Graph Kernels between Point Clouds

Francis R. Bach FRANCIS.BACH@MINES.ORG
INRIA - WILLOW Project-Team, Laboratoire d'Informatique de l'Ecole Normale Supérieure, Paris, France

Abstract

Point clouds are sets of points in two or three dimensions. Most kernel methods for learning on sets of points have not yet dealt with the specific geometrical invariances and practical constraints associated with point clouds in computer vision and graphics. In this paper, we present extensions of graph kernels for point clouds, which allow one to use kernel methods for such objects as shapes, line drawings, or any three-dimensional point clouds. In order to design rich and numerically efficient kernels with as few free parameters as possible, we use kernels between covariance matrices and their factorizations on probabilistic graphical models. We derive polynomial time dynamic programming recursions and present applications to recognition of handwritten digits and Chinese characters from few training examples.

1. Introduction

In recent years, kernels for structured data have been designed in many domains, such as bioinformatics (Vert et al., 2004), text processing (Lodhi et al., 2002) and computer vision (Harchaoui & Bach, 2007; Parsana et al., 2008). They provide an elegant way of including known *a priori* information, by using directly the natural topological structure of objects. Using *a priori* knowledge through kernels on structured data have proved beneficial because it allows (a) to reduce the number of training examples, (b) to reuse existing data representations that are already well developed by experts of those domains and (c) to bring to bear the rapidly developing kernel machinery, and in particular semi-supervised learning—see, e.g., Chapelle et al. (2006)—and hyperparameter learning for supervised kernel methods—see, e.g., Bach et al. (2004).

In this paper, we propose a positive definite kernel between point clouds, with applications to classification of line drawings—such as handwritten digits (LeCun et al., 1998) or Chinese characters (Srihari et al., 2007)—or shapes (Belongie et al., 2002). The natural geometrical structure of point clouds is hard to represent in a few real-valued features (see, e.g., Forsyth and Ponce (2003)), in particular because of (a) the required local or global invariances by rotation, scaling, and/or translation, (b) the lack of preestablished registrations of the point clouds (i.e., points from one cloud are not given matched to points from another cloud), and (c) the noise and occlusion that impose that only portions of two point clouds ought to be compared.

One of the leading principles for designing kernels between structured objects is to decompose each object into parts and to compare all parts of one object to all parts of another object (Shawe-Taylor & Cristianini, 2004). Even if there is an exponential number of such decompositions, which is a common case, this is numerically possible under two conditions: (a) the object must lead itself to an efficient enumeration of subparts, and (b) the similarity function between subparts (i.e., the *local kernel*), beyond being a positive definite kernel, must be simple enough so that the sum over a potentially exponential number of terms can be recursively performed in polynomial time through factorization.

One of the most striking instantiations of this design principle are the *string kernels* (see, e.g., Shawe-Taylor and Cristianini (2004)), which consider all substrings of a given string but still allow efficient computation in polynomial time. The same principle can also be applied to graphs: intuitively, the *graph kernels* (Ramon & Gärtner, 2003; Kashima et al., 2004; Borgwardt et al., 2005) consider all possible subgraphs and compare and count matching subgraphs. However, the set of subgraphs (or even the set of paths) has exponential size and cannot be efficiently described recursively. By choosing appropriate substructures, such as *walks* or *tree-walks*, and fully factorized local kernels, matrix inversion formulations (Kashima et al., 2004) and efficient dynamic programming recursions (Harchaoui & Bach, 2007) allow one to sum over an exponential number of substructures in polynomial time (for more details

Appearing in *Proceedings of the 25^{th} International Conference on Machine Learning*, Helsinki, Finland, 2008.

on graph kernels, see Section 2.1).

In this paper, we consider the application of graph kernels to point clouds. Indeed, we assume that each point cloud has a graph structure (most often a neighborhood graph); then, our graph kernels consider all partial matches between two neighborhood graphs and sum over those. However, the straightforward application of graph kernels poses a major problem: in the context of computer vision, substructures correspond to matched sets of points, and dealing with local invariances by rotation and/or translation imposes to use a local kernel that cannot be readily expressed as a product of separate terms for each pair of points, and the usual dynamic programming and matrix inversion approaches cannot then be directly applied. One of the main contributions of this paper is to design a local kernel that is not fully factorized but can be instead factorized according to the graph underlying the substructure. This is naturally done through probabilistic graphical models and the design of positive definite kernels for covariance matrices that factorize on graphical models (see Section 3). With this novel local kernel, we derive new polynomial time dynamic programming recursions in Section 4. In Section 5, we present simulations on handwritten character recognition.

2. Graph Kernels

In this section, we consider two labelled undirected graphs $G = (V, E, a, x)$ and $H = (W, F, b, y)$, where V, W are vertex sets, E, F are edge sets and a, b, x, y are vertex labelling functions (Diestel, 2005). Two types of labels are considered: *attributes*, which are denoted $a(v) \in \mathcal{A}$ for vertex $v \in V$ and $b(w) \in \mathcal{A}$ for vertex $w \in W$ and *positions*, which are denoted $x(v) \in \mathcal{X}$ and $y(w) \in \mathcal{X}$. We assume that the graphs have no self-loops. Our motivating examples are line drawings, where $\mathcal{X} = \mathcal{A} = \mathbb{R}^2$ (i.e., the position is itself also an attribute). In this case, the graph is naturally obtained from the drawings by considering 4-connectivity or 8-connectivity (Forsyth & Ponce, 2003). In other cases, graphs can be easily obtained from nearest-neighbor graphs.

2.1. Related work

Graph data occur in many application domains, and kernels for attributed graphs have received increased interest in the applied machine learning literature, in particular in bioinformatics (Kashima et al., 2004; Borgwardt et al., 2005) and computer vision (Harchaoui & Bach, 2007). Note that in this paper, we only consider kernels between graphs (each data point is a graph), as opposed to kernels for a single dataset with associated graph information between data points (see, e.g., Shawe-Taylor and Cristianini (2004)).

Current graph kernels can roughly be divided in two classes: the first class is composed of non positive definite similarity measures based on existing techniques from the graph matching literature, that can be made positive definite by *ad hoc* matrix transformations; this includes the edit-distance kernel (Neuhaus & Bunke, 2006) and the optimal assignment kernel (Fröhlich et al., 2005; Vert, 2008).

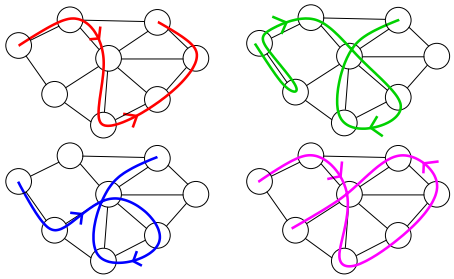

Figure 1. (top left) path, (top right) 1-walk which is not a 2-walk, (bottom left) 2-walk which is not a 3-walk, (bottom right) 4-walk.

Another class of graph kernels relies on a set of substructures of the graphs. The most natural ones are paths, subtrees and more generally subgraphs; however, they do not lead to positive definite kernels with polynomial time computation algorithms—see, in particular, NP-hardness results by Ramon and Gärtner (2003)—and recent work has focused on larger sets of substructures. In particular, *random walk* kernels consider all possible walks and sum a local kernel over all possible walks of the graphs (with all possible lengths). With a proper length-dependent factor, the computation can be achieved by solving a large sparse linear system (Kashima et al., 2004; Borgwardt et al., 2005), whose running time complexity has been recently reduced (Vishwanathan et al., 2007). When considering fixed-length walks, efficient dynamic programming recursions can de derived (Harchaoui & Bach, 2007) that drive down the computation time, at the cost of considering a smaller feature space. These however have the advantage of allowing extensions to other types of substructures, namely "tree-walks" (Ramon & Gärtner, 2003), that we now present.

2.2. Paths, Walks, Subtrees and Tree-walks

Given an undirected graph G with vertex set V, a *path* is a sequence of distinct connected vertices, while a *walk* is a sequence of possibly non distinct connected vertices. In order to prevent the walks from going back and forth too quickly (a phenomenon referred to as *tottering* by Mahé and Vert (2006)), we further restrain the set of walks; that is, for any positive integer β, we define β-walks as walks such that any $\beta+1$ successive vertices are distinct (1-walks are regular walks); see examples in Figure 1. Note that when the graph G is a tree (no cycles), then the set of 2-walks is equal to the set of paths. More generally, for any graph, β-walks of length $\beta + 1$ are exactly paths of length $\beta+1$. Note that the integer β corresponds to the "memory"

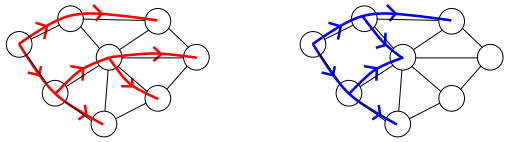

Figure 2. (left) binary 2-tree-walk, which in fact a subtree, (right) binary 1-tree-walk which is not a 2-tree-walk.

of the walk, i.e., the number of past vertices it needs to remember before going on.

A subtree of G is a subgraph of G with no cycles. A subtree of G can thus be seen as a connected subset of distinct nodes of G with an underlying tree structure. The notion of walk is extending the notion of path by allowing nodes to be equal; similarly, we can extend the notion of subtrees to *tree-walks*, which can have nodes that are equal. More precisely, we define an α-ary tree-walk of depth γ of G as a rooted labelled α-ary tree of depth γ with nodes labelled by vertices in G, and such that the labels of neighbors in the tree-walk must be neighbors in G (we refer to all allowed such set of labels as *consistent* labels). We assume that the tree-walks are not necessarily complete trees, i.e., each node may have less than α children. Tree-walks can be plotted on top of the original graph, as shown in Figure 2, and may be represented by a tree structure T over the vertex set $\{1, \dots, |T|\}$ and a tuple of consistent but possibly non distinct labels $I \in V^{|T|}$ (i.e., the labels of neighboring vertices in T must be neighboring vertices in G). Finally, in this paper, we consider only rooted subtrees, i.e., subtrees where a specific node is identified as the root; moreover, all the trees that we consider are unordered trees (i.e., no order is considered among siblings).

We can also define β-tree-walks, as tree-walks such that for each node in T, its label (which is an element of the original vertex set V) and the ones of all its descendants up to the β-th generation are all distinct. With that definition, 1-tree-walks are regular tree-walks (see Figure 2), and if $\alpha = 1$, we get back β-walks. From now on, we refer to the descendants up to the β-th generation as the β-descendants.

We let denote $\mathcal{T}_{\alpha,\gamma}$ the set of rooted tree structures of depth less than γ and with at most α children per node; for example, $\mathcal{T}_{1,\gamma}$ is exactly the set of chain graphs of length less than γ. For $T \in \mathcal{T}_{\alpha,\gamma}$, we denote $\mathcal{J}_\beta(T, G)$ the set of consistent labellings of T by vertices in V leading to β-tree-walks. With these definitions, a β-tree-walk of G is characterized by (a) a tree structure $T \in \mathcal{T}_{\alpha,\gamma}$ and (b) a labelling $I \in \mathcal{J}_\beta(T, G)$.

2.3. Graph Kernels

We assume that we are given a positive definite kernel between tree-walks that share the same tree structure, which we refer to as the *local kernel*. This kernel depends on the tree structure T and the set of attributes and positions associated with the nodes in the tree-walks (remember that each node of G and H has two labels, a position and an attribute). Given a tree structure T and consistent labellings $I \in \mathcal{J}_\beta(T, G)$ and $J \in \mathcal{J}_\beta(T, H)$, we let denote $q_{T,I,J}(G, H)$ the value of the local kernel between two tree-walks defined by the same structure T and labellings I and J.

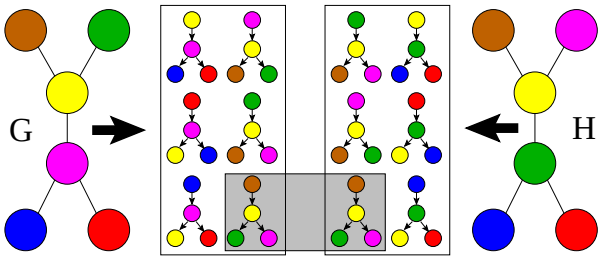

Figure 3. Graph kernels between two graphs (each color represents a different label). We display all binary 1-tree walks with a specific tree structure, extracted from two simple graphs; the graph kernels is computing and summing the local kernels between all those extracted tree-walks. In the case of the Dirac kernel (hard matching), only one pair of tree-walks is matched (for both labels and structures).

Following Ramon and Gärtner (2003), we can define the *tree-kernel* as the sum over all matching tree-walks of G and H of the local kernel, i.e.:

$$k^{\mathcal{T}}_{\alpha,\beta,\gamma}(G, H) = \sum_{T \in \mathcal{T}_{\alpha,\gamma}} f_{\lambda,\nu}(T) \times \sum_{I \in \mathcal{J}_\beta(T,G)} \sum_{J \in \mathcal{J}_\beta(T,H)} q_{T,I,J}(G, H). \quad (1)$$

When considering 1-walks (i.e., $\alpha = \beta = 1$), and letting the maximal walk length γ tend to $+\infty$, we get back the random walk kernel (Ramon & Gärtner, 2003; Kashima et al., 2004). If the kernel $q_{T,I,J}(G, H)$ has nonnegative values and is equal to 1 if the two tree-walks are equal, it can be seen as a soft matching indicator, and then the kernel in Eq. (1) simply counts the softly matched tree-walks in the two graphs (see Figure 3 for an illustration with hard matching).

We add a nonnegative penalization $f_{\lambda,\nu}(T)$ depending only on the tree-structure. Besides the usual penalization of the number of nodes $|T|$, we also add a penalization of the number of leaf nodes $\ell(T)$ (i.e., nodes with no children). More precisely, we use the penalization $f_{\lambda,\nu} = \lambda^{|T|}\nu^{\ell(T)}$. This penalization, suggested by Mahé and Vert (2006), is essential in our situation to avoid that trees with nodes of higher degrees dominate the sum.

If $q_{T,I,J}(G, H)$ is obtained from a positive definite kernel between (labelled) tree-walks, then $k^{\mathcal{T}}_{\alpha,\beta,\gamma}(G, H)$ also defines a positive definite kernel. The kernel $k^{\mathcal{T}}_{\alpha,\beta,\gamma}(G, H)$ sums the *local kernel* $q_{T,I,J}(G, H)$ over all tree-walks of G and H that share the same tree structure; the number

of such matching tree-walks is exponential in the depth γ, thus, in order to deal with potentially deep trees, a recursive definition is needed. As we now detail, it requires a specific type of local kernels, which can be decomposed according to tree structures.

2.4. Local Kernels

The local kernel is used between tree-walks which can have large depths (note that everything we propose will turn out to have linear time complexity in the depth γ). We use the product of a kernel for attributes and a kernel for positions. For attributes, we use the following usual factorized form $q_{\mathcal{A}}(a(I), b(J)) = \prod_{p=1}^{|I|} k_{\mathcal{A}}(a(I_p), b(J_p))$, where $k_{\mathcal{A}}$ is a positive definite kernel on $\mathcal{A} \times \mathcal{A}$. This allows the separate comparison of each matched pair of points and efficient dynamic programming recursions (Harchaoui & Bach, 2007). However, for our local kernel on positions, we need a kernel that *jointly* depends on the whole vectors $x(I) \in \mathcal{X}^{|I|}$ and $y(J) \in \mathcal{X}^{|J|}$, and not only on the p pairs $(x(I_p), y(J_p)) \in \mathcal{X} \times \mathcal{X}$. Indeed, we do not assume that the pairs are *registered*, i.e., we do not know the matching between points indexed by I in the first graph and the ones indexed by J in the second graph.

In this paper, we focus on $\mathcal{X} = \mathbb{R}^d$ and *translation invariant* local kernels, which implies that the local kernel for positions may only depend on differences $x(i) - x(i')$ and $y(j) - y(j')$ for $(i, i') \in I \times I$ and $(j, j') \in J \times J$. We further reduce these to kernel matrices corresponding to a translation invariant positive definite kernel $k_{\mathcal{X}}(x_1 - x_2)$. Depending on the application, $k_{\mathcal{X}}$ may or may not be rotation invariant. In simulations, we use the rotation invariant Gaussian kernel of the form $k_{\mathcal{X}}(x_1, x_2) = e^{-\upsilon \|x_1 - x_2\|^2}$.

Thus, we reduce the set of all positions in $\mathcal{X}^{|V|}$ and $\mathcal{X}^{|W|}$ to full kernel matrices $K \in \mathbb{R}^{|V| \times |V|}$ and $L \in \mathbb{R}^{|W| \times |W|}$ for each graph, defined as $K(v, v') = k_{\mathcal{X}}(x(v) - x(v'))$ (and similarly for L). These matrices are by construction symmetric positive semi-definite and, for simplicity, we assume that these matrices are positive definite (i.e., invertible), which can be enforced by adding a multiple of the identity matrix. The local kernel will thus only depend on the submatrices $K_I = K_{I,I}$ and $L_J = L_{J,J}$, which are positive definite matrices. Note that we use kernel matrices K and L to represent the geometry of each graph, and that we use a positive definite kernel on such kernel matrices.

We consider the following positive definite kernel on positive matrices K and L, the (squared) Bhattacharyya kernel $k_{\mathcal{B}}$, defined as (Kondor & Jebara, 2003):

$$k_{\mathcal{B}}(K, L) = |K|^{1/2} |L|^{1/2} \left| \tfrac{K+L}{2} \right|^{-1}, \qquad (2)$$

where $|K|$ denotes the determinant of K.

By taking the product of the attribute-based local kernel and the position-based local kernel, we get the following local kernel $q^0_{T,I,J}(G, H) = k_{\mathcal{B}}(K_I, L_J) q_{\mathcal{A}}(a(I), b(J))$. However, this local kernel $q^0_{T,I,J}(G, H)$ does not yet depend on the tree structure T and the recursion may be efficient only if $q^0_{T,I,J}(G, H)$ can be computed recursively. The factorized term $q_{\mathcal{A}}(a(I), b(J))$ does not cause any problems; however, for the term $k_{\mathcal{B}}(K_I, L_J)$, we need an approximation based on T. As we show in Section 3, this can be obtained by a factorization according to the appropriate graphical model, i.e., we will replace each kernel matrix of the form K_I by a projection onto a subset of kernel matrices which allow efficient recursions.

3. Positive Matrices and Graphical Models

The main idea underlying the factorization of the kernel is to consider symmetric positive definite matrices as covariance matrices and to look at probabilistic graphical models defined for Gaussian random vectors with those covariance matrices. The goal of this section is to show that by appropriate graphical model techniques, we can design properly factorized approximations of Eq. (2), namely through Eq. (6) and Eq. (7).

More precisely, we assume that we have n random variables $Z_1, \dots, Z_n$ with probability distribution $p(z) = p(z_1, \dots, z_n)$. Given a kernel matrix K (in our case defined as $K_{ij} = e^{-\upsilon \|x_i - x_j\|^2}$, for positions $x_1, \dots, x_n$), we consider jointly Gaussian distributed random variables $Z_1, \dots, Z_n$ such that $\operatorname{cov}(Z_i, Z_j) = K_{ij}$. In this section, with this identification, we consider covariance matrices as kernel matrices, and vice-versa.

3.1. Graphical Models and Junction Trees

Graphical models provide a flexible and intuitive way of defining factorized probability distributions. Given any undirected graph Q with vertices in $\{1, \dots, n\}$, the distribution $p(z)$ is said to factorize in Q if it can be written as a product of potentials over all cliques (completely connected subgraphs) of the graph Q. When the distribution is Gaussian with covariance matrix $K \in \mathbb{R}^{n \times n}$, the distribution factorizes if and only if $(K^{-1})_{ij} = 0$ for each (i, j) which is not an edge in Q (Lauritzen, 1996).

In this paper, we only consider *decomposable* graphical models, for which the graph Q is *triangulated* (i.e., there exists no chordless cycle of length strictly larger than 3). In this case, the joint distribution is uniquely defined from its marginals $p_C(z_C)$ on the cliques C of the graph Q. Namely, if $\mathcal{C}(Q)$ is the set of maximal cliques of Q, we can build a tree of cliques, a *junction tree*, such that $p(z) = \prod_{C \in \mathcal{C}(Q)} p_C(z_C) / \prod_{C, C' \in \mathcal{C}(Q), C \sim C'} p_{C \cap C'}(z_{C \cap C'})$ (see Figure 4 for an example of a graphical model and a junction tree). The sets $C \cap C'$ are usually referred to as *separators* and we let denote $\mathcal{S}(Q)$ the set of such separators. Note that for a zero mean normally distributed vector, the marginals

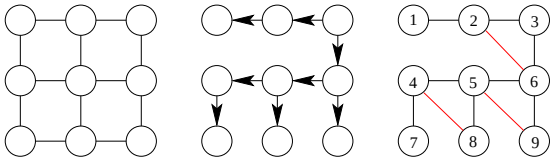

Figure 4. (left) original graph, (middle) a single extracted tre-walk, (right) decomposable graphical model $Q_1(T)$ with added edges in red, defined in Section 3.4. The junction tree is a chain composed of the cliques $\{1,2\},\{2,3,6\},\{5,6,9\},\{4,5,8\},\{4,7\}$.

$p_C(z_C)$ are characterized by the marginal covariance matrix $K_C = K_{C,C}$. Projecting onto a graphical model will preserve the marginal over all maximal cliques, and thus preserve the local kernel matrices, while imposing zeros in the inverse of K.

3.2. Graphical Models and Projections

We let denote $\Pi_Q(K)$ the covariance matrix that factorizes in Q which is closest to K for the Kullback-Leibler divergence between normal distributions. In this paper, we essentially replace K by $\Pi_Q(K)$; i.e., we project all our covariance matrices onto a graphical model, which is a classical tool in probabilistic modelling (Lauritzen, 1996). We leave the study of the approximation properties of such a projection (i.e., for a given K, how dense the graph should be to approximate the full local kernel correctly?) to future work—see, e.g., Caetano et al. (2006) for related results.

Practically, since our kernel on kernel matrices involves determinants, we simply need to compute $|\Pi_Q(K)|$ efficiently. For decomposable graphical models, $\Pi_Q(K)$ can be obtained in closed form (Lauritzen, 1996) and its determinant has the following simple expression:

$$\log|\Pi_Q(K)| = \sum_{C\in\mathcal{C}(Q)} \log|K_C| - \sum_{S\in\mathcal{S}(Q)} \log|K_S|. \quad (3)$$

The determinant $|\Pi_Q(K)|$ is thus a ratio of terms (determinants over cliques and separators), which will restrict the applicability of the projected kernels (see Proposition 1). In order to keep only products, we consider the following equivalent form: if the junction tree is rooted (by choosing any clique as the root), then for each clique but the root, a unique parent clique is defined, and we have:

$$\begin{aligned}\log|\Pi_Q(K)| &= \textstyle\sum_{C\in\mathcal{C}(Q)} \log\frac{|K_C|}{|K_{p_Q(C)}|} \\ &= \textstyle\sum_{C\in\mathcal{C}(Q)} \log|K_{C|p_Q(C)}|, \end{aligned} \quad (4)$$

where $p_Q(C)$ is the parent clique of Q (and $\varnothing$ for the root clique) and the conditional covariance matrix is defined, as usual, as $K_{C|p_Q(C)} = K_{C,C} - K_{C,p_Q(C)}K^{-1}_{p_Q(C),p_Q(C)}K_{p_Q(C),C}$ (Lauritzen, 1996).

3.3. Graphical Models and Kernels

We now propose several ways of defining a kernel adapted to graphical models. All of them are based on replacing determinants $|M|$ by $|\Pi_Q(M)|$, and their different decompositions in Eq. (3) and Eq. (4). Simply using Eq. (3), we obtain the similarity measure:

$$k^Q_{\mathcal{B},0}(K,L) = \prod_{C\in\mathcal{C}(Q)} k_{\mathcal{B}}(K_C,L_C) \prod_{S\in\mathcal{S}(Q)} k_{\mathcal{B}}(K_S,L_S)^{-1}. \quad (5)$$

which turns out not to be a positive definite kernel for general covariance matrices:

Proposition 1 *For any decomposable model Q, the kernel $k^Q_{\mathcal{B},0}$ defined in Eq. (5) is a positive definite kernel on the set of covariance matrices K such that for all separators $S\in\mathcal{S}(Q)$, $K_{S,S} = I$. In particular, when all separators have cardinal one, this is a kernel on correlation matrices.*

In order to remove the condition on separators (i.e., we want more sharing between cliques than through a single variable), we consider the rooted junction tree representation in Eq. (4). A straightforward kernel is to compute the product of the Bhattacharyya kernels $k_{\mathcal{B}}(K_{C|p_Q(C)}, L_{C|p_Q(C)})$ for each conditional covariance matrix. However, this does not lead to a true distance on covariance matrices that factorize on Q because the set of conditional covariance matrices do not characterize entirely those distributions. Rather, we consider the following kernel:

$$k^Q_{\mathcal{B}}(K,L) = \textstyle\prod_{C\in\mathcal{C}(Q)} k^{C|p_Q(C)}_{\mathcal{B}}(K,L); \quad (6)$$

for the root clique, we define $k^{R|\varnothing}_{\mathcal{B}}(K,L) = k_{\mathcal{B}}(K_R,L_R)$ and the kernels $k^{C|p_Q(C)}_{\mathcal{B}}(K,L)$ are defined as kernels between conditional Gaussian distributions of Z_C given $Z_{p_Q(C)}$. We use

$$k^{C|p_Q(C)}_{\mathcal{B}}(K,L) = \frac{|K_{C|p_Q(C)}|^{1/2}|L_{C|p_Q(C)}|^{1/2}}{\left|\frac{1}{2}K_{C|p_Q(C)}+\frac{1}{2}L_{C|p_Q(C)}+MM^\top\right|}, \quad (7)$$

where the additional term M is equal to $\frac{1}{2}(K_{C,p_Q(C)}K^{-1}_{p_Q(C)} - L_{C,p_Q(C)}L^{-1}_{p_Q(C)})$. This exactly corresponds to putting a prior with identity covariance matrix on variables $Z_{p_Q(C)}$ and considering the kernel between the resulting joint covariance matrices on variables indexed by $(C,p_Q(C))$. We now have a positive definite kernel on all covariance matrices:

Proposition 2 *For any decomposable model Q, the kernel $k^Q_{\mathcal{B}}(K,L)$ defined in Eq. (6) and Eq. (7) is a positive definite kernel on the set of covariance matrices.*

Note that the kernel is not invariant by the choice of the particular root of the junction tree. However, in our setting, this is not an issue because we have a natural way of rooting the junction trees (i.e, following the rooted tree-walk, see Section 3.4). Note that these kernels could be useful in other domains than point clouds and computer vision.

In Section 4, we will use the notation $k_{\mathcal{B}}^{I_1|I_2,J_1|J_2}(K,L)$ for $|I_1| = |I_2|$ and $|J_1| = |J_2|$ to denote the kernel between covariance matrices $K_{I_1\cup I_2}$ and $L_{I_1\cup I_2}$ adapted to the conditional distributions $I_1|I_2$ and $J_1|J_2$, defined through Eq. (7).

3.4. Choice of Graphical Models

Given the rooted tree structure T of a β-tree-walk, we now need to define the graphical model $Q_\beta(T)$ that we use to project our kernel matrices. A natural candidate is T itself; however, as shown in Section 4, in order to compute efficiently the kernel we simply need that the local kernel is a product of terms that only involve a node and its β-descendants. The densest graph (remember that denser graphs lead to better approximations when projecting onto the graphical model) we may use is exactly the following: we define $Q_\beta(T)$ such that for all nodes in T, the node together with all its β-descendants form a clique, i.e., a node is connected to its β-descendants and all β-descendants are also mutually connected (see Figure 4 for example for $\beta = 1$): the set of cliques are thus the set of *families* of depth $\beta+1$ (i.e., with $\beta+1$ generations). Thus, our final kernel is:

$$k_{\alpha,\beta,\gamma}^{\mathcal{T}}(G,H) = \sum_{T\in\mathcal{T}_{\alpha,\gamma}} f_{\lambda,\nu}(T)\times$$
$$\sum_{I\in\mathcal{J}_\beta(T,G)}\ \sum_{J\in\mathcal{J}_\beta(T,H)} k_{\mathcal{B}}^{Q_\beta(T)}(K_I,L_J)q_{\mathcal{A}}(a(I),b(J)). \quad (8)$$

The main intuition behind this definition is to sum local similarities over all matching subgraphs. In order to obtain a tractable formulation, we simply needed (a) to extend the set of subgraphs (to tree-walks of depth γ) and (b) to factorize the local similarities along the graphs. We now show how these elements can be combined to derive efficient recursions.

4. Dynamic Programming Recursions

In order to derive dynamic programming recursions, we follow Mahé and Vert (2006) and rely on the fact that α-ary β-tree-walks of G can essentially be defined through 1-tree-walks on the augmented graph of all rooted subtrees of G of depth at most β and arity less than α. We thus consider the set $V_{\alpha,\beta}$ of non complete rooted (unordered) subtrees of $G = (V,E)$, of depths less than β and arity less than α. Given two different rooted unordered labelled trees, they are said *equivalent* (or isomorphic) if they share the same tree structure, and this is denoted $\sim_t$.

On this set $V_{\alpha,\beta}$, we define a *directed* graph with edge set $E_{\alpha,\beta}$ as follows: $R_0 \in V_{\alpha,\beta}$ is connected to $R_1 \in V_{\alpha,\beta}$ if "the tree R_1 extends the tree R_0 one generation further", i.e., if and only if (a) the first $\beta-1$ generations of R_1 are exactly equal to one of the complete subtree of R_0 rooted at a child of the root of R_0, and (b) the nodes of depth β of R_1 are distinct from the nodes in R_0. This defines a graph $G_{\alpha,\beta} = (V_{\alpha,\beta}, E_{\alpha,\beta})$ and a neighborhood $\mathcal{N}_{G_{\alpha,\beta}}(R)$ for $R \in V_{\alpha,\beta}$ (see Figure 5 for an example). Similarly we define a graph $H_{\alpha,\beta} = (W_{\alpha,\beta}, F_{\alpha,\beta})$ for the graph H. Note that when $\alpha = 1$, $V_{1,\beta}$ is the set of paths of length less than or equal to β.

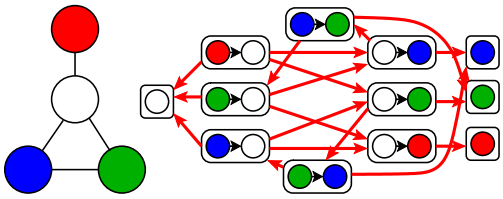

Figure 5. (left) undirected graph G, (right) graph $G_{1,2}$.

For a β-tree-walk, the root with its β-descendants must have distinct vertices and thus corresponds exactly to an element of $V_{\alpha,\beta}$. We denote $k_{\alpha,\beta,\gamma}^{\mathcal{T}}(G,H,R_0,S_0)$ the same kernel as defined in Eq. (8), but restricted to tree-walks that start respectively with R_0 and S_0. Note that if R_0 and S_0 are not equivalent, then $k_{\alpha,\beta,\gamma}^{\mathcal{T}}(G,H,R_0,S_0) = 0$.

We obtain the following recursion between depths γ and depth $\gamma-1$, for all $R_0 \in V_{\alpha,\beta}$ and and $S_0 \in W_{\alpha,\beta}$ such that $R_0 \sim_t S_0$:

$$k_{\alpha,\beta,\gamma}^{\mathcal{T}}(G,H,R_0,S_0) = k_{\alpha,\beta,\gamma-1}^{\mathcal{T}}(G,H,R_0,S_0)$$
$$+\sum_{p=1}^{\alpha} \sum_{\substack{R_1,\dots,R_p \in \mathcal{N}_{G_{\alpha,\beta}}(R_0) \\ R_1,\dots,R_p \text{ disjoint}}} \sum_{\substack{S_1,\dots,S_p \in \mathcal{N}_{H_{\alpha,\beta}}(S_0) \\ S_1,\dots,S_p \text{ disjoint}}}$$
$$\left[\lambda \prod_{i=1}^{p} k_{\mathcal{A}}(a(\mathrm{root}(R_i)),b(\mathrm{root}(S_i)))\times \right.$$
$$\left. \frac{k_{\mathcal{B}}^{\cup_{i=1}^p R_i|R_0,\cup_{i=1}^p S_i|S_0}(K,L)}{\prod_{i=1}^p k_{\mathcal{B}}^{R_i,S_i}(K,L)} \left(\prod_{i=1}^{p} k_{\alpha,\beta,\gamma-1}^{\mathcal{T}}(G,H,R_i,S_i)\right)\right].$$

Note that if any of the trees R_i is not equivalent to S_i, it does not contribute to the sum. The recursion is initialized with $k_{\alpha,\beta,\gamma}^{\mathcal{T}}(G,H,R_0,S_0) = \lambda^{|R_0|}\nu^{\ell(R_0)}q_{\mathcal{A}}(a(R_0),b(S_0))k_{\mathcal{B}}(K_{R_0},L_{S_0})$ while the final kernel is obtained by summing over all R_0 and S_0, i.e, $k_{\alpha,\beta,\gamma}^{\mathcal{T}}(G,H) = \sum_{R_0\sim_t S_0} k_{\alpha,\beta,\gamma}^{\mathcal{T}}(G,H,R_0,S_0)$.

Computational Complexity The complexity of computing one kernel between two graphs is linear in γ (the depth of the tree-walks), and quadratic in the size of $V_{\alpha,\beta}$ and $W_{\alpha,\beta}$. However, those sets may have exponential size in β and α in general (in particular if graphs are densely connected). And thus, we are limited to small values (typically $\alpha \leqslant 3$ and $\beta \leqslant 6$) which are sufficient for good classification performance (in particular, higher β or α do not necessarily mean better performance, see Section 5). Overall, one can deal with any graph size, as long as the "sufficient statistics" (i.e., the unique local neighorhoods in $V_{\alpha,\beta}$) are not too numerous.

Figure 6. For digits and Chinese characters: (left) original characters, (right) thinned and subsampled characters.

For example, for the handwritten digits we use in simulations, the average number of nodes in the graphs is 18 ± 4, while the average cardinal of $V_{\alpha,\beta}$ and running times[1] for one kernel evaluation are, for walk kernels of depth 24: $|V_{\alpha,\beta}| = 36$, $T = 2$ ms ($\alpha = 1$, $\beta = 2$), $|V_{\alpha,\beta}| = 37$, $T = 3$ ms ($\alpha = 1$, $\beta = 4$); and for tree-kernels: $|V_{\alpha,\beta}| = 56$, $T = 25$ ms ($\alpha = 2$, $\beta = 2$), $|V_{\alpha,\beta}| = 70$, $T = 32$ ms ($\alpha = 2$, $\beta = 4$).

Finally, we may reduce the computational load by considering a set of trees of smaller arity in the previous recursions; i.e., we can consider $V_{1,\beta}$ instead of $V_{\alpha,\beta}$ with tree-kernels of arity $\alpha > 1$.

5. Application to Character Recognition

We have tested our new kernels on the task of isolated handwritten character recognition, handwritten arabic numerals (MNIST dataset) and Chinese characters (ETL9B dataset). We selected the first 100 examples for the ten classes in the MNIST dataset, while for the ETL9B dataset, we selected the five hardest classes to discriminate among 3,000 classes (by computing distances between class means) and then selected the first 50 examples per class. Our learning task it to classify those characters; we use a one-vs-rest multiclass scheme with 1-norm support vector machines (see, e.g., Shawe-Taylor and Cristianini (2004)).

We consider characters as drawings in $\mathbb{R}^2$, which are sets of possibly intersecting contours. Those are naturally represented as undirected planar graphs. We have thinned and subsampled uniformly each character to reduce the sizes of the graphs (see two examples in Figure 6).

The kernel on positions is $k_{\mathcal{X}}(x, y) = \exp(-\tau\|x-y\|^2) + \kappa\delta(x, y)$, but could take into account different weights on horizontal and vertical directions. We add the positions from the center of the bounding box as features, to take into account the global positions, i.e., we use $k_{\mathcal{A}}(x, y) = \exp(-\upsilon\|x-y\|^2)$. This is necessary because the problem of handwritten character recognition is not globally translation invariant.

[1]Those do not take into account preprocessing and were evaluated on an Intel Xeon 2.33 GHz processor from MATLAB/C code, and are to be compared to the simplest recursions which correspond to the usual random walk kernel ($\alpha = 1$, $\beta = 1$), where $T = 1$ ms.

In this paper, we have defined a family of kernels, corresponding to different values of the following free parameters (shown with their possible values): arity of tree-walks ($\alpha = 1, 2$), order of tree-walks ($\beta = 1, 2, 4, 6$), depth of tree-walks ($\gamma = 1, 2, 4, 8, 16, 24$), penalization on number of nodes ($\lambda = 1$), penalization on number of leaf nodes ($\nu = .1, .01$), bandwidth for kernel on positions ($\tau = .05, .01, .1$), ridge parameter ($\kappa = .001$), bandwidth for kernel on attributes ($\upsilon = .05, .01, .1$).

The first two sets of parameters $(\alpha, \beta, \gamma, \lambda, \nu)$ are parameters of the graph kernel, independent of the application, while the last set (τ, κ, ν) are parameters of the kernels for attributes and positions. Note that with only a few important scale parameters (τ and ν), we are able to characterize complex interactions between the vertices and edges of the graphs. In practice, this is important to avoid considering many more distinct parameters for all sizes and topologies of subtrees.

In simulations, we performed two loops of 5-fold cross-validation: in the outer loop, we consider 5 different training folds with their corresponding testing folds. On each training fold, we consider all possible values of α and β. For all of those values, we select all other parameters (including the regularization parameters of the SVM) by 5-fold cross-validation (the inner folds). Once the best parameters are found only by looking only at the training fold, we train on the whole training fold, and test on the testing fold. We output the means and standard deviations of the testing errors for each testing fold. We show in Figure 7 the performance for various values of α and β. We compare those favorably to three baseline kernels with hyperparameters learned by cross-validation in the same way: (a) the *Gaussian-RBF kernel* on the vectorized original images, which leads to testing errors of $11.6 \pm 5.4\%$ (MNIST) and $50.4 \pm 6.2\%$ (ETL9B); (b) the regular *random walk kernel* which sums over all walk lengths, which leads to testing errors of $8.6 \pm 1.3\%$ (MNIST) and $34.8 \pm 8.4\%$ (ETL9B); and (c) the *pyramid match kernel* (Grauman & Darrell, 2007), which is commonly used for image classification and leads here to testing errors of $10.8 \pm 3.6\%$ (MNIST) and $45.2 \pm 3.4\%$ (ETL9B).

These results show that our new family of kernels that use the natural structure of line drawings are outperforming other kernels on structured data (regular random walk kernel and pyramid match kernel) as well as the "blind" Gaussian-RBF kernel which does not take into account explicitly the structure of images but still leads to very good performance with more training data (LeCun et al., 1998). Note that for arabic numerals, higher arity does not help, which is not surprising since most digits have a linear structure (i.e, graphs are chains). On the contrary, for Chinese characters, which exhibit higher connectivity, best performance is achieved for binary tree-walks.

	MNIST $\alpha = 1$	MNIST $\alpha = 2$	ETL9B $\alpha = 1$	ETL9B $\alpha = 2$
$\beta = 1$	11.6 ± 4.6	9.2 ± 3.9	36.8 ± 4.6	32 ± 8.4
$\beta = 2$	5.6 ± 3.1	5.6 ± 3.0	29.2 ± 8.8	$\mathbf{25.2 \pm 2.7}$
$\beta = 4$	$\mathbf{5.4 \pm 3.6}$	$\mathbf{5.4 \pm 3.1}$	32.4 ± 3.9	29.6 ± 4.3
$\beta = 6$	5.6 ± 3.3	6 ± 3.5	29.6 ± 4.6	28.4 ± 4.3

Figure 7. Error rates (multiplied by 100) on handwritten character classification tasks.

6. Conclusion

We have presented a new kernel for point clouds which is based on comparisons of local subsets of the point clouds. Those comparisons are made tractable by (a) considering subsets based on tree-walks and walks, and (b) using a specific factorized form for the local kernels between tree-walks, namely a factorization on a properly defined probabilistic graphical model.

Moreover, we have reported applications to handwritten character recognition where we showed that the kernels were able to capture the relevant information to allow good predictions from few training examples. We are currently investigating other domains of applications of points clouds, such as shape mining in computer vision (Belongie et al., 2002), and prediction of protein functions from their three-dimensional structures (Qiu et al., 2007).

Acknowledgements

We would like to thank Zaïd Harchaoui and Jean-Philippe Vert for fruitful discussions related to this work.

References

Bach, F. R., Lanckriet, G. R. G., & Jordan, M. I. (2004). Multiple kernel learning, conic duality, and the SMO algorithm. *Proc. ICML*.

Belongie, S., Malik, J., & Puzicha, J. (2002). Shape matching and object recognition using shape contexts. *IEEE Trans. PAMI, 24*, 509–522.

Borgwardt, K. M., Ong, C. S., Schönauer, S., Vishwanathan, S. V. N., Smola, A. J., & Kriegel, H.-P. (2005). Protein function prediction via graph kernels. *Bioinformatics, 21*.

Caetano, T., Caelli, T., Schuurmans, D., & Barone, D. (2006). Graphical models and point pattern matching. *IEEE Trans. PAMI, 28*, 1646–1663.

Chapelle, O., Schölkopf, B., & Zien, A. (Eds.). (2006). *Semi-supervised learning (adaptive computation and machine learning)*. MIT Press.

Diestel, R. (2005). *Graph theory*. Springer-Verlag.

Forsyth, D. A., & Ponce, J. (2003). *Computer vision: A modern approach*. Prentice Hall.

Fröhlich, H., Wegner, J. K., Sieker, F., & Zell, A. (2005). Optimal assignment kernels for attributed molecular graphs. *Proc. ICML*.

Grauman, K., & Darrell, T. (2007). The pyramid match kernel: Efficient learning with sets of features. *J. Mach. Learn. Res., 8*, 725–760.

Harchaoui, Z., & Bach, F. (2007). Image classification with segmentation graph kernels. *Proc. CVPR*.

Kashima, H., Tsuda, K., & Inokuchi, A. (2004). Kernels for graphs. *Kernel Methods in Comp. Biology*. MIT Press.

Kondor, R. I., & Jebara, T. (2003). A kernel between sets of vectors. *Proc. ICML*.

Lauritzen, S. (1996). *Graphical models*. Oxford U. Press.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proc. IEEE, 86*, 2278–2324.

Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., & Watkins, C. (2002). Text classification using string kernels. *J. Mach. Learn. Res., 2*, 419–444.

Mahé, P., & Vert, J.-P. (2006). *Graph kernels based on tree patterns for molecules* (Tech. report HAL-00095488).

Neuhaus, M., & Bunke, H. (2006). Edit distance based kernel functions for structural pattern classification. *Pattern Recognition, 39*, 1852–1863.

Parsana, M., Bhattacharyya, C., Bhattacharya, S., & Ramakrishnan, K. R. (2008). Kernels on attributed pointsets with applications. *Adv. NIPS*.

Qiu, J., Hue, M., Ben-Hur, A., Vert, J.-P., & Noble, W. S. (2007). A structural alignment kernel for protein structures. *Bioinformatics, 23*, 1090–1098.

Ramon, J., & Gärtner, T. (2003). Expressivity versus efficiency of graph kernels. *First International Workshop on Mining Graphs, Trees and Sequences*.

Shawe-Taylor, J., & Cristianini, N. (2004). *Kernel methods for pattern analysis*. Cambridge Univ. Press.

Srihari, S. N., Yang, X., & Ball, G. R. (2007). Offline Chinese handwriting recognition: A survey. *Frontiers of Computer Science in China*.

Vert, J.-P. (2008). *The optimal assignment kernel is not positive definite* (Tech. report HAL-00218278).

Vert, J.-P., Saigo, H., & Akutsu, T. (2004). Local alignment kernels for biological sequences. *Kernel Methods in Comp. Biology*. MIT Press.

Vishwanathan, S. V. N., Borgwardt, K. M., & Schraudolph, N. (2007). Fast computation of graph kernels. *Adv. NIPS*.