and Klaus Obermayer, editors, *Advances in Neural Information Processing Systems*, volume 15. The MIT Press, 2003.

C. Leslie, E. Eskin, J. Weston, and W. S. Noble. Mismatch string kernels for SVM protein classification. In Suzanna Becker, Sebastian Thrun, and Klaus Obermayer, editors, *Advances in Neural Information Processing Systems*, volume 15. The MIT Press, 2003.

Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, 2:419–444, 2002.

R. B. Lyngsø, C. N. S. Pedersen, and H. Nielsen. Metrics and similarity measures for hidden markov models. In *Proc. of the Seventh International Conference on Intelligent Systems for Molecular Biology*, 1999.

Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, Massachusetts, 1999. URL `citeseer.nj.nec.com/540990.html`.

W.J. Rugh. *Linear System Theory*. Prentice Hall, 1995.

B. Schölkopf and A. J. Smola. *Learning with Kernels*. The MIT Press, Cambridge, MA, 2002.

H. Shimodaira, K.-I. Noma, M. Nakai, and S. Sagayama. Dynamic time-alignment kernel in support vector machine. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 921–928, Cambridge, MA, 2002. MIT Press.

A. Srinivasan, S. Muggleton, R. D. King, and M. Sternberg. Theories for mutagenicity: a study of first-order and feature based induction. *Artificial Intelligence*, 85(1-2):277–299, 1996.

E. Takimoto and M. K. Warmuth. Path kernels and multiplicative updates. In *Proc. of the Fifteenth Annual Conference on Computational Learning Theory*, pages 74–89, 2002.

K. Tsuda, T. Kin, and K. Asai. Marginalized kernels for biological sequences. *Bioinformatics*, 18:S268–S275, 2002.

S. V. N. Vishwanathan and A. J. Smola. Fast kernels for string and tree matching. In Suzanna Becker, Sebastian Thrun, and Klaus Obermayer, editors, *Advances in Neural Information Processing Systems*, volume 15. The MIT Press, 2003.

C. Watkins. Dynamic alignment kernels. In A.J. Smola, P.L. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 39–50. MIT Press, 2000.