Journal of Machine Learning Research 10 (2009) 1–41 Submitted 05/08; Published xx/09

Graph Kernels

S.V. N. Vishwanathan VISHY@STAT.PURDUE.EDU
Departments of Statistics and Computer Science
Purdue University
250 N University Street, West Lafayette, IN 47907-2066, USA

Nicol N. Schraudolph SCHRAUDOLPH@ADAPTIVETOOLS.COM
adaptive tools AG
Canberra ACT 2602, Australia

Risi Kondor RISI@GATSBY.UCL.AC.UK
Gatsby Computational Neuroscience Unit
University College London
17 Queen Square, WC1N 3AR London, United Kingdom

Karsten M. Borgwardt KARSTEN.BORGWARDT@TUEBINGEN.MPG.DE
Interdepartmental Bioinformatics Group,
Max Planck Institute for Developmental Biology and
Max Planck Institute for Biological Cybernetics
Spemannstr. 38, 72076 Tübingen, Germany

Editor: John Lafferty

Abstract

We present a unified framework to study graph kernels, special cases of which include the random walk (Gärtner et al., 2003; Borgwardt et al., 2005), marginalized (Kashima et al., 2003, 2004; Mahé et al., 2004), and geometric (Gärtner, 2002) graph kernels. Through extensions of linear algebra to Reproducing Kernel Hilbert Spaces (RKHS) and reduction to a Sylvester equation, we construct an algorithm that improves the time complexity of kernel computation from $O(n^6)$ to $O(n^3)$. When the graphs are sparse, conjugate gradient solvers or fixed-point iterations bring our algorithm into the sub-cubic domain. Experiments on graphs from bioinformatics and other application domains show that this can speed up computing the kernel by an order of magnitude or more. We also show that rational kernels (Cortes et al., 2002, 2003, 2004) when specialized to graphs reduce to our random walk graph kernel. Finally, we relate our framework to R-convolution kernels (Haussler, 1999) and provide a kernel that is close to the optimal assignment kernel of Fröhlich et al. (2006) yet provably positive semi-definite.

Keywords: Linear Algebra in RKHS, Sylvester Equations, Bioinformatics, Rational Kernels, Transducers, Semirings, Random Walks.

1. Introduction

Machine learning in domains such as bioinformatics (Sharan and Ideker, 2006), chemoinformatics (Bonchev and Rouvray, 1991), drug discovery (Kubinyi, 2003), web data mining (Washio and Motoda, 2003), and social networks (Kumar et al., 2006) involves the study

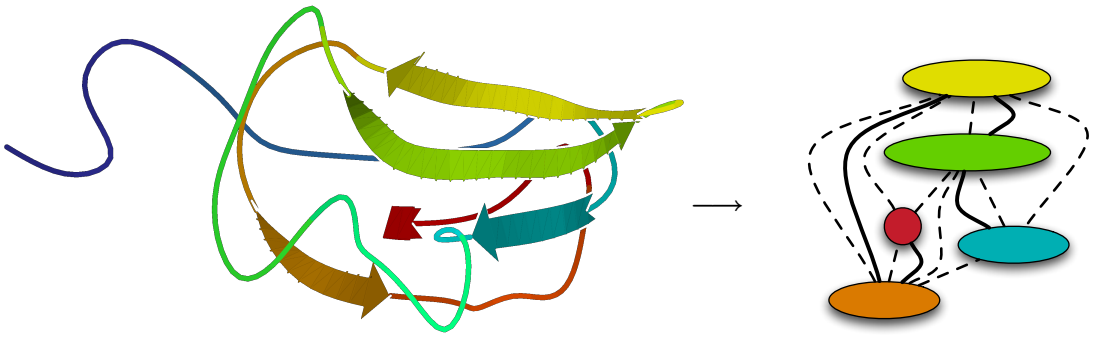

Figure 1: Left: Structure of *E. coli* protein fragment APO-BCCP87 (Yao et al., 1997), ID 1a6x in the Protein Data Bank (Berman et al., 2000). Right: Borgwardt et al.'s (2005) graph representation for this protein fragment. Nodes represent secondary structure elements, and edges encode neighborhood along the amino acid chain (solid) *resp.* in Euclidean 3D space (dashed).

of relationships between structured objects. Graphs are natural data structures to model such structures, with nodes representing objects and edges the relations between them. In this context, one often encounters two questions: "How similar are two nodes in a given graph?" and "How similar are two graphs to each other?"

In protein function prediction, for instance, one might want to predict whether a given protein is an enzyme or not. Computational approaches infer protein function by finding proteins with similar sequence, structure, or chemical properties. A very successful recent method is to model the protein as a graph (see Figure 1), and assign similar functions to similar graphs (Borgwardt et al., 2005). In Section 5.2 we compute graph kernels to measure the similarity between proteins and enzymes represented in this fashion.

Another application featured in Section 5.2 involves predicting the toxicity of chemical molecules by comparing their three-dimensional structure. Here the molecular structure is modeled as a graph, and the challenge is to compute the similarity between molecules of known and unknown toxicity.

Finally, consider the task of finding web pages with related content. Since documents on the web link to each other, one can model each web page as the node of a graph, and each link as an edge. Now the problem becomes that of computing similarities between the nodes of a graph. Taking this one step further, detecting mirrored sets of web pages requires computing the similarity between the graphs representing them.

Kernel methods (Schölkopf and Smola, 2002) offer a natural framework to study these questions. Roughly speaking, a kernel $k(x, x')$ is a measure of similarity between objects x and x'. It must satisfy two mathematical requirements: it must be symmetric, that is, $k(x, x') = k(x', x)$, and positive semi-definite (*p.s.d.*). Comparing nodes in a graph involves constructing a kernel between nodes, while comparing graphs involves constructing a kernel between graphs. In both cases, the challenge is to define a kernel that captures the semantics inherent in the graph structure and is reasonably efficient to evaluate.

The idea of constructing kernels *on* graphs (*i.e.,* between the nodes of a single graph) was first proposed by Kondor and Lafferty (2002), and extended by Smola and Kondor (2003). In contrast, in this paper we focus on kernels *between* graphs. The first such kernels were proposed by Gärtner (2002) (geometric kernel) and Gärtner et al. (2003) (random walk kernel), and later extended by Borgwardt et al. (2005). Much at the same time, the idea of marginalized kernels (Tsuda et al., 2002) was extended to graphs by Kashima et al. (2003, 2004), and further refined by Mahé et al. (2004). Another algebraic approach to graph kernels has appeared recently (Kondor and Borgwardt, 2008). A seemingly independent line of research investigates the so-called rational kernels, which are kernels between finite state automata based on the algebra of abstract semirings (Cortes et al., 2002, 2003, 2004).

The aim of this paper is twofold: on the one hand we present theoretical results showing that all the above graph kernels are in fact closely related, on the other hand we present new algorithms for efficiently computing such kernels. We begin by establishing some notation and reviewing pertinent concepts from linear algebra and graph theory.

1.1 Paper Outline

The first part of this paper (Sections 2–5) elaborates on a recent conference publication (Vishwanathan et al., 2007) to present a unifying framework for graph kernels encompassing many known kernels as special cases, and to discuss connections to yet others. After defining some basic concepts in Section 2, we describe the framework in Section 3, prove that it leads to *p.s.d.* kernels, and discuss random walk, geometric, and marginalized graph kernels as special cases. For ease of exposition we will work with real matrices in the main body of the paper and relegate the RKHS extensions to Appendix A. In Section 4 we present three efficient ways to compute random walk graph kernels, namely 1. via reduction to a Sylvester equation, 2. using a conjugate gradient (CG) solver, and 3. using a fixed-point iteration. Experiments on a variety of real and synthetic datasets in Section 5 illustrate the computational advantages of our approach, which generally reduces the time complexity of kernel computations from $O(n^6)$ to $O(n^3)$. The experiments of Section 5.3 were previously presented at a bioinformatics symposium (Borgwardt et al., 2007).

The second part of the paper (Sections 6–7) draws further connections to existing kernels on structured objects. In Section 6 we present a simple proof that rational kernels (Cortes et al., 2002, 2003, 2004) are *p.s.d.*, and show that specializing them to graphs yields random walk graph kernels. In Section 7 we discuss the relation between R-convolution kernels (Haussler, 1999) and various graph kernels, all of which can in fact be shown to be instances of R-convolution kernels. Extending the framework through the use of semirings does not always result in a *p.s.d.* kernel though; a case in point is the optimal assignment kernel of Fröhlich et al. (2006). We establish sufficient conditions for R-convolution kernels in semirings to be *p.s.d.*, and provide a "mostly optimal assignment kernel" that is provably *p.s.d.* We conclude in Section 8 with an outlook and discussion.

2. Preliminaries

Here we define the basic concepts and notation from linear algebra and graph theory that will be used in the remainder of the paper.

2.1 Linear Algebra Concepts

We use $\mathbf{e}_i$ to denote the i^{th} standard basis vector (that is, a vector of all zeros with the i^{th} entry set to one), $\mathbf{e}$ to denote a vector with all entries set to one, $\mathbf{0}$ to denote the vector of all zeros, and $\mathbf{I}$ to denote the identity matrix. When it is clear from the context we will not mention the dimensions of these vectors and matrices.

Definition 1 *Given real matrices $A \in \mathbb{R}^{n\times m}$ and $B \in \mathbb{R}^{p\times q}$, the Kronecker product $A\otimes B \in \mathbb{R}^{np\times mq}$ and column-stacking operator $\operatorname{vec}(A) \in \mathbb{R}^{nm}$ are defined as*

$$A \otimes B := \begin{bmatrix} A_{11}B & A_{12}B & \dots & A_{1m}B \\ \vdots & \vdots & \vdots & \vdots \\ A_{n1}B & A_{n2}B & \dots & A_{nm}B \end{bmatrix}, \quad \operatorname{vec}(A) := \begin{bmatrix} A_{*1} \\ \vdots \\ A_{*m} \end{bmatrix},$$

where A_{*j} denotes the j^{th} column of A.

The Kronecker product and vec operator are linked by the well-known property (*e.g.*, Bernstein, 2005, proposition 7.1.9):

$$\operatorname{vec}(ABC) = (C^\top \otimes A)\operatorname{vec}(B). \tag{1}$$

Another well-known property of the Kronecker product which we make use of is (Bernstein, 2005, proposition 7.1.6):

$$(A \otimes B)(C \otimes D) = AC \otimes BD. \tag{2}$$

Finally, the Hadamard product of two real matrices $A, B \in \mathbb{R}^{n\times m}$, denoted by $A \odot B \in \mathbb{R}^{n\times m}$, is obtained by element-wise multiplication. It interacts with the Kronecker product via

$$(A \otimes B) \odot (C \otimes D) = (A \odot C) \otimes (B \odot D). \tag{3}$$

All the above concepts can be extended to a Reproducing Kernel Hilbert Space (RKHS) (See Appendix A for details).

2.2 Graph Concepts

A graph G consists of an ordered set of n vertices $V = \{v_1, v_2, \dots, v_n\}$, and a set of directed edges $E \subset V \times V$. A vertex v_i is said to be a neighbor of another vertex v_j if they are connected by an edge, *i.e.,* if $(v_i, v_j) \in E$; this is also denoted $v_i \sim v_j$. We do not allow self-loops, *i.e.,* $(v_i, v_i) \notin E$ for any i. A walk of length k on G is a sequence of indices $i_0, i_1, \dots i_k$ such that $v_{i_{r-1}} \sim v_{i_r}$ for all $1 \le r \le k$. A graph is said to be strongly connected if any two pairs of vertices can be connected by a walk. In this paper we will always work with strongly connected graphs. A graph is said to be undirected if $(v_i, v_j) \in E \iff (v_j, v_i) \in E$.

In much of the following we will be dealing with weighted graphs, which are a slight generalization of the above. In a weighted graph, each edge (v_i, v_j) has an associated weight $w_{ij} > 0$ signifying its "strength". If v_i and v_j are not neighbors, then $w_{ij} = 0$. In an undirected weighted graph $w_{ij} = w_{ji}$.

When G is unweighted, we define its adjacency matrix as the $n \times n$ matrix $\widetilde{\mathrm{A}}$ with $\widetilde{\mathrm{A}}_{ij} = 1$ if $v_j \sim v_i$, and 0 otherwise. For weighted graphs, $\widetilde{\mathrm{A}}_{ij} = w_{ji}$. While some authors would call these matrices the transpose of the adjacency matrix, for our purposes the present defnitions will be more convenient. For undirected graphs $\widetilde{\mathrm{A}}$ is symmetric, and the two definitions coincide. The diagonal entries of $\widetilde{\mathrm{A}}$ are always zero.

The adjacency matrix has a normalized cousin, defined $A := \widetilde{\mathrm{A}}\, D^{-1}$, which has the property that each of its columns sums to one, and it can therefore serve as the transition matrix for a stochastic process. Here, D is a diagonal matrix of node degrees, *i.e.,* $D_{ii} = d_i = \sum_j \widetilde{\mathrm{A}}_{ij}$. A random walk on G is a process generating sequences of vertices $v_{i_1}, v_{i_2}, v_{i_3}, \ldots$ according to $\mathbb{P}(i_{k+1}|i_1, \ldots i_k) = A_{i_{k+1},i_k}$, that is, the probability at v_{i_k} of picking $v_{i_{k+1}}$ next is proportional to the weight of the edge $(v_{i_k}, v_{i_{k+1}})$. The t^{th} power of A thus describes t-length walks, *i.e.,* $(A^t)_{ij}$ is the probability of a transition from vertex v_j to vertex v_i via a walk of length t. If p_0 is an initial probability distribution over vertices, then the probability distribution p_t describing the location of our random walker at time t is $p_t = A^t p_0$. The j^{th} component of p_t denotes the probability of finishing a t-length walk at vertex v_j. We will use this intuition to define generalized random walk graph kernels.

Let $\mathcal{X}$ be a set of labels which includes the special label ζ. Every edge-labeled graph G is associated with a label matrix $X \in \mathcal{X}^{n \times n}$ in which X_{ij} is the label of the edge (v_j, v_i) and $X_{ij} = \zeta$ if $(v_j, v_i) \notin E$. Let $\mathcal{H}$ be the RKHS induced by a *p.s.d.* kernel $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, and let $\phi : \mathcal{X} \to \mathcal{H}$ denote the corresponding feature map, which we assume maps ζ to the zero element of $\mathcal{H}$. We use $\Phi(X)$ to denote the feature matrix of G (see Appendix A for details). For ease of exposition we do not consider labels on vertices here, though our results hold for that case as well. Henceforth we use the term labeled graph to denote an edge-labeled graph.

Two graphs $G = (V, E)$ and $G' = (V', E')$ are *isomorphic* (denoted by $G \cong G'$) if there exists a bijective mapping $g : V \to V'$ (called the isomorphism function) such that $(v_i, v_j) \in E$ iff $(g(v_i), g(v_j)) \in E'$.

3. Random Walk Graph Kernels

Our generalized random walk graph kernels are based on a simple idea: given a pair of graphs, perform random walks on both, and count the number of matching walks. We show that this simple concept underlies random walk, marginalized, and geometric graph kernels. In order to do this, we first need to introduce direct product graphs.

3.1 Direct Product Graphs

Given two graphs $G(V, E)$ and $G'(V', E')$, their direct product $G_\times$ is a graph with vertex set

$$V_\times = \{(v_i, v'_r) : v_i \in V,\ v'_r \in V'\}, \tag{4}$$

and edge set

$$E_\times = \{((v_i, v'_r), (v_j, v'_s)) : (v_i, v_j) \in E \wedge (v'_r, v'_s) \in E'\}. \tag{5}$$

In other words, $G_\times$ is a graph over pairs of vertices from G and G', and two vertices in $G_\times$ are neighbors if and only if the corresponding vertices in G and G' are both neighbors; see

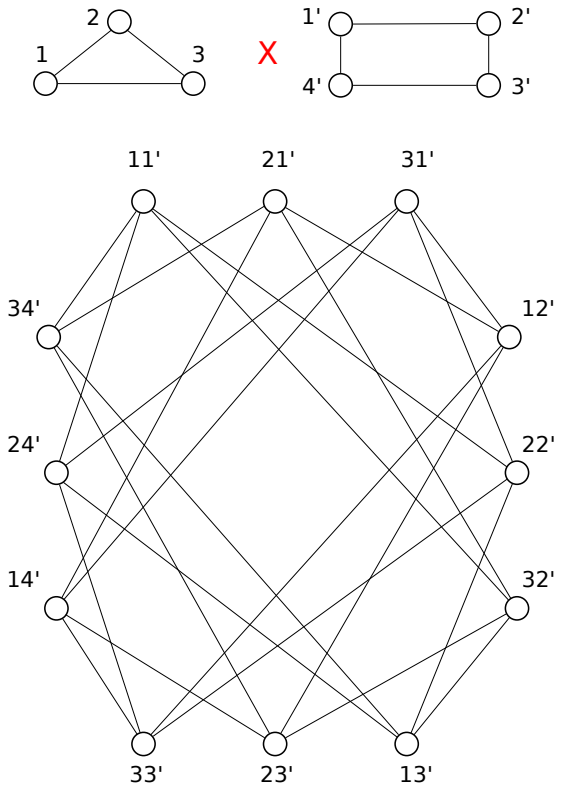

Figure 2: Two graphs (top left & right) and their direct product (bottom). Each node of the direct product graph is labeled with a pair of nodes (4); an edge exists in the direct product if and only if the corresponding nodes are adjacent in both original graphs (5). For instance, nodes $11'$ and $32'$ are adjacent because there is an edge between nodes 1 and 3 in the first, and $1'$ and $2'$ in the second graph.

Figure 2 for an illustration. If $\widetilde{\mathrm{A}}$ and $\widetilde{\mathrm{A}}'$ are the respective adjacency matrices of G and G', then the adjacency matrix of $G_\times$ is $\widetilde{\mathrm{A}}_\times = \widetilde{\mathrm{A}} \otimes \widetilde{\mathrm{A}}'$. Similarly, $A_\times = A \otimes A'$.

Performing a random walk on the direct product graph is equivalent to performing a simultaneous random walk on G and G' (Imrich and Klavžar, 2000). If p and p' denote initial probability distributions over the vertices of G and G', then the corresponding initial probability distribution on the direct product graph is $p_\times := p \otimes p'$. Likewise, if q and q' are stopping probabilities (that is, the probability that a random walk ends at a given vertex), then the stopping probability on the direct product graph is $q_\times := q \otimes q'$.

Let $|V| =: n$ and $|V'| =: n'$. If G and G' are edge-labeled, we can associate a weight matrix $W_\times \in \mathbb{R}^{nn' \times nn'}$ with $G_\times$ using our extension of the Kronecker product (Definition 1) into RKHS (Definition 11 in Appendix A):

$$W_\times = \Phi(X) \otimes \Phi(X'). \tag{6}$$

As a consequence of the definition of $\Phi(X)$ and $\Phi(X')$, the entries of $W_\times$ are non-zero only if the corresponding edge exists in the direct product graph. If we simply let $\mathcal{H} = \mathbb{R}$, $\Phi(X) = \widetilde{\mathrm{A}}$, and $\Phi(X') = \widetilde{\mathrm{A}}'$ then (6) reduces to $\widetilde{\mathrm{A}}_\times$, the adjacency matrix of the direct product graph. Normalization can be incorporated by letting $\phi(X_{ij}) = 1/d_i$ if $(v_j, v_i) \in E$ or zero otherwise.[1] Then $\Phi(X) = A$ and $\Phi(X') = A'$, and consequently $W_\times = A_\times$.

If the edges of our graphs take on labels from a finite set, without loss of generality $\{1, 2, \ldots, d\}$, we can let $\mathcal{H}$ be $\mathbb{R}^d$ endowed with the usual inner product. For each edge $(v_j, v_i) \in E$ we set $\phi(X_{ij}) = \mathbf{e}_l / d_i$ if the edge (v_j, v_i) is labeled l; all other entries of $\Phi(X)$ are $\mathbf{0}$. Thus the weight matrix (6) has a non-zero entry iff an edge exists in the direct product graph and the corresponding edges in G and G' have the same label. Let ${}^l\!A$ denote the normalized adjacency matrix of the graph filtered by the label l, that is, ${}^l\!A_{ij} = A_{ij}$ if $X_{ij} = l$, and zero otherwise. Some simple algebra (omitted for the sake of brevity) shows that the weight matrix of the direct product graph can then be written as

$$W_\times = \sum_{l=1}^{d} {}^l\!A \otimes {}^l\!A'. \tag{7}$$

In Section 4 we will develop efficient methods to compute kernels defined using the weight matrix of the direct product graph. The applicability and time complexity of a particular method will depend on whether the graphs are unlabeled ($W_\times = A_\times$), have discrete edge labels (7), or — in the most general case — employ an arbitrary edge kernel (6); see Table 1 for a summary.

3.2 Kernel Definition

As stated above, performing a random walk on the direct product graph $G_\times$ is equivalent to performing a simultaneous random walk on the graphs G and G' (Imrich and Klavžar, 2000). Therefore, the $((i-1)n' + r,\ (j-1)n' + s)^{\text{th}}$ entry of $A_\times^k$ represents the probability of simultaneous length k random walks on G (starting from vertex v_j and ending in vertex v_i) and G' (starting from vertex v'_s and ending in vertex v'_r). The entries of $W_\times$ (6) represent similarity between edges: The $((i-1)n'+r,\ (j-1)n'+s)$ entry of $W_\times^k$ represents the similarity between simultaneous length k random walks on G and G', measured via the kernel function κ. Given initial and stopping probability distributions $p_\times$ and $q_\times$ one can compute $q_\times^\top W_\times^k p_\times$, which is the expected similarity between simultaneous length k random walks on G and G'.

To define a kernel which computes the similarity between G and G', one natural idea is to simply sum up $q_\times^\top W_\times^k p_\times$ for all values of k. However, this sum might not converge, leaving the kernel value undefined. To overcome this problem, we introduce appropriately chosen non-negative coefficients $\mu(k)$, and define the kernel between G and G' as

$$k(G, G') := \sum_{k=1}^{\infty} \mu(k)\, q_\times^\top W_\times^k p_\times. \tag{8}$$

This definition is very flexible and offers the kernel designer many parameters to adjust in an application-specific manner: Appropriately choosing $\mu(k)$ allows one to (de-)emphasize

1. Strictly speaking this is not an edge-wise feature map anymore, since $\phi(X_{ij})$ depends on d_i as well as X_{ij}

walks of different lengths; if initial and stopping probabilities are known for a particular application, then this knowledge can be incorporated into the kernel; and finally, appropriate kernels or similarity measures between edges can be incorporated via the weight matrix $W_\times$. Despite its flexibility, this kernel is guaranteed to be *p.s.d.* and — as we will see in Section 4 — can be computed efficiently by exploiting the special structure of $W_\times$. To show that (8) is a valid *p.s.d.* kernel we need the following technical lemma:

Lemma 2 $\forall\, k \in \mathbb{N}: \quad W_\times^k p_\times = \mathrm{vec}[\Phi(X')^k p' \,(\Phi(X)^k p)^\top]$.

Proof By induction over k. Base case: $k = 1$. Observe that

$$p_\times = (p \otimes p')\,\mathrm{vec}(1) = \mathrm{vec}(p' p^\top). \tag{9}$$

Using (9) and Lemma 12 — which extends (1) into RKHS — $W_\times p_\times$ can be written as

$$\begin{aligned} W_\times p_\times = [\Phi(X) \otimes \Phi(X')]\,\mathrm{vec}(p' p^\top) &= \mathrm{vec}[\Phi(X') p' p^\top \Phi(X)^\top] \\ &= \mathrm{vec}[\Phi(X') p' (\Phi(X) p)^\top]. \end{aligned} \tag{10}$$

Induction from k to $k{+}1$: Using the induction assumption $W_\times^k p_\times = \mathrm{vec}[\Phi(X')^k p' \,(\Phi(X)^k p)^\top]$ and Lemma 12 we obtain

$$\begin{aligned} W_\times^{k+1} p_\times = W_\times W_\times^k p_\times &= (\Phi(X) \otimes \Phi(X'))\,\mathrm{vec}[\Phi(X')^k p' \,(\Phi(X)^k p)^\top] \\ &= \mathrm{vec}[\Phi(X')\Phi(X')^k p' \,(\Phi(X)^k p)^\top \Phi(X)^\top] \\ &= \mathrm{vec}[\Phi(X')^{k+1} p' \,(\Phi(X)^{k+1} p)^\top]. \end{aligned} \tag{11}$$

Base case (10) and induction (11) together imply Lemma 2 $\forall\, k \in \mathbb{N}$. ■

Lemma 3 *If the coefficients $\mu(k)$ are such that (8) converges, then (8) defines a valid* p.s.d. *kernel.*

Proof Using Lemmas 12 and 2 we can write

$$\begin{aligned} q_\times^\top W_\times^k p_\times &= (q \otimes q')^\top\,\mathrm{vec}[\Phi(X')^k p' \,(\Phi(X)^k p)^\top] \\ &= \mathrm{vec}[q'^\top \Phi(X')^k p' \,(\Phi(X)^k p)^\top q] \\ &= \underbrace{(q^\top \Phi(X)^k p)^\top}_{\rho_k(G)^\top} \underbrace{(q'^\top \Phi(X')^k p')}_{\rho_k(G')}. \end{aligned} \tag{12}$$

Each individual term of (12) equals $\rho_k(G)^\top \rho_k(G')$ for some function ρ_k, and is therefore a valid *p.s.d.* kernel. The lemma follows because the class of *p.s.d.* kernels is closed under non-negative linear combinations and pointwise limits (Berg et al., 1984). ■

3.3 Special Cases

Kashima et al. (2004) define a kernel between labeled graphs via walks and their label sequences. Recall that a walk of length t on G is a sequence of indices $i_1, i_2, \dots i_{t+1}$ such that $v_{i_k} \sim v_{i_{k+1}}$ for all $1 \leq k \leq t$. In our setting (where we do not consider node labels), the label sequence $h = h_1, \dots, h_t$ associated with a walk is simply the sequence of edge labels encountered during the walk. Let P denote a transition probability matrix, where P_{ij} denotes the probability of transition from node v_i to node v_j. For instance, P might be the normalized adjacency matrix of G. Furthermore, let p and q denote starting and stopping probabilities. Then one can compute the probability of a walk $i_1, i_2, \dots i_{t+1}$ and hence the label sequence h associated with it as

$$p(h|G) := q_{i_{t+1}} \prod_{j=1}^{t} P_{i_j, i_{j+1}} \, p_{i_1}. \tag{13}$$

Now let $\hat{\phi}$ denote a feature map on edge labels, and define a kernel between label sequences of length t by

$$\kappa(h, h') := \prod_{i=1}^{t} \kappa(h_i, h'_i) = \prod_{i=1}^{t} \left\langle \hat{\phi}(h_i), \hat{\phi}(h'_i) \right\rangle \tag{14}$$

if h and h' have the same length t, and zero otherwise. Using (13) and (14) we can define a kernel between graphs via marginalization:

$$k(G, G') := \sum_{h} \sum_{h'} \kappa(h, h') \, p(h|G) \, p(h|G'). \tag{15}$$

Kashima et al. (2004, Eq. 1.19) show that (15) can be written as

$$k(G, G') = q_\times^\top (\mathbf{I} - T_\times)^{-1} p_\times, \tag{16}$$

where $T_\times = [\mathrm{vec}(P)\,\mathrm{vec}(P')^\top] \odot [\hat{\Phi}(X) \otimes \hat{\Phi}(X')]$. (As usual, X and X' denote the edge label matrices of G and G', respectively, and $\hat{\Phi}$ the corresponding feature matrices.)

Although this kernel is differently motivated, it can be obtained as a special case of our framework. Towards this end, assume $\mu(k) = \lambda^k$ for some $\lambda > 0$. We can then write

$$k(G, G') \;=\; \sum_{k} \lambda^k q_\times^\top W_\times^k p_\times \;=\; q_\times^\top (\mathbf{I} - \lambda W_\times)^{-1} p_\times. \tag{17}$$

To recover the marginalized graph kernels let $\lambda = 1$, and define $\Phi(X_{ij}) = P_{ij} \hat{\Phi}(X_{ij})$, in which case $W_\times = T_\times$, thus recovering (16).

Given a pair of graphs, Gärtner et al. (2003) also perform random walks on both, but then *count* the number of matching walks. Their kernel can be defined as (Gärtner et al., 2003, Definition 6):

$$k(G, G') = \sum_{i=1}^{n} \sum_{j=1}^{n'} \sum_{k=1}^{\infty} \lambda^k [A_\times^k]_{ij}. \tag{18}$$

To obtain (18) in our framework, set $\mu(k) = \lambda^k$, assume a uniform distribution over the vertices of G and G', *i.e.*, $p_i = q_i = 1/n$ and $p'_i = q'_i = 1/n'$, and let $\Phi(X) := A$ and $\Phi(X') = A'$. Consequently, $p_\times = q_\times = \mathbf{e}/(nn')$, and $W_\times = A_\times$. This allows us to rewrite (8) to obtain

$$k(G, G') = \sum_{k=1}^{\infty} \lambda^k q_\times^\top A_\times^k p_\times = \frac{1}{n^2 n'^2} \sum_{i=1}^{n} \sum_{j=1}^{n'} \sum_{k=1}^{\infty} \lambda^k [A_\times^k]_{ij}, \tag{19}$$

which recovers (18) to within a constant factor. Gärtner et al. (2003) also extend their kernel to graphs with labels from a finite set by replacing $A_\times$ in (18) with the weight matrix $W_\times$ from (7). The reduction to our framework extends to this setting in a straightforward manner.

The kernels of Gärtner et al. (2003) differ from our definition (8) in a number of ways: First, they employ a fixed exponential decay factor λ to down-weight the contribution of long walks to the kernel.[2] Second, they do not take starting or stopping probabilities into account, and third, instead of our general weight matrix (6) which allows for kernels on edges it only handles graphs whose labels are drawn from a finite set.

Finally, the so-called geometric kernel is defined as (Gärtner, 2002)

$$k(G, G') = \sum_{i=1}^{n} \sum_{j=1}^{n'} [e^{\lambda A_\times}]_{ij} = \mathbf{e}^\top e^{\lambda A_\times} \mathbf{e}, \tag{20}$$

using the matrix exponential. This is obtained in our framework by setting $p_i = q_i = 1/n$, $p'_i = q'_i = 1/n'$, $\Phi(X) := A$, $\Phi(X') = A'$, and $\mu(k) = \lambda^k/k!$, then proceeding as for (19).

4. Efficient Computation

Computing a random walk graph kernel essentially reduces to inverting the matrix $(\mathbf{I} - \lambda W_\times)$. If both G and G' have n vertices, then $(\mathbf{I} - \lambda W_\times)$ is an $n^2 \times n^2$ matrix. Given that the complexity of inverting a matrix is essentially cubic in its dimensions, direct computation of (17) would require $O(n^6)$ time. Below we develop methods based on Sylvester equations (Section 4.1), conjugate gradients (Section 4.2), and fixed-point iterations (Section 4.3) that can be used to speed up this computation. In Section 4.4 we show that the geometric kernel can also be computed efficiently, and Section 4.5 introduces an approximation that can further speed up the kernel computation for labeled graphs.

Table 1 summarizes our results, listing the worst-case time complexity of our methods as a function of graph density and labeling. Exact computation of a kernel between dense, unlabeled graphs (leftmost column) is cubic in the graph size (number of nodes) for all our algorithms; for the iterative methods this must be multiplied by the number of iterations, which is given by the effective rank r of the weight matrix $W_\times$ for conjugate gradient, and by (32) for fixed-point iterations.

For these two algorithms the cost increases by another factor of d for graphs with edge labels from a finite set of d symbols or an edge kernel with d-dimensional feature map;

2. The choice of λ is critical: It must be small enough for the sum in (18) to converge, depending on the spectrum of $W_\times$ (Vishwanathan, 2002, Chapter 6).

Table 1: Worst-case time complexity (in $O(\cdot)$ notation) of our methods to compute graph kernels, where n = size of graph (number of nodes), d = size of label set *resp.* dimensionality of feature map, r = effective rank of $W_\times$, k = number of fixed-point iterations (32), and k' = number of power iterations (34).

sparsity	dense				sparse
edge labels	none	finite set	finite-dim.	∞-dim.	any
Method (Section) $W_\times =$	$A \otimes A'$	(7)	kernel (6)		
Sylvester Equation (4.1)	n^3	unknown	—		—
Conjugate Gradient (4.2)	rn^3	rdn^3		rn^4	rn^2
Fixed-Point Iterations (4.3)	kn^3	kdn^3		kn^4	kn^2
Geometric Kernel (4.4)	n^3	n^6			—
Nearest Kron. Product (4.5)	1	$k'dn^2$	$k'n^4$		$k'n^2$

for an arbitrary edge kernel (whose feature map may be infinite-dimensional) this factor becomes n. Our technique for fast computation of the geometric kernel does not apply to labeled graphs, and the Sylvester equation method only if the labels come from a finite set of symbols, and then with unknown time complexity. A nearest Kronecker product approximation can be used, however, to approximate the direct product of labeled graphs with a weight matrix that can be handled by any of our methods for unlabeled graphs. This approximation requires k' (34) iterations, each costing $O(dn^2)$ time when the labels come from a finite set of d symbols, and $O(n^4)$ in general.

Finally, when the graphs are sparse (*i.e.,* only have $O(n)$ edges each; rightmost column in Table 1) our iterative methods (conjugate gradient, fixed-point, and nearest Kronecker product) take only $O(n^2)$ time per iteration, regardless of how the graphs are labeled. We cannot authoritatively state the time complexity for sparse graphs of solving Sylvester equations or performing spectral decompositions (for the geometric kernel).

4.1 Sylvester Equation Methods

Consider the following equation, commonly known as the Sylvester or Lyapunov equation:

$$M = SMT + M_0. \tag{21}$$

Here, $S, T, M_0 \in \mathbb{R}^{n \times n}$ are given and we need for solve for $M \in \mathbb{R}^{n \times n}$. These equations can be readily solved in $O(n^3)$ time with freely available code (Gardiner et al., 1992), such as Matlab's `dlyap` method. Solving the generalized Sylvester equation

$$M = \sum_{i=1}^{d} S_i M T_i + M_0 \tag{22}$$

involves computing generalized simultaneous Schur factorizations of d symmetric matrices (Lathauwer et al., 2004). Although technically involved, this can also be solved efficiently, albeit at a higher computational cost. The computational complexity of this generalized factorization is at present unknown.

We now show that for graphs with discrete edge labels, whose weight matrix $W_\times$ can be written as (7), the problem of computing the graph kernel (17) can be reduced to solving the following generalized Sylvester equation:

$$M = \sum_{i=1}^{d} \lambda \, {}^{i}\!A' \, M \, {}^{i}\!A^\top + M_0, \tag{23}$$

where $\mathrm{vec}(M_0) = p_\times$. We begin by *flattening* (23):

$$\mathrm{vec}(M) = \lambda \sum_{i=1}^{d} \mathrm{vec}({}^{i}\!A' M \, {}^{i}\!A^\top) + p_\times. \tag{24}$$

Using Lemma 12 (which extends (1) into an RKHS) we can rewrite (24) as

$$(\mathbf{I} - \lambda \sum_{i=1}^{d} {}^{i}\!A \otimes {}^{i}\!A') \, \mathrm{vec}(M) = p_\times, \tag{25}$$

use (7), and solve (25) for $\mathrm{vec}(M)$:

$$\mathrm{vec}(M) = (\mathbf{I} - \lambda W_\times)^{-1} p_\times. \tag{26}$$

Multiplying both sides of (26) by $q_\times^\top$ yields

$$q_\times^\top \mathrm{vec}(M) = q_\times^\top (\mathbf{I} - \lambda W_\times)^{-1} p_\times. \tag{27}$$

The right-hand side of (27) is the graph kernel (17). Given the solution M of the Sylvester equation (23), the graph kernel can be obtained as $q_\times^\top \mathrm{vec}(M)$ in $O(n^2)$ time. The same argument applies for unlabeled graphs by simply setting $d = 1$, which turns (23) into a simple Sylvester equation. Since solving that only takes $O(n^3)$ time, computing the random walk graph kernel in this fashion is much faster than the $O(n^6)$ time required by the direct approach.

One drawback of this strategy is that Sylvester equation solvers are quite sophisticated and typically available only as black-box library routines, which limits their applicability. Matlab's `dlyap` solver, for instance, does not exploit sparsity, and only handles the cases $d = 1$ and $d = 2$. A solver for the simple Sylvester equation (21) can still be used to efficiently compute kernels between labeled graphs though by employing the nearest Kronecker product approximation (Section 4.5).

4.2 Conjugate Gradient Methods

Given a matrix M and a vector b, conjugate gradient (CG) methods solve the system of equations $Mx = b$ efficiently (Nocedal and Wright, 1999). While they are designed

for symmetric *p.s.d.* matrices, CG solvers can also be used to solve other linear systems efficiently. They are particularly efficient if the matrix is rank deficient, or has a small *effective rank*, that is, number of distinct eigenvalues. Furthermore, if computing matrix-vector products is cheap — because M is sparse, for instance — the CG solver can be sped up significantly (Nocedal and Wright, 1999). Specifically, if computing Mv for an arbitrary vector v requires $O(m)$ time, and the effective rank of M is r, then a CG solver takes $O(r)$ iterations, and hence only $O(rm)$ time, to solve $Mx = b$.

The graph kernel (17) can be computed by a two-step procedure: First we solve the linear system

$$(\mathbf{I} - \lambda W_\times)\, x = p_\times, \tag{28}$$

for x, then we compute $q_\times^\top x$. We now focus on efficient ways to solve (28) with a CG solver. Recall that if G and G' contain n vertices each then $W_\times$ is an $n^2 \times n^2$ matrix. Naively, multiplying W by some vector y inside the CG algorithm requires $O(n^4)$ operations. However, by our extension of the vec-ABC formula (1) into RKHS (Lemma 12), introducing the matrix $Y \in \mathbb{R}^{n \times n}$ with $y = \mathrm{vec}(Y)$, and recalling that $W_\times = \Phi(X) \otimes \Phi(X')$, by Lemma 12 we can write

$$W_\times y = (\Phi(X) \otimes \Phi(X'))\,\mathrm{vec}(Y) = \mathrm{vec}(\Phi(X')Y\,\Phi(X)^\top). \tag{29}$$

If $\phi(\cdot) \in \mathbb{R}^d$ then the above matrix-vector product can be computed in $O(dn^3)$ time. If $\Phi(X)$ and $\Phi(X')$ are sparse, then $\Phi(X')Y\,\Phi(X)^\top$ can be computed yet more efficiently: If there are $O(n)$ non-ζ entries in $\Phi(X)$ and $\Phi(X')$, then computing (29) takes only $O(n^2)$ time.

4.3 Fixed-Point Iterations

Fixed-point methods begin by rewriting (28) as

$$x = p_\times + \lambda W_\times x. \tag{30}$$

Now, solving for x is equivalent to finding a fixed point of (30) taken as an iteration (Nocedal and Wright, 1999). Letting x_t denote the value of x at iteration t, we set $x_0 := p_\times$, then compute

$$x_{t+1} = p_\times + \lambda W_\times x_t \tag{31}$$

repeatedly until $\|x_{t+1} - x_t\| < \varepsilon$, where $\|\cdot\|$ denotes the Euclidean norm and ε some pre-defined tolerance. This is guaranteed to converge if all eigenvalues of $\lambda W_\times$ lie inside the unit disk; this can be ensured by setting $\lambda < |\xi_1|^{-1}$, where ξ_1 is the largest-magnitude eigenvalue of $W_\times$. Assuming that each iteration of (31) contracts x to the fixpoint by a factor of $\lambda\xi_1$, we converge to within ε of the fixpoint in k iterations, where

$$k = O\left(\frac{\ln \varepsilon}{\ln \lambda + \ln |\xi_1|}\right). \tag{32}$$

The above is closely related to the power method used to compute the largest eigenvalue of a matrix (Golub and Van Loan, 1996); efficient preconditioners can also be used to

speed up convergence (Golub and Van Loan, 1996). Since each iteration of (31) involves computation of the matrix-vector product $W_\times x_t$, all speed-ups for computing the matrix-vector product discussed in Section 4.2 are applicable here. In particular, we exploit the fact that $W_\times$ is a sum of Kronecker products to reduce the worst-case time complexity to $O(dn^3)$ per iteration in our experiments, in contrast to Kashima et al. (2004) who computed the matrix-vector product explicitly.

4.4 Geometric Kernel

We now turn our attention to Gärtner's (2002) geometric kernel (20). If G and G' are unlabeled graphs with n vertices, then $A_\times$ is an $n^2 \times n^2$ matrix, therefore a naive implementation of the geometric kernel takes $O(n^6)$ time. We now show how to reduce this to $O(n^3)$.

Lemma 4 *The geometric kernel (20) between two unlabeled graphs with n vertices each can be computed in $O(n^3)$ time.*

Proof Let $A = PDP^\top$ denote the spectral decomposition of A, that is, columns of P are the eigenvectors of A and D is a diagonal matrix of corresponding eigenvalues (Stewart, 2000). Similarly $A' = P'D'P'^\top$. The spectral decomposition of an $n \times n$ matrix can be computed efficiently in $O(n^3)$ time (Golub and Van Loan, 1996).

Using Propositions 7.1.10, 7.1.6, and 7.1.3 of Bernstein (2005) it is easy to show that the spectral decomposition of $A_\times$ is $(P \otimes P')(D \otimes D')(P \otimes P')^\top$. Furthermore, the matrix exponential $e^{\lambda A_\times}$ can be written as $(P \otimes P')\, e^{\lambda D \otimes D'} (P \otimes P')^\top$ (Bernstein, 2005, proposition 11.2.3). This and (2) allow us to rewrite (20) as

$$\begin{aligned} k(G, G') &= (\mathbf{e} \otimes \mathbf{e})^\top (P \otimes P')\, e^{\lambda D \otimes D'} (P \otimes P')^\top (\mathbf{e} \otimes \mathbf{e}) \\ &= (\mathbf{e}^\top P \otimes \mathbf{e}^\top P')\, e^{\lambda D \otimes D'} (P^\top \mathbf{e} \otimes P'^\top \mathbf{e}). \end{aligned} \tag{33}$$

The proof follows by observing that each of the three terms in (33) as well as their product can be computed in $O(n^2)$ time. ■

Lemma 4 applies only to the computation of the geometric kernel between unlabeled graphs. The key technical difficulty in extending it to labeled graphs is that a sum of matrices in the exponent cannot be separated unless they commute, *i.e.,* generally $e^{A+B} \neq e^A e^B$ unless $AB = BA$. We can use the nearest Kronecker product (Section 4.5), however, to compute an approximate geometric kernel between labeled graphs.

4.5 Nearest Kronecker Product Approximation

As we have seen above, some of our fast methods for computing random walk graph kernels may become computationally expensive, or not even be available, for labeled graphs, in particular when the number d of distinct labels is large or a general edge kernel is employed. In such cases we can find the *nearest Kronecker product* to $W_\times$, *i.e.,* compute matrices S and T such that $W_\times \approx S \otimes T$, then use any of our methods on $S \otimes T$ as if it were the adjacency matrix of a direct product of *unlabeled* graphs.

Finding the nearest Kronecker product approximating a matrix such as $W_\times$ is a well-studied problem in numerical linear algebra, and efficient algorithms which can exploit the

sparsity of $W_\times$ are available (Pitsianis, 1992; Van Loan, 2000). Formally, these methods minimize the Frobenius norm $\|W_\times - S \otimes T\|_F$ by computing the largest singular value of $\hat{W}_\times$, a permuted version of $W_\times$. We employ the power method[3] for this purpose, each iteration of which entails computing the matrix-vector product $\hat{W}_\times \mathrm{vec}(T')$, where $T' \in \mathbb{R}^{n \times n}$ is the current approximation of T. The result of the matrix-vector product is then reshaped into an $n \times n$ matrix to form T' for the next iteration (Pitsianis, 1992). It is easy to see that computing $\hat{W}_\times \mathrm{vec}(T')$ requires $O(n^4)$ time.

If $W_\times$ can be written as a sum of d Kronecker products (7), then so can $\hat{W}_\times$ (Pitsianis, 1992; Van Loan, 2000), and the cost per iteration hence drops to $O(dn^2)$. Furthermore, if the two graphs are sparse with $O(n)$ edges each, then $W_\times$ will have $O(n^2)$ non-zero entries, and each iteration only takes $O(n^2)$ time. The number k' of iterations required is

$$k' = O\left(\frac{\ln n}{\ln |\xi_1| - \ln |\xi_2|}\right), \tag{34}$$

where ξ_1 and ξ_2 are the eigenvalues of $W_\times$ with largest *resp.* second-largest magnitude.

5. Experiments

Numerous studies have applied random walk graph kernels to problems such as protein function prediction (Borgwardt et al., 2005) and chemoinformatics (Kashima et al., 2004). In our experiments we therefore focus on the runtime of computing the kernels, rather than their utility in any given application. We present three sets of experiments: First, we study the scaling behaviour of our algorithms on random graphs. Second, we assess the practical impact of our algorithmic improvement on four real-world datasets whose size mandates fast kernel computation. Third, we devise novel methods for protein-protein interaction (PPI) network comparison using graph kernels. The algorithmic challenge here is to efficiently compute kernels on large sparse graphs.

The baseline for comparison in all our experiments is the direct approach of Gärtner et al. (2003), implemented via a sparse LU factorization; this already runs orders of magnitude faster on our datasets than a dense (*i.e.,* non-sparse) implementation. Our code was written in Matlab Release 2008a, and all experiments were run under Mac OS X 10.5.5 on an Apple Mac Pro with a 3.0 GHz Intel 8-Core processor and 16 GB of main memory. We employed Lemma 12 to speed up matrix-vector multiplication for both CG and fixed-point methods (*cf.* Section 4.2), and used the function `dlyap` from Matlab's control toolbox to solve the Sylvester equation. By default, we used a value of $\lambda = 10^{-4}$, and set the convergence tolerance for both CG solver and fixed-point iteration to 10^{-6}. For the real-world datasets, the value of λ was chosen to ensure that the random walk graph kernel converges. Since our methods are exact and produce the same kernel values (to numerical precision), we only report the CPU time of each algorithm.

5.1 Random Graphs

The aim here is to study the scaling behaviour of our algorithms as a function of graph size and sparsity. We generated several sets of random graphs. For the first set we began with

3. Lanczos iterations are typically faster but more difficult to handle numerically.

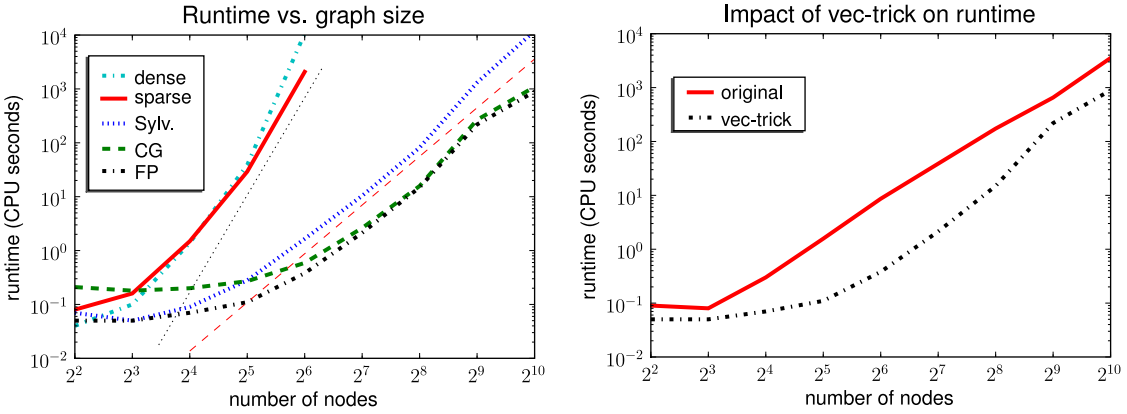

Figure 3: Time to compute a 10×10 kernel matrix on random graphs with n nodes and $3n$ edges as a function of the graph size n. Left: The Sylvester equation (Sylv.), conjugate gradient (CG), and fixed-point iteration (FP) approaches compared to the dense and sparse direct method. The thin straight lines indicate $O(n^6)$ (black dots) *resp.* $O(n^3)$ (red dashes) scaling. Right: Kashima et al.'s (2004) fixed-point iteration (original) compared to our version, which exploits Lemma 12 (vec-trick).

an empty graph of $n = 2^k$ nodes, where $k = 2, 3, \ldots, 10$, randomly added $3n$ edges, then checked the graph's connectivity. For each k we repeated this process until we had collected 10 strongly connected random graphs.

The time required to compute the 10×10 kernel matrix between these graphs for each value of n is shown in Figure 3 (left). We see that the direct approach scales asymptotically as $O(n^6)$ in both the dense and the sparse implementation. For a graph of 64 nodes the direct approach already takes over half an hour (sparse) *resp.* 3 hours (dense) of CPU time. Our Sylvester equation (Sylv.), conjugate gradient (CG) and fixed-point iteration (FP) methods, by contrast, all scale as $O(n^3)$, and can thus be applied to far larger graphs.

We also examined the impact of Lemma 12 on enhancing the runtime performance of the fixed-point iteration approach as originally proposed by Kashima et al. (2004). For this experiment, we again computed the 10×10 kernel matrix on the above random graphs, once using the original fixed-point iteration, and once using fixed-point iteration enhanced by Lemma 12. As Figure 3 (right) shows, our approach consistently outperforms the original version, sometimes by over an order of magnitude.

For the next set of experiments we fixed the graph size at 32 nodes (the largest size that the direct method could handle comfortably), and randomly added edges until the fill factor (*i.e.,* the number of non-zero entries in the adjacency matrix) reached x%, where $x = 5, 10, 20, 30, \ldots, 100$. For each x, we generated 10 such graphs and computed the 10×10 kernel matrix between them. Figure 4 (left) shows that as expected, the sparse direct method is faster than its dense counterpart for small fill factors but slower for larger ones. Both however are consistently outperformed by our three methods, which are up to three orders of magnitude faster.

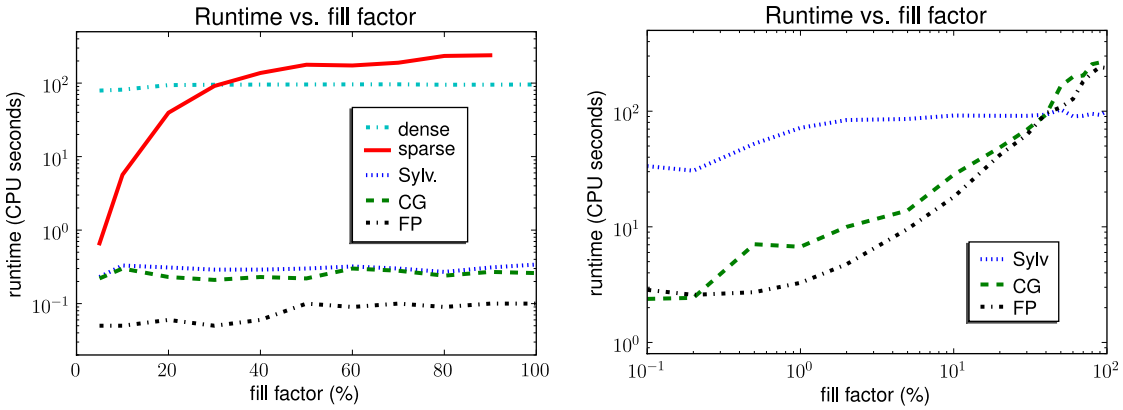

Figure 4: Time to compute a 10×10 kernel matrix on random graphs as a function of their fill factor. Left: The dense and sparse direct method on 32-node graphs, compared to our Sylvester equation (Sylv.), conjugate gradient (CG), and fixed point iteration (FP) approaches. Right: Our approaches on larger graphs with 256 nodes, where the direct method is infeasible.

To better understand how our algorithms take advantage of sparsity, we generated a set of larger random graphs (with 256 nodes) by the same procedure as before, but with a geometric progression of fill factors: $x = 0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50, 100$. The direct methods are infeasible here. The CPU times taken by our algorithms to compute a 10×10 kernel matrix is shown in Figure 4 (right). The runtime of the Sylvester equation solver is fairly independent of the fill factor because our black-box `dlyap` solver does not exploit sparsity in the adjacency matrices. Both conjugate gradient and fixed point iteration methods, by contrast, have runtimes roughly proportional to the fill factor. Although they are therefore typically much faster than our Sylvester equation solver, it is interesting to note that the latter is fastest on dense graphs. We thus predict that a *sparse* Sylvester equation solver, if available, could provide the most efficient way to compute graph kernels.

5.2 Real-World Datasets

Our next set of experiments used four real-world datasets: Two sets of molecular compounds (MUTAG and PTC), and two datasets describing protein tertiary structure (Protein and Enzyme). Graph kernels provide useful measures of similarity for all of these. We now briefly describe each dataset, and discuss how graph kernels are applicable.

Chemical Molecules. Toxicity of chemical molecules can be predicted to some degree by comparing their three-dimensional structure. We employed graph kernels to measure similarity between molecules from the MUTAG and PTC datasets (Toivonen et al., 2003). The average number of nodes per graph in these datasets is 17.72 *resp.* 26.70; the average number of edges is 38.76 *resp.* 52.06.

Table 2: Time to compute kernel matrix for unlabeled graphs from various datasets.

dataset	MUTAG		PTC		Enzyme		Protein	
nodes/graph	17.7		26.7		32.6		38.6	
edges/node	2.2		1.9		3.8		3.7	
#graphs	100	230	100	417	100	600	100	1128
Sparse	31”	1’45”	45”	7’23”	1’52”	1h21’	23’23”	2.1d*
Sylvester	10”	54”	28”	7’33”	31”	23’28”	5’25”	11h29’
Conj. Grad.	23”	1’29”	26”	4’29”	14”	10’00”	45”	39’39”
Fixed-Point	8”	43”	15”	2’38”	5”	5’44”	43”	22’09”

*extrapolated number of days; run did not finish in time available.

Protein Graphs. A standard approach to protein function prediction involves classifying proteins into enzymes and non-enzymes, then further assigning enzymes to one of the six top-level classes of the Enzyme Commission (EC) hierarchy. Towards this end, Borgwardt et al. (2005) modeled a dataset of 1128 proteins as graphs in which vertices represent secondary structure elements, and edges represent neighborhood within the 3-D structure or along the amino acid chain, as illustrated in Figure 1.

Comparing these graphs via a modified random walk graph kernel and classifying them with a Support Vector Machine (SVM) led to function prediction accuracies competitive with state-of-the-art approaches (Borgwardt et al., 2005). We used Borgwardt et al.’s (2005) data to test the efficacy of our methods on a large dataset. The average number of nodes and edges per graph in this data is 38.57 *resp.* 143.75. We used a single label on the edges, and the delta kernel to define similarity between edges.

Enzyme Graphs. We repeated the above experiment on an enzyme graph dataset, also due to Borgwardt et al. (2005). This dataset contains 600 graphs, with 32.63 nodes and 124.27 edges on average. Graphs in this dataset represent enzymes from the BRENDA enzyme database (Schomburg et al., 2004). The biological challenge on this data is to correctly assign the enzymes to one of the EC top-level classes.

5.2.1 Unlabeled Graphs

For this experiment, we computed kernels taking into account only the topology of the graph, *i.e.,* we did not consider node or edge labels. Table 2 lists the CPU time required to compute the full kernel matrix for each dataset, as well as — for comparison purposes — a 100×100 submatrix. The latter is also shown graphically in Figure 5 (left).

On these unlabeled graphs, conjugate gradient and fixed-point iteration — sped up via Lemma 12 — are consistently faster than the sparse direct method. The Sylvester equation approach is very competitive on smaller graphs (outperforming CG on MUTAG) but slows down with increasing number of nodes per graph. Even so, the Sylvester equation approach still outperforms the sparse direct method under most conditions.

5.2.2 Labeled Graphs

For this experiment, we compared graphs with edge labels. Note that node labels can be dealt with by concatenating them to the edge labels of adjacent edges. On the two protein datasets we employed a linear kernel to measure similarity between edge labels representing distances (in Ångströms) between secondary structure elements. On the two chemical datasets we used a delta kernel to compare edge labels reflecting types of bonds in molecules; for the Sylvester equation approach we then employed the nearest Kronecker product approximation. We report CPU times for the full kernel matrix as well as a 100×100 submatrix in Table 3; the latter is also shown graphically in Figure 5 (right).

On labeled graphs, the conjugate gradient and the fixed-point iteration always outperform the sparse direct approach, more so on the larger graphs and with the linear kernel. The Sylvester equation approach (at least with the Sylvester solver we used) cannot take advantage of sparsity, but still manages to perform almost as well as the sparse direct method.

5.3 Protein-Protein Interaction Networks

In our third experiment, we used random walk graph kernels to tackle a large-scale problem in bioinformatics involving the comparison of fairly large protein-protein interaction (PPI) networks. Using a combination of human PPI and clinical microarray gene expression data, the task is to predict the disease outcome (dead or alive, relapse or no relapse) of cancer patients. As before, we set $\lambda = 0.001$ and the convergence tolerance to 10^{-6} for all our experiments reported below.

5.3.1 Co-Integration of Gene Expression and PPI Data

We co-integrated clinical microarray gene expression data for cancer patients with known human PPI from Rual et al. (2005). Specifically, a patient's gene expression profile was transformed into a graph as follows: A node was created for every protein which — according to Rual et al. (2005) — participates in an interaction, and whose corresponding gene expression level was measured on this patient's microarray. We connect two proteins in this graph

Table 3: Time to compute kernel matrix for labeled graphs from various datasets.

kernel	delta, $d=7$		delta, $d=22$		linear, $d=1$			
dataset	MUTAG		PTC		Enzyme		Protein	
#graphs	100	230	100	417	100	600	100	1128
Sparse	42"	2'44"	1'07"	14'22"	1'25"	57'43"	12'38"	1.1d*
Sylvester	1'08"	6'05"	1'06"	18'20"	2'13"	76'43"	19'20"	11h19'
Conj. Grad.	39"	3'16"	53"	14'19"	20"	13'20"	41"	57'35"
Fixed-Point	25"	2'17"	37"	7'55"	10"	6'46"	25"	31'09"

*extrapolated number of days; run did not finish in time available.

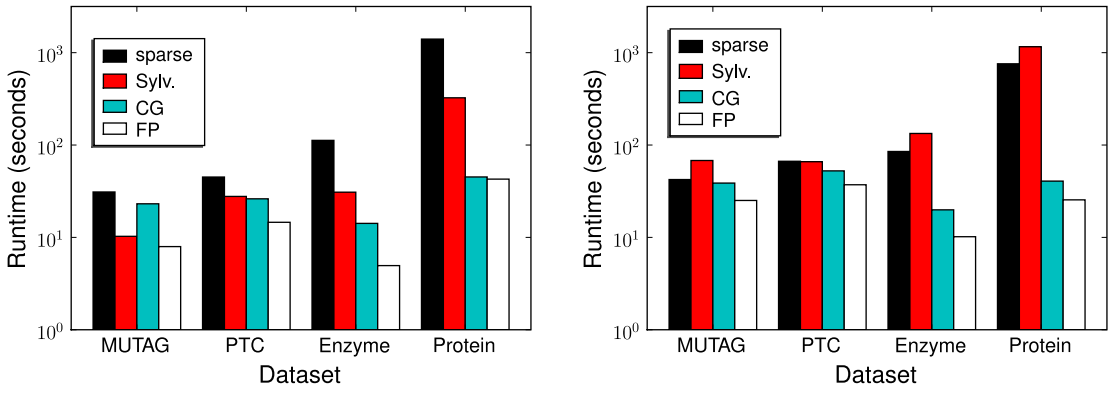

Figure 5: Time (in seconds on a log-scale) to compute 100×100 kernel matrix for unlabeled (left) *resp.* labeled (right) graphs from several datasets, comparing the conventional sparse method to our fast Sylvester equation, conjugate gradient (CG), and fixed-point iteration (FP) approaches.

by an edge if Rual et al. (2005) list these proteins as interacting, and both genes are up- *resp.* downregulated with respect to a reference measurement. Each node bears the name of the corresponding protein as its label.

This approach of co-integrating PPI and gene expression data is built on the assumption that genes with similar gene expression levels are translated into proteins that are more likely to interact. Recent studies confirm that this assumption holds significantly more often for co-expressed than for random pairs of proteins (Fraser et al., 2004; Bhardwaj and Lu, 2005). To measure similarity between these networks in a biologically meaningful manner, we compare which groups of proteins interact and are co-regulated in each patient. For this purpose, a random walk graph kernel is the natural choice, as a random walk in this graph represents a group of proteins in which consecutive proteins along the walk are co-expressed and interact. As each node bears the name of its corresponding protein as its node label, the size of the product graph is at most that of the smaller of the two input graphs.

5.3.2 Composite Graph Kernel

The presence of an edge in a graph signifies an interaction between the corresponding nodes. In chemoinformatics, for instance, edges indicate chemical bonds between two atoms; in PPI networks, edges indicate interactions between proteins. When studying protein interactions in disease, however, the *absence* of a given interaction can be as significant as its presence. Since existing graph kernels cannot take this into account, we propose to modify them appropriately. Key to our approach is the notion of a complement graph:

Definition 5 *Let $G = (V, E)$ be a graph with vertex set V and edge set E. Its complement $\bar{G} = (V, \bar{E})$ is a graph over the same vertices but with complementary edges $\bar{E} := (V \times V) \backslash E$.*

Table 4: Average time to compute kernel matrix on protein-protein interaction networks.

dataset	Leukemia		Breast Cancer	
kernel	vanilla	composite	vanilla	composite
Sparse	24”	52”	39”	1’19”
Sylvester	11’55”	24’02”	19’59”	43’32”
Conj. Grad.	6”	13”	12”	26”
Fixed-Point	4”	7”	7”	13”

In other words, the complement graph consists of exactly those edges *not* present in the original graph. Using this notion we define the *composite* graph kernel

$$k_{comp}(G, G') := k(G, G') + k(\bar{G}, \bar{G}'). \tag{35}$$

This deceptively simple kernel leads to substantial gains in performance in our experiments comparing co-integrated gene expression/protein-protein interaction networks.

5.3.3 Datasets

Leukemia. Bullinger et al. (2004) provide a dataset of microarrays of 119 leukemia patients. Since 50 patients survived after a median follow-up time of 334 days, always predicting a lethal outcome here would result in a baseline prediction accuracy of 1 - 50/119 = 58.0%. Co-integrating this data with human PPI, we found 2,167 proteins from Rual et al. (2005) for which Bullinger et al. (2004) report expression levels among the 26,260 genes they examined.

Breast Cancer. This dataset consists of microarrays of 78 breast cancer patients, of which 44 had shown no relapse of metastases within 5 years after initial treatment (van’t Veer et al., 2002). Always predicting survival thus gives a baseline prediction accuracy of 44/78 = 56.4% on this data. When generating co-integrated graphs, we found 2,429 proteins from Rual et al. (2005) for which van’t Veer et al. (2002) measure gene expression out of the 24,479 genes they studied.

5.3.4 Results

The CPU runtimes of our conjugate gradient, fixed-point, and Sylvester equation approaches to graph kernel computation on the cancer patients modeled as graphs is contrasted with that of the direct sparse method in Table 4. Using the computed kernel and a support vector machine (SVM) we tried to predict the survivors, either with a “vanilla” graph kernel (17), or our composite graph kernel (35) in 10-fold cross-validation.

On both datasets, our fast graph kernel computation methods result in a speed gain. With respect to prediction accuracy, the vanilla random walk graph kernel performs slightly better than the baseline classifier on one task (Leukemia: 59.2 % vs 58.0 %), and gives identical results on the other (Breast Cancer: both 56.4 %). The composite graph kernel

attains 5 percentage points above baseline in both experiments (Leukemia: 63.3 %; Breast cancer: 61.5 %).

The vanilla kernel suffers from its inability to measure network discrepancies, the paucity of the graph model employed, and the fact that only a small minority of genes could be mapped to interacting proteins; due to these problems, its accuracy remains close to the baseline. The composite kernel, by contrast, also models missing interactions. With it, even our simple graph model, which only considers 10% of the genes examined in both studies, is able to capture some relevant biological information, which in turn leads to better classification accuracy on these challenging datasets (Warnat et al., 2005).

6. Rational Kernels

Rational kernels (Cortes et al., 2004) were conceived to compute similarity between variable-length sequences and, more generally, weighted automata. For instance, the output of a large-vocabulary speech recognizer for a particular input speech utterance is typically a weighted automaton compactly representing a large set of alternative sequences. The weights assigned by the system to each sequence are used to rank different alternatives according to the models the system is based on. It is therefore natural to compare two weighted automata by defining a kernel.

As discussed in Section 3, random walk graph kernels have a very different basis: They compute the similarity between two random graphs by matching random walks. Here the graph itself is the object to be compared, and we want to find a semantically meaningful kernel. Contrast this with a weighted automaton, whose graph is merely a compact representation of the set of variable-length sequences which we wish to compare. Despite these differences we find, somewhat surprisingly, that rational kernels and random walk graph kernels are actually closely related.

To understand the connection recall that every random walk on a labeled graph produces a sequence of edge labels encountered during the walk. Viewing the set of all label sequences generated by random walks on a graph as a language, one can design a weighted transducer which accepts this language, with the weight assigned to each label sequence being the probability of a random walk generating this sequence. In this section we formalize this observation and thus establish connections between rational kernels on transducers (Cortes et al., 2004) and random walk graph kernels. In particular, we show that composition of transducers is analogous to computing product graphs, and that rational kernels on weighted transducers may be viewed as generalizations of random walk graph kernels to weighted automata. In order to make these connections explicit we adapt slightly non-standard notation for weighted transducers, extensively using matrices and tensors wherever possible.

6.1 Semirings

At the most general level, weighted transducers are defined over semirings. In a semiring addition and multiplication are generalized to abstract operations $\bar{\oplus}$ and $\bar{\odot}$ with the same distributive properties:

Definition 6 (Mohri, 2002) *A semiring is a system* $(\mathbb{K}, \bar{\oplus}, \bar{\odot}, \bar{0}, \bar{1})$ *such that*

1. $(\mathbb{K}, \bar{\oplus}, \bar{0})$ *is a commutative monoid in which* $\bar{0} \in \mathbb{K}$ *is the identity element for* $\bar{\oplus}$ *(i.e., for any* $x, y, z \in \mathbb{K}$*, we have* $x \bar{\oplus} y \in \mathbb{K}$*,* $(x \bar{\oplus} y) \bar{\oplus} z = x \bar{\oplus} (y \bar{\oplus} z)$*,* $x \bar{\oplus} \bar{0} = \bar{0} \bar{\oplus} x = x$ *and* $x \bar{\oplus} y = y \bar{\oplus} x$*);*

2. $(\mathbb{K}, \bar{\odot}, \bar{1})$ *is a monoid in which* $\bar{1}$ *is the identity operator for* $\bar{\odot}$ *(i.e., for any* $x, y, z \in \mathbb{K}$*, we have* $x \bar{\odot} y \in \mathbb{K}$*,* $(x \bar{\odot} y) \bar{\odot} z = x \bar{\odot} (y \bar{\odot} z)$*, and* $x \bar{\odot} \bar{1} = \bar{1} \bar{\odot} x = x$*);*

3. $\bar{\odot}$ *distributes over* $\bar{\oplus}$*, i.e., for any* $x, y, z \in \mathbb{K}$*,*

$$(x \bar{\oplus} y) \bar{\odot} z = (x \bar{\odot} z) \bar{\oplus} (y \bar{\odot} z)$$
$$\text{and} \quad z \bar{\odot} (x \bar{\oplus} y) = (z \bar{\odot} x) \bar{\oplus} (z \bar{\odot} y);$$

4. $\bar{0}$ *is an annihilator for* $\bar{\odot}$*:* $\forall x \in \mathbb{K}$*,* $x \bar{\odot} \bar{0} = \bar{0} \bar{\odot} x = \bar{0}$*.*

Thus, a semiring is a ring that may lack negation. $(\mathbb{R}, +, \cdot, 0, 1)$ is the familiar semiring of real numbers. Other examples include

Boolean: $(\{\textsc{False}, \textsc{True}\}, \vee, \wedge, \textsc{False}, \textsc{True})$;

Logarithmic: $(\mathbb{R} \cup \{-\infty\}, \bar{\oplus}_{\ln}, +, -\infty, 0)$, where $\forall x, y \in \mathbb{K} : \; x \bar{\oplus}_{\ln} y \; := \; \ln(e^x + e^y)$;

Tropical: $(\mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0)$.

Linear algebra operations such as matrix addition and multiplication as well as Kronecker products can be carried over to a semiring in a straightforward manner. For instance, for $M, M' \in \mathbb{K}^{n \times n}$ we have

$$[M \bar{\odot} M']_{i,j} = \bigoplus_{k=1}^{n} M_{ik} \bar{\odot} M'_{kj}. \tag{36}$$

The $(\bar{\oplus}, \bar{\odot})$ operations in some semirings can be mapped into ordinary $(+, \cdot)$ operations by applying an appropriate *morphism*:

Definition 7 *Let* $(\mathbb{K}, \bar{\oplus}, \bar{\odot}, \bar{0}, \bar{1})$ *be a semiring. A function* $\psi : \mathbb{K} \to \mathbb{R}$ *is a morphism if*

$$\psi(x \bar{\oplus} y) = \psi(x) + \psi(y);$$
$$\psi(x \bar{\odot} y) = \psi(x) \cdot \psi(y);$$
$$\psi(\bar{0}) = 0 \quad \text{and} \quad \psi(\bar{1}) = 1.$$

In the following, by 'morphism' we will always mean a morphism from a semiring to the real numbers. Not all semirings have such morphisms: For instance, the logarithmic semiring has a morphism — namely, the exponential function — but the tropical semiring does not have one. If the semiring has a morphism ψ, applying it to the matrix product (36), for instance, yields

$$\begin{aligned} \psi([M \bar{\odot} M']_{i,j}) \; &= \; \psi\left(\bigoplus_{k=1}^{n} M_{ik} \bar{\odot} M'_{kj}\right) \\ &= \; \sum_{k=1}^{n} \psi(M_{ik} \bar{\odot} M'_{kj}) \; = \; \sum_{k=1}^{n} \psi(M_{ik}) \cdot \psi(M'_{kj}). \end{aligned} \tag{37}$$

As in Appendix A, we can extend the morphism ψ to matrices (and analogously to vectors) by defining $[\Psi(M)]_{ij} := \psi(M_{ij})$. We can then write (37) concisely as

$$\Psi(M \,\bar{\odot}\, M') \;=\; \Psi(M)\Psi(M'). \tag{38}$$

6.2 Weighted Transducers

Loosely speaking, a transducer is a weighted automaton with an input and an output alphabet. We will work with the following slightly specialized definition:[4]

Definition 8 *A weighted finite-state transducer T over a semiring $(\mathbb{K}, \bar{\oplus}, \bar{\odot}, \bar{0}, \bar{1})$ is a 5-tuple $T = (\Sigma, Q, H, p, q)$, where Σ is a finite input-output alphabet, Q is a finite set of n states, $p \in \mathbb{K}^n$ is a vector of initial weights, $q \in \mathbb{K}^n$ is a vector of final weights, and H is a four-dimensional tensor in $\mathbb{K}^{n\times|\Sigma|\times|\Sigma|\times n}$ which encodes transitions and their corresponding weights.*

For $a, b \in \Sigma$ we will use the shorthand H_{ab} to denote the $n \times n$ slice H_{*ab*} of the transition tensor, which represents all valid transitions on input symbol a emitting the output symbol b. The output weight assigned by T to a pair of strings $\alpha = a_1 a_2 \ldots a_l$ and $\beta = b_1 b_2 \ldots b_l$ is

$$[\![T]\!](\alpha, \beta) = q^\top \,\bar{\odot}\, H_{a_1 b_1} \,\bar{\odot}\, H_{a_2 b_2} \,\bar{\odot} \ldots \bar{\odot}\, H_{a_l b_l} \,\bar{\odot}\, p. \tag{39}$$

A transducer is said to accept a pair of strings (α, β) if it assigns non-zero output weight to them, *i.e.*, $[\![T]\!](\alpha, \beta) \neq \bar{0}$. A transducer is said to be regulated if the output weight it assigns to any pair of strings is well-defined in $\mathbb{K}$. Since we disallow ϵ transitions, our transducers are always regulated.

The inverse of $T = (\Sigma, Q, H, p, q)$, denoted by T^{-1}, is obtained by transposing the input and output labels of each transition. Formally, $T^{-1} = (\Sigma, Q, H^\top, p, q)$ where $H^\top_{ab} := H_{ba}$. The composition of two transducers $T = (\Sigma, Q, H, p, q)$ and $T' = (\Sigma, Q', H', p', q')$ is a transducer $T_\times = T \circ T' = (\Sigma, Q_\times, H_\times, p_\times, q_\times)$, where $Q_\times = Q \times Q'$, $p_\times = p \,\bar{\otimes}\, p'$,[5] $q_\times := q \,\bar{\otimes}\, q'$, and $(H_\times)_{ab} = \bar{\bigoplus}_{c\in\Sigma} H_{ac} \,\bar{\otimes}\, H'_{cb}$. It can be shown that

$$[\![T_\times]\!](\alpha, \beta) = [\![T \circ T']\!](\alpha, \beta) = \overline{\bigoplus_{\gamma}} [\![T]\!](\alpha, \gamma) \,\bar{\odot}\, [\![T']\!](\gamma, \beta). \tag{40}$$

Composing T with its inverse yields $T \circ T^{-1} = (\Sigma, Q \times Q, H^*, p \,\bar{\otimes}\, p, q \,\bar{\otimes}\, q)$, where $H^*_{ab} = \bar{\bigoplus}_{c\in\Sigma} H_{ac} \,\bar{\otimes}\, H_{bc}$. There exists a general and efficient algorithm for composing transducers which takes advantage of the sparseness of the input transducers (Mohri et al., 1996; Pereira and Riley, 1997).

6.3 Weighted Automata

A weighted automaton is a transducer with identical input and output symbols. The transition matrix of a weighted automaton is therefore a three-dimensional tensor in $\mathbb{K}^{n\times|\Sigma|\times n}$.

4. We disallow ϵ transitions, and use the same alphabet for both input and output. Furthermore, in a departure from tradition, we represent the transition function as a four-dimensional tensor.
5. We use $\bar{\otimes}$ to denote the Kronecker product using the semiring operation $\bar{\odot}$, in order to distinguish it from the regular Kronecker product $\otimes$.

As before, we will use the shorthand H_a to denote the $n \times n$ slice H_{*a*} of the transition tensor, which represents all valid transitions on the input symbol a emitting output symbolx a. If Σ contains d symbols, then by specializing (39) it is easy to see that a weighted automaton accepts a string $\alpha = a_1 a_2 \dots a_l$ with weight

$$[\![T]\!](\alpha) = q^\top \bar{\odot} H_{a_1} \bar{\odot} H_{a_2} \bar{\odot} \dots \bar{\odot} H_{a_l} \bar{\odot} p. \tag{41}$$

The composition of two weighted automata $T = (\Sigma, Q, H, p, q)$ and $T' = (\Sigma, Q', H', p', q')$ is an automaton $T_\times = T \circ T' = (\Sigma, Q_\times, H_\times, p_\times, q_\times)$, where $Q_\times = Q \times Q'$, $p_\times = p \bar{\otimes} p'$, $q_\times := q \bar{\otimes} q'$, and $(H_\times)_a = H_a \bar{\otimes} H'_a$. The composition operation is also defined for a weighted automaton W and a transducer T:

$$[\![W \circ T]\!](\alpha, \beta) = [\![W]\!](\alpha) \bar{\odot} [\![T]\!](\alpha, \beta). \tag{42}$$

Every random walk on a labeled graph results in a sequence of edge labels encountered during the walk. The set of all label sequences generated by random walks on a given graph is a language. One can construct a weighted automaton which accepts this language as follows: Use the standard semiring $(\mathbb{R}, +, \cdot, 0, 1)$, let the alphabet Σ consist of the labels $\{1, \dots, d\}$ of the graph, and identify the nodes of the graph with the states of the weighted automaton. Let the starting and stopping probabilities p and q on the graph equal those of the weighted automaton, and complete the construction by identifying for each $l \in \Sigma$ the label-filtered adjacency matrix ${}^l\!A$ of the graph with H_l, the transition tensor of the weighted automaton for that symbol.

Under the above mapping (41) has a natural interpretation: The weight assigned by the automaton to a string of symbols is the probability of encountering the corresponding labels while performing a random walk on the corresponding labeled graph. The composition of weighted automata, when specialized to labeled graphs, is equivalent to computing a direct product graph.

An unlabeled graph corresponds to a weighted automaton whose input-output alphabet contains exactly one symbol, and which therefore only accepts strings of the form $a^k = aa \dots a$. The transition matrix of such a graph (equivalently, its adjacency matrix) is a 2-dimensional tensor in $\mathbb{K}^{n \times n}$. If A denotes the adjacency matrix of a graph G, then the output weight assigned by G to a^k is $[\![G]\!](a^k) = q^\top AA \dots Ap = q^\top A^k p$.

6.4 The Rational Kernel for Strings

Given a weighted transducer T and a function $\psi : \mathbb{K} \to \mathbb{R}$, the rational kernel between two strings $\alpha = a_1 a_2 \dots a_l$ and $\beta = b_1 b_2 \dots b_l$ is defined as (Cortes et al., 2004):

$$k(\alpha, \beta) := \psi\left([\![T]\!](\alpha, \beta)\right). \tag{43}$$

Cortes et al. (2004) show that a generic way to obtain *p.s.d.* rational kernels is to replace T in (43) by $T \circ T^{-1}$, and let ψ be a semiring morphism. We now present an alternate proof which uses properties of the Kronecker product. Since ψ is a semiring morphism, by specializing (39) to $T \circ T^{-1}$, we can write $k(\alpha, \beta) = \psi\left([\![T \circ T^{-1}]\!](\alpha, \beta)\right)$ as

$$\Psi(q \bar{\otimes} q)^\top \Psi\left(\overline{\bigoplus_{c_1}} H_{a_1 c_1} \bar{\otimes} H_{b_1 c_1}\right) \dots \Psi\left(\overline{\bigoplus_{c_l}} H_{a_l c_l} \bar{\otimes} H_{b_l c_l}\right) \Psi(p \bar{\otimes} p). \tag{44}$$

Rules analogous to (38) give us

$$\Psi\left(\overline{\bigoplus_{c\in\Sigma}} H_{ac} \bar{\otimes} H_{bc}\right) = \sum_{c\in\Sigma} \Psi(H_{ac}) \otimes \Psi(H_{bc}). \tag{45}$$

Using (45) we can rewrite (44) as

$$\sum_{c_1 c_2 \ldots c_l} \Psi(q)^\top \otimes \Psi(q)^\top \left(\Psi(H_{a_1 c_1}) \otimes \Psi(H_{b_1 c_1})\right) \ldots \left(\Psi(H_{a_l c_l}) \otimes \Psi(H_{b_l c_l})\right) \Psi(p) \otimes \Psi(p). \tag{46}$$

Finally, successively applying (2) to (46) yields

$$k(\alpha,\beta) = \sum_{c_1 c_2 \ldots c_l} \underbrace{\left(\Psi(q)^\top \Psi(H_{a_1 c_1}) \ldots \Psi(H_{a_l c_l}) \Psi(p)\right)}_{\rho(\alpha)} \underbrace{\left(\Psi(q)^\top \Psi(H_{b_1 c_1}) \ldots \Psi(H_{b_l c_l}) \Psi(p)\right)}_{\rho(\beta)}, \tag{47}$$

Each term of (47) equals $\rho(\alpha)\,\rho(\beta)$ for some scalar function ρ, and is therefore a valid *p.s.d.* kernel. Since *p.s.d.* kernels are closed under addition and pointwise limits (Berg et al., 1984), $k(\alpha,\beta)$ is a valid *p.s.d.* kernel.

6.5 The Rational Kernel for Weighted Automata

Rational kernels on strings can be naturally extended to weighted automata S and U via (Cortes et al., 2004):

$$\begin{aligned} k(S,U) &= \psi\left(\overline{\bigoplus_{\alpha,\beta}} [\![S]\!](\alpha) \,\bar{\odot}\, [\![T]\!](\alpha,\beta) \,\bar{\odot}\, [\![U]\!](\beta)\right) \\ &= \psi\left(\overline{\bigoplus_{\alpha,\beta}} [\![S \circ T \circ U]\!](\alpha,\beta)\right), \end{aligned} \tag{48}$$

where we obtained (48) by using (42) twice. If ψ is a semiring morphism, then we can use Definition 7 to rewrite (48) as

$$k(S,U) = \sum_{\alpha,\beta} \psi\left([\![S \circ T \circ U]\!](\alpha,\beta)\right). \tag{49}$$

Since *p.s.d.* kernels are closed under addition and pointwise limits, if $\psi\left([\![S \circ T \circ U]\!](\alpha,\beta)\right)$ is a *p.s.d.* kernel for any given α and β, then so is (49).

6.6 Recovering Random Walk Graph Kernels

In order to recover random walk graph kernels we use the standard $(\mathbb{R}, +, \cdot, 0, 1)$ ring as our semiring, and hence set ψ to be the identity function. Next we set the transducer T to simply transform any input string of length k into an identical output string with weight $\mu(k) \geq 0$. With these restrictions (49) can be written as

$$k(S,U) = \sum_{\alpha} \mu(|\alpha|) [\![S \circ U]\!](\alpha), \tag{50}$$

where $|\alpha|$ denotes the length of α. Let us rearrange (50) to

$$k(S,U) = \sum_k \mu(k) \left(\sum_{a_1,a_2,\ldots,a_k} [\![S \circ U]\!](a_1, a_2, \ldots, a_k) \right). \tag{51}$$

Specializing the definition of $\circ$ to weighted automata, and letting H_a (resp. H'_a) denote the transition tensor of S (resp. U), we can rewrite (51) as

$$\begin{aligned}
k(S,U) &= \sum_k \mu(k) \left(\sum_{a_1,a_2,\ldots,a_k \in \Sigma^k} (q \otimes q')^\top (H_{a_1} \otimes H'_{a_1}) \ldots (H_{a_k} \otimes H'_{a_k})(p \otimes p') \right) \\
&= \sum_k \mu(k) (q \otimes q')^\top \left(\sum_{a_1,a_2,\ldots,a_k \in \Sigma^k} (H_{a_1} \otimes H'_{a_1}) \ldots (H_{a_k} \otimes H'_{a_k}) \right) (p \otimes p') \\
&= \sum_k \mu(k) (q \otimes q')^\top \left(\sum_{a \in \Sigma} H_a \otimes H'_a \right) \ldots \left(\sum_{a \in \Sigma} H_a \otimes H'_a \right) (p \otimes p') \\
&= \sum_k \mu(k) (q \otimes q')^\top \left(\sum_a H_a \otimes H'_a \right)^k (p \otimes p'). \qquad (52)
\end{aligned}$$

Next, we identify H_a (resp. H'_a) with the label-filtered adjacency matrix ${}^a\!A$ (resp. ${}^a\!A'$) of a graph G (resp. G') with discrete edge labels. It easy to see that $H_\times := \sum_a H_a \otimes H'_a$ is the weight matrix (7) of the direct product of G and G'. Letting $p_\times = p \otimes p'$ and $q_\times = q \otimes q'$, (52) reduces to

$$k(G,G') = \sum_k \mu(k)\, q_\times^\top H_\times^k\, p_\times, \tag{53}$$

which recovers the random walk graph kernel (8) with $W_\times = H_\times$.

The generality of the rational kernel comes at a computational cost: Even when restricted as in (50), it requires the composition $S \circ U$ of two transducers, which takes up to $O((|Q_S| + |E_S|)(|Q_U| + |E_U|))$ time, where $|Q|$ is the number of states and $|E|$ the number of transitions (Cortes et al., 2004, Section 4.1). In our setting $|Q| = n$, the number of nodes in the graph, and $|E|$ is the number of its edges, which can be of $O(n^2)$. The worst-case time complexity of the composition operation is therefore $O(n^4)$. Contrast this with the $O(n^3)$ worst-case complexity for computing the random walk kernel that we achieve via the methods described in Section 4.

Thus even though random walk graph kernels can be considered a special case of rational kernels, the algorithms for computing them differ: We have succeeded in leveraging our narrower framework into lower computational complexity. Our key insight is that we never explicitly construct the composition (*i.e.,* direct product graph) to compute the kernel.

For ease of exposition we derived (53) by setting T to be an identity transducer. Instead, one can use a weighted transducer which allows for more flexible matching between strings in the alphabet. Basically, the transducer now plays the role of the kernel function κ, and this in turn leads to a more flexible similarity matrix $W_\times$.

There is one important difference between graph kernels and rational kernels. Graph kernels can handle arbitrary edge kernels, including continuous edge labels via the weight matrix $W_\times$. In contrast, rational kernels, which were designed to work with strings and automata, assume that the alphabet (set of labels) is finite. As we saw above, they can incorporate flexible similarity matrices $W_\times$ in this setting, but cannot handle continuous edge labels. Furthermore, extending rational kernels to deal with labels mapped to an RKHS remains an open problem.

7. R-convolution Kernels

Haussler's (1999) R-convolution kernels provide a generic way to construct kernels for discrete compound objects. Let $x \in \mathcal{X}$ be such an object, and $\boldsymbol{x} := (x_1, x_2, \ldots, x_D)$ denote a decomposition of x, with each $x_i \in \mathcal{X}_i$. We can define a boolean predicate

$$R : \mathcal{X} \times \boldsymbol{\mathcal{X}} \to \{\textsc{True}, \textsc{False}\}, \tag{54}$$

where $\boldsymbol{\mathcal{X}} := \mathcal{X}_1 \times \ldots \times \mathcal{X}_D$ and $R(x, \boldsymbol{x})$ is True whenever $\boldsymbol{x}$ is a valid decomposition of x. Now consider the inverse of (54), the set of all valid decompositions of an object:

$$R^{-1}(x) := \{\boldsymbol{x} | R(x, \boldsymbol{x}) = \textsc{True}\}. \tag{55}$$

Like Haussler (1999) we assume that (55) is countable. We define the R-convolution $\star$ of the kernels $\kappa_1, \kappa_2, \ldots, \kappa_D$ with $\kappa_i : \mathcal{X}_i \times \mathcal{X}_i \to \mathbb{R}$ to be

$$k(x, x') \;=\; \kappa_1 \star \kappa_2 \star \ldots \star \kappa_D(x, x') \;:=\; \sum_{\substack{\boldsymbol{x} \in R^{-1}(x) \\ \boldsymbol{x}' \in R^{-1}(x')}} \mu(\boldsymbol{x}, \boldsymbol{x}') \prod_{i=1}^{D} \kappa_i(x_i, x_i'), \tag{56}$$

where μ denotes a set of non-negative coefficients on $\boldsymbol{\mathcal{X}} \times \boldsymbol{\mathcal{X}}$, which ensures that the sum in (56) converges.[6] Haussler (1999) showed that $k(x, x')$ is *p.s.d.* and hence admissible as a kernel (Schölkopf and Smola, 2002), provided that all the individual κ_i are. The deliberate vagueness of this setup in regard to the nature of the underlying decomposition leads to a rich framework: Many different kernels can be obtained by simply changing the decomposition.

7.1 Graph Kernels as R-Convolutions

To apply R-convolution kernels to graphs, one decomposes the graph into smaller substructures, and builds the kernel based on similarities between those components. Most graph kernels are — knowingly or not — based on R-convolutions; they mainly differ in the way they decompose the graph for comparison and the similarity measure they use to compare the components.

Gärtner et al. (2003) observed that a hypothetical graph kernel which takes *all* substructures (*i.e.,* subgraphs) of a graph into account could be used to determine whether two graphs G and G' are isomorphic: Simply compute $d(G, G') := k(G, G) - k(G, G')$; since

6. Haussler (1999) implicitly assumed this sum to be well-defined, hence did not use μ in his definition.

any structural difference between the graphs would yield a non-zero $d(G, G')$, they are isomorphic iff $d(G, G') = 0$. The graph isomorphism problem, however, is widely believed to be not solvable in polynomial time (Garey and Johnson, 1979). The choice of substructures used in defining a graph kernel is therefore generally motivated by runtime considerations.

Random walks provide a straightforward graph decomposition that — as we have seen in Section 4 — leads to kernels that can be computed efficiently. To see that our random walk graph kernel (8) is indeed an R-convolution kernel, note that the definition of our weight matrix (6) and the RKHS Kronecker product (Definition 11) imply

$$\begin{aligned}[W_\times]_{(i-1)n'+r,\,(j-1)n'+s} &= [\Phi(X) \otimes \Phi(X')]_{(i-1)n'+r,\,(j-1)n'+s} \\ &= \big\langle \phi(v_i, v_j), \phi(v'_r, v'_s) \big\rangle_{\mathcal{H}} =: \kappa((v_i, v_j), (v'_r, v'_s)), \end{aligned} \tag{57}$$

where κ is our edge kernel. We can thus expand (8) by explicitly taking all paths through the repeated matrix products, giving

$$\begin{aligned} k(G, G') &:= \sum_{k=1}^{\infty} \mu(k)\, q_\times^\top W_\times^k p_\times = \sum_{k=1}^{\infty} \mu(k)\, q_\times^\top \Big(\prod_{i=1}^{k} W_\times \Big) p_\times \\ &= \sum_{k=1}^{\infty} \mu(k) \sum_{\substack{v_0, v_1, \ldots v_k \in V \\ v'_0, v'_1, \ldots v'_k \in V'}} q_{v_k} q'_{v'_k} \Big(\prod_{i=1}^{k} \kappa((v_{i-1}, v_i), (v'_{i-1}, v'_i)) \Big) p_{v_0}\, p'_{v'_0}. \end{aligned} \tag{58}$$

This is easily identified as an instance of the R-convolution kernel (56), where the decomposition is into all equal-length sequences $\boldsymbol{v}, \boldsymbol{v}'$ of nodes from V and V', respectively, and

$$\mu(\boldsymbol{v}, \boldsymbol{v}') := \mu(|\boldsymbol{v}|)\, q_{v_{|\boldsymbol{v}|}} q'_{v'_{|\boldsymbol{v}|}} p_{v_0}\, p'_{v'_0}, \tag{59}$$

where $|\cdot|$ in (59) denotes the length of a sequence. Finally, note that by definition of our edge kernel κ, only pairs of sequences that are both actual walks on their respective graphs will make a non-zero contribution to (58).

Random walk graph kernels as proposed by Gärtner et al. (2003) likewise decompose a graph into random walks, but then employ a delta kernel between nodes. Borgwardt et al. (2005), on the other hand, use a kernel defined on both nodes and edges. The marginalized graph kernels of Kashima et al. (2004) are closely related but subtly different in that they decompose the graph into all possible *label* sequences generated by a walk. Mahé et al. (2004) extend this approach in two ways: They enrich the labels via the so-called Morgan index, and modify the kernel definition to prevent *tottering*, that is, the generation of high similarity scores by multiple, similar, small substructures. Both these extensions are particularly relevant for chemoinformatics applications.

Further afield, Horvath et al. (2004) decompose a graph into cyclic patterns, then count the number of common cyclic patterns which occur in both graphs. Their kernel is plagued by computational issues; in fact they show that computing the cyclic pattern kernel of a general graph is NP-hard. They consequently restrict their attention to practical problem classes where the number of simple cycles is bounded.

Ramon and Gärtner (2003) consider subtree patterns to define graph kernels. Starting from a given node v, a tree is created by adding all the nodes that can be reached from v in

$1, \ldots, h$ steps, where h is the height of the tree. If more than one walk connects two nodes, then each one of these is used to define a distinct subtree. This means that the same node is counted several times, thus leading to tottering. Furthermore, the number of candidate trees grows exponentially with the height of the subtree under consideration, thus severely limiting the depth of graph structure one can probe at reasonable computational cost.

Borgwardt and Kriegel (2005) define a kernel (60) based on shortest paths. They represent a graph $G = (V, E)$ by a complete graph $S = (V, \bar{E})$ over the same vertices, wherein the weight of each edge in $\bar{E}$ equals the length of the shortest path between the corresponding nodes in G. Their shortest path kernel is then defined as

$$k_{\mathrm{sp}}(G, G') = \sum_{e \in \bar{E}} \sum_{e' \in \bar{E}'} \kappa(e, e'), \tag{60}$$

where κ is any kernel defined on the edges of S and S'.

Shervashidze et al. (2009) use subgraphs of fixed size to define kernels. Their key idea is to represent the graph by a normalized frequency vector which counts the frequency of occurrence of various fixed-size subgraphs. The kernel is then simply computed as the dot product between these vectors.

Other decompositions of graphs which are well suited for particular application domains include molecular fingerprints based on various types of depth-first searches (Ralaivola et al., 2005) and structural elements such as rings or functional groups (Fröhlich et al., 2006).

7.2 R-Convolutions in Abstract Semirings

There have been a few attempts to extend the R-convolution kernel (56) to abstract semirings, by defining:

$$k(x, x') := \overline{\bigoplus_{\substack{\boldsymbol{x} \in R^{-1}(x) \\ \boldsymbol{x}' \in R^{-1}(x')}}} \mu(\boldsymbol{x}, \boldsymbol{x}') \,\bar{\odot}\, \overline{\bigodot_{i=1}^{D}} \kappa_i(x_i, x_i'). \tag{61}$$

The optimal assignment graph kernel of Fröhlich et al. (2006) is motivated along these lines, using the tropical semiring. It can be defined as

$$k(x, x') = \max_{\substack{\boldsymbol{x} \in R^{-1}(x) \\ \boldsymbol{x}' \in R^{-1}(x')}} \left(\mu(\boldsymbol{x}, \boldsymbol{x}') + \sum_{i=1}^{D} \kappa_i(x_i, x_i') \right). \tag{62}$$

Unfortunately (62) is not always *p.s.d.* (Vert, 2008). The problem is that the class of *p.s.d.* kernels is not closed under the max operation (Berg et al., 1984).

For semirings that have a morphism ψ to the reals, however, we can rewrite (61) as

$$\psi(k(x, x')) = \sum_{\substack{\boldsymbol{x} \in R^{-1}(x) \\ \boldsymbol{x}' \in R^{-1}(x')}} \mu(\boldsymbol{x}, \boldsymbol{x}') \prod_{i=1}^{D} \psi(\kappa_i(x_i, x_i')). \tag{63}$$

Comparing (63) with (56) makes it clear that $\psi \circ k$ is *p.s.d.* and hence admissible if all $\psi \circ \kappa_i$ are. This can be used to construct *p.s.d.* R-convolution kernels in such semirings.

For instance, take the logarithmic semiring $(\mathbb{R} \cup \{-\infty\}, \bar{\oplus}_{\ln}, +, -\infty, 0)$ augmented with an inverse temperature parameter $\beta > 0$, so that $x \,\bar{\oplus}_{\ln}\, y := \ln(e^{\beta x} + e^{\beta y})/\beta$. This has the morphism $\psi(x) = e^{\beta x}$. We can thus specialize (63) to define

$$k(x, x') := \sum_{\substack{\boldsymbol{x} \in R^{-1}(x) \\ \boldsymbol{x}' \in R^{-1}(x')}} e^{\beta \kappa(\boldsymbol{x}, \boldsymbol{x}')}, \quad \text{where} \quad \kappa(\boldsymbol{x}, \boldsymbol{x}') := \mu(\boldsymbol{x}, \boldsymbol{x}') + \sum_{i=1}^{D} \kappa_i(x_i, x_i'), \tag{64}$$

which is a valid *p.s.d.* kernel if all $e^{\beta \kappa_i}$ are. Note that if κ_i is a *p.s.d.* kernel, then since $\beta > 0$ so is $\beta \kappa_i$, and since *p.s.d.* kernels are closed under exponentiation (Genton, 2001, Equation 5) so is $e^{\beta \kappa_i}$.

What makes (64) interesting is that when the temperature approaches zero ($\beta \to \infty$), the augmented logarithmic semiring approaches the tropical semiring, as $x \,\bar{\oplus}_{\ln}\, y \to \max(x, y)$. We thus obtain a kernel that approximates (an exponentiated version of) the optimal assignment kernel (62) yet is provably *p.s.d.* Since at low temperatures the value of (64) is dominated by the optimal assignment, one might call it the "mostly optimal assignment kernel."

The finite range of floating-point computer arithmetic unfortunately limits how low a temperature (64) can be used with in practice, though this can be greatly extended via suitable software, such as the `extnum` C++ class (`http://darwin.nmsu.edu/molb_resources/bioinformatics/extnum/extnum.html`).

8. Discussion and Outlook

As evidenced by the large number of recent papers, random walk graph kernels and marginalized graph kernels have received considerable research attention. Although the connections between these two kernels were hinted at by Kashima et al. (2004), no effort was made to pursue this further. Our aim in presenting a unified framework for random walk, marginalized, and geometric graph kernels that combines the best features of previous formulations is to highlight the similarities as well as the differences between these approaches. Furthermore, it allows us to use extended linear algebra in an RKHS to efficiently compute all these kernels by exploiting common structure inherent in these problems.

As more and more graph-structured data (e.g., molecular structures and protein interaction networks) becomes available in fields such as biology, web data mining, *etc.*, graph classification will gain importance over the coming years. Hence there is a pressing need to speed up the computation of similarity metrics on graphs. We have shown that sparsity, low effective rank, and Kronecker product structure can be exploited to greatly reduce the computational cost of graph kernels; taking advantage of other forms of structure in $W_\times$ remains a computational challenge. Now that the computation of random walk graph kernels is viable for practical problem sizes, it will open the doors for their application in hitherto unexplored domains.

A major deficiency of random walk graph kernels is that the admissible range of values of the decay parameter λ in (17) depends on the spectrum of the weight matrix $W_\times$. Since

this is typically unknown, in practice one often resorts to very low values of λ — but this makes the contributions of higher-order terms (corresponding to long walks) to the kernel negligible. In fact in many applications a naive kernel which simply computes the average kernel between all pairs of edges in the two graphs has performance comparable to the random walk graph kernel.

Trying to remedy this situation by normalizing the matrices involved leads to another phenomenon called *tottering* (Mahé et al., 2004). Roughly speaking tottering occurs when short self-repeating walks make a disproportionately large contribution to the kernel value. Consider two adjacent vertices v and v' in a graph. Because of tottering, contributions due to walks of the form $v \to v' \to v \to \ldots$ dominate the kernel value. Unfortunately a kernel using self-avoiding walks (walks which do not visit the same vertex twice) cannot be computed in polynomial time.

A natural question to ask is the following: Since diffusion can be viewed as a continuous time limit of random walks, can the ideas behind the random walk kernel be extended to diffusion? Unfortunately, the Laplacian of the product graph does not decompose into the Kronecker product of the Laplacian matrices of the constituent graphs; this rules out a straightforward extension.

Although rational kernels have always been viewed as distinct from graph kernels, we have shown that in fact these two research areas are closely related. It is our hope that this will facilitate cross-pollination of ideas such as the use of semirings and transducers in defining graph kernels. We also hope that tensor and matrix notation become more prevalent in the transducer community.

It is fair to say that R-convolution kernels are the mother of all kernels on structured data. It is enlightening to view various graph kernels as instances of R-convolution kernels since this brings into focus the relevant decomposition used to define a given kernel, and the similarities and differences between various kernels. Extending R-convolutions to abstract semirings, however, does not always result in a valid *p.s.d.* kernel. We have shown that a morphism to the reals is sufficient to successfully transport an R-convolution kernel into a semiring; whether it is necessary remains an open problem.

We do not believe that the last word on graph comparison has been said yet. Thus far, simple decompositions like random walks have been used to compare graphs. This is mainly driven by computational considerations and not by the application domain at hand. The algorithmic challenge of the future is to integrate higher-order structures such as spanning trees in graph comparisons, and to compute such kernels efficiently.

Acknowledgments

We thank Markus Hegland and Tim Sears for enlightening discussions, Alex Smola for pointing out that the optimal assignment kernel may fail to be *p.s.d.*, and the anonymous reviewers for their detailed comments and suggestions which greatly helped improve this paper.

This work was supported by NICTA, funded by the Australian Government through the Backing Australia's Ability and Centre of Excellence programs, by the IST Program of the European Community under the FP7 Network of Excellence, ICT-216886-NOE, by the German Ministry for Education, Science, Research and Technology (BMBF) under grant

No. 031U112F within the BFAM (Bioinformatics for the Functional Analysis of Mammalian Genomes) project, part of the German Genome Analysis Network (NGFN), and by NIH grant GM063208-05 "Tools and Data Resources in Support of Structural Genomics."

References

C. Berg, J. P. R. Christensen, and P. Ressel. *Harmonic Analysis on Semigroups*. Springer, New York, 1984.

H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne. The protein data bank. *Nucleic Acids Research*, 28:235–242, 2000.

D. S. Bernstein. *Matrix Mathematics*. Princeton University Press, 2005.

N. Bhardwaj and H. Lu. Correlation between gene expression profiles and protein-protein interactions within and across genomes. *Bioinformatics*, 21(11):2730–2738, June 2005.

D. Bonchev and D. H. Rouvray, editors. *Chemical Graph Theory: Introduction and Fundamentals*, volume 1. Gordon and Breach Science Publishers, London, UK, 1991.

K. M. Borgwardt and H.-P. Kriegel. Shortest-path kernels on graphs. In *Proceedings of the International Conference on Data Mining*, pages 74–81, 2005.

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 *Proceedings of Intelligent Systems in Molecular Biology (ISMB)*, Detroit, USA, 2005. http://www.stat.purdue.edu/~vishy/papers/BorOngSchVisetal05.pdf.

K. M. Borgwardt, H.-P. Kriegel, S. V. N. Vishwanathan, and N. Schraudolph. Graph kernels for disease outcome prediction from protein-protein interaction networks. In R. B. Altman, A. K. Dunker, L. Hunter, T. Murray, and T. E. Klein, editors, *Proceedings of the Pacific Symposium of Biocomputing 2007*, Maui Hawaii, January 2007. World Scientific.

L. Bullinger, K. Dohner, E. Bair, S. Frohling, R. F. Schlenk, R. Tibshirani, H. Dohner, and J. R. Pollack. Use of gene-expression profiling to identify prognostic subclasses in adult acute myeloid leukemia. *New England Journal of Medicine*, 350(16):1605–1616, Apr 2004.

C. Cortes, P. Haffner, and M. Mohri. Rational kernels. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, volume 14, Cambridge, MA, 2002. MIT Press.

C. Cortes, P. Haffner, and M. Mohri. Positive definite rational kernels. In B. Schölkopf and M. K. Warmuth, editors, *Procedings of the Annual Conference on Computational Learning Theory*, pages 41–56, 2003.

C. Cortes, P. Haffner, and M. Mohri. Rational kernels: Theory and algorithms. *Journal of Machine Learning Research*, 5:1035–1062, 2004.

H. B. Fraser, A. E. Hirsh, D. P. Wall, and M. B. Eisen. Coevolution of gene expression among interacting proteins. *Proceedings of the National Academy of Science USA*, 101 (24):9033–9038, Jun 2004.

H. Fröhlich, J. K. Wegner, F. Siker, and andreas Zell. Kernel functions for attributed molecular graphs — a new similarity based approach to ADME prediction in classification and regression. *QSAR and Combinatorial Science*, 25(4):317–326, 2006.

J. D. Gardiner, A. L. Laub, J. J. Amato, and C. B. Moler. Solution of the Sylvester matrix equation $AXB^\top + CXD^\top = E$. *ACM Transactions on Mathematical Software*, 18(2): 223–231, 1992.

M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Series of Books in Mathematical Sciences. W. H. Freeman, 1979.

T. Gärtner. Exponential and geometric kernels for graphs. In *NIPS workshop on unreal data*, principles of modeling nonvectorial data, 2002.

T. Gärtner, P. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In B. Schölkopf and M. K. Warmuth, editors, *Proceedings of the Annual Conference on Computational Learning Theory*, pages 129–143. Springer, 2003.

M. G. Genton. Classes of kernels for machine learning: A statistics perspective. *Journal of Machine Learning Research*, 2:299–312, 2001.

G. H. Golub and C. F. Van Loan. *Matrix Computations*. John Hopkins University Press, Baltimore, MD, 3rd edition, 1996.

D. Haussler. Convolutional kernels on discrete structures. Technical Report UCSC-CRL-99-10, Computer Science Department, UC Santa Cruz, 1999.

T. Horvath, T. Gärtner, and S. Wrobel. Cyclic pattern kernels for predictive graph mining. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 158–167, 2004.

W. Imrich and S. Klavžar. *Product Graphs, Structure and Recognition*. Wiley, 2000.

H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labeled graphs. In *Proceedings of the International Conference on Machine Learning*, pages 321–328, San Francisco, CA, 2003. Morgan Kaufmann.

H. Kashima, K. Tsuda, and A. Inokuchi. Kernels for graphs. In K. Tsuda, B. Schölkopf, and J. Vert, editors, *Kernels and Bioinformatics*, pages 155–170, Cambridge, MA, 2004. MIT Press.

R. Kondor and K. Borgwardt. The skew spectrum of graphs. In *Proceedings of the International Conference on Machine Learning*, pages 496–503. ACM, 2008.

R. Kondor and J. D. Lafferty. Diffusion kernels on graphs and other discrete structures. In *Proceedings of the International Conference on Machine Learning*, pages 315–322, San Francisco, CA, 2002. Morgan Kaufmann.

H. Kubinyi. Drug research: myths, hype and reality. *Nature Reviews: Drug Discovery*, 2 (8):665–668, August 2003.

R. Kumar, J. Novak, and A. Tomkins. Structure and evolution of online social networks. In T. Eliassi-Rad, L. H. Ungar, M. Craven, and D. Gunopulos, editors, *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, August 20-23, 2006*, pages 611–617. ACM, 2006. ISBN 1-59593-339-5.

L. D. Lathauwer, B. D. Moor, and J. Vandewalle. Computation of the canonical decomposition by means of a simultaneous generalized Schur decomposition. *SIAM Journal on Matrix Analysis and Applications*, 26(2):295–327, 2004.

P. Mahé, N. Ueda, T. Akutsu, J.-L. Perret, and J.-P. Vert. Extensions of marginalized graph kernels. In *Proceedings of the Twenty-First International Conference on Machine Learning*, pages 552–559, 2004.

M. Mohri. Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata, Languages and Combinatorics*, 7(3):321–350, 2002.

M. Mohri, F. C. N. Pereira, and M. D. Riley. Weighted automata in text and speech processing. In A. Kornai, editor, *Extended Finite State Models of Language: Proceedings of the ECAI'96 Workshop*, pages 46–50, 1996.

J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research. Springer, 1999.

F. C. N. Pereira and M. D. Riley. Speech recognition by composition of weighted finite automata. In *Finite-State Language Processing*, pages 431–453. MIT Press, 1997.

N. P. Pitsianis. *The Kronecker Product in Approximation and Fast Transform Generation*. PhD thesis, Department of Computer Science, Cornell University, 1992.

L. Ralaivola, S. J. Swamidass, H. Saigo, and P. Baldi. Graph kernels for chemical informatics. *Neural Networks*, 18(8):1093–1110, October 2005.

J. Ramon and T. Gärtner. Expressivity versus efficiency of graph kernels. Technical report, First International Workshop on Mining Graphs, Trees and Sequences (held with ECML/PKDD'03), 2003.

J. F. Rual, K. Venkatesan, T. Hao, T. Hirozane-Kishikawa, A. Dricot, N. Li, et al. Towards a proteome-scale map of the human protein-protein interaction network. *Nature*, 437 (7062):1173–1178, Oct 2005.

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

I. Schomburg, A. Chang, C. Ebeling, M. Gremse, C. Heldt, G. Huhn, and D. Schomburg. Brenda, the enzyme database: updates and major new developments. *Nucleic Acids Research*, 32D:431–433, Jan 2004.

R. Sharan and T. Ideker. Modeling cellular machinery through biological network comparison. *Nature Biotechnology*, 24(4):427–433, Apr 2006.

N. Shervashidze, S. V. N. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt. Efficient graphlet kernels for large graph comparison. In M. Welling and D. van Dyk, editors, *Proceedings of the International Workshop on Artificial Intelligence and Statistics*. Society for Artificial Intelligence and Statistics, 2009.

A. J. Smola and R. Kondor. Kernels and regularization on graphs. In B. Schölkopf and M. K. Warmuth, editors, *Proceedings of the Annual Conference on Computational Learning Theory*, Lecture Notes in Comput. Sci., pages 144–158, Heidelberg, Germany, 2003. Springer-Verlag.

G. W. Stewart. Decompositional approach to matrix computation. *Computing in Science and Engineering*, 2(1):50–59, February 2000.

H. Toivonen, A. Srinivasan, R. D. King, S. Kramer, and C. Helma. Statistical evaluation of the predictive toxicology challenge 2000-2001. *Bioinformatics*, 19(10):1183–1193, July 2003.

K. Tsuda, T. Kin, and K. Asai. Marginalized kernels for biological sequences. *Bioinformatics*, 18 (Suppl. 2):S268–S275, 2002.

C. F. Van Loan. The ubiquitous Kronecker product. *Journal of Computational and Applied Mathematics*, 123(1–2):85–100, 2000.

L. J. van't Veer, H. Dai, M. J. van de Vijver, Y. D. He, A. A. M. Hart, et al. Gene expression profiling predicts clinical outcome of breast cancer. *Nature*, 415:530–536, 2002.

J.-P. Vert. The optimal assignment kernel is not positive definite. Technical Report 0801.4061v1, arXiv, May 2008. http://aps.arxiv.org/pdf/0801.4061v1.

S. V. N. Vishwanathan. *Kernel Methods: Fast Algorithms and Real Life Applications*. PhD thesis, Indian Institute of Science, Bangalore, India, November 2002. http://www.stat.purdue.edu/~vishy/papers/Vishwanathan02.pdf.

S. V. N. Vishwanathan, K. Borgwardt, and N. N. Schraudolph. Fast computation of graph kernels. In B. Schölkopf, J. Platt, and T. Hofmann, editors, *Advances in Neural Information Processing Systems 19*, Cambridge MA, 2007. MIT Press.

P. Warnat, R. Eils, and B. Brors. Cross-platform analysis of cancer microarray data improves gene expression based classification of phenotypes. *BMC Bioinformatics*, 6:265, Nov 2005.

T. Washio and H. Motoda. State of the art of graph-based data mining. *SIGKDD Explorations*, 5(1):59–68, 2003.

X. Yao, D. Wei, C. Soden Jr., M. F. Summers, and D. Beckett. Structure of the carboxy-terminal fragment of the apo-biotin carboxyl carrier subunit of Escherichia coli acetyl-CoA carboxylase. *Biochemistry*, 36:15089–15100, 1997.

Appendix A. Extending Linear Algebra to RKHS

It is well known that any continuous, symmetric, positive definite kernel $\kappa\colon \mathcal{X}\times\mathcal{X}\to\mathbb{R}$ has a corresponding Hilbert space $\mathcal{H}$, called the Reproducing Kernel Hilbert Space or RKHS, which induces a feature map $\phi:\mathcal{X}\to\mathcal{H}$ satisfying $\kappa(x,x')=\langle\phi(x),\phi(x')\rangle_{\mathcal{H}}$. The natural extension of this so-called feature map to matrices is $\Phi\colon \mathcal{X}^{n\times m}\to\mathcal{H}^{n\times m}$ defined $[\Phi(A)]_{ij}:=\phi(A_{ij})$. In what follows, we use Φ to lift tensor algebra from $\mathcal{X}$ to $\mathcal{H}$, extending various matrix products to the RKHS, and proving some of their their useful properties. Straightforward extensions via the commutativity properties of the operators have been omitted for the sake of brevity.

A.1 Matrix Product

Definition 9 *Let $A\in\mathcal{X}^{n\times m}$, $B\in\mathcal{X}^{m\times p}$, and $C\in\mathbb{R}^{m\times p}$. The matrix products $\Phi(A)\Phi(B)\in\mathbb{R}^{n\times p}$ and $\Phi(A)\,C\in\mathcal{H}^{n\times p}$ are given by*

$$[\Phi(A)\Phi(B)]_{ik}:=\sum_j\langle\phi(A_{ij}),\phi(B_{jk})\rangle_{\mathcal{H}}\qquad\text{and}\qquad[\Phi(A)\,C]_{ik}:=\sum_j\phi(A_{ij})\,C_{jk}.$$

It is straightforward to show that the usual properties of matrix multiplication — namely associativity, transpose-commutativity, and distributivity with addition — hold for Definition 9 above, with one exception: associativity does *not* hold if the elements of all three matrices involved belong to the RKHS. In other words, given $A\in\mathcal{X}^{n\times m}$, $B\in\mathcal{X}^{m\times p}$, and $C\in\mathcal{X}^{p\times q}$, generally $[\Phi(A)\Phi(B)]\Phi(C)\neq\Phi(A)[\Phi(B)\Phi(C)]$. The technical difficulty is that in general

$$\langle\phi(A_{ij}),\phi(B_{jk})\rangle_{\mathcal{H}}\,\phi(C_{kl})\neq\phi(A_{ij})\,\langle\phi(B_{jk}),\phi(C_{kl})\rangle_{\mathcal{H}}\,. \tag{65}$$

Further examples of statements like (65), involving properties which do not hold when extended to an RKHS, can be found for the other matrix products at (67) and (74) below.

Definition 9 allows us to state a first RKHS extension of the vec(ABC) formula (1):

Lemma 10 *If $A\in\mathbb{R}^{n\times m}$, $B\in\mathcal{X}^{m\times p}$, and $C\in\mathbb{R}^{p\times q}$, then*

$$\operatorname{vec}(A\,\Phi(B)\,C))=(C^\top\otimes A)\operatorname{vec}(\Phi(B))\ \in\mathcal{X}^{nq\times 1}\,.$$

Proof Analogous to Lemma 12 below. ■

A.2 Kronecker Product

Definition 11 *Let $A\in\mathcal{X}^{n\times m}$ and $B\in\mathcal{X}^{p\times q}$. The Kronecker product $\Phi(A)\otimes\Phi(B)\in\mathbb{R}^{np\times mq}$ is defined as*

$$[\Phi(A)\otimes\Phi(B)]_{(i-1)p+k,(j-1)q+l}:=\langle\phi(A_{ij}),\phi(B_{kl})\rangle_{\mathcal{H}}\,.$$

Similarly to (65) above, for matrices in an RKHS

$$* \qquad (\Phi(A) \otimes \Phi(B))(\Phi(C) \otimes \Phi(D)) = (\Phi(A)\,\Phi(C)) \otimes (\Phi(B)\,\Phi(D)) \tag{66}$$

does *not* necessarily hold. The technical problem with (66) is that generally

$$\langle \phi(A_{ir}), \phi(B_{ks}) \rangle_{\mathcal{H}} \langle \phi(C_{rj}), \phi(D_{sl}) \rangle_{\mathcal{H}} \neq \langle \phi(A_{ir}), \phi(C_{rj}) \rangle_{\mathcal{H}} \langle \phi(B_{ks}), \phi(D_{sl}) \rangle_{\mathcal{H}} \,. \tag{67}$$

In Section A.3 we show that analogous properties (Lemmas 14 and 15) do hold for the *heterogeneous* Kronecker product between RKHS and real matrices.

Definition 11 gives us a second extension of the vec(ABC) formula (1) to RKHS:

Lemma 12 *If $A \in \mathcal{X}^{n\times m}$, $B \in \mathbb{R}^{m\times p}$, and $C \in \mathcal{X}^{p\times q}$, then*

$$\operatorname{vec}(\Phi(A)\, B\, \Phi(C)) = (\Phi(C)^\top \otimes \Phi(A)) \operatorname{vec}(B) \;\; \in \mathbb{R}^{nq\times 1} \,.$$

Proof We begin by rewriting the k^{th} column of $\Phi(A)B\Phi(C)$ as

$$\begin{aligned}
[\Phi(A)B\Phi(C)]_{*k} = \Phi(A) \sum_j B_{*j}\, \phi(C_{jk}) &= \sum_j \phi(C_{jk}) \Phi(A) B_{*j} \\
&= [\phi(C_{1k})\Phi(A), \phi(C_{2k})\Phi(A), \ldots \phi(C_{nk})\Phi(A)] \underbrace{\begin{bmatrix} B_{*1} \\ B_{*2} \\ \vdots \\ B_{*n} \end{bmatrix}}_{\operatorname{vec}(B)} \\
&= ([\phi(C_{1k}), \phi(C_{2k}), \ldots \phi(C_{nk})] \otimes \Phi(A)) \operatorname{vec}(B).
\end{aligned} \tag{68}$$

To obtain Lemma 12 we stack up the columns of (68):

$$\begin{aligned}
\operatorname{vec}(\Phi(A)\, B\, \Phi(C)) &= \left(\begin{bmatrix} \phi(C_{11}) & \phi(C_{21}) & \ldots & \phi(C_{n1}) \\ \vdots & \vdots & \ddots & \vdots \\ \phi(C_{1n}) & \phi(C_{2n}) & \ldots & \phi(C_{nn}) \end{bmatrix} \otimes \Phi(A) \right) \operatorname{vec}(B) \\
&= (\Phi(C)^\top \otimes \Phi(A)) \operatorname{vec}(B).
\end{aligned}$$

■

Direct computation of the right-hand side of Lemma 12 requires $nmpq$ kernel evaluations; when m, p, and q are all $O(n)$ this is $O(n^4)$. If $\mathcal{H}$ is finite-dimensional, however — in other words, if the feature map can be taken to be $\phi\colon \mathcal{X} \to \mathbb{R}^d$ with $d < \infty$ — then the left-hand side of Lemma 12 can be obtained in $O(n^3 d)$ operations. Our efficient computation schemes in Section 4 exploit this observation.

A.3 Heterogeneous Kronecker Product

Definition 13 *Let $A \in \mathcal{X}^{n\times m}$ and $B \in \mathbb{R}^{p\times q}$. The heterogeneous Kronecker product $\Phi(A)\otimes B \in \mathcal{X}^{np\times mq}$ is given by*

$$[\Phi(A) \otimes B]_{(i-1)p+k,(j-1)q+l} = \phi(A_{ij})\, B_{kl}.$$

Recall that the standard Kronecker product obeys (2); here we prove two extensions:

Lemma 14 *If $A \in \mathcal{X}^{n\times m}$, $B \in \mathcal{X}^{p\times q}$, $C \in \mathbb{R}^{m\times o}$, and $D \in \mathbb{R}^{q\times r}$, then*

$$(\Phi(A) \otimes \Phi(B))(C \otimes D) = (\Phi(A)\, C) \otimes (\Phi(B)\, D).$$

Proof Using the linearity of the inner product we directly verify

$$\begin{aligned}
[(\Phi(A) \otimes \Phi(B))(C \otimes D)]_{(i-1)p+k,(j-1)q+l} &= \sum_{r,s} \langle \phi(A_{ir}), \phi(B_{ks})\rangle_{\mathcal{H}}\, C_{rj} D_{sl} \\
&= \left\langle \sum_r \phi(A_{ir}) C_{rj},\ \sum_s \phi(B_{ks}) D_{sl} \right\rangle_{\mathcal{H}} \\
&= \langle [\Phi(A)\, C]_{ij}, [\Phi(B)\, D]_{kl}\rangle_{\mathcal{H}} \\
&= [(\Phi(A)\, C) \otimes (\Phi(B)\, D)]_{(i-1)p+k,(j-1)q+l}
\end{aligned}$$

■

Lemma 15 *If $A \in \mathcal{X}^{n\times m}$, $B \in \mathbb{R}^{p\times q}$, $C \in \mathcal{X}^{m\times o}$, and $D \in \mathbb{R}^{q\times r}$, then*

$$(\Phi(A) \otimes B)(\Phi(C) \otimes D) = (\Phi(A)\, \Phi(C)) \otimes (B\, D).$$

Proof Using the linearity of the inner product we directly verify

$$\begin{aligned}
[(\Phi(A) \otimes B)(\Phi(C) \otimes D)]_{(i-1)p+k,(j-1)q+l} &= \sum_{r,s} \langle \phi(A_{ir}) B_{ks}, \phi(C_{rj}) D_{sl}\rangle_{\mathcal{H}} \\
&= \sum_r \langle \phi(A_{ir}), \phi(C_{rj})\rangle_{\mathcal{H}} \sum_s B_{ks} D_{sl} \\
&= [\Phi(A)\, \Phi(C)]_{ij}\, [B\, D]_{kl} \\
&= [(\Phi(A)\, \Phi(C)) \otimes (B\, D)]_{(i-1)p+k,(j-1)q+l}
\end{aligned}$$

■

Using the heterogeneous Kronecker product, we can state four more RKHS extensions of the vec-ABC formula (1):

Lemma 16 *If $A \in \mathcal{X}^{n\times m}$, $B \in \mathbb{R}^{m\times p}$, and $C \in \mathbb{R}^{p\times q}$, then*

$$\operatorname{vec}(\Phi(A)\, B\, C) = (C^\top \otimes \Phi(A)) \operatorname{vec}(B) \quad \in \mathcal{X}^{nq\times 1}.$$

Proof Analogous to Lemma 12. ■

Lemma 17 *If $A \in \mathbb{R}^{n\times m}$, $B \in \mathbb{R}^{m\times p}$, and $C \in \mathcal{X}^{p\times q}$, then*

$$\operatorname{vec}(A\, B\, \Phi(C)) = (\Phi(C)^\top \otimes A) \operatorname{vec}(B) \quad \in \mathcal{X}^{nq\times 1}.$$

Proof Analogous to Lemma 12. ■

Lemma 18 *If $A \in \mathcal{X}^{n\times m}$, $B \in \mathcal{X}^{m\times p}$, and $C \in \mathbb{R}^{p\times q}$, then*

$$\operatorname{vec}(\Phi(A)\, \Phi(B)\, C) = (C^\top \otimes \Phi(A)) \operatorname{vec}(\Phi(B)) \quad \in \mathbb{R}^{nq\times 1}.$$

Proof Analogous to Lemma 12. ■

Lemma 19 *If $A \in \mathbb{R}^{n\times m}$, $B \in \mathcal{X}^{m\times p}$, and $C \in \mathcal{X}^{p\times q}$, then*

$$\operatorname{vec}(A\, \Phi(B)\, \Phi(C)) = (\Phi(C)^\top \otimes A) \operatorname{vec}(\Phi(B)) \quad \in \mathbb{R}^{nq\times 1}.$$

Proof Analogous to Lemma 12. ■

Note that there is no analogous lemma for $\operatorname{vec}(\Phi(A)\, \Phi(B)\, \Phi(C))$ since this term is not well-defined due to non-associativity (65).

A.4 Kronecker Sum

A concept closely related to the Kronecker product is that of the Kronecker sum, which is defined for real matrices $A \in \mathbb{R}^{n\times m}$ and $B \in \mathbb{R}^{p\times q}$ as

$$A \oplus B := A \otimes \mathbf{I}_{pq} + \mathbf{I}_{nm} \otimes B, \tag{69}$$

with $\mathbf{I}_{nm}$ (*resp.* $\mathbf{I}_{pq}$) denoting the $n \times m$ (*resp.* $p \times q$) identity matrix. Many of its properties can be derived from those of the Kronecker product. Unlike the Kronecker product, however, the Kronecker sum of two matrices in an RKHS is a matrix in the RKHS. From Definition 1 and (69) we find that

$$[A \oplus B]_{(i-1)p+k,(j-1)q+l} := A_{ij}\delta_{kl} + \delta_{ij}B_{kl}. \tag{70}$$

We can extend (70) to RKHS, defining analogously:

Definition 20 *Let $A \in \mathcal{X}^{n\times m}$ and $B \in \mathcal{X}^{p\times q}$. The Kronecker sum $\Phi(A)\oplus\Phi(B) \in \mathcal{X}^{np\times mq}$ is defined as*

$$[\Phi(A) \oplus \Phi(B)]_{(i-1)p+k,(j-1)q+l} := \phi(A_{ij})\delta_{kl} + \delta_{ij}\phi(B_{kl}).$$

In other words, in an RKHS the Kronecker sum is defined just as in (69):

$$\Phi(A) \oplus \Phi(B) \;=\; \Phi(A) \otimes \mathbf{I}_B \;+\; \mathbf{I}_A \otimes \Phi(B), \tag{71}$$

where $\mathbf{I}_M$ denotes the real-valued identity matrix of the same dimensions (not necessarily square) as matrix M. In accordance with Definition 13, the result of (71) is an RKHS matrix.

The equivalent of the vec-ABC formula (1) for Kronecker sums is:

$$\begin{aligned}(A \oplus B)\,\mathrm{vec}(C) &= (A \otimes \mathbf{I}_B + \mathbf{I}_A \otimes B)\,\mathrm{vec}(C)\\ &= (A \otimes \mathbf{I}_B)\,\mathrm{vec}(C) + (\mathbf{I}_A \otimes B)\,\mathrm{vec}(C)\\ &= \mathrm{vec}(\mathbf{I}_B\, C A^\top) + \mathrm{vec}(B C\, \mathbf{I}_A^\top)\\ &= \mathrm{vec}(\mathbf{I}_B\, C A^\top + B C\, \mathbf{I}_A^\top).\end{aligned} \tag{72}$$

This also works for matrices in an RKHS:

Lemma 21 *If $A \in \mathcal{X}^{n\times m}$, $B \in \mathcal{X}^{p\times q}$, and $C \in \mathcal{X}^{q\times m}$, then*

$$(\Phi(A) \oplus \Phi(B))\,\mathrm{vec}(\Phi(C)) = \mathrm{vec}(\mathbf{I}_B\,\Phi(C)\,\Phi(A)^\top + \Phi(B)\,\Phi(C)\,\mathbf{I}_A^\top) \;\in \mathbb{R}^{np\times 1}.$$

Proof Analogous to (72), using Lemmas 18 and 19. ■

Furthermore, we have two valid heterogeneous forms that map into the RKHS:

Lemma 22 *If $A \in \mathcal{X}^{n\times m}$, $B \in \mathcal{X}^{p\times q}$, and $C \in \mathbb{R}^{q\times m}$, then*

$$(\Phi(A) \oplus \Phi(B))\,\mathrm{vec}(C) = \mathrm{vec}(\mathbf{I}_B\, C\,\Phi(A)^\top + \Phi(B)\, C\,\mathbf{I}_A^\top) \;\in \mathcal{X}^{np\times 1}.$$

Proof Analogous to (72), using Lemmas 16 and 17. ■

Lemma 23 *If $A \in \mathbb{R}^{n\times m}$, $B \in \mathbb{R}^{p\times q}$, and $C \in \mathcal{X}^{q\times m}$, then*

$$(A \oplus B)\,\mathrm{vec}(\Phi(C)) = \mathrm{vec}(\mathbf{I}_B\,\Phi(C)\,A^\top + B\,\Phi(C)\,\mathbf{I}_A^\top) \;\in \mathcal{X}^{np\times 1}.$$

Proof Analogous to (72), using Lemma 10. ■

A.5 Hadamard Product

While the extension of the Hadamard (element-wise) product to an RKHS is not required to implement our fast graph kernels, the reader may find it interesting in its own right.

Definition 24 *Let $A, B \in \mathcal{X}^{n\times m}$ and $C \in \mathbb{R}^{n\times m}$. The Hadamard products $\Phi(A) \odot \Phi(B) \in \mathbb{R}^{n\times m}$ and $\Phi(A) \odot C \in \mathcal{H}^{n\times m}$ are given by*

$$[\Phi(A) \odot \Phi(B)]_{ij} = \langle \phi(A_{ij}), \phi(B_{ij}) \rangle_{\mathcal{H}} \quad \text{and} \quad [\Phi(A) \odot C]_{ij} = \phi(A_{ij})\, C_{ij}.$$

We prove two extensions of (3):

Lemma 25 *If $A \in \mathcal{X}^{n\times m}$, $B \in \mathcal{X}^{p\times q}$, $C \in \mathbb{R}^{n\times m}$, and $D \in \mathbb{R}^{p\times q}$, then*

$$(\Phi(A) \otimes \Phi(B)) \odot (C \otimes D) = (\Phi(A) \odot C) \otimes (\Phi(B) \odot D).$$

Proof Using the linearity of the inner product we directly verify

$$\begin{aligned}
[(\Phi(A) \otimes \Phi(B)) \odot (C \otimes D)]_{(i-1)p+k,(j-1)q+l} &= \langle \phi(A_{ij}), \phi(B_{kl}) \rangle_{\mathcal{H}} C_{ij} D_{kl} \\
&= \langle \phi(A_{ij}) C_{ij}, \phi(B_{kl}) D_{kl} \rangle_{\mathcal{H}} \\
&= \langle [\Phi(A) \odot C]_{ij}, [\Phi(B) \odot D]_{kl} \rangle_{\mathcal{H}} \\
&= [(\Phi(A) \odot C) \otimes (\Phi(B) \odot D)]_{(i-1)p+k,(j-1)q+l}
\end{aligned}$$

■

Lemma 26 *If $A \in \mathcal{X}^{n\times m}$, $B \in \mathbb{R}^{p\times q}$, $C \in \mathcal{X}^{n\times m}$, and $D \in \mathbb{R}^{p\times q}$, then*

$$(\Phi(A) \otimes B) \odot (\Phi(C) \otimes D) = (\Phi(A) \odot \Phi(C)) \otimes (B \odot D).$$

Proof Using the linearity of the inner product we directly verify

$$\begin{aligned}
[(\Phi(A) \otimes B) \odot (\Phi(C) \otimes D)]_{(i-1)p+k,(j-1)q+l} &= \langle \phi(A_{ij}) B_{kl}, \phi(C_{ij}) D_{kl} \rangle_{\mathcal{H}} \\
&= \langle \phi(A_{ij}), \phi(C_{ij}) \rangle_{\mathcal{H}} B_{kl} D_{kl} \\
&= [\Phi(A) \odot \Phi(C)]_{ij} [B \odot D]_{kl} \\
&= [(\Phi(A) \odot \Phi(C)) \otimes (B \odot D)]_{(i-1)p+k,(j-1)q+l}
\end{aligned}$$

■

As before,

$$* \qquad (\Phi(A) \otimes \Phi(B)) \odot (\Phi(C) \otimes \Phi(D)) = (\Phi(A) \odot \Phi(C)) \otimes (\Phi(B) \odot \Phi(D)) \tag{73}$$

does *not* necessarily hold, the difficulty with (73) being that in general,

$$\langle \phi(A_{ij}), \phi(B_{kl}) \rangle_{\mathcal{H}} \langle \phi(C_{ij}), \phi(D_{kl}) \rangle_{\mathcal{H}} \neq \langle \phi(A_{ij}), \phi(C_{ij}) \rangle_{\mathcal{H}} \langle \phi(B_{kl}), \phi(D_{kl}) \rangle_{\mathcal{H}}. \tag{74}$$