# Inexact graph matching based on kernels for object retrieval in image databases

Justine Lebrun, Philippe-Henri Gosselin, Sylvie Philipp-Foliguet

# Inexact graph matching based on kernels for object retrieval in image databases

Justine Lebrun, Philippe-Henri Gosselin and Sylvie Philipp-Foliguet

*ETIS, CNRS, ENSEA, Univ Cergy-Pontoise, F-95000 Cergy-Pontoise*

**Abstract**

In the framework of online object retrieval with learning, we address the problem of graph matching using kernel functions. An image is represented by a graph of regions where the edges represent the spatial relationships. Kernels on graphs are built from kernel on walks in the graph. This paper firstly proposes new kernels on graphs and on walks, which are very efficient for graphs of regions. Secondly we propose fast solutions for exact or approximate computation of these kernels. Thirdly we show results for the retrieval of images containing a specific object with the help of very few examples and counter-examples in the framework of an active retrieval scheme.

*Keywords:* Online, Interactive, Database, Content-Based, Object Retrieval, Image Retrieval, Machine Learning, Kernel Methods, Graph matching, Inexact match
*PACS:*

## 1. Introduction

One of the goals of the content-based image retrieval is to retrieve images containing a particular object or type of object, animal or person, whose shape can be very variable and set in a background also very variable. Global signatures are not a good way to solve this problem, especially if the context (background) brings no information. Approaches based on points of interest are interesting, but must be used with a high number of points to be efficient, and thus have a very high computational complexity. A promising approach is to represent an object by a set of regions characterized on one hand by intrinsic features (such as color, texture or shape), and on the other hand by spatial relations between them. The adjacency graph of regions constitutes a structure well adapted to represent objects in their infinite variability. However the segmentation into regions is very difficult, since there is no unique solution (it depends on the level of detail expected for this segmentation) and it is very sensitive to changes in the lighting, in the scale and in the aspect of the object. The number and the characteristics of the regions representing the same object are thus very variable from one image to the other. The problem of retrieving images including a type of object can thus be considered as a problem of inexact graph matching.

A retrieval system needs a similarity measure and a retrieval engine. The most popular - because the most efficient - way to perform classification or browsing in a

database is the Support Vector Machines (SVM). SVM are state-of-the-art large margin classifiers which have demonstrated remarkable performances in image retrieval, when associated with adequate kernel functions.

The problem of graph comparison is a topic which has been widely studied in the literature for several decades [1]. One reason is that this problem occurs in many domains as various as computer or social networks, chemistry, or pattern recognition. Another reason is that graphs may be of very various kinds, in their size, their structure, in the type of information they represent and so on and thus they gave rise to many different methods to compare and classify them, all methods passing through graph matching.

A first way to classify methods of graph matching concerns the structure of the graphs : there are two main categories of methods, depending whether the structure of both graphs is the same or if it may differ. The first category addresses the graph isomorphisms for which both graphs have the same number of vertices and the same number of edges, each vertex of one graph being matched with one and only one vertex of the other graph and the same for the edges. There is a wide literature on the problem of finding the best isomorphism between graphs or sub-graphs [2]. This type of method is used for example in chemistry, or in computer-aided design, where vertices and edges are affected with symbolic labels, such as “carbon” or “hydrogen” for vertices, and “over” or “under” to characterize edges in pattern recognition. The constraint of having exactly the same structure is often too strong and is relaxed in the second category where one vertex can match zero, one or several vertices of the other graph. The problem of comparing graphs with unlabeled vertices and edges is a NP-hard problem. When the vertices and the edges are labeled with symbols, the same problem is much simpler, since the combinatorial is much smaller: a vertex with label $l$ is only matched with a vertex with the same label $l$, but it is still a NP-hard problem [3]. In our problem of image retrieval from datasets, we deal with graphs whose both vertices and edges are assigned with vectors of values. And we need not only to compare graphs in terms of structure but we need a similarity also taking into account the similarity between vertices and between edges.

The problem we address in this paper has two main challenges : compare graphs of various structures and deal with vertices and edges attributed with numerical values. As vertices and edges include numerical information, these graphs are called Attributed Relational Graphs (ARG). In our case, vertices represent regions of the image and edges represent adjacencies between regions (neighbor). Edges are directed since they are described by information such as : how much one region is over another one. There are no multiple edges between two vertices (two regions), but loops are possible in order to allow multiple matches (one vertex to several).

Because of the computational cost, algorithms to compute distances between graphs are either complete (they give the optimal solution, possibly with an exponential complexity) or incomplete (the complexity is polynomial but the optimal solution is not guaranteed [4]). Concerning the former ones, most methods use search trees and filtering to prune these trees. A* or “branch and bound" algorithm is then used to solve the problem [5, 6, 7]. In [8] A* algorithm is used to perform ARG isomorphism for an image retrieval task. A way to find the isomorphism between graphs is to represent them in a canonical way and then to compare these representations. The algorithm

developed by McKay [9] is "regarded by many authors as the fastest isomorphism algorithm available today" [1]. Another solution is graph editing [10], which consists in deforming one graph into the other one. The drawback of complete methods is that, because of the computational cost, they are limited to small graphs [5]. Another algorithm based on graph deformations is the Graph Transform Matching, which is applied to image registration [11]. In many cases, such as clustering or similar document retrieval, the exact distance between graphs is not crucial and an approximation is sufficient. In incomplete methods, combinatorial optimization algorithms are used, with quadratic optimization like Softassign [12], or with estimation of distribution [13], or with taboo search [14], etc. Recently Vishwanathan et al. [15] proposed a method to compute a graph kernel from kernels on walks, which improves the time complexity. It is particularly efficient for sparse graphs, but limited to graphs with unlabeled vertices.

Recent approaches of graph comparison consider graphs as sets of substructures such as chains, walks, trees and even graphlets (small subgraphs). As we are interested in matching only a part of the image (the object and not its background), this approach seems able to measure a similarity between sets of regions with their layout. We thus propose to build kernels on graphs from kernels on walks to compute the similarity between images. In the previous papers using random walks [16, 17, 18, 19, 15], authors only compare walks of equal length. But in our application, we need to compute a similarity between graphs of different orders, which means that one vertex can be matched with several vertices of the other graph, this will be achieved by allowing loops in the walks.

We will show that the search tree is a representation well suited to a recursive building of the walks in a graph and that the branch and bound algorithm allows a fast computation of the best match. Moreover with this algorithm and the similarities we propose, we are able to compute either the exact distance or an approximation.

The novelty of this paper is firstly to propose new kernels between graphs and between walks, which are more efficient and faster than existing ones (Section 2). Secondly we propose solutions to the inexact graph matching problem for attributed graphs of regions (Section 3). Thirdly we show results for the retrieval of images containing a specific object with the help of very few examples and counter-examples in the framework of an active retrieval scheme (Section 4).

## 2. Kernels on Graphs

Kernel-based methods, such as Support Vector Machines (SVM), have shown their robustness for image retrieval and many other domains, thanks to convex minimization criterion. Kernel functions can be seen as similarity functions, which respect properties known as Mercer properties [20].

The idea of syntax-driven kernels [3] as opposed to model-driven kernels is to define a kernel on graph from kernel on parts of the graphs. Such a kernel was first defined by Haussler with the convolution kernel [21]. Then Kondor et al. [22] defined kernels over discrete structures which can be regarded as the discretization of Gaussian kernels. Since 2003 many different kernels have been defined that can be arranged according to the kind of structure they consider :

- random walks in [17, 18, 19, 15]
- paths in [23] : a path is a walk which does not go twice through the same vertex
- trees in [24, 25]
- graphlets in [26] : graphlets are subgraphs of small order, typically 3 to 5 vertices. In [26], they only capture the structure of the graph, they do not carry any information on vertices and edges.

Most of these kernels have been designed for chemical or bioinformatics applications, where vertices and edges carry very few information, usually only a label and sometimes a vector of small dimension (less than 4 attributes). Moreover these methods lead with graphs of small order, except [26] which deals with graphs of several dozens or even several hundreds of vertices (but unlabeled).

There are two main approaches to define kernels on graphs, depending on the way the embedding of the graph into a vector space is performed.

The first approach is explicit. This means that only a subset of features extracted from the graphs can be considered (edge number, walks, spectrum...). In order to choose such features, prototypes are built using techniques like K-Means, PCA [27], MIL-based techniques [28] or randomized forests [29]. From a set of prototypes or of frequent patterns, an explicit embedding consists for example [30] in computing the distance to each of the prototypes and then to use a classical vector-based kernel. In [31] the vertices are embedded into a vector space thanks to a membership function to the pattern, this membership function can either be binary or obtained by diffusion of the pattern through the edges of the graph.

This approach bounds the dimension of vectors in the induced space, since they need to be explicitly stored. Moreover it leads to a global parametrization (such as the number of prototypes) which needs to be tuned for each database or each query. Then, the comparison of two graphs always depends on this global parameter. A solution to this problem can be to perform an online computation of the prototypes during the retrieval [32, 33].

The second approach performs an implicit embedding of graphs into a vector space, which means that the vectors in the space induced by the kernel function are never

computed. This can be done using spectral techniques for matching pairs [34] or for high order matching, with tensors [35]. Other kernels were proposed, which behave like the similarity functions based on votes, but with respect to mathematical properties [36, 37, 38].

In this paper, we focus on this last approach and especially on kernels based on randow walks, since they are in our opinion the most adapted to compare graphs whose information is carried by the vertices and the edges, rather than by the structure of the graph.

Kashima et al. [17] and Gärtner et al. [16] built graph kernels from random walks. These kernels look for common walks and weight them either by their length [16] or by their probability of appearance [17]. These two kernels were defined for labeled graphs (graphs of molecules) and they were extended to continuous values by Borgwardt [18]. In [15], the kernel is computed by counting the number of common walks (only the edges are labeled). All these papers compute the kernel on graphs from all random walks from the graphs. The number of these walks may be infinite (especially if cycles are allowed). Suard [19] proposed to reduce the number of walks and showed that $n^2$ walks are enough (where $n$ is the number of vertices), which considerably reduces the computational cost.

### *2.1. Kernel functions*

We present in this section the main ideas of the kernel methods without entering into details which can be found in [20].

Let $x$ be the representation of an image in an input space $\mathcal{X}$. This representation can be a vector or a more complex structure. The first idea is no more to work on the initial image representation $x$, but on an image $\phi(x)$ in a Hilbert space $\mathcal{H}$. In this Hilbert space, images are represented by vectors $\phi(x)$, and the dot product $\langle \phi(x), \phi(x') \rangle$ can be used as a similarity function. Then, any vector-based learning technique can be used.

At this point, the problem is to create a mapping function $\phi : \mathcal{X} \rightarrow \mathcal{H}$ which maps any complex representation $x$ to a vector representation $\phi(x)$. A common way to perform this, as we will present below, is to work in a very large Hilbert space, up to the infinite. For the largest ones, the direct computation of the mapped vector becomes impossible. No matter the size of the Hilbert space, since the similarity between two images $x$ and $x'$ is computed through the dot product of their images in $\mathcal{H}$: $k(x, x') = \langle \phi(x), \phi(x') \rangle$. It has been shown that many operations can be performed on vector data without having an explicit representation, but only with the dot product [39]. For example the Gaussian kernel which is one of the most frequently used kernel functions is : $k(x, x') = \exp(-\frac{1}{2} d(x, x')^2 / \sigma^2)$, with $d(x, x')$ a metric. All methods able to work only through a dot product are called kernel methods. For instance, the SVM classification function never needs an explicit expression of a mapped vector:

$$f(x) = \sum_i \alpha_i y_i k(x, x_i)$$

where $(x_i, y_i)$ represents one example ($x_i$ is the image, and $y_i$ its class +1 or -1) and $(\alpha_i)_i$ are the parameters of the hyperplane separating both classes.

In order to build these kernel functions, we use the following properties:

$$\forall \lambda > 0, k \text{ is a kernel function } \Rightarrow \lambda k \text{ is a kernel function.} \quad (1)$$

$$\forall p \geq 1, k \text{ is a kernel function } \Rightarrow k^p \text{ is a kernel function.} \quad (2)$$

$$k, k' \text{ are kernel functions } \Rightarrow k + k' \text{ is a kernel function.} \quad (3)$$

$$k, k' \text{ are kernel functions } \Rightarrow kk' \text{ is a kernel function.} \quad (4)$$

Let $G$ be the Gram matrix on a data-set $X = (x_i)_{i \in [1,n]}$. The Gram matrix for a kernel function $k$ is the matrix of all values of $k$:

$$\forall i, j \in [1, n] \; G_{ij} = k(x_i, x_j) \quad (5)$$

Gram matrices are semi-definite positive, which is equivalent to say that all their eigenvalues are positive or null.

Reciprocally, if a matrix $G$ defined as in Eq. (5) is semi-definite positive, then the following property holds:

$$\exists \phi : \mathcal{X} \rightarrow \mathcal{H} \mid \forall i, j \in [1, n] \; k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle. \quad (6)$$

This last property means that, even if the similarity function $k$ does not respect Mercer properties, it can be used on a data set where the Gram matrix is semi-definite positive.

### *2.2. Kernels on Bags*

The first step to kernels on graphs was the kernels on sets of vectors, or kernels on bags [40, 36]. An image $x_i$ is represented by a set of unordered elements $B_i = \{b_{ri}\}_r$, a "bag of features", for instance all points of interest or all regions in an image.

A way to build a kernel function from this representation is to first map the elements of the bag in a Hilbert space. When the elements are vectors, many kernel functions on vectors are available, from polynomial to Gaussian kernels. If we denote by $\phi$ the mapping function for a feature and by $k$ the corresponding kernel function, then the mapping of bag $B_i$ is $\Phi(B_i) = \sum_r \phi(b_{ri})$, which leads to the following kernel on bags [39]:

$$\begin{aligned} K(B_i, B_j) &= \langle \Phi(B_i), \Phi(B_j) \rangle \\ &= \sum_r \sum_s \langle \phi(b_{ri}), \phi(b_{sj}) \rangle \\ &= \langle \sum_r \phi(b_{ri}), \sum_s \phi(b_{sj}) \rangle \\ &= \sum_r \sum_s k(b_{ri}, b_{sj}) \end{aligned} \quad (7)$$

This kernel on bags satisfies Mercer properties, but it does not behave well since it does not perform any matching-like operations, it only sums up the similarities of all vectors of image $i$ with all features of image $j$. Several propositions have been made to improve this function, for instance in [40]:

$$K(B_i, B_j) = \frac{1}{|B_i|} \sum_r \max_s k(b_{ri}, b_{sj}) + \frac{1}{|B_j|} \sum_s \max_r k(b_{ri}, b_{sj}) \quad (8)$$

Let us note that, because of the $\max$ operation in the expression, this kernel function does not satisfy the Mercer properties. However, the loss of the semi-definite positiveness of the Gram matrix occurs in very special cases, and never happened on our datasets. Then, property of Eq. (6) can be used.

The validity of these kernels on bags depends on the minor kernel function $k$. They were introduced with minor kernel on vectors, but kernel functions on more complex data can be used. For instance we will show in the next section that kernel on graphs can be defined from kernel on walks.

### *2.3. Kernels on graphs and kernels on bags of walks*

We denote by $G \in \mathcal{G}$ a graph defined by a pair $G = (V, E)$, where $V$ is a set of vertices, and $E \subseteq V \times V$ is a set of edges. In the case of ARG, such a graph is built by representing each region by a vertex $v \in V$, and each edge $e = (v_1, v_2) \in V \times V$ represents an adjacency between two regions.

A way to create a kernel function on graphs $K(G, G')$ is to directly produce such a function from existing similarity measures. In many similarity measures $S(G, G')$ between two graphs $G = (V, E)$ and $G' = (V', E')$, the idea is to find the best matches between vertices and edges. For example, Sorlin [41] proposes a similarity measure which is the average value of the similarities between vertices and between edges of the best graph match. However most of these similarity measures do not possess the usual properties of a metric measure such as symmetry or triangular inequality. Furthermore, the operators usually involved in these measures are hardly "kernelizable".

Another way to create a kernel function on graphs is to consider a set of walks in the graphs. A walk $h$ is an ordered $n$-uple of vertices $(v_1 v_2 ... v_n)$ linked by edges of $E$. The number of walks one can extract from a graph is infinite, since the same vertex and the same edge can be used several times. This number can be reduced by specifying the length or the type of the walks. We will study this in section 4 and show that a graph can be completely described by a small amount of walks. In this section we will note $H(.)$ a function which maps a graph $G$ to a given set of walks. Kernels on graphs can thus be seen as kernels on bags of walks : one can reuse the kernel on bags of Eq. (7) with a kernel on walks $K_C(h, h')$ on a set of walks $H(G)$ of a graph $G$:

$$K(G, G') = \sum_{h \in H(G)} \sum_{h' \in H(G')} K_C(h, h') \tag{9}$$

This assumes that there is a kernel on walks $K_C(h, h')$ able to deal with walks of different lengths. Several kernels involving only walks of the same length have been proposed [16, 17, 18, 15]. For instance Kashima et al. proposed the following kernel [17]:

$$K_{Kashima}(G, G') = \sum_{h \in H(G)} \sum_{\substack{h' \in H(G') \\ |h'|=|h|}} K_C(h, h') p(h|G) p(h'|G') \tag{10}$$

with $p(h|G)$ the probability of finding walk $h$ in graph $G$ and $|h|$ the length of $h$ i.e. its number of edges.

This class of kernel is used in the framework of graphs of molecules, where the similarity between vertices is binary, a vertex (an atom) is or is not the same as the vertex of the other graph. But when the similarity between two vertices takes real values, this function tends to bury the similarities between walks in the sum. For example, if there are 3 matches (high similarity value $a$) among 100 possible matches (97 small similarity values $b$), then the total similarity equals $3a + 97b$. The 3 strong matches are not sufficient towards the 97 small matches.

To deal with this problem, and also to reduce the computation time, another kernel we will call $K_{max}$ takes the maximum of all similarities of all walks of same length.

$$K_{max}(G, G') = \max_{h \in H(G)} \max_{\substack{h' \in H(G') \\ |h'|=|h|}} K_C(h, h') \tag{11}$$

A similar kernel was used in FReBIR [42] without the restriction of walks of similar lengths.

In between these two kernels, Suard kernel [19] uses the maximum of these values, instead of using the mean of the matching values as Kashima.

$$K_{Suard}(G, G') = \frac{1}{2} \left( \sum_{h \in H(G)} \max_{\substack{h' \in H(G') \\ |h'|=|h|}} K_C(h, h') + \sum_{h' \in H(G')} \max_{\substack{h \in H(G) \\ |h|=|h'|}} K_C(h, h') \right) \tag{12}$$

This formula is symmetric because of the use of two terms.

Since there is a $max$ in the last two formula, these two functions do not respect the Mercer conditions. But these conditions are violated only in very specific cases, which never occurred in the databases we used, and the Gram matrix is always semi-definite positive (*cf.* property of Eq. (6)). This type of functions was also used in [36] and [40] with the same conclusions.

However, the presence of a $max$ operator authorizes the use of fast algorithms and approximate solutions.

### *2.4. New kernel on bags of walks*

Kernels like the one of Eq. 10 uses the similarities between all pairs of walks of same length. It is obvious that in the bag, some pairs of walks are not similar at all. It is useless to include them in the sum. On the contrary, $K_{max}$ only takes into account the best pair of walks, which is obviously not sufficient to completely describe the graph.

We propose a compromise between these two behaviors. In order to take into account the graphs in their totality, we want that each vertex is included at least in one walk. Thus we will start one walk from each vertex $v$ of $G$. In order to reduce combinatorial complexity, we will look for the vertex of $G'$ the most similar to $v$ (denoted $m_{G(}(v)$), and we start the walk in $G'$ by this vertex. The kernel between both graphs is then the average value of the best matches of the walks starting from each vertex of $G$ (plus the symmetric). More formally, the proposed kernel is :

$$K_{new}(G,G') = \frac{1}{|V|} \sum_{v\in G} \max_{\substack{h\in s_G(v) \\ h'\in s_{G'}(m_{G'}(v))}} K_C(h,h') + \frac{1}{|V'|} \sum_{v'\in G'} \max_{\substack{h'\in s_{G'}(v') \\ h\in s_G(m_G(v'))}} K_C(h,h') \quad (13)$$

$$\text{with } \begin{cases} h \in s_G(v) \Leftrightarrow v \text{ is the first vertex of } h \in H(G) \\ m_G(u) = \underset{w\in G}{\operatorname{argmax}}(k_V(w,u)) \end{cases}$$

This function is symmetric thanks to the two terms. The first term returns the best match value between any walk $h$ of $G$ starting from $v$ and any walk $h'$ of $G'$ starting from the closest vertex of $G'$ to $v$. This choice has two interesting properties. The first one is that we restrict the search of the best match by starting with the couple $(v, m_{G'}(v))$ of $v$ and its best match $m_{G'}(v)$, which increases the discrimination of the final function, as opposed to Kashima kernel. The second one is that this scheme allows the use of a branch and bound algorithm, which dramatically decreases the computation time, as opposed to Suard kernel.

Moreover as each term is an average value of the best matches between sets of walk of both graphs, the kernel has good generalisation properties. Let us note that this idea is close to the ones of Wallraven [40] and Suard [19], except that we added some restrictions that make the kernel more discriminant and faster to compute.

As kernels defined by Philipp [42], Wallraven [40] and Suard [19], this function does not fully satisfy Mercer conditions, but the Gram matrix is always semi-definite positive ($sdp$) on our datasets, and property of Eq. (6) also holds.

However, for databases where the Gram matrix is not semi-definite positive ($sdp$), several methods can be followed to use the function as a kernel. Let us recall that we focus on the interactive retrieval of objects in image databases, which means that the whole data is available before any search, and the learning methods are not used on unknown data. Furthermore, during the offline stage of the database, post-processing can be performed. The semi-definite positiveness of the Gram matrix is first checked. In the case where this property holds, retrieval sessions can begin. In the other case, a first solution is to identify the graphs that lead to a non-semi-definite positiveness of the Gram matrix. Then, some manual modifications can be performed to get the desired property. A second solution is to use non-$sdp$ techniques, for instance SVM for non-$sdp$ kernels [43] or Indefinite Kernel Fisher Discriminant [44].

### *2.5. Kernels on walks*

All the kernels on graphs expound in the previous section are based on kernels of walks $K_C(h,h')$, with $h \in H(G)$ and $h' \in H(G')$. We present now various propositions for these kernels on walks. They involve kernels on vertices $K_V$, and kernels on edges $K_E$, called minor kernels. In the following, we assume that we use Gaussian kernels, which return values between 0 and 1.

Kashima et al. used the following kernel on walks, which is a product of all similarities of vertices and of edges of both walks [17]:

$$K_{C_{mul}}(h,h') = K_V(v_0,v'_0)\prod_{i=1}^{|h|} K_E(e_i,e'_i)K_V(v_i,v'_i) \tag{14}$$

When using minor kernels that return values between 0 and 1, this kernel function always decreases with the length of walks $h$, which makes the similarity of long walks very small. This kernel function disadvantages long walks.

Philipp-Foliguet used the sum instead of the product [42] :

$$K_{C_{sum}}(h,h') = K_V(v_0,v'_0) + \sum_{i=1}^{|h|} K_E(e_i,e'_i)K_V(v_i,v'_i) \tag{15}$$

When using minor kernels that return positive values, this kernel function always increases with the length of walk $h$, which makes the similarity of short walks very small. This kernel function disadvantages short walks.

**First proposition.** We first propose an improved version of product kernel $K_{C_{mul}}$ (Eq. 14):

$$K_{C_{new1}}(h,h') = K_V(v_0,v'_0)\prod_{i=1}^{|h|}(1 + K_E(e_i,e'_i)K_V(v_i,v'_i)) \tag{16}$$

Adding 1 to the product of the minor kernels transforms the behaviour of the kernel when the walk length increases. This new product kernel increases with the length of the walk, getting thus the same behaviour than the sum kernel of Eq. (15).

**Second proposition.** The second proposition is also an improvement of the product kernel which only modifies the minor kernel on vertices:

$$K_{C_{new2}}(h,h') = K_V(v_0,v'_0)\prod_{i=1}^{|h|} K_E(e_i,e'_i)(1 + K_V(v_i,v'_i)) \tag{17}$$

Let us remind that $K_V$ is the minor kernel on vertices (which in our case represent regions) and $K_E$ the minor kernel on edges (which represent spatial relationships). In the case where walks $h$ and $h'$ link regions with the same spatial relationships, this kernel function will increase with the length of walks. In the other cases, a single difference between the spatial relationships of two regions will significantly decreases the similarity. This kernel is not monotonic with the walk length.

This kernel emphasizes region layout which has a strong semantic meaning. Moreover, when used with the efficient algorithm we present in the next section, this reduces the number of required walk comparisons.

| walk lengths | $H_{\Omega l}$ | $H_{El}$ | $H_{\bar{c}l}$ | $H_{\Omega}$ | $H_{E}$ | $H_{\bar{c}}$ |
|---|---|---|---|---|---|---|
| $\|h\| = 0$ | 5 | 5 | 5 | 5 | 5 | 5 |
| $\|h\| = 1$ | 25 | 25 | 20 | 20 | 20 | 20 |
| $\|h\| = 2$ | 125 | 100 | 100 | 90 | 60 | 60 |
| $\|h\| = 3$ | 625 | 380 | 320 | 320 | 180 | 120 |

Figure 1: Number of walks in the complete graph $K_5$ for different walk sets.

## 3. Computation of kernels

The main problem of the kernels on walks is the computational cost, especially in the applications of image retrieval, where the walk comparisons have to be performed for each image of a learning set and several iterations of classification. In order to reduce the complexity we propose several choices :

- Use a set of walks, able to properly represent the graph, but avoiding as most as possible the redundancy in order to be more efficient (see section 4.1).
- Use a search tree and similarity functions compatible with recursion : the similarity of walks of length $l$ is computed from similarity of walks of length $l-1$ (see section 4.2.2).
- Allow an approximate solution of walk comparisons, which is possible thanks to the use of the branch and bound algorithm, which allows the tree pruning.

### *3.1. Sets of walks*

The computation time of graph kernel depends on the number of walks of $H(G)$. This number depends on the number of vertices and on the "density" of the graph. We give in the first column of table in Fig. 1 the number of walks of lengths 0 to 3 for the complete graph $K_5$, which is a not oriented graph, without loop and whose all nodes are connected. Let us note that although the graph is not oriented, walks are oriented, and for example the walk between two vertices $ab$ is different from walk $ba$. The number of walks still increases if loops are added to the walks.

As there is much redundancy between all walks, a solution consists in reducing the number of walks by removing the walks supposed to be the most redundant.

Suard et al. only keeps acyclic walks which do not contain twice the same vertex. Another reduction consists in taking only Eulerian walks, which do not include twice the same edge.

More formally, if we denote $h = abc...$ a walk of graph $G$, with $a, b, c \in V$ , various sets of walks $H(G)$ can be defined :

- $H_{\Omega}(G)$ : walks without loops; (a loop is an edge $(v, v)$);
- $H_{\Omega l}(G)$ : walks with one loop for each vertex of the graph $G$ (used by Kashima [17]);
- $H_{E}(G)$ : Eulerian walks (an edge cannot be duplicated);

- $H_{El}(G)$ : Eulerian walks with loops (used by Philipp-Foliguet [42]);
- $H_{\bar{c}}(G)$ : walks without cycles (used by Suard [19]); (a cycle is a walk with first and last vertices equal);
- $H_{\bar{c}l}(G)$ : walks without cycles, but with loops.

The number of walks of each set in $K_5$ is compared in Fig. 1 for lengths 0 to 3.

In order to reduce the number of walks, Suard chooses a more specific set of walks. This set is composed of all walks which are a shortest walk between two vertices of the graph. The size of this set equals the number of pairs of vertices $|V|^2$, and the number of possible matches is bounded by $|V|^4$. Mahé et al. studied tottering in [24], where a walk can return to visited vertex just after leaving it.

### *3.2. Computation algorithm*

The computational complexity depends on the number of walks in the graphs, on the similarity function and on the kernel on walks. $K_{Kashima}$ kernel (Eq. (10)) requires an exhaustive comparison of all walks, since it performs the sum of all similarities between walks. If an incomplete solution is sufficient, the only way to reduce the computation with this kernel is to bound the lengths of the walks. On the contrary, to compute the $K_{max}$ kernel (Eq. (11)), only a part of the walks needs to be compared, the search of the maximum can be easily obtained by the branch and bound algorithm, and an incomplete solution is also reachable through a pruning of the search tree. The $K_{Suard}$ kernel (Eq. (12)) and our two proposed kernels can be computed without computing all walk comparisons.

In this section we discuss the complexity of graph kernel computation and the possible optimization. We present our representation which minimizes the complexity. Then we present the way we use it and how we use the branch and bound algorithm with our graph kernels.

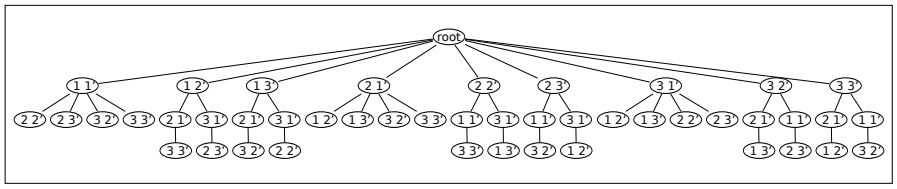
#### *3.2.1. Search tree*

An interesting property of the walks is that they can be built recursively. A search tree is used to represent all the walk comparisons. This representation allows to perform the computation with the "branch and bound" algorithm, whose recursivity well suits the recursivity of walks.

For two graphs $G = (V, E)$ and $G' = (V', E')$ and a generative function of set walks $H$, our search tree is composed of :

- a root
- nodes : each node represents a match of two vertices $n = (v, v') \in V \times V'$.
- a link between two nodes $n_1 = (v_1, v'_1)$ and $n_2 = (v_2, v'_2)$ means that there is an edge $e_{1,2}$ between $v_1$ and $v_2$ in $H(G)$, and an edge $e'_{1,2}$ between $v'_1$ and $v'_2$ in $H(G')$.


(a) An example of graph $G$ (b) An example of graph $G'$




(c) The corresponding search tree, with no cycles nor loops.

Figure 2: A example of search tree. Each path from the root until a leaf is a possible match between two walks. For instance, the corresponding path to the comparison of walks $(213)$ and $(3'1'2')$ is $(root) \rightarrow (23') \rightarrow (11') \rightarrow (32')$.

An example of search tree is shown in Fig. 2. We will call a path a walk in the search tree, starting from the root. A path corresponds to a comparison of two walks.

The main interest of the search tree is that it does not need to be built completely, it can be pruned during its building. For instance, for a search of a maximum, only the useful branches are built. The branch and bound algorithm allows this pruning.

*3.2.2. Application of branch and bound algorithm to kernel on walks*

The branch and bound algorithm aims at finding optimal solutions to problems whose goal is to find the maximal value of a function. It is especially adapted to solve the search of the maximum for functions whose bounds can be predicted on a given subset (in our case, a subset of walks).

In our case, the function is the kernel function $K_C$. To be employed with a search tree, this function must be computed recursively. Adding a node to a path in the tree from the root means adding a vertex to each of the walks compared in this path. More formally, let $h_i$ be a walk of $H(G)$ and $h'_i$ a walk of $H(G')$. If, in graph $G$, vertex $v$ (or edge $e$) is added to walk $h_i$, we obtain walk $h_{i+1}$, and in graph $G'$, if vertex $v'$ (or edge $e'$) is added to walk $h'_i$, we obtain walk $h'_{i+1}$. The node $(v, v')$ is added to the tree as a prolongation to this path representing $(h, h')$. To be efficient, the computation of $K_C(h_{i+1}, h'_{i+1})$ must be performed from the computation of $K_C(h_i, h'_i)$. All the kernels on walks presented in section 3.5 have this property.

The recursivity formula for these kernels are :

$$K_{C_{mul}}(h_{i+1}, h'_{i+1}) = K_{C_{mul}}(h_i, h'_i) K_E(e, e') K_V(v, v')$$

$$K_{C_{sum}}(h_{i+1}, h'_{i+1}) = K_{C_{sum}}(h_i, h'_i) + K_E(e, e') K_V(v, v')$$

$$K_{C_{new1}}(h_{i+1}, h'_{i+1}) = K_{C_{new1}}(h_i, h'_i)(1 + K_E(e, e') K_V(v, v'))$$

$$K_{C_{new2}}(h_{i+1}, h'_{i+1}) = K_{C_{new2}}(h_i, h'_i) K_E(e, e')(1 + K_V(v, v'))$$

Since we use Gaussian kernels, the values returned by $K_E$ and $K_V$ are always between 0 and 1. Thus, the value returned by $K_C$ can be easily bounded, and from the value of a node, the range of values of its descendants can be predicted. For instance, for walks $h$ of length $|h|$, the $K_{C_{sum}}$ function always returns values between 0 and $(1 + |h|)$. If we consider the subtree whose root is the node at the end of $(h_i, h_{i'})$ in the main tree, then the value of any node at depth $d$ of the subtree will be between $K_{C_{sum}}(h_i, h'_i)$ and $K_{C_{sum}}(h_i, h'_i) + d$. That shows the ability of our graph kernels to be used with the "branch and bound" algorithm.

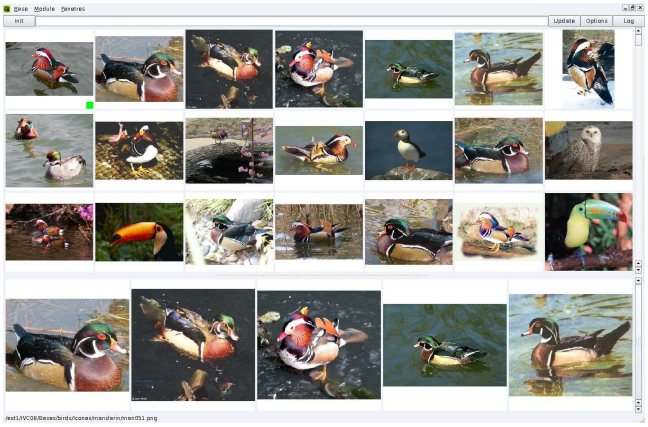


Figure 3: The RETIN graphic user interface. First three lines: retrieved images, ranked from one query (green square); Last line: images selected by the active learner and displayed for annotation.

## 4. Application to image retrieval

In order to browse or to classify a database, we use interactive machine learning. The user annotates the images to lead the search through a graphic user interface (*cf.* Fig. 3). The learning set is iteratively built thanks to the user, who labels the images as relevant or irrelevant to his query. The number of images a user can reasonably label in an on-line use is less than 100.

In this section we present our experiments on kernels on graphs. In a first approach, we compared the evolution of the retrieval task with three kernels on graphs and four kernels on walks. These experiments were first performed on a toy database. Then we used a public database for further experimentation and comparison to other methods.

For the first experiments, we built a toy database composed of 600 images. It is made of 50 objects of 12 views each and put on a random background. The objects come from the Columbia database[1], and the background is an image of landscape issued from the ANN database[2]. Examples of this database are displayed in Fig. 4.

[1] http://www1.cs.columbia.edu/CAVE/software/softlib/coil-100.php
[2] http://www.cs.washington.edu/research/imagedatabase/


Figure 4: Objects of Columbia on random backgrounds from ANN.

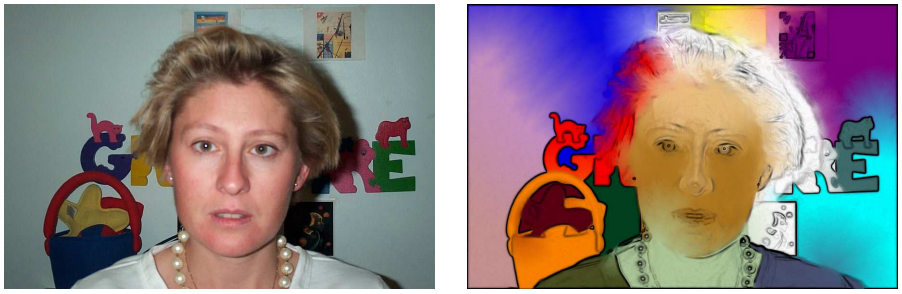


Figure 5: Example of fuzzy segmentation. Each colour corresponds to one of the 16 regions of the image. The more saturated the pixel, the more it belongs to the region.

For the second experiments, we used an image database of birds[3]. This database is composed of 6 categories of 100 photographs.

For the third experiments, we used the VOC database, a large generalist image database from the Pascal network challenge[4]. This database has about 6,000 images in 10 categories.

### *4.1. Image representation by an ARG*

Images are segmented into fuzzy regions, which allows to segment a whole database without tuning any parameter (only an interval for the number of regions is specified for the whole database [42]). The regions are fuzzy sets, which overlap more or less according to the colorimetric variation between them (*cf.* Fig. 5). In the toy database, images are represented by planar graphs of between 5 and 25 vertices, whereas in the bird database, there are between 10 and 50 vertices. The number of regions composing the object is variable from an image to the other, for example a face can be constituted by only one fuzzy region in an image and by five regions in another image. In the toy database, objects are generally covered by one to five regions, the other regions composing the background. In both databases, each region is represented by a histogram of 32 values, 8 values for the color (chrominances of $L^\star a^\star b^\star$) and 24 values for the gradient relative to the principal orientation of the region (3 scales and 8 orientations).

Images are represented by Attributed Relational Graphs, whose edges describe the layout of the regions. An edge between two adjacent regions $R_i$ and $R_j$ is described by 4 features : above, below, left to and right to.

Let us consider the set $F_{ij}$ of pixel couples $(p_i, p_j) \in R_i \times R_j$ neighbour in 4-connectivity. We define the following features, where $|\cdot|$ represents the cardinal number

[3] http://www-cvr.ai.uiuc.edu/ponce_grp/data/

[4] http://www.pascal-network.org/

of a set :

$$T_{ij}^{\mathcal{R}_t} = \frac{|\{(p_i, p_j) \in F_{ij}\ ,\ p_j \mathcal{R}_t p_i\}|}{|F_{ij}|} \text{ with } \begin{cases} p_i \mathcal{R}_1 p_j & \Leftrightarrow & p_i \text{ is above } p_j \\ p_i \mathcal{R}_2 p_j & \Leftrightarrow & p_i \text{ is below } p_j \\ p_i \mathcal{R}_3 p_j & \Leftrightarrow & p_i \text{ is left of } p_j \\ p_i \mathcal{R}_4 p_j & \Leftrightarrow & p_i \text{ is right of } p_j \end{cases} \quad (18)$$

Edge $e_{ij}$ between two regions $R_i$ and $R_j$ is then represented by a 4-dimension vector $e_{ij} = (T_{ij}^{\mathcal{R}_1} T_{ij}^{\mathcal{R}_2} T_{ij}^{\mathcal{R}_3} T_{ij}^{\mathcal{R}_4})$.

### *4.2. Experimental protocol*

In order to simulate an interactive search for an object or for a category, and also to be able to evaluate the results, each database is provided with a ground truth. Once a kernel function is chosen, a SVM classifier is trained on a training set composed of positive and negative examples allowing the ranking of images by relevance. The kernels also allow the use of active learning techniques in order to select the best images to be annotated by the user [45]. We evaluate each method by simulating a large number of retrieval sessions. For each session, a category is chosen. An image is randomly chosen within this category, and is annotated as positive. A first ranking of the database is then performed, only based on the similarity. Then the images selected by the active learning technique are annotated according to the category (as positive or negative). The classifier is then trained with this first set of examples, leading to a better classification of all images of the database. The ranking process is then repeated with the same principle of selection and of classification. The simulation of retrieval sessions are repeated a hundred of times for each category. The average quality of the ranking can be measured at each step of annotation thanks to the usual criterion of Average Precision used for example in TRECVID evaluation campaign [46]. At last, in order to have a global quality measure of our system, we compute the Mean Average Precision (MAP) on all categories.

We also compute the average computational time of each kernel in order to evaluate the tractability of large databases. The computational time is the average time for computing a value of a kernel on graphs $K(G, G')$ over the whole database.

For each experiment we used a Gaussian kernel with $\sigma = 1$ and $\chi^2$ distance as a kernel on vertices $K_V$ and a Gaussian kernel with a $L_1$ distance as a kernel on edges $K_E$. In all experiments carried out, the semi-definite positiveness of the Gram was checked before the simulated retrieval sessions, and during the each SVM training.

### *4.3. Experiments on the toy database*

We first used our toy database to compare the three kernels on graphs $K_{Kashima}$, $K_{max}$ and $K_{new}$ and the four kernels on walks $K_{C_{sum}}$, $K_{C_{mul}}$, $K_{C_{new1}}$, and $K_{C_{new2}}$ with three walk lengths. The set of walks is defined by $H_{\bar{c}}$, that is to say they have no cycles nor loops. Results are displayed in Fig. 6. Each graphic corresponds to a combination of a kernel on graphs $K$ and a kernel on walks $K_C$. Each graphic includes one to three curves, corresponding to walk lengths $|h| = 1$, 2 or 3. When the MAP is less than 60%, the curve does not appear.

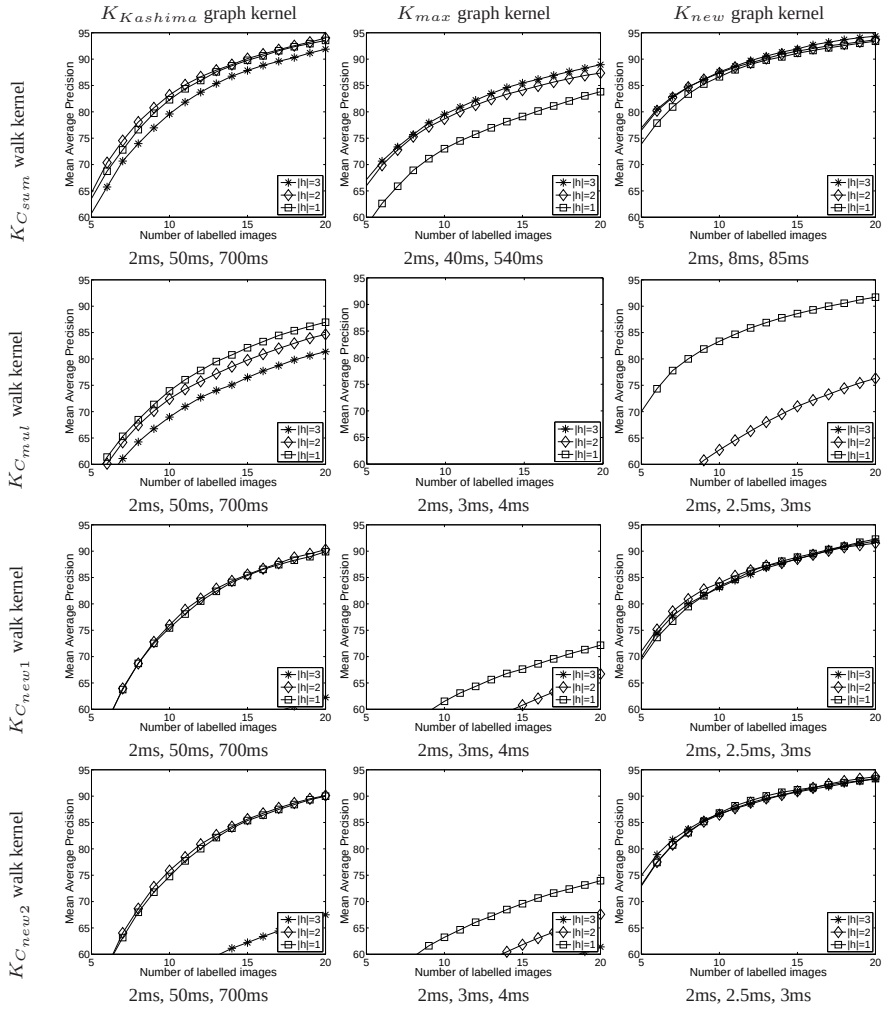
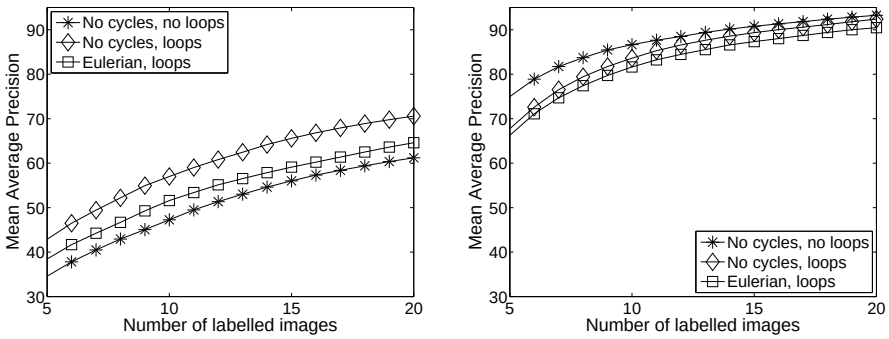



Figure 6: Comparison of kernels on graphs and kernels on walks on the toy database. Each column uses the same kernel on graph, and each row uses the same kernel on walk. Below each figure we show the average time required for the computation of one value of the graph kernel $K(G, G')$ for walks of length 1, 2 and 3. Let us note that in some cases, curves are not visible since their values are below 60%.




(a) $K_{max}$ graph kernel (b) $K_{new}$ graph kernel

Figure 7: Comparison of walk sets on the toy database.

**Kashima kernel**. Let us first comment the first column of Fig. 6, which concerns $K_{Kashima}$ kernel on graphs (Eq. 10). Whatever the kernel on walks, increasing the walk length decreases the performances. That can be explained by the fact that $K_{Kashima}$ kernel sums all similarities between all walks of the graph, the highest ones as well as the lowest ones. Thus, the sum of a large number of small similarity values tends to be larger than a single high similarity value as explained in section 3.3. However, except increasing computational time, the performances are about the same for all lengths. Now comparing the kernels on walks with $K_{Kashima}$, it turns out that $K_{C_{sum}}$ ( Eq. 15) is the most interesting . And finally, considering computational time, with all kernels on walks we have the same average time : 2ms for walks of length 1, 50ms for walks of length 2, and 700ms for walks of length 3. Thus, with walks of length 3, the comparison of one image to all images of a database of 10,000 images would require about 2 hours, which is completely intractable.

**Max kernel**. Let us now focus on the second column of Fig. 6, with kernel on graph $K_{max}$ (Eq. 11). Considering kernels on walks, we have very different behaviors. With kernel on walks $K_{C_{sum}}$ (Eq. 15), performances increase with the walk length. With kernel on walks $K_{C_{mul}}$ (Eq. 14), we have very bad results (curves are below 60%). This is certainly because the $K_{C_{mul}}$ function quickly decreases with the length of walks, which makes the comparison of long walks very difficult. Furthermore, since the $K_{max}$ graph kernel returns only one similarity between two walks (the best one), no averaging behavior like in $K_{kashima}$ can counterpart this. Considering the kernels on walks we introduce in this paper, even if they reduce the problems of the $K_{C_{mul}}$, this is not enough to get good results with the $K_{max}$ graph kernel. Finally, thanks to the branch and bound algorithm, the average time is very good (except with the $K_{C_{sum}}$) : from 2 to 4ms.

**New graph kernel**. Now we focus on the third column of Fig. 6, with the kernel on graph we introduce in this paper, $K_{new}$ (Eq. 11). Except for $K_{C_{mul}}$ walk kernels, performances are about the same whatever the length of the walk. This is due to the set of walks used by this kernel. Let us remind that each vertex is taken as the start of one walk, which is matched with a walk of the other graph. Because of the recursion

of the walk building, a walk of length 2 is built as a prolongation of a walk of length 1 and a walk of length 3 as a prolongation of a walk of length 2. It seems from these results that with this kernel, the structure of the graph is completely grasped by a set of edges (walks of length 1), at least for our goal of graph comparison. Moreover all results with this kernel outperforms those with $K_{max}$ and $K_{Kashima}$.

**Best combination**. The experiments on the toy database show that the best combination of kernels on graphs and kernels on walks is the graph kernel $K_{new}$ used with a walk kernel $K_{C_{new2}}$. This combination also has the lowest computational complexity, since walks of length 1 are enough. Thus, the exact comparison of one image to all images of a database of 10,000 images needs about 20s. This time can be reduced if approximate solutions are sufficient, which is generally true for online applications.

**Comparison of walk sets**. We also compared different sets of walks $H(G)$. We added to the sets of walks either walks with loops or walks with cycles (but restricted to Eulerian ones). Results are shown in Fig. 7 for $K_{max}$ and $K_{new}$ graph kernels, with $K_{C_{new2}}$ walk kernel and walks of length 3. $K_{Kashima}$ has not been evaluated since its computational complexity is intractable with Eulerian walk sets. With $K_{max}$ kernel, using a larger walk set increases the MAP. Since this graph kernel selects the best similarity between two walk sets, adding more walks increases the chance to find a better match. With $K_{new}$, as expected, performances are not improved, especially when few images are labeled.

**Example of retrieval session**. We present in Fig. 8 an example of interactive search with $K_{new}$ graph kernel and $K_{C_{new2}}$ walk kernel. The system is initialized with one image containing the query object, a red can. As can be seen on the top screen shot of Fig. 8, the system has returned many images with the same background (lake). This result is as expected: since the system does not known which parts of the image are relevant, it returns the images containing the regions which have the best matches with the query regions – in this example regions of lake. The user gives then 4 new labels, and asks for a system update. The new ranking is shown in the middle screen shot of Fig. 8, where many more cans are returned. There are still some unwanted images, certainly brought by the background regions in the second positive example (road, grass and trees). The user gives 5 more labels in order to help the system to find the discriminant region walks. The final ranking is shown in the bottom screen shot of Fig. 8, where 21 cans are present. Let us note that the 3 remaining cans of the database (not visible on this last screen shot), are the next ones in the ranking.

### *4.4. Experiments on a generalist database*

**Simulations**. We carried out the same experiments on the bird database, except for $K_{kashima}$ graph kernel and $K_{C_{sum}}$ walk kernel because of their high computational complexity. As one can see on Fig. 9, the kernel behaviors are almost the same. These results are interesting for the following reasons.

Firstly, the bird database is composed of real photographs, which shows that the conclusions we made on the toy database are also true for real data. Furthermore, although the sizes of the categories is very different – 100 images per category in birds (17% of the database), 12 images in toy (2% of the database) – the behavior of the system is the same.

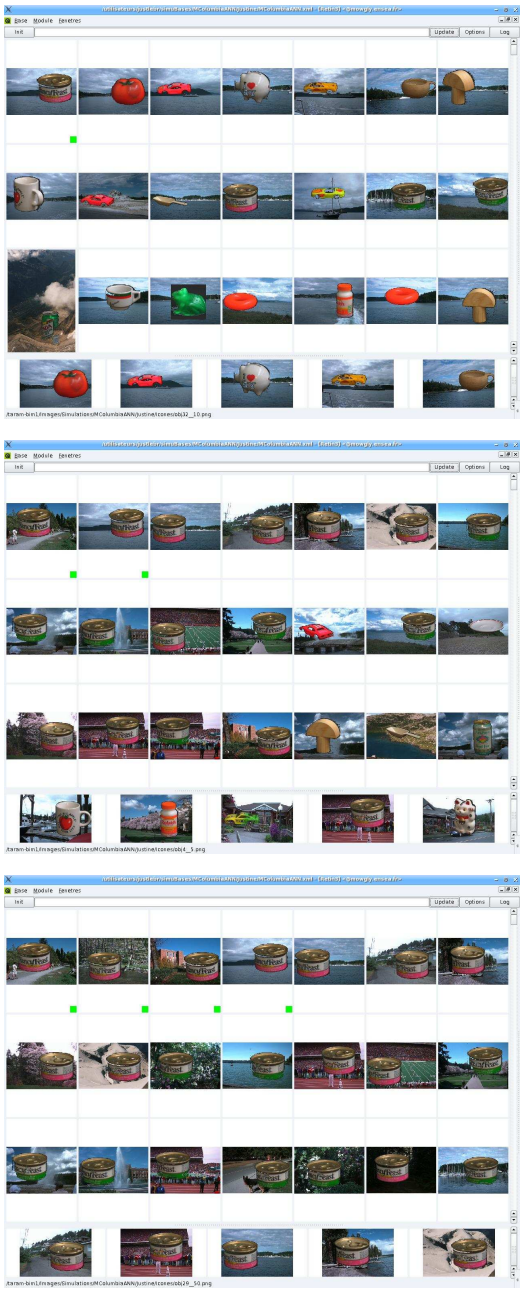


Figure 8: Example of retrieval session on the toy database. In each screenshot, the three first rows show the ranking of images according to the current classifier, and the last row shows the images selected by the active learner. Images with a small green square are images labeled as positive by the user. Top : Initial ranking with one query image. Middle : Result with 2 positive labels and 3 negative labels. Bottom : Result with 4 positive labels and 6 negative labels.

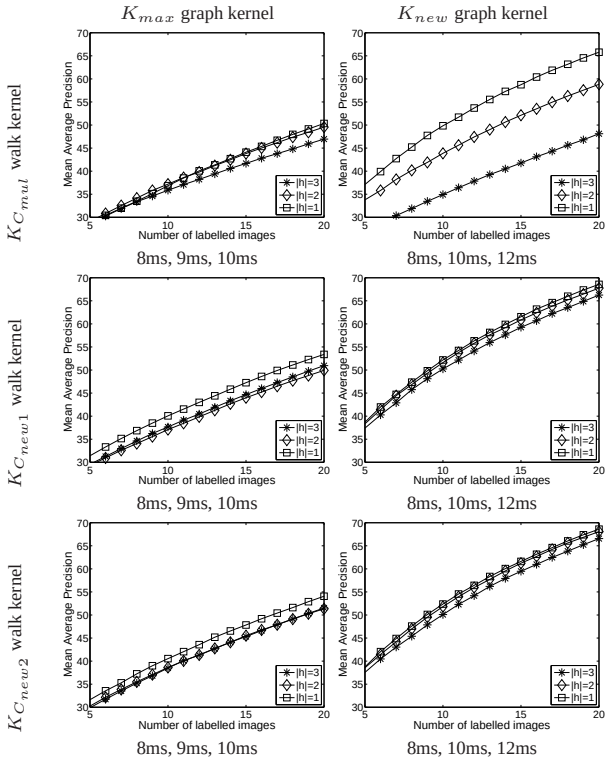



Figure 9: Comparison of kernels on graphs and kernels on walks on the bird database. Each column uses the same kernel on graph, and each row uses the same kernel on walk. Below each figure we show the average time required for the computation of one value of the graph kernel $K(G, G')$ for walks of length 1, 2 and 3.

Secondly, in this database, there is a correlation between the background and the objects – on the contrary to the toy database, where background is nothing but noise. For instance, some birds are always surrounded by water. That shows the ability of our learning scheme to deal with both cases, with or without meaningful context. Let us also remind that we use global annotation, i.e. the user never has to select the relevant parts of an image, he just has to inform the system if an image contains something he is looking for or not. The system is able to select by itself the discriminant parts (from object and/or background) through the combination of SVM and graph kernels .

Thirdly, the birds are made of larger sub-graphs than the objects in the toy database. Despite this difference, and especially for the graph kernel we introduced in this paper, the system is still able to catch matches even with walks of length 1.

Finally, the low-level features (color and gradient histograms) are less efficient on the bird database, as can be seen on the beginning of the curves, which starts with a MAP of about 30%. However, the performances quickly increase with more labels. For instance, with the kernels we propose (bottom right graphic of Fig. 9), with 20 labels the MAP goes up to 68%. That shows that our learning scheme is able to quickly catch the discriminant walks in the graphs. Let us note that, with 50 labels and the kernels we propose, the MAP is over 90%, which is comparable to other results on the same database, obtained with a larger learning set [47].

**Example of retrieval session**. We present in Fig. 10 an example of interactive search with $K_{new}$ graph kernel and $K_{C_{new2}}$ walk kernel on the bird database. The system is initialized with one image containing the query object, a white owl. Let us note that in the query image, the owl is partially occluded. As one can see on the top screen shot of Fig. 10, the system has returned images with owls, but also other birds. In the next feedback step with 8 labels (*cf.* middle screen shot of Fig. 10), as in the previous session example, new relevant objects are retrieved, but also new images are returned. This is a typical behavior of the system in the first feedback steps, where the ranking changes a lot. At those steps, the system tries to find the discriminant parts of images. Since new positive labels also bring new backgrounds, the system needs negative examples to remove them. Then, with those labels and new positive ones to reinforce the relevant parts, the systems tends to return only images with an owl (*cf.* bottom screen shot of Fig. 10).

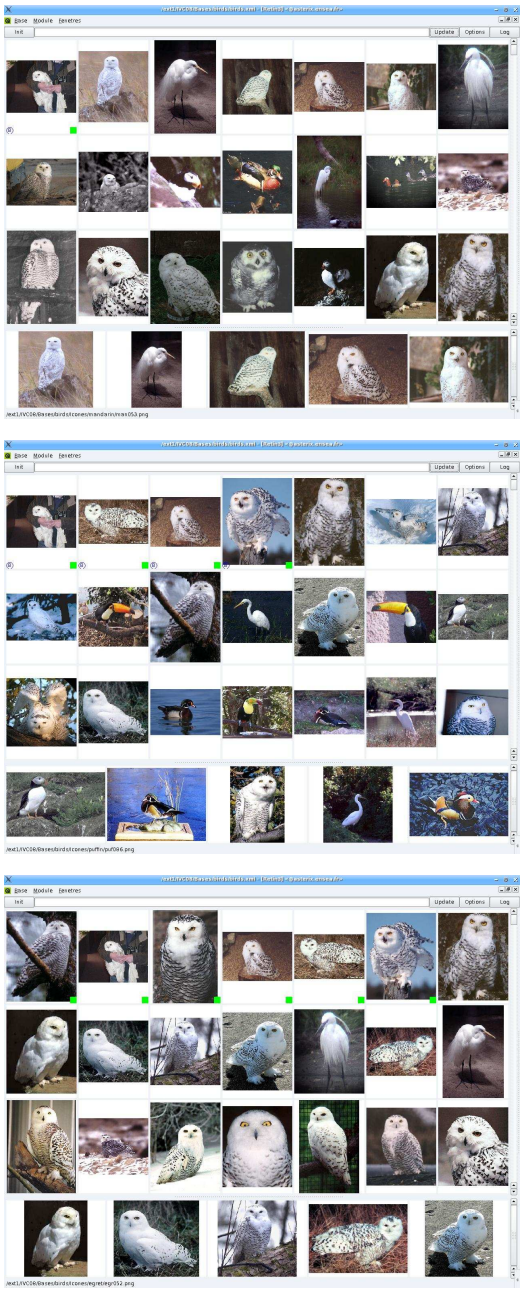


Figure 10: Example of retrieval session on the toy database. In each screenshot, the three first rows show the ranking of images according to the current classifier, and the last row shows the images selected by the active learner. Images with a small green square are images labeled as positive by the user. Top : Initial ranking with one query image. Middle : Result with 4 positive labels and 4 negative labels. Bottom : Result with 6 positive labels and 9 negative labels.

*4.5. Experiments on a large generalist database*

We carried out experiments on the VOC database in order to compare graph kernels to bags kernel. This database has 6,000 images in 10 categories. The categories are heterogeneous : some of them consider small objects, other scenes, objects with a relevant context, some without, etc. This is a very interesting database for the evaluation of methods in the most generic way, since a lot of retrieval problems in generalist databases can be found.

The following methods have been compared:

- Global histogram – each image is represented by a histogram of 64 colors and 64 textures. A Gaussian kernel with a $\chi^2$ distance was used.
- Bags of keypoints – each image is represented by a bag of about 100 SIFT vectors computed on MSER regions. The kernel on bag of Wallraven has been used [40]. Let us note that we did not tested graphs of keypoints since it is computationally untractable.
- Bags of regions – each image is represented by a bag of regions (the same as for graphs). The kernel on bag of Wallraven has also been used in this case.
- Graphs of regions – each image is represented by a graph of regions. The graph kernel proposed in this paper has been used with $K_{C_{new2}}$ walk kernel. On the contrary to previous experiments, an inexact similarity computation is performed, with a speed up of about 20%.

Global results are shown in Fig. 12, and results per category are shown in Fig. 13.

Considering features, the region-based ones are globally the best ones (*cf.* Fig. 12). However, as one can see on Fig. 13, there are very different results for each category. Even if they have no structural information, the global histograms give honorable results. This is certainly because, for several categories, the context is discriminant. Thus, retrieving the scene is about the same than retrieving the object. The keypoints have the best results on two categories (bicycles and dogs). This can be explained by the fact that there are near-identical visual features on the objects and no discriminant context. And finally, the region-based methods give the best results, and especially when employing graphs. This result is interesting since it shows the ability of our model to handle heterogeneous categories.

Considering computation time, the global histogram is by far the fastest method. The average computation of one value of a kernel point is around $1\mu s$. On the other side, the bags of SIFT are the slowest with about $15ms$. The bags of regions require about $150\mu s$ per bag comparison, which is reasonable for the improvement (about 10% with 50 labels). For the graphs of regions, the exact similarity computation requires $2.5ms$ per comparison, but with approximation, about the same results (the ones presented in the figures) were obtained with $2ms$ per comparison. Compared to bags, the small increase of computational time is worth the performance increase.

In order to evaluate the performances of the methods from the user viewpoint, we show in Fig. 14 the Top 100 for each category, with 25 labels[5]. This shows the average

[5]With 50 labels, the 100 first images returned by the system are all relevant with most of the methods.

number of images the user sees in the first screens of the graphic interface. The global behavior of the methods is about the same. The graphs of regions give the best results, with an average Top of 85.

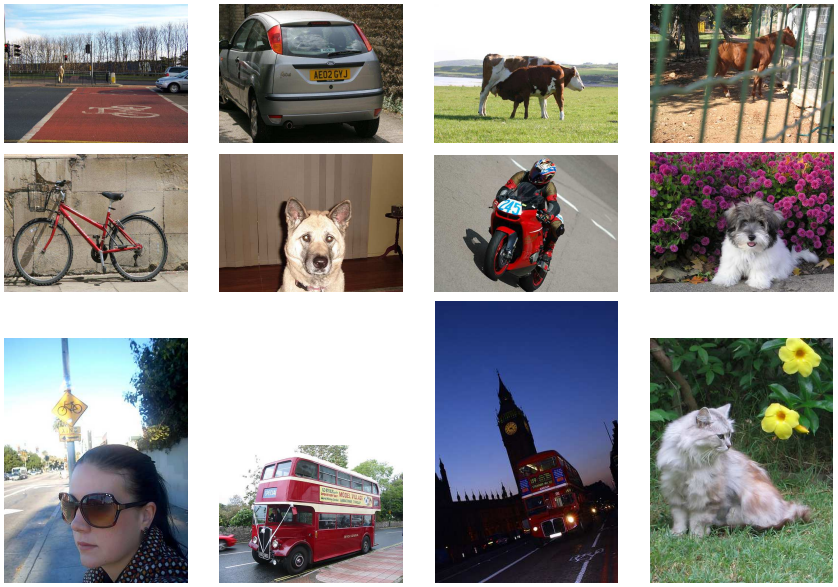


Figure 11: Images from the VOC database.

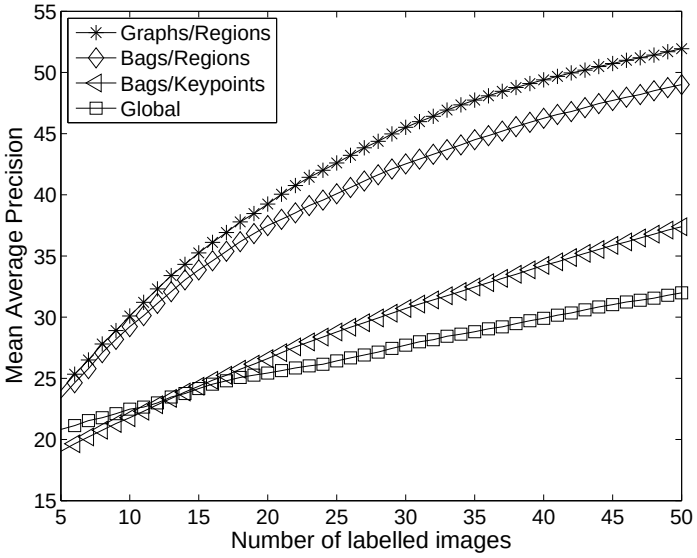



Figure 12: Comparison of features and kernels on the VOC database.

| category | Global | Bags/Keypoints | Bags/Regions | Graphs/Regions |
|---|---|---|---|---|
| cat | 27 | 33 | **48** | 45 |
| car | 53 | 60 | 61 | **68** |
| cow | 32 | 28 | **59** | **59** |
| horse | 20 | 28 | **42** | **42** |
| sheep | 43 | 34 | 59 | **65** |
| motorbike | 27 | 22 | 22 | **49** |
| bicycle | 27 | **53** | 44 | 47 |
| dog | 22 | **35** | 33 | 34 |
| person | 37 | 41 | **51** | **51** |
| bus | 32 | 40 | 51 | **59** |
| total | 32 | 37 | 48 | **52** |

Figure 13: Average Precision (%) on the categories of the VOC database, with 50 labels.

| category | Global | Bags/Keypoints | Bags/Regions | Graphs/Regions |
|---|---|---|---|---|
| cat | 49 | 52 | 84 | 83 |
| car | 80 | **100** | 98 | **100** |
| cow | 59 | 34 | 89 | **92** |
| horse | 33 | 37 | **67** | 65 |
| sheep | 72 | 60 | 95 | **96** |
| motorbike | 42 | 29 | 74 | **79** |
| bicycle | 42 | **96** | 92 | 94 |
| dog | 38 | 58 | 54 | **64** |
| person | 47 | 70 | 86 | **87** |
| bus | 45 | 62 | 83 | **90** |

Figure 14: Number of relevant images within the 100 first ones returned by the system (Top 100), on the categories of the VOC database, with 25 labels.

## 5. Conclusion

We have shown in this paper that the similarity between two graphs can be efficiently and effectively computed through a set of walks within the graphs. We have proposed a new kernel on graphs which improves the effectiveness of Kashima graph kernel, while dramatically decreasing the computational time. This kernel is suitable for any type of graph matching - and especially for inexact graph matching - and for labeled or attributed vertices or edges. We have also proposed two improvements of the kernel on walks used by Kashima. The combination of these new kernels on graphs and kernels on walks on two databases gives high results.

More surprising with the new kernel on graphs applied to image regions, instead of computing the similarities between all pairs of walks from both graphs, as previous work did, the length of the walks can be reduced until considering only pairs of regions, without loss of performance. This set of matched pairs must cover both graphs (one edge starting from each vertex of each graph). The structure of the graph is thus captured by a set of edges, which is sufficient to achieve the graph matching.

In other applications longer walks are necessary, for example molecules studied or skeletons in pattern recognition. For these cases, the inexact matching of graphs can be achieved with the method we propose thanks to branch and bound algorithm, which well suits the recursive building of the walks. Inexact solutions can also be obtained more rapidly with this algorithm. The algorithm is thus general and can be applied to any problem of graph matching.

The results on several databases show that the method is usable for on-line queries. Applications concern partial queries such as retrieving images including a particular object, whatever the background. Furthermore, the learning is weakly-supervised, which means that there is no need for the user to indicate the area of interest in images.

## References


[1] D. Conte, P. Foggia, C. Sansone, M. Vento, Thirty years of graph matching in pattern recognition, International Journal of Pattern Recognition and Artificial Intelligence 18 (3) (2004) 265–298.

[2] H. Bunke, X. Jiang, Graph matching and similarity, in: H.-N. Teodorescu, D. Mlynek, A. Kandel, H.-J. Zimmermann (Eds.), Intelligent Systems and Interfaces, International series in intelligent technologies, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2000, pp. 281–301.

[3] T. Gärtner, A survey of kernels for structured data, SIGKDD Explorations 5 (1) (2003) 49–58.

[4] O. Sammoud, S. Sorlin, C. Solnon, K. Ghedira, A comparative study of ant colony optimization and reactive search for graph matching problems, in: 6th European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP 2006), 2006, pp. 287–301.

[5] R. Ambauen, S. Fischer, H. Bunke, Graph edit distance with node splitting and merging, and its application to diatom identification, in: IAPR-TC15 Workshop on Graph-based Representation in Pattern Recognition, 2003, pp. 95–106.

[6] M. Neuhaus, K. Riesen, H. Bunke, Fast suboptimal algorithms for the computation of graph edit distance, in: SSPR/SPR, 2006, pp. 163–172.

[7] L. Cordella, P. Foggia, C. Sansone, M. Vento, Subgraph transformation for the inexact matching of attributed relational graphs, Computing 12 (1998) 43–52.

[8] S. Berretti, A. D. Bimbo, E. Vicario, Efficient matching and indexing of graph models in content-based retrieval, IEEE Trans. on PAMI 23 (10) (2001) 1089–1105.

[9] B. McKay, Practical graph isomorphism, Congressus Numerantium 30 (1981) 45–87.

[10] K. Riesen, M. Neuhaus, H. Bunke, Bipartite graph matching for computing the edit distance of graphs, in: GbRPR, 2007, pp. 1–12.

[11] W. Aguilar, Y. Frauel, F. Escolano, M. E. Martinez-Perez, A. Espinosa-Romero, M. A. Lozano, A robust graph transformation matching for non-rigid registration, Image and Vision Computing 27 (7) (2009) 897–910.

[12] S. Gold, A. Rangarajan, A graduated assignment algorithm for graph matching, IEEE Transactions on Pattern Analysis and Machine Intelligence 18 (4) (1996) 377–388.

[13] E. Bengoetxea, P. Larrañaga, I. Bloch, A. Perchant, C. Boeres, Inexact graph matching by means of estimation of distribution algorithms, Pattern Recognition 35 (12) (2002) 2867–2880.

[14] S. Sorlin, C. Solnon, Reactive taboo search for measuring graph similarity, in: IAPR Workshop on Graph-based Representation in Pattern Recognition, 2005, pp. 172–182.

[15] S. V. N. Vishwanathan, N. N. Schraudolph, I. R. Kondor, K. M. Borgwardt, Graph kernels, Journal of Machine Learning Research (2009) submitted.

[16] T. Gärtner, P. A. Flach, S. Wrobel, On graph kernels: Hardness results and efficient alternatives, in: COLT, 2003, pp. 129–143.

[17] H. Kashima, Y. Tsuboi, Kernel-based discriminative learning algorithms for labeling sequences, trees and graphs, in: International Conference on Machine Learning (ICML), Banff, Alberta, Canada, 2004, p. 58.

[18] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. V. N. Vishwanathan, A. J. Smola, H.-P. Kriegel, Protein function prediction via graph kernels, in: ISMB (Supplement of Bioinformatics), 2005, pp. 47–56.

[19] F. Suard, V. Guigue, A. Rakotomamonjy, A. Bensrhair, Pedestrian detection using stereo-vision and graph kernels, in: Intelligent Vehicles Symposium, Las Vegas, Nevada, 2005, pp. 267–272.

[20] B. Schölkopf, A. Smola, Learning with Kernels, MIT Press, Cambridge, MA, 2002.

[21] D. Haussler, Convolution kernels on discrete structures, Tech. rep., Institute of California at Santa Cruz (1999).

[22] R. I. Kondor, J. D. Lafferty, Diffusion kernels on graphs and other discrete input spaces, in: ICML, 2002, pp. 315–322.

[23] L. Ralaivola, S. J. Swamidass, H. Saigo, P. Baldi, Graph kernels for chemical informatics, Neural Networks 18 (8) (2005) 1093–1110.

[24] P. Mahé, J.-P. Vert, Graph kernels based on tree patterns for molecules, Machine Learning 75 (1) (2009) 3–35.

[25] N. Shervashidze, K. M. Borgwardt, Fast subtree kernel on graphs, in: NIPS, 2009, p. to appear.

[26] N. Shervashidze, S. V. N. Vishwanathan, T. Petri, K. Mehlhorn, K. M. Borgwardt, Efficient graphlet kernels for large graph comparison, in: AISTATS, 2009, p. to appear.

[27] R. C. Wilson, E. R. Hancock, Pattern spaces from graph polynomials, in: 12th International Conference On Image Analysis And Processing, 2003, pp. 480–485.

[28] Y. Chen, J. Wang, Image categorization by learning and reasoning with regions, International Journal on Machine Learning Research 5 (2004) 913–939.

[29] F. Moosmann, E. Nowak, F. Jurie, Randomized clustering forests for image classification, IEEE Trans. on Pattern Analysis and Machine Intelligence 30 (2008) 1632–1646.

[30] H. Bunke, K. Riesen, A family of novel graph kernels for structural pattern recognition, in: CIARP, Vol. 4756, 2007, pp. 20–31.

[31] A. M. Smalter, J. Huan, G. H. Lushington, Gpm: A graph pattern matching kernel with diffusion for chemical compound classification, in: BIBE, 2008, pp. 1–6.

[32] I. Gondra, D. Heisterkamp, Learning in region-based image retrieval with generalized support vector machines, in: IEEE International Conference on Computer Vision and Pattern Recognition Workshop (CVPRW), Vol. 27, 2004, pp. 149–149.

[33] J. Mairal, F. Bach, J. Ponce, G. Sapiro, Online dictionary learning for sparse coding, in: International Conference on Machine Learning (ICML), 2009, pp. 689–696.

[34] M. Leordeanu, M. Hebert, A spectral technique for correspondence problems using pairwise constraints, in: International Conference of Computer Vision (ICCV), 2005, pp. 1482–1489.

[35] O. Duchenne, F. Bach, I. Kweon, J. Ponce, A tensor-based algorithm for high-order graph matching, in: IEEE International Conference on Computer Vision and Pattern Recognition (CVPR), 2009, pp. 1980–1987.

[36] J. Eichhorn, O. Chapelle, Object categorization with svm: kernels for local features, Tech. rep., Max Planck Institute (2004).

[37] S. Lyu, Mercer kernels for object recognition with local features, in: IEEE International Conference on Computer Vision and Pattern Recognition, Vol. 2, San Diego, CA, 2005, pp. 223–229.

[38] P. Gosselin, M. Cord, S. Philipp-Foliguet, Kernel on bags of fuzzy regions for fast object retrieval, in: IEEE International Conference on Image Processing, Vol. 1, San Antonio, Texas, USA, 2007, pp. 177–180.

[39] J. Shawe-Taylor, N. Cristianini, Kernel methods for Pattern Analysis, Cambridge University Press, ISBN 0-521-81397-2, 2004.

[40] C. Wallraven, B. Caputo, A. Graf, Recognition with local features: the kernel recipe, in: International Conference on Computer Vision (ICCV), Vol. 2, 2003, pp. 257–264.

[41] S. Sorlin, O. Sammoud, C. Solnon, J.-M. Jolion, Mesurer la similarite de graphes, in: Extraction de Connaissance a partir d'Images (ECOI'06), Atelier de Extraction et Gestion de Connaissances (EGC'06), 2006, pp. 21–30.

[42] S. Philipp-Foliguet, J. Gony, FReBIR : Fuzzy regions-based image retrieval, in: Information Processing and Management of Uncertainty (IPMU), Paris, France, 2006, pp. 693–707.

[43] B. Haasdonk, Feature space interpretation of svms with indefinite kernels, IEEE Transactions on Pattern Analysis and Machine Intelligence 27 (4) (2005) 482–492. doi:http://doi.ieeecomputersociety.org/10.1109/TPAMI.2005.78.

[44] B. Haasdonk, E. Pekalska, Indefinite kernel fisher discriminant, in: Int. Conf. on Pattern Recognition (ICPR), Tampa, USA, 2008, pp. 1–4.

[45] P. Gosselin, M. Cord, Precision-oriented active selection for interactive image retrieval, in: IEEE International Conference on Image Processing, Atlanta, GA, USA, 2006, pp. 3197–3200.

[46] TREC Video Retrieval Evaluation Campain, http://www-nlpir.nist.gov/projects/trecvid/.

[47] S. Lazebnik, C. Schmid, J. Ponce, A maximum entropy framework for part-based texture and object recognition, in: ICCV, 2005, pp. 832–838.