Tree-walk kernels for computer vision

Zaid Harchaoui, Francis Bach

Chapter 1

Tree-walk kernels for computer vision

ZAID HARCHAOUI

LEAR project-team
INRIA and LJK
655, avenue de l'Europe
38330 Montbonnot Saint-Martin. FRANCE
Email: zaid.harchaoui@inria.fr

FRANCIS BACH

SIERRA project-team
INRIA and LIENS
23, avenue d'Italie
75214 Paris Cedex 13. FRANCE
Email: francis.bach@inria.fr

1.1 Introduction

Kernel-based methods have proven highly effective in many applications because of their wide generality. As soon as a similarity measure leading to a symmetric positive-definite kernel can be designed, a wide range of learning algorithms working on Euclidean dot-products of the examples can be used, such as support vector machines (1). Replacing the euclidian dot-products by the kernel evaluations then corresponds to applying the learning algorithm on dot products of feature maps of the examples into a higher-dimensional

space. Kernel-based methods are now well-established tools for computer vision—see, e.g., (2) for a review.

A recent line of research consists in designing kernels for structured data, to address problems arising in bioinformatics (3) or text processing. Structured data are data that can be represented through a labelled discrete structure, such as strings, labelled trees or labelled graphs (4). 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 can be beneficial. It usually allows to reduce the number of training examples to reach a high performance accuracy. It is also an elegant way to leverage existing data representations that are already well developed by experts of those domains. Last but not least, as soon as a new kernel on structured data is defined, one can bring to bear the rapidly developing kernel machinery, and in particular semi-supervised learning—see, e.g., (5)—and kernel learning —see, e.g., (6).

We propose a general framework to build positive-definite kernels between images, allowing us to leverage, e.g., shape and appearance information to build meaningful similarity measures between images. On the one hand, we introduce positive-definite kernels between appearances as coded in region adjacency graphs, with applications to classification of images of object categories, as by (7, 8). On the other hand, we introduce positive-definite kernels between shapes as coded in point clouds, with applications to classification of line drawings, as in (9; 10; 11). Both types of information are difficult to encode into vector features without loss of information. For instance, in the case of shapes defined through point clouds, local and global invariances with respect to rotation/translation need to be encoded in the feature space representation.

A leading principle 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 (1). 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* (1), 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* (12; 13) 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 (14) and efficient dynamic programming recursion allow one to sum over an exponential number of substructures in polynomial time. We first review previous work on graph kernels. Then we present our general framework for tree-walk kernels in Section 1.2. In Section 1.3, we show how one can instantiate tree-walk kernels to build

a powerful positive-definite similarity measure between region adjacency graphs. Then, in Section 1.4, we show how to instantiate tree-walk kernels to build an efficient positive-definite similarity measure between point clouds. Finally, in Section 1.5, we present experimental results of our tree-walk kernels resp. on object categorization, natural image classification, and handwritten digit classification tasks. This chapter is based on two conference papers (15; 16).

1.1.1 Related work

The underlying idea of our tree-walk kernels is to provide an expressive relaxation of graph-matching similarity scores yet leading to a positive-definite kernel. The idea of matching graphs for the purpose of image classification has a long history in computer vision, and has been investigated by many authors (17; 18; 19). However, the general *graph matching* problem is especially hard as most of the simple operations which are simple on strings (such as matching and edit distances) are NP-hard for general undirected graphs. Namely, while exact graph matching is unrealistic in our context, inexact graph matching is NP-hard and subgraph matching is NP-complete (4). Hence most work so far has focused on finding ingenious approximate solutions to this challenging problem (20). An interesting line of research consists in overcoming graph matching problems by projecting graphs on strings by the so-called seriation procedure (18), and then use string edit distance (21) as a proxy to graph edit distance.

Designing kernels for image classification is also an active research topic. Since the work of (22) who investigated image classification with kernels on color histograms, many kernels for image classification were proposed. The bag-of-pixels kernels proposed in (23; 24) compare color distribution of two images by using kernels between probability measures, and was extended to hierarchical multi-scale settings in (24; 25); see (8) for in-depth review.

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 (14; 13) and computer vision. Note that in this work, we only consider kernels between graphs (each data point is a graph), as opposed to settings where kernels are used to build a neighborhood graph from all the data points of a dataset (1).

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 (26) and the optimal assignment kernel (27; 28). In (26), the authors propose to use the edit distance between two graphs and directly plug it into a kernel to define a similarity measure between graphs. Kernels between shock graphs for pedestrian detection were also proposed in (29), allowing them to capture the topological

structure of images. Our method efficiently circumvents the graph matching problem by soft-matching tree-walks, i.e., virtual substructures of graphs, in order to obtain kernels computable in polynomial time. Our kernels keep the underlying topological structure of graphs by describing them through tree-walks, while in graph seriation the topological structure somehow fades away by only keeping one particular substring of the graph. Moreover our kernels encompasses local information, by using segment histograms as local features, as well as global information, by summing up all soft matchings of tree-walks.

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 (12)—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 (14; 30), whose running time complexity has been recently reduced (31). When considering fixed-length walks, efficient dynamic programming recursions allow to drive down the computation time, at the cost of considering a smaller feature space. This is the approach we chose to follow here. Indeed, such an approach allows extensions to other types of substructures, namely "tree-walks", whose abstract definition were first proposed by (12), that we now present. Several works later extended our approach to other settings, such as (32) for feature extraction from images and (33) for scene interpretation.

1.2 Tree-walk kernels as graph kernels

We first describe in this section the general framework in which we define tree-walk kernels. We shall then instantiate the tree-walk kernels resp. on region adjacency graphs in Section 1.3 and on point clouds in Section 1.4.

Let us consider two labelled undirected graphs $\mathcal{G} = (\mathcal{V}_\mathcal{G}, \mathcal{E}_\mathcal{G})$ and $\mathcal{H} = (\mathcal{V}_\mathcal{H}, \mathcal{E}_\mathcal{H})$, where $\mathcal{V}_\mathcal{G}$ and $\mathcal{V}_\mathcal{H}$ denote the set of vertices resp. of $\mathcal{G}$ and $\mathcal{H}$, $\mathcal{E}_\mathcal{G}$ and $\mathcal{H}$ denote the set of edges resp. of $\mathcal{G}$ and $\mathcal{H}$. For the sake of clarity, we shall assume in the remainder that the vertex sets have the same cardinality; yet our framework can also handle vertex sets with different cardinality. Here, both the vertices and the edges can be labelled. Two types of labels are considered: *attributes*, which are denoted $a(v) \in \mathcal{A}_\mathcal{G}$ for vertex $v \in \mathcal{V}_\mathcal{G}$, and $b(v') \in \mathcal{A}_\mathcal{H}$ for vertex $v' \in \mathcal{V}_\mathcal{H}$ and *positions*, which are denoted $x(v) \in \mathcal{X}_\mathcal{G}$ and $y(v') \in \mathcal{X}_\mathcal{H}$. We make the distinction here between attributes and positions, since different kernels should be used to mesure similarity between attributes (resp. positions).

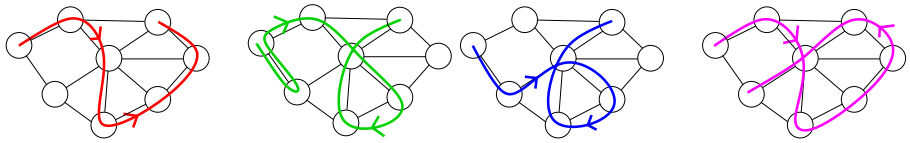

Figure 1.1: From left to right: path (red), 1-walk which is not a 2-walk (green), 2-walk which is not a 3-walk (blue), 4-walk (magenta).

1.2.1 Paths, Walks, Subtrees and Tree-walks

Given an undirected graph $\mathcal{G}$ with vertex set $\mathcal{V}$, 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, we further restrain the set of walks. Such phenomenon was termed *tottering* by (34)). 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 1.1. Note that when the graph $\mathcal{G}$ 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" of the walk, i.e., the number of past vertices it needs to "remember" before going on.

A *subtree* of $\mathcal{G}$ is a subgraph of $\mathcal{G}$ with no cycles (4). A subtree of $\mathcal{G}$ 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 vertices to be equal. Similarly, we can extend the notion of subtrees to *tree-walks*, where equal vertices can occur. More precisely, we define an α-ary tree-walk of depth γ of $\mathcal{G}$ as a rooted labelled α-ary tree of depth γ whose vertices share the same labels as the labels of the corresponding vertices in $\mathcal{G}$. In addition, the labels of neighbors in the tree-walk must be neighbors in $\mathcal{G}$ (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 1.5, and may be represented by a tree structure T over the vertex set $\{1,\ldots,|T|\}$ 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 $\mathcal{G}$). Finally, we only consider 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). Yet, it is worthwhile to note that taking into account the order becomes more important when considering subtrees of region adjacency graphs.

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 1.2), 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.

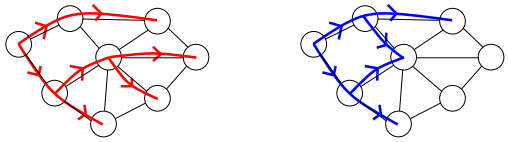

Figure 1.2: (left) binary 2-tree-walk, which in fact a subtree, (right) binary 1-tree-walk which is not a 2-tree-walk.

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 $I \in \mathcal{J}_\beta(T,G)$.

1.2.2 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 $\mathcal{G}$ and $\mathcal{H}$ 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 $q_{T,\mathcal{J},\mathcal{J}}(\mathcal{G},\mathcal{H})$ the value of the local kernel between two tree-walks defined by the same structure T and labellings I and J.

We can define the *tree-walk* kernel as the sum over all matching tree-walks of $\mathcal{G}$ and $\mathcal{H}$ of the local kernel, i.e.:

$$k^{\mathcal{T}}_{\alpha,\beta,\gamma}(\mathcal{G},\mathcal{H}) = \sum_{T \in \mathcal{T}_{\alpha,\gamma}} f_{\lambda,\nu}(T) \left\{ \sum_{I \in \mathcal{J}_\beta(T,\mathcal{G})} \sum_{J \in \mathcal{J}_\beta(T,\mathcal{H})} q_{T,I,J}(\mathcal{G},\mathcal{H}) \right\} . \tag{1.1}$$

Details on the efficient computation of the tree-walk kernel are given in the next section. 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 (12; 14). If the kernel $q_{T,I,J}(\mathcal{G},\mathcal{H})$ 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 simply counts the softly matched tree-walks in the two graphs (see Figure 1.3 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 (34), is essential in our situation to avoid that trees with nodes of higher degrees dominate the sum. Yet, we shall most often drop this penalization in subsequent derivations to highlight essential

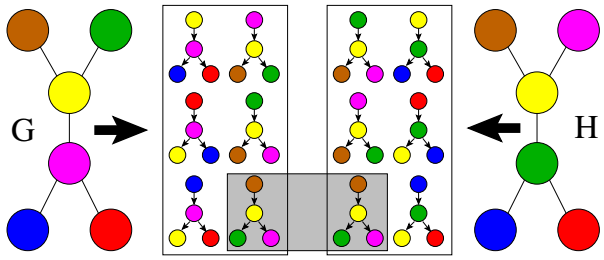

Figure 1.3: Graph kernels between two graphs (each color represents a different label). We display all binary 1-tree walks with a specific tree structure, extracted from two simple graphs. The graph kernels is computing and summing the local kernels between all those extracted tree-walks. In the case of the Dirac kernel (hard matching), only one pair of tree-walks is matched (for both labels and structures).

calculations and to keep light notations.

If $q_{T,I,J}(G,H)$ is obtained from a positive definite kernel between (labelled) tree-walks, then $k^{\mathcal{T}}_{\alpha,\beta,\gamma}(\mathcal{G},\mathcal{H})$ also defines a positive definite kernel. The kernel $k^{\mathcal{T}}_{\alpha,\beta,\gamma}(\mathcal{G},\mathcal{H})$ sums the *local kernel* $q_{T,I,J}(G,H)$ over all tree-walks of $\mathcal{G}$ and $\mathcal{H}$ 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. The local kernel shall depend on the application at hand. In the next sections, we show how one can design meaningful local kernels resp. for region adjacency graphs and point clouds.

1.3 The region adjacency graph kernel as a tree-walk kernel

We propose to model appearance of images using region adjacency graphs obtained by morphological segmentation. If enough segments are used, i.e., if the image (and hence the objects of interest) is *over-segmented*, then the segmentation enables us to reduce the dimension of the image while preserving the boundaries of objects. Image dimensionality goes from millions of pixels down to hundreds of segments, with little loss of information. Those segments are naturally embedded in a planar graph structure. Our approach takes into account graph planarity. The goal of this section is to show that one can feed kernel-based learning methods with a kernel measuring appropriate region adjacency graph similarity, called the *region adjacency graph kernel*, for the purpose of image classification. Note that this kernel was previously referred to as the *segmentation graph kernel* in the original paper (15).

1.3.1 Morphological region adjacency graphs

Among the many methods available for the segmentation of natural images (35; 36; 37), we chose to use the watershed transform for image segmentation (35), which allows a fast segmentation of large images into a

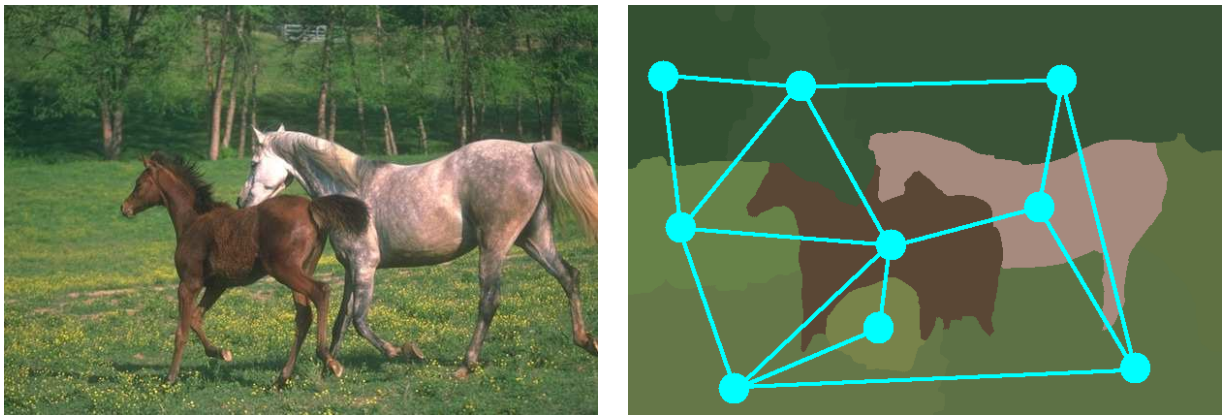

Figure 1.4: An example of natural image (left), and its segmentation mosaic (right) obtained by using the median RGB color in each segmented region. The associated region adjacency graph is depicted in light blue

given number of segments. First, given a color image, a gray-scale gradient image is computed from oriented energy filters on the LAB representation of the image, with two different scales and eight orientations (38). Then, the watershed transform is applied to this gradient image, and the number of resulting regions is reduced to a given number p ($p = 100$ in our experiments) using the hierarchical framework of (35). We finally obtain p singly-connected regions, together with the planar neighborhood graph. An Example of segmentation is shown in Figure 1.4.

We have chosen a reasonable value of p because our similarity measure implicitly relies on the fact that images are mostly over-segmented, i.e., the objects of interest may span more than one segment, but very few segments span several objects. This has to be contrasted with the usual (and often unreached) segmentation goal of obtaining one segment per object. In this respect, we could have also used other image segmentation algorithms, such as the Superpixels algorithm (39; 40). In this section, we always use the same number of segments; a detailed analysis of the effect of choosing a different number of segments, or a number which depends on the given image, is outside the scope of this work.

In this section, images will be represented by a *region adjacency graph*, i.e., an undirected labelled planar graph, where each vertex is one singly connected region, with edges joining neighboring regions. Since our graph is obtained from neighboring singly connected regions, the graph is planar, i.e., it can be embedded in a plane where no edge intersects (4). The only property of planar graphs that we are going to use is the fact that for each vertex in the graph, there is a natural notion of cyclic ordering of the vertices. Another typical feature of our graphs is their sparsity: indeed, the degree (number of neighbors) of each node is usually small, and the maximum degree typically does not exceed 10.

There are many ways to assign labels to regions, based on shape, color or texture, leading to highly

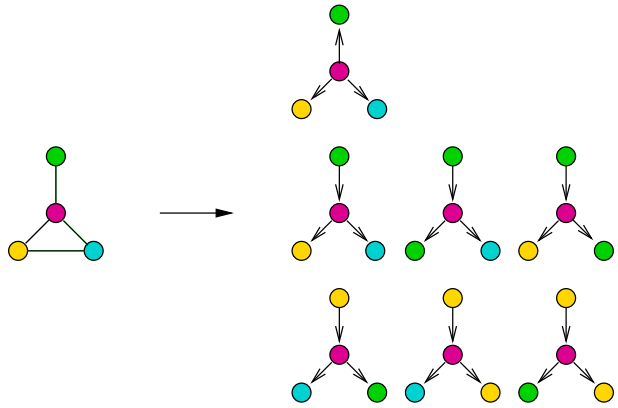

Figure 1.5: Examples of tree-walks from a graph. Each color represents a different label.

multivariate labels, which is to be contrasted with the usual application of structured kernels in bioinformatics or natural language processing, where labels belong to a small discrete set. Our family of kernels simply require a simple kernel $k(\ell, \ell')$ between labels ℓ and ℓ' of different regions, which can be any positive semi-definite kernel (1), and not merely a Dirac kernel which is commonly used for exact matching in bioinformatics applications. We could base such a kernel on any relevant visual information. In this work we considered kernels between color histograms of each segment, as well as weights corresponding to the size (area) of the segment.

1.3.2 Tree-walk kernels on region adjacency graphs

We now make explicit the derivation of tree-walks kernels on region adjacency graphs, in order to highlight the dynamic programming recursion which makes their computation tractable in pratice.

The tree-walk kernel between region adjacency graphs $k_{p,\alpha}(\mathcal{G}, \mathcal{H})$ is defined as the sum over all tree-walks in G and all tree-walks in $\mathcal{H}$ (sharing the same tree structure) of the products of the local kernels between corresponding individual vertices of the subtree-patterns. Note that in the context of exact matching, investigated in (41), this kernel simply counts the number of common subtree-patterns. Note also that, as long as the local kernel is a positive semi-definite kernel, the resulting tree-walk kernel is also an positive semi-definite kernel as well (1).

Local kernels In order to apply the tree-walk kernels to region adjacency graphs, we need to specify the kernels involved at the lower level of computation. Here the local kernels correspond to kernels between segments.

There are plenty of choices for defining relevant features for segment comparisons, ranging from median color to sophisticated local features (42). We chose to focus on RGB color histograms since previous work (43) thoroughly investigated their relevance for image classification, when used on the whole image

without any segmentation. This allows us to fairly evaluate the efficiency of our kernels to make a smart use of segmentations for classification.

Experimental results of kernels between color histograms taken as discretized probability distributions $P = (p_i)_{i=1}^N$ were given in (43). For the sake of simplicity, we shall focus here on the χ^2-kernel defined as follows. The symmetric χ^2-divergence between two distributions P and Q is defined as

$$d_\chi^2(P,Q) = \frac{1}{N}\sum_{j=1}^{N} \frac{(p_i - q_i)^2}{p_i + q_i} , \tag{1.2}$$

and the χ^2-kernel is defined as

$$k_\chi(P,Q) = \exp(-\mu d_\chi^2(P,Q)) , \tag{1.3}$$

with μ a free parameter to be tuned. Following results of (43) and (44), since this kernel is positive semi-definite, it can be used as a local kernel. If we denote P_ℓ the histogram of colors of a region labelled by ℓ, then it defines a kernel between labels as $k(\ell,\ell') = k_\chi(P_\ell, P_{\ell'})$.

We need to deal with the too strong diagonal dominance of the obtained kernel matrices, a usual issue with kernels on structured data (3)). We propose to include a constant term λ that controls the maximal value of $k(\ell,\ell')$. We thus use the following kernel:

$$k(\ell,\ell') = \lambda \exp(-\mu d_\chi^2(P_\ell, P_{\ell'})) ,$$

with free parameters λ, μ. Note that, by doing so, we also ensure the positive semi-definiteness of the kernel.

It is natural to give more weight in the overall sum to massive segments than to tiny ones. Hence we incorporate this into the segment kernel as:

$$k(\ell,\ell') = \lambda A_\ell^\gamma A_{\ell'}^\gamma \exp(-\mu d_\chi^2(P_\ell, P_{\ell'})),$$

where A_ℓ is the area of the corresponding region, and γ is an additional free parameter in $[0,1]$ to be tuned.

Dynamic programming recursion In order to derive an efficient dynamic programming formulation, we now need to restrict the set of subtrees. Indeed, if d is an upper bound on the degrees of the vertices in $\mathcal{G}$ and $\mathcal{H}$, then at each depth, the q-ary tree-walk may go through any subsets of size α of the set of neighbors of a given vertex, and thus the matching complexity would be $O(d^{2\alpha})$ ($O(d^\alpha)$ for each of the two graphs). We restrict the allowed subsets of neighbors in a tree-walk, by requiring that these subsets are intervals for the natural cyclic ordering of the neighbors of a given vertex (this is possible because the graph is planar). See Figure 1.6 for an enumeration of the intervals of size $\alpha = 2$. For a given vertex r in $\mathcal{G}$ (resp. s in $\mathcal{H}$), we denote by $\mathcal{A}_\mathcal{G}^\alpha(r)$ (resp. $\mathcal{A}_\mathcal{H}^\alpha(s)$) the set of non empty intervals of length at most α around r in $\mathcal{G}$ (resp. around s in $\mathcal{H}$). In the remaining of the section, we assume that all our subtree-patterns (referred to as tree-walks

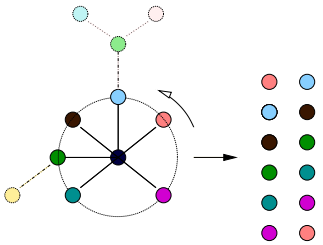

Figure 1.6: Enumeration of neighbor intervals of size two.

from now on) are restricted to intervals of neighbors. Let $k_{p,\alpha}(\mathcal{G},\mathcal{H},r,s)$ denote the sum over all tree patterns starting from vertex r in $\mathcal{G}$ and s in $\mathcal{H}$.

The following result shows that the recursive dynamic programming formulation which allows us to efficiently compute tree-walk kernels. In the case of the tree-walk kernel for region adjacency graphs, the final kernel writes as:

$$k(\mathcal{G},\mathcal{H}) = \sum_{r\in\mathcal{V}_\mathcal{G}, s\in\mathcal{V}_\mathcal{H}} k_{p,\alpha}(\mathcal{G},\mathcal{H},r,s). \tag{1.4}$$

Assuming equal size intervals $\mathrm{Card}(A) = \mathrm{Card}(B)$, the kernel values $k_{p,\alpha}(\mathcal{G},\mathcal{H},r,s)$ can be recursively computed through:

$$k_{p,\alpha}(\mathcal{G},\mathcal{H},r,s) = k(\ell_\mathcal{G}(r),\ell_\mathcal{H}(s)) \sum_{A\in\mathcal{A}^\alpha_\mathcal{G}(r), B\in\mathcal{A}^\alpha_\mathcal{H}(s)} \prod_{r'\in A,\ s'\in B} k_{p-1,\alpha}(\mathcal{G},\mathcal{H},r',s')\,. \tag{1.5}$$

The above equation defines a dynamic programming recursion which allows us to efficiently compute the values of $k_{p,\alpha}(\mathcal{G},\mathcal{H},\cdot,\cdot)$ from $p=1$ to any desired p. We first compute $k_{1,\alpha}(\mathcal{G},\mathcal{H},\cdot,\cdot)$, using that $k(\mathcal{G},\mathcal{H},r,s) = k(\ell_\mathcal{G}(r),\ell_\mathcal{H}(s))$ for r a segment of $\mathcal{G}$ and s a segment of $\mathcal{H}$. Then $k_{2,\alpha}(\mathcal{G},\mathcal{H},\cdot,\cdot)$ can be computed from $k_{1,\alpha}(\mathcal{G},\mathcal{H},\cdot,\cdot)$ using (1.5) with $p=2$. And so on. Finally, note that when $\alpha=1$ (intervals of size 1), the tree-walk kernel reduces to the walk kernel (12).

Running time complexity Given labelled graphs $\mathcal{G}$ and $\mathcal{H}$ with $n_\mathcal{G}$ and $n_\mathcal{H}$ vertices each and maximum degrees $d_\mathcal{G}$ and $d_\mathcal{H}$, we assume that the kernel between labels $k(\ell,\ell')$ can be computed in constant time. Hence the total cost of computing $k(\ell_\mathcal{G}(r),\ell_\mathcal{H}(s))$ for all $r\in\mathcal{V}_\mathcal{G}$ and $s\in\mathcal{V}_\mathcal{H}$ is $O(n_\mathcal{G}n_\mathcal{H})$.

For walk kernels, the complexity of each recursion is $O(d_\mathcal{G}d_\mathcal{H})$. Thus, computation of all q-th walk kernels for $q\leqslant p$ needs $O(pd_\mathcal{G}d_\mathcal{H}n_\mathcal{G}n_\mathcal{H})$ operations.

For tree-walk kernels, the complexity of each recursion is $O(\alpha^2 d_\mathcal{G}d_\mathcal{H})$. Therefore, computation of all q-th α-ary tree walk kernels for $q\leqslant p$ needs $O(p\alpha^2 d_\mathcal{G}d_\mathcal{H}n_\mathcal{G}n_\mathcal{H})$ operations, i.e., leading to polynomial-time complexity.

We now pause in the exposition of tree-walk kernels applied to region adjacency graphs, and move on to tree-walk kernels applied to point clouds. We shall get back to tree-walk kernels on region adjacency graphs in Section 1.5.

1.4 The point cloud kernel as a tree-walk kernel

We propose to model shapes of images using 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. We shall assume that the graphs have no self-loops. Our motivating examples are line drawings, where $\mathcal{A} = \mathbb{R}^2$ (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 (25). In other cases, graphs can be easily obtained from nearest-neighbor graphs.

We show here how 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 probabilistic graphical models. With this novel local kernel, we derive new polynomial time dynamic programming recursions.

1.4.1 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). Recall here that, given a tree structure T and consistent labellings $I \in \mathcal{J}_\beta(T,G)$ and $J \in \mathcal{J}_\beta(T,H)$, the quantity $q_{T,I,J}(G,H)$ denotes the value of the local kernel between two tree-walks defined by the same structure T and labellings I and J. 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. However, for our local kernel on positions, we need a kernel that *jointly* depends on the whole vectors $x(I) \in \mathsf{X}^{|I|}$ and $y(J) \in \mathsf{X}^{|J|}$, and not only on the p pairs $(x(I_p),y(J_p))$. 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.

We shall focus here on $\mathsf{X} = \mathcal{R}^d$ 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

$k_{\mathsf{X}}(x_1 - x_2)$. Depending on the application, k_{X} may or may not be rotation invariant. In experiments, we use the rotation invariant Gaussian kernel of the form $k_{\mathsf{X}}(x_1, x_2) = \exp(-\upsilon \|x_1 - x_2\|^2)$.

Thus, we reduce the set of all positions in $\mathsf{X}^{|V|}$ and $\mathsf{X}^{|W|}$ to full kernel matrices $K \in \mathcal{R}^{|V|\times|V|}$ and $L \in \mathcal{R}^{|W|\times|W|}$ for each graph, defined as $K(v,v') = k_{\mathsf{X}}(x(v) - x(v'))$ (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 (45):

$$k_{\mathcal{B}}(K,L) = |K|^{1/2}|L|^{1/2}\left|\tfrac{K+L}{2}\right|^{-1} , \tag{1.6}$$

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 $q_{\mathcal{A}}(a(I), b(J))$ is not an issue. However, for the term $k_{\mathcal{B}}(K_I, L_J)$, we need an approximation based on T. As we show in the next section, this can be obtained by a factorization according to the appropriate probabilistic 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.

1.4.2 Positive matrices and probabilistic 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 probabilistic graphical model techniques, we can design properly factorized approximations of Eq. 1.6, namely through Eq. 1.11 and Eq. 1.12.

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

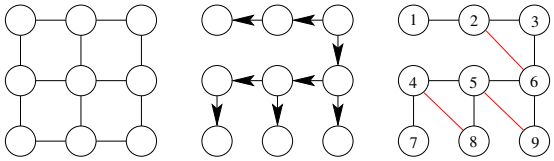

Figure 1.7: (left) original graph, (middle) a single extracted tree-walk, (right) decomposable graphical model $Q_1(T)$ with added edges in red. The junction tree is a chain composed of the cliques $\{1,2\},\{2,3,6\},\{5,6,9\},\{4,5,8\},\{4,7\}$.

Probabilistic graphical models and junction trees Probabilistic 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 $K \in \mathcal{R}^{n\times n}$, the distribution factorizes if and only if $(K^{-1})_{ij} = 0$ for each (i,j) which is not an edge in Q (46).

In this section, we only consider *decomposable* probabilistic 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

$$p(z) = \frac{\prod_{C\in\mathcal{C}(Q)} p_C(z_C)}{\prod_{C,C'\in\mathcal{C}(Q),C\sim C'} p_{C\cap C'}(z_{C\cap C'})} .$$

Figure 1.7 shows an example of a probabilistic 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 $p_C(z_C)$ are characterized by the marginal covariance matrix $K_C = K_{C,C}$. Projecting onto a probabilistic 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.

Probabilistic 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. We essentially replace K by $\Pi_Q(K)$. In other words, we project all our covariance matrices onto a probabilistic graphical model, which is a classical tool in statistics (46; 47). 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., (48) for related results.

Practically, since our kernel on kernel matrices involves determinants, we simply need to compute $|\Pi_Q(K)|$ efficiently. For decomposable probabilistic graphical models, $\Pi_Q(K)$ can be obtained in closed

form (46) and its determinant has the following simple expression:

$$\log|\Pi_Q(K)| = \sum_{C\in\mathcal{C}(Q)} \log|K_C| - \sum_{S\in\mathcal{S}(Q)} \log|K_S|. \tag{1.7}$$

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:

$$\log|\Pi_Q(K)| = \sum_{C\in\mathcal{C}(Q)} \log|K_{C|p_Q(C)}|, \tag{1.8}$$

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

$$K_{C|p_Q(C)} = K_{C,C} - K_{C,p_Q(C)} K^{-1}_{p_Q(C),p_Q(C)} K_{p_Q(C),C}\,. \tag{1.9}$$

Probabilistic graphical models and kernels We now propose several ways of defining a kernel adapted to probabilistic graphical models. All of them are based on replacing determinants $|M|$ by $|\Pi_Q(M)|$, and their different decompositions in Eq. 1.7 and Eq. 1.8. Simply using Eq. 1.7, we obtain the similarity measure:

$$k^Q_{\mathcal{B},0}(K,L) = \prod_{C\in\mathcal{C}(Q)} k_{\mathcal{B}}(K_C,L_C) \prod_{S\in\mathcal{S}(Q)} k_{\mathcal{B}}(K_S,L_S)^{-1}. \tag{1.10}$$

This similarity measure 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. 1.10 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. 1.8. 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:

$$k^Q_{\mathcal{B}}(K,L) = \textstyle\prod_{C\in\mathcal{C}(Q)} k^{C|p_Q(C)}_{\mathcal{B}}(K,L); \tag{1.11}$$

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

$$k_{\mathcal{B}}^{C|p_Q(C)}(K,L) = \frac{|K_{C|p_Q(C)}|^{1/2}|L_{C|p_Q(C)}|^{1/2}}{\left|\frac{1}{2}K_{C|p_Q(C)}+\frac{1}{2}L_{C|p_Q(C)}+MM^\top\right|}, \tag{1.12}$$

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 $k_{\mathcal{B}}^{Q}(K,L)$ defined in Eq. 1.11 and Eq. 1.12 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). Note that these kernels could be useful in other domains than computer vision.

In the next derivations, 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. 1.12.

Choice of probabilistic graphical models Given the rooted tree structure T of a β-tree-walk, we now need to define the probabilistic graphical model $Q_\beta(T)$ that we use to project our kernel matrices. A natural candidate is T itself. However, as shown later, 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 probabilistic 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 1.7 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:

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

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.

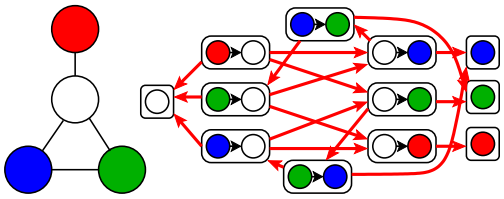

Figure 1.8: (left) undirected graph G, (right) graph $G_{1,2}$.

1.4.3 Dynamic programming recursions

In order to derive dynamic programming recursions, we follow (34) 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. Recall that the *arity* of a tree is the maximum number of children of the root and internal vertices of the tree. 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 1.8 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.

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 $k^{\mathcal{T}}_{\alpha,\beta,\gamma}(G,H,R_0,S_0)$ the same kernel as defined in Eq. 1.4.2, 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 $k^{\mathcal{T}}_{\alpha,\beta,\gamma}(G,H,R_0,S_0)=0$.

Denote $\rho(S)$ the root of a subtree S. 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 $R_0 \sim_t S_0$:

$$k^{\mathcal{T}}_{\alpha,\beta,\gamma}(G,H,R_0,S_0) = k^{\mathcal{T}}_{\alpha,\beta,\gamma-1}(G,H,R_0,S_0) + \mathcal{R}^{\mathcal{T}}_{\alpha,\beta,\gamma-1}\,, \tag{1.13}$$

where $\mathcal{R}^{\mathcal{T}}_{\alpha,\beta,\gamma-1}$ is given by:

$$\mathcal{R}^{\mathcal{T}}_{\alpha,\beta,\gamma-1} = \sum_{p=1}^{\alpha} \sum_{\substack{R_1,\ldots,R_p \in \mathcal{N}_{G_{\alpha,\beta}}(R_0) \\ R_1,\ldots,R_p \text{ disjoint}}} \sum_{\substack{S_1,\ldots,S_p \in \mathcal{N}_{H_{\alpha,\beta}}(S_0) \\ S_1,\ldots,S_p \text{ disjoint}}} \cdots \left[\lambda \prod_{i=1}^{p} k_{\mathcal{A}}(a(\rho(R_i)), b(\rho(S_i))) \frac{k_{\mathcal{B}}^{\cup_{i=1}^{p} R_i | R_0, \cup_{i=1}^{p} S_i | S_0}(K,L)}{\prod_{i=1}^{p} k_{\mathcal{B}}^{R_i,S_i}(K,L)} \left(\prod_{i=1}^{p} k^{\mathcal{T}}_{\alpha,\beta,\gamma-1}(G,H,R_i,S_i) \right) \right]. \quad (1.14)$$

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

$$k^{\mathcal{T}}_{\alpha,\beta,\gamma}(G,H,R_0,S_0) = \lambda^{|R_0|} \nu^{\ell(R_0)} q_{\mathcal{A}}(a(R_0), b(S_0)) k_{\mathcal{B}}(K_{R_0}, L_{S_0}) \quad (1.15)$$

while the final kernel is obtained by summing over all R_0 and S_0, i.e, $k^{\mathcal{T}}_{\alpha,\beta,\gamma}(G,H) = \sum_{R_0 \sim_i S_0} k^{\mathcal{T}}_{\alpha,\beta,\gamma}(G,H,R_0,S_0)$.

Computational complexity The above equations define a dynamic programming recursion which allows us to efficiently compute the values of $k^{\mathcal{T}}_{\alpha,\beta,\gamma}(G,H,R_0,S_0)$ from $\gamma = 1$ to any desired γ. We first compute $k^{\mathcal{T}}_{\alpha,\beta,1}(G,H,R_0,S_0)$ using Eq. 1.15. Then $k^{\mathcal{T}}_{\alpha,\beta,2}(G,H,R_0,S_0)$ can be computed from $k^{\mathcal{T}}_{\alpha,\beta,1}(G,H,R_0,S_0)$ using Eq. 1.13 with $\gamma = 2$. And so on.

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 satisfactory classification performance (in particular, higher β or α do not necessarily mean better performance). 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.

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

[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 time $= 1$ ms.

1.5 Experimental results

We present here experimental results of tree-walk kernels respectively on: i) region adjacency graphs for object categorization and natural image classification, ii) point clouds for handwritten digits classification.

1.5.1 Application to object categorization

We have tested our kernels on the task of object categorization (Coil100 dataset), and natural image classification (Corel14 dataset) both in fully-supervised and in semi-supervised settings.

Experimental setting Experiments have been carried out on both `Corel14` (22) and `Coil100` datasets. Our kernels were put to the test step by step, going from the less sophisticated version to the most complex one. Indeed, we compared on a multi-class classification task performances of the usual histogram kernel (**H**), the walk-based kernel (**W**), the tree-walk kernel (**TW**), the weighted-vertex tree-walk kernel (**wTW**) and the combination of the above by multiple kernel learning (**M**). We report here their performances averaged in an outer loop of 5-fold cross-validation. The hyperparameters are tuned in an inner loop in each fold by 5-fold cross-validation (see in the sequel for further details). `Coil100` consists in a database of 7200 images of 100 *objects in a uniform background*, with 72 images per object. Data are color images of the objects taken from different angles, with steps of 5 degrees. `Corel14` is a database of 1400 *natural images* of 14 different classes, which are usually considered much harder to classify. Each class contains 100 images, with a non-negligible proportion of outliers.

Feature extraction Each image's segmentation outputs a labelled graph with 100 vertices, with each vertex labelled with the RGB-color histogram within each corresponding segment. We used 16 bins per dimension, as in (43), yielding 4096-dimensional histograms. Note that we could use LAB histograms as well. Average vertex degree was around 3 for `Coil100` and 5 for `Corel14`. In other words region adjacency graphs are very sparsely connected.

Free parameter selection For the multi-class classification task, the usual SVM classifier was used in a one-versus-all setting (1). For each family of kernels, hyper-parameters corresponding to kernel design and the SVM regularization parameter C were learned by cross-validation, with the following usual machine learning procedure: we randomly split the full dataset in 5 parts of equal size, then we consider successively each of the 5 parts as the testing set (the outer testing fold), learning being performed on the four other parts (the outer training fold). This is in contrast to other evaluation protocols. Assume instead one is trying out different values of the free parameters on the outer training fold, computing the prediction accuracy on the corresponding testing fold, repeating five times for each of the five outer folds, and compute average

performance. Then assume one selects the best hyper-parameter and report its performance. A major issue with such a protocol is that it leads to an optimistic estimation of the prediction performance (49).

It is preferable to consider each outer training fold, and split those into 5 equal parts, and learn the hyper-parameters using cross-validation on those inner folds, and use the resulting parameter, train on the full outer training fold with this set of hyperparameters (which might be different for each outer fold) and test on the outer testing fold. The prediction accuracies which we report (in particular in the boxplots of Figure 1.11) are the prediction accuracies on the outer testing folds. This two-stage approach leads to more numerous estimations of SVM parameters but provide a fair evaluation of performance.

In order to choose the values of the free parameters, we use values of free parameters on the finite grid in Figure 9.

Parameter	Values
γ	$0.0, 0.2, 0.4, 0.6, 0.8$
α	$1, 2, 3$
p	$1, 2, 3, 4, 5, 6, 7, 8, 9, 10$
C	$10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}, 10^{3}, 10^{4}$

Figure 1.9: Range of values of the free parameters.

The respective performances in average test error rates (on the five testing outer folds) in multi-class classification of the different kernels are listed in Figure 10.. See Figure 1.11 for corresponding boxplots for the `Corel14` dataset.

	H	W	TW	wTW	M
`Coil100`	1.2%	0.8%	0.0%	0.0%	0.0%
`Corel14`	10.36%	8.52%	7.24%	6.12%	5.38%

Figure 1.10: Best test error performances of histogram, walk, tree-walk, tree-walk with weighted segments, kernels

`Corel14` **dataset** We have compared test error rate performances of SVM-based multi-class classification with histogram kernel (**H**), walk kernel (**W**), tree-walk kernel (**TW**), and the tree-walk kernel with weighted segments (**wTW**). Our methods, i.e., **TW** and **wTW** clearly outperforms global histogram kernels and simple walk kernels. This corroborates the efficiency of tree-walk kernels to capture the topological structure of natural images. Our weighting scheme also seems to be reasonable.

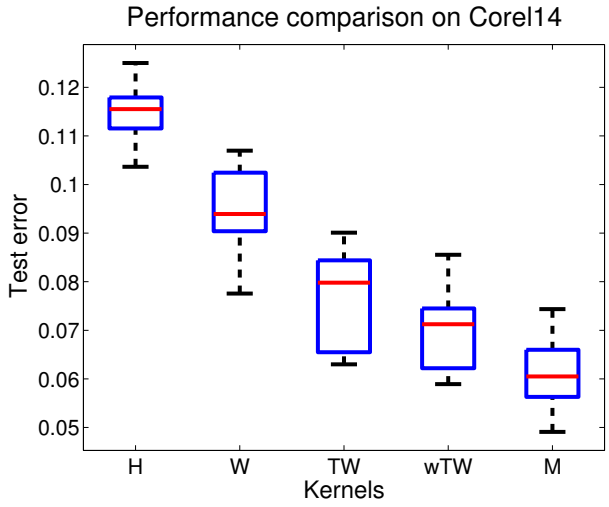

Figure 1.11: Test errors for supervised multi-class classification on `Corel14`, for **H**istogram, **W**alk, **T**ree-**W**alk, **w**eighted **T**ree-**W**alk, kernels and optimal **M**ultiple kernel combination.

Multiple kernel learning We first tried the combination of histogram kernels with walk-based kernels, which did not yield significant performance enhancement. This suggests that histograms do not carry any supplemental information over walk-based kernels: the global histogram information is implicitly retrieved in the summation process of walk-based kernels.

We ran the multiple kernel learning (MKL) algorithm of (50) with 100 kernels (i.e., a rough subset of the set of all parameter settings with parameters taken values detailed in Figure 9). As seen in Figure 1.11, the performance increases as expected. It is also worth looking at the kernels which were selected. We here give results for one of the five outer cross-validation folds, where 5 kernels were selected, as displayed in Figure 12.

p,α,γ	$10,3,0.6$	$7,1,0.6$	$10,3,0.3$	$5,3,0.0$	$8,1,0.0$
η	0.12	0.17	0.10	0.07	0.04

Figure 1.12: Weight η of the kernels with the largest magnitude (see (50) for details on normalization of η).

It is worth noting that various values of γ, α and p are selected, showing that each setting may indeed capture different discriminative information from the segmented images.

Semi-supervised classification Kernels allow us to tackle many tasks, from unsupervised clustering to multi-class classification and manifold learning (1). To further explore the expressiveness of our region adjacency graph kernels, we give below the evolution of test error performances on `Corel14` dataset for multi-class classification with 10% labelled examples, 10% test examples, and an increasing amount ranging from 10% to 80% of unlabelled examples. Indeed, all semi-supervised algorithms derived from statistically

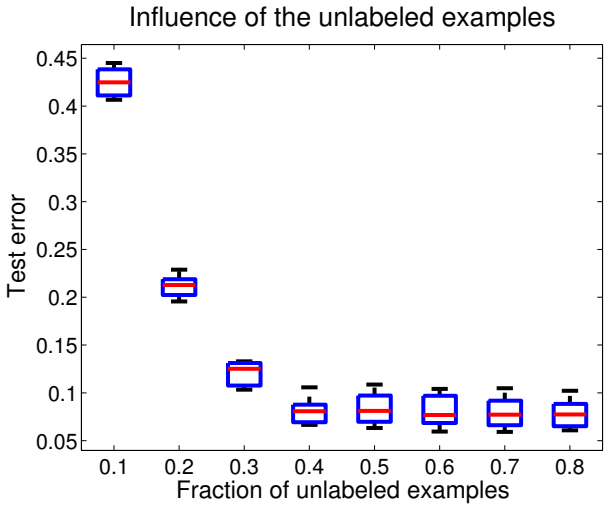

Figure 1.13: Test error evolution for semi-supervised multi-class classification as the number of unlabelled examples increases.

consistent supervised ones see their test errors falling down as the number of labelled examples is increased and the number of unlabelled examples kept constant. However, such experimental convergence of the test errors as the number of labelled examples is kept fixed and the number of unlabelled ones is increased is much less systematic (51). We used the publicly available code for the low density separation (LDS) algorithm of (51), since it reached good performances on `Coil100` image dataset.

Since we are more interested in showing the flexibility of our approach than the semi-supervised learning problem in general, we simply took as a kernel the optimal kernel learned on the whole supervised multi-classification task by the multiple kernel learning method. Although this may lead to a slight over-estimation of the prediction accuracies, this allowed us to bypass the kernel selection issue in semi-supervised classification, which still remains unclear and under active investigation. For each amount of unlabelled examples, as an outer loop we randomly selected 10 different splits into labelled and unlabelled examples. As an inner loop we optimized the hyperparameters, namely the regularization parameter C and ρ the cluster squeezing parameter (see (51) for details), by leave-one-out cross-validation. The boxplots in Figure 1.13 shows the variability of performances within the outer loop. Keeping in mind that the best test error performance on `Corel14` dataset of histogram kernels is around 10% for *completely supervised* multi-class classification, the results are very promising; we see that our kernel reaches this level of performance for as little as 40% unlabelled examples and 10% labelled example.

We now present the experimental results obtained with tree-walk kernels on point clouds in a character recognition task.

Figure 1.14: For digits and Chinese characters: (left) original characters, (right) thinned and subsampled characters.

1.5.2 Application to character recognition

We have tested our kernels on the task of isolated handwritten character recognition, handwritten arabic numerals (MNIST dataset) and Chinese characters (ETL9B dataset).

1.5.3 Experimental setting

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., (1)).

Feature extraction We consider characters as drawings in $\mathcal{R}^2$, 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 1.14). 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.

Free parameter selection The tree-walk kernels on point clouds have 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

	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 1.15: Error rates (multiplied by 100) on handwritten character classification tasks.

distinct parameters for all sizes and topologies of subtrees. In experiments, 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 1.15 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* (52), 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 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 (9). 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.

1.6 Conlusion

We have presented a family of kernels for computer vision tasks, along with two instantiations of these kernels: i) tree-walk kernels for region adjacency graphs, ii) tree-walk kernels for point clouds. For i), we have showed how one can efficiently compute kernels in polynomial time with respect to the size of

the region adjacency graphs and their degrees. For ii), we proposed an efficient dynamic programming algorithm using a specific factorized form for the local kernels between tree-walks, namely a factorization on a properly defined probabilistic graphical model. We have reported applications to object categorization and natural image classification, as well as handwritten character recognition, where we showed that the kernels were able to capture the relevant information to allow good predictions from few training examples.

1.7 Acknowledgements

This work was partially supported by grants from the Agence Nationale de la Recherche (MGA Project) and from the European Research Council (SIERRA Project), and the PASCAL 2 Network of Excellence.

Bibliography

[1] J. Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*. Cambridge Univ. Press, 2004.

[2] C. H. Lampert, "Kernel methods in computer vision," *Found. Trends. Comput. Graph. Vis.*, vol. 4, pp. 193–285, March 2009.

[3] J.-P. Vert, H. Saigo, and T. Akutsu, *Local Alignment Kernels for Biological Sequences*. MIT Press, 2004.

[4] R. Diestel, *Graph Theory*. Springer-Verlag, 2005.

[5] O. Chapelle, B. Schölkopf, and A. Zien, Eds., *Semi-Supervised Learning (Adaptive Computation and Machine Learning)*. MIT Press, 2006.

[6] F. Bach, "Consistency of the group lasso and multiple kernel learning," *Journal of Machine Learning Research*, vol. 9, pp. 1179–1225, 2008.

[7] J. Ponce, M. Hebert, C. Schmid, and A. Zisserman, *Toward Category-Level Object Recognition (Lecture Notes in Computer Science)*. Springer, 2007.

[8] J. Zhang, M. Marszalek, S. Lazebnik, and C. Schmid, "Local features and kernels for classification of texture and object categories: a comprehensive study," *International Journal of Computer Vision*, vol. 73, no. 2, pp. 213–238, 2007.

[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[10] S. N. Srihari, X. Yang, and G. R. Ball, "Offline Chinese handwriting recognition: A survey," *Frontiers of Computer Science in China*, 2007.

[11] S. Belongie, J. Malik, and J. Puzicha, "Shape matching and object recognition using shape contexts," *IEEE Trans. PAMI*, vol. 24, no. 24, pp. 509–522, 2002.

[12] J. Ramon and T. Gärtner, "Expressivity versus efficiency of graph kernels," in *First International Workshop on Mining Graphs, Trees and Sequences*, 2003.

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

[14] H. Kashima, K. Tsuda, and A. Inokuchi, "Kernels for graphs," in *Kernel Methods in Comp. Biology*. MIT Press, 2004.

[15] Z. Harchaoui and F. Bach, "Image classification with segmentation graph kernels," in *CVPR*, 2007.

[16] F. R. Bach, "Graph kernels between point clouds," in *Proceedings of the 25th international conference on Machine learning*, ser. ICML '08. New York, NY, USA: ACM, 2008, pp. 25–32.

[17] C. Wang and K. Abe, "Region correspondence by inexact attributed planar graph matching," in *Proc. ICCV*, 1995.

[18] A. Robles-Kelly and E. Hancock, "Graph edit distance from spectral seriation," *IEEE PAMI*, vol. 27, no. 3, pp. 365–378, 2005.

[19] C. Gomila and F. Meyer, "Graph based object tracking," in *Proc. ICIP*, 2003, pp. 41–44.

[20] B. Huet, A. D. Cross, and E. R. Hancock, "Graph matching for shape retrieval," in *Adv. NIPS*, 1999.

[21] D. Gusfield, *Algorithms on Strings, Trees, and Sequences*. Cambridge Univ. Press, 1997.

[22] O. Chapelle and P. Haffner, "Support vector machines for histogram-based classification," *IEEE Trans. Neural Networks*, vol. 10, no. 5, pp. 1055–1064, 1999.

[23] T. Jebara, "Images as bags of pixels," in *Proc. ICCV*, 2003.

[24] M. Cuturi, K. Fukumizu, and J.-P. Vert, "Semigroup kernels on measures," *J. Mac. Learn. Research*, vol. 6, pp. 1169–1198, 2005.

[25] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *Proc. CVPR*, 2006.

[26] M. Neuhaus and H. Bunke, "Edit distance based kernel functions for structural pattern classification," *Pattern Recognition*, vol. 39, no. 10, pp. 1852–1863, 2006.

[27] H. Fröhlich, J. K. Wegner, F. Sieker, and A. Zell, "Optimal assignment kernels for attributed molecular graphs," in *Proc. ICML*, 2005.

[28] J.-P. Vert, "The optimal assignment kernel is not positive definite, Tech. Rep. HAL-00218278, 2008.

[29] F. Suard, V. Guigue, A. Rakotomamonjy, and A. Benshrair, "Pedestrian detection using stereo-vision and graph kernels," in *IEEE Symposium on Intelligent Vehicule*, 2005.

[30] 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." *Bioinformatics*, vol. 21, 2005.

[31] S. V. N. Vishwanathan, K. M. Borgwardt, and N. Schraudolph, "Fast computation of graph kernels," in *Adv. NIPS*, 2007.

[32] J.-P. Vert, T. Matsui, S. Satoh, and Y. Uchiyama, "High-level feature extraction using svm with walk-based graph kernel," in *Proceedings of the 2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, ser. ICASSP '09, 2009.

[33] M. Fisher, M. Savva, and P. Hanrahan, "Characterizing structural relationships in scenes using graph kernels," in *ACM SIGGRAPH 2011 papers*, ser. SIGGRAPH '11, 2011.

[34] P. Mahé and J.-P. Vert, "Graph kernels based on tree patterns for molecules," *Machine Learning Journal*, vol. 75, pp. 3–35, April 2009.

[35] F. Meyer, "Hierarchies of partitions and morphological segmentation," in *Scale-Space and Morphology in Computer Vision*. Springer-Verlag, 2001.

[36] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE PAMI*, vol. 22, no. 8, pp. 888–905, 2000.

[37] D. Comaniciu and P. Meer, "Mean shift: a robust approach toward feature space analysis," *IEEE PAMI*, vol. 24, no. 5, pp. 603–619, 2002.

[38] J. Malik, S. Belongie, T. K. Leung, and J. Shi, "Contour and texture analysis for image segmentation," *Int. J. Comp. Vision*, vol. 43, no. 1, pp. 7–27, 2001.

[39] X. Ren and J. Malik, "Learning a classification model for segmentation," *Computer Vision, IEEE International Conference on*, vol. 1, p. 10, 2003.

[40] A. Levinshtein, A. Stere, K. N. Kutulakos, D. J. Fleet, S. J. Dickinson, and K. Siddiqi, "Turbopixels: fast superpixels using geometric flows." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 12, pp. 2290–2297, 2009.

[41] T. Gärtner, P. A. Flach, and S. Wrobel, "On graph kernels: Hardness results and efficient alternatives," in *COLT*, 2003.

[42] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comp. Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[43] M. Hein and O. Bousquet, "Hilbertian metrics and positive-definite kernels on probability measures," in *AISTATS*, 2004.

[44] C. Fowlkes, S. Belongie, F. Chung, and J. Malik, "Spectral grouping using the Nyström method," *IEEE PAMI*, vol. 26, no. 2, pp. 214–225, 2004.

[45] R. I. Kondor and T. Jebara, "A kernel between sets of vectors." in *Proc. ICML*, 2003.

[46] S. Lauritzen, *Graphical Models*. Oxford U. Press, 1996.

[47] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[48] T. Caetano, T. Caelli, D. Schuurmans, and D. Barone, "Graphical models and point pattern matching," *IEEE Trans. PAMI*, vol. 28, no. 10, pp. 1646–1663, 2006.

[49] R. Kohavi and G. John, "Wrappers for feature subset selection," *Artificial Intelligence*, vol. 97, no. 1-2, pp. 273–324, 1997.

[50] F. R. Bach, R. Thibaux, and M. I. Jordan, "Computing regularization paths for learning multiple kernels," in *Adv. NIPS*, 2004.

[51] O. Chapelle and A. Zien, "Semi-supervised classification by low density separation," in *Proc. AISTATS*, 2004.

[52] K. Grauman and T. Darrell, "The pyramid match kernel: Efficient learning with sets of features," *J. Mach. Learn. Res.*, vol. 8, pp. 725–760, 2007.

Index