Symbolic Learning vs. Graph Kernels: An Experimental Comparison in a Chemical Application

Luc Brun[1], Donatello Conte[3], Pasquale Foggia[3], Mario Vento[3], and Didier Villemin[2]

[1] GREYC UMR CNRS 6072,
[2] LCMT UMR CNRS 6507
ENSICAEN-Université de Caen Basse-Normandie,
14050 Caen, France
luc.brun@greyc.ensicaen.fr, didier.villemin@ensicaen.fr
[3] Dipartimento di Ingegneria dell'Informazione e di Ingegneria Elettrica,
Università di Salerno, Via Ponte Don Melillo, 1 I-84084 Fisciano (SA), Italy
{dconte, pfoggia, mvento}@unisa.it

Abstract. In this paper we present a quantitative comparison between two approaches, Graph Kernels and Symbolic Learning, within a classification scheme. The experimental case-study is the predictive toxicology evaluation, that is the inference of the toxic characteristics of chemical compounds from their structure. The results demonstrate that both approaches are comparable in terms of accuracy, but present pros and cons that are discussed in the last part of the paper.

1 Introduction

Many scientific areas are populated by applications dealing with structured information, i.e., complex information which can be seen as made of simpler parts suitably interconnected. Structured information is widely used in many applicative domains as software engineering, pattern recognition, chemistry, medicine, linguistics, etc. Usually structured information is represented by means of graphs because of their high expressive power.

Despite their attractiveness in terms of representational power, structural methods (i.e., methods dealing with structured information) imply complex procedures both in the recognition and in the learning processes. The difficulty of defining effective algorithms for facing this task is so high that the problem is considered still open.

In this paper we focus our attention on two different approaches using graphs for classification tasks. The first one is based on the idea of facing the learning problem directly in the representation space of the structured data (we will call it Symbolic Learning). The second is based on the notion of Graph Kernels that gives a measure of similarity between graphs which corresponds to a metric thus making possible the adoption of well-known statistical/neural paradigms. Here

we present several Graph Kernels based on the notion of bag of patterns and graph edit distance that both preserve most of the structural information of graphs.

The chosen application domain is the Predictive Toxicology Evaluation, that is the characterization of the toxic nature of chemical compounds from their chemical structure. Namely, we used data from the Predictive Toxicology Challenge [4], a competition based on the prediction of cancerogenic molecules. This domain is appropriate for the analysis because of the structured nature of chemical compounds.

The paper is organized as follows: in section 2 a brief survey of the two approaches is presented, while details on the two methods will be given in the section 3; section 4 reports the results of an experimental case-study of both the approaches. Finally, in the last section, conclusions are outlined.

2 Related Works

The Symbolic Learning can be summarized as follows: the goal of the system is to build a structural description for each class (prototypes) so that the classification is conducted by mean of a structural matching between the prototypes and the structured representation of the object to classify.

Papers dealing with this approach can be ascribed to two rather different categories. The first category collects methods based on the assumption that the prototypical descriptions are built by interacting, more or less deeply, with an expert of the domain. To this concern, some authors [12] start from some ideal prototypes and refine manually, in successive approximations, a model of the possible deformations. Others [10] assume a deformational model and use a small training set for tuning the parameters contained in it. Despite the advantage of facing the problem without a training set or at least with a small training set, this approach is effective only in simple problems. The inadequacy of human knowledge to find a set of prototypes really representative of a given class significantly increases the risk of errors, especially in domains containing either many data or many different classes.

The methods ascribed to the second category face the problem in a formal way ([8]) by considering it as a symbolic machine learning problem, so formulated: "Given a suitably chosen set of input data, whose class is known and possibly some background domain knowledge, find out a set of optimal prototypical descriptions for each class".

Graph kernels methods are based on an implicit embedding of graphs within a vector space of large dimension. Several graph kernels are introduced in literature but graph kernels built on the notion of bag of patterns seem to be very promising. The design of such kernels, implies the following choices or heuristics:

1. Selection of a bag of patterns from a graph,

[4] http://www.predictive-toxicology.org

2. Definition of a kernel between bags based on a kernel between patterns,
3. Definition of a kernel between patterns.

Kernels between graphs then correspond to kernels between bags of patterns. The implicit assumption being here that bags of patterns capture most of the structural information of graphs. Several kernels have been proposed based on different types of patterns: Vert [7] proposed to compare the set of sub-trees of two graphs; Graphlets kernels [13] are based on the number of common subgraphs of two graphs. However, the comparison of complex patterns requires important processing times for an insight which is not always clearly demonstrated [7]. As a consequence most of graph kernels are based on simpler patterns such as walks [6], trails [2] or paths. Bags of patterns may be finite or infinite. Infinite bags, such as bag of walks are computed implicitly using random walks defined on both graphs being compared [6]. Bags of patterns may also be computed explicitly [2] by enumerating all patterns of a given graph. Given two bags of patterns the similarity between both bags is often measured using convolution kernels [4]. Both bags may alternatively be considered as two sets in the feature space induced by the kernel between patterns. This point of view allows to measure the similarity between bags using general kernels between sets. Finally, the similarity between patterns should be designed according both to the feature available on each element of the pattern and to the structural noise which may corrupt patterns. Using walks on labeled graphs the well known kernel [6] being equal to 1 if both walks are similar, an 0 otherwise, encodes the fact that labels are either equal or not comparable. The construction of a bag of patterns from a graph removes connections between elementary patterns within input graphs which are then only represented by their bags. As previously mentioned, assumption of bag of pattern kernels is that most of the relevant information about graphs is contained in the pattern and thus that most of the information is preserved by the transformation from a graph to a bag. Bunke [1] proposed several kernels based on the graph edit distance which do not rely on such an assumption.

3 Methods Description

3.1 Symbolic Learning

The Symbolic Learning Method presented in this section (hereafter called SLM) is inspired by Quinlan's FOIL algorithm particularizing it for Attributed Relational Graph (ARG) representation. An extension of this representation has been presented for describing prototypes of Attributed Relational Graphs; these graphs, called Generalized Attributed Relational Graphs (GARGs), contain generalized nodes, edges and attributes. Then, a learning algorithm has been formulated to build such prototypes by means of a set of operations directly defined on graphs. The algorithm preserves the generality of the prototypes generated by the classical machine learning algorithms and moreover, similarly to most

machine learning systems ([8]), the prototypes obtained by this system are consistent, i.e. each prototype covers samples of a same class.

An ARG can be defined as a 6-tuple $(N, E, A_N, A_E, a_N, a_E)$, where N and $E \subset N \times N$ are, respectively, the sets of nodes and edges of the ARG, A_N and A_E the sets of nodes and edge attributes and, finally, a_N and a_E the functions which associate to each node or edge of the graph the corresponding attributes.

A GARG is used for representing a prototype of a set of ARGs. In order to allow a GARG to match a set of possibly different ARGs, in the SLM the attribute definition has been extended. First of all, the set of types of node and edge attributes is extended with the special type Φ, carrying no parameter and allowed to match any attribute type, ignoring the attribute parameters. We say that a GARG $G^* = (N, E, A_N, A_E, a_N, a_E)$ covers a sample G and we use the notation $G^* \triangleright G$ (the symbol $\triangleright$ denotes the relation called covering) if there is a mapping $\mu : N^* \rightarrow N$ such that μ is a *monomorphism* and the attributes of the nodes and of the edges of G^* are *compatible* with the corresponding ones of G. The first condition requires that each primitive and each relation in the prototype is present also in the sample; note that the converse condition does not hold. The latter condition constrains the monomorphism to be consistent with the attributes of the prototype and of the sample in the sense that the type of the attribute of the prototype must be either equal to the special type Φ or to the type of the corresponding attribute of the sample. In the latter case, all the parameters of the attribute, which are actually sets of values, must contain the value of the corresponding parameter of the sample. The goal of the learning algorithm can be stated as follows: there is a (possibly infinite) set S^* of all the patterns that may occur, partitioned into C different classes $S_1^*, \ldots, S_C^*$, with $S_i^* \cap S_j^* = \phi$ for $i \neq j$; to the algorithm is given a finite subset $S \subset S^*$ (training set) of labeled patterns ($S = S_1 \cup \cdots \cup S_C$ with $S_i = S \cap S_i^*$), from which it tries to find a sequence of prototype graphs $G_1^*, G_2^*, \ldots, G_p^*$, each labeled with a class identifier, such that:

$$\forall G \in S^* \exists i : G_i^* \triangleright G (\textit{completeness} \text{ of the prototype set}) \quad (1)$$

$$\forall G \in S^*, G_i^* \triangleright G \Rightarrow \textit{class}(G) = \textit{class}(G_i^*) (\textit{consistency} \text{ of the prototype set}) \quad (2)$$

where $\textit{class}(G)$ and $\textit{class}(G^*)$ refer to the class associated with sample G and prototype G^*, respectively. Of course, this is an ideal goal since only a finite subset of S^* is available to the algorithm; in practice, the algorithm can only demonstrate that completeness and consistency hold for the samples in S. On the other hand, (1) dictates that, in order to get as close as possible to the ideal case, the prototypes generated should be able to model samples also not found in S. However, they should not be too general otherwise (2) will not be satisfied.

A description of the learning algorithm is presented in the following: the algorithm starts with an empty list L of prototypes and tries to cover the training set by successively adding consistent prototypes. When a new prototype is found, the samples covered by it are eliminated and the process continues on the remaining samples of the training set. Then a sample is compared sequentially against the prototypes in the same order in which they have been generated, and

it is attributed to the class of the first prototype that covers it. In this way, each prototype implicitly entails the condition that the sample is not covered by any previous prototype. The algorithm fails if no consistent prototype covering the remaining samples can be found. It is worth noting that the test of consistency in the algorithm actually checks whether the prototype is almost consistent, i.e. almost all the samples covered by G belongs to the same class:

$$Consistent(G^*) \Leftrightarrow \max_i \left(|S_i(G^*)| \,/\, |S(G^*)|\right) \geq \theta \tag{3}$$

where $S(G^*)$ denotes the sets of all the samples of the training set covered by a prototype G^*, and $S_i(G^*)$ the samples of the class i covered by G^* and θ is a threshold close to 1. For further details about the representation and the algorithm you can refer to [3].

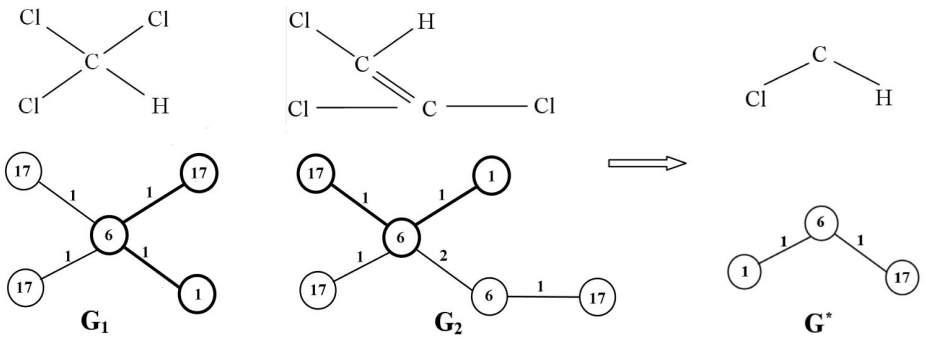

Fig. 1. An example of GARG covering two graphs. The example is extracted from the application domain used in the experimental phase. In bold the parts of the two graphs covered by G^*

An heuristic function H is introduced for evaluating how promising is the provisional prototype. The heuristic function used in the algorithm is given by the product of H_{compl} and H_{cons}:

$$H = H_{compl}(S, G^*) \cdot H_{cons}(S, G^*) \tag{4}$$

and it is based on estimation of consistency and completeness of prototype. The use of H_{cons} will drive the algorithm towards consistent prototypes while the term H_{compl} is introduced in order to privilege general prototypes with respect to prototypes which, albeit consistent, cover only a small number of samples.

See Figure 1 for an example of a GARG G^* that covers the graphs G_1 and G_2 representing two chemical compounds.

3.2 Graph Kernels

In this section we will introduce two kind of Graph Kernels: Bag of trails Kernel and Laplacian Kernel.

Bag of trails Kernel Let us consider a *graph* $G = (V, E)$ where V denotes the set of vertices and $E \subset V \times V$ the set of edges. We define a *simple-graph* as a graph with no multiple edges between two nodes and no loop (an edge linking a node with itself). We define a *walk* as an alternating sequence of nodes and edges, a *trail* as a walk with distinct edges and a *path* as a trail with distinct nodes. The description of a graph by a bag of walks is penalized by a tottering effect since long walks may remain on few edges hereby describing small parts of a graph. Bags of trails reduce drastically this tottering effects and may capture non linear features such as loops which are not described by open Paths. We thus use bags of trails, in the remaining part of this paper. The cardinality of a bag T being denoted by $|T|$. The kernel between trails is denoted K_{trail}.

Suard [16] has proposed numerous kernels for bags of paths which can be easily extended to bags of trails. We proposed to extend the mean kernel to bag of trails. By considering bag of trails as sets and a trail kernel K_{trail}, we can design a convolution kernel called *mean kernel* as follows:

$$K_{mean}(T_1, T_2) = \frac{1}{|T_1|}\frac{1}{|T_2|}\sum_{t\in T_1}\sum_{t'\in T_2} K_{trail}(t, t'). \tag{5}$$

where T_1 and T_2 denote two bags of trails.

Following Haussler [4], this kernel is positive definite on the bag of trails domain if and only if K_{trail} is positive definite on the trail domain.

However, the mean kernel compares all the paths of both bags without considering their relevance in both bags. As a consequence, irrelevant paths may corrupt the overall value of the kernel due to the "meaning" effect. Recently Dupé et al. [2], proposed to limit this effect by using a relevance measure of trails. Let T_1 and T_2 two bags of trails, the *weighted mean kernel* between these two bags is then defined as:

$$K_{weighted}(T_1, T_2) = \frac{1}{|T_1|}\frac{1}{|T_2|}\sum_{t\in T_1}\sum_{t'\in T_2} <R_{T_1}(t), R_{T_2}(t')>^d K_{trail}(t, t'). \tag{6}$$

where $d \in \mathbb{R}^+$ allows to weight the importance of the relevance measure and $R_T(t)$ denote the relevance of trail t according to the bag T. This relevance measure is defined as the similarity (the kernel) between the input trail and the mean trail of the bag ($R_T(t) = \frac{1}{|T|}\sum_{t'\in T} K_{trail}(t, t')$).

Kernel defined by equation 6 is a convolution kernel and so is positive definite if K_{trail} is positive definite.

The above bag of trail kernels are based on a trail kernel K_{trail}. A kernel between two trails $t = (v_1, e_1, \ldots, v_n)$ and $t' = (v'_1, e'_1, \ldots, v'_p)$ can be derived from the path kernel proposed by Kashima [6]. This kernel is defined as 0 if both trails have not the same size and as follows otherwise:

$$K_{trail}(t, t') = \delta(\varphi(v_1), \varphi(v'_1))\prod_{i=2}^{|t|}\delta(\psi(e_{v_{i-1}v_i}), \psi(e_{v'_{i-1}v'_i}))\delta(\varphi(v_i), \varphi(v'_i)), \tag{7}$$

where $\varphi(v)$ and $\psi(e)$ denote respectively the labels of vertex v and edge e while δ is equal to 1 if its both arguments are equal and 0 otherwise. The kernel K_{trail} is definite positive as a tensor product of definite positive kernels.

Laplacian Kernel The graph edit distance between two graphs is defined as the minimal number of vertices and edge removal/addition/relabeling required to transform one graph into an other [9]. The computational complexity of the exact computation of the edit distance is exponential in the number of vertices. Fortunately, Riesen and Bunke [11] proposed an heuristic to efficiently approximate the edit distance between two graphs.

Unfortunately, the edit distance does not define a metric and, as a consequence, kernels directly based on the edit distance does not define a valid kernel.

Let us consider a set of input graphs $\{G_1, \ldots, G_n\}$ defining our graph test database. Given a kernel k, the gram matrix K associated to this database is an $n \times n$ matrix defined by $K_{i,j} = k(G_i, G_j)$. As denoted by Steinke [15], the inverse of K (or its pseudo inverse if K is not invertible) may be considered as a regularization operator on the set of vector of dimension n. Conversely, the inverse (or pseudo inverse) of any definite positive regularization operator may be considered as a kernel. More precisely, within an interpolation or classification scheme, the optimal mapping of a real value f^* to each graph $\{G_1, \ldots, G_n\}$ which interpolates a vector of know values y while minimizing the norm induced by K in $\mathbb{R}^n$ is achieved by solving the following minimization problem:

$$f^* = \arg min_{f \in \mathbb{R}^n} CLoss(f, y, K) + f^t K^{-1} f$$

where $CLoss(\cdot, \cdot, \cdot)$ denotes a given loss function encoding the distance between vector f and the vector of known values y.

One scheme to design a kernel consists thus to first build a definite positive regularization operator and then to take its inverse (or its pseudo inverse) to obtain a kernel. Having an interpolation problem in mind, we would like that our regularization operator penalizes different values of f^* mapped onto graphs with a small edit distance. We thus consider the Laplacian operator defined as follows: given the set of graphs, $\{G_1, \ldots, G_n\}$, we first consider the $n \times n$ adjacency matrix $W_{i,j} = e^{-\frac{d(G_i, G_j)}{\sigma}}$ where σ is a tuning variable and $d(\cdot, \cdot)$ denotes the edit distance [11]. The normalized Laplacian of $\{G_1, \ldots, G_n\}$ is then defined as $L = I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$ where D is a diagonal matrix defined by $D_{i,i} = \sum_{j=1}^{n} W_{i,j}$.

Well know results from spectral graph theory establish that L is a symmetric, semi definite positive matrix whose eigenvalues belongs to $[0, 2]$. Unfortunately, the Laplacian is also well know for being non invertible. The only semi definite property of the Laplacian matrix forbids a direct inversion of this matrix. Moreover, the pseudo inverse of the Laplacian induces numerical instabilities which does not lead to a reliable kernel. Therefore, following Smola [14], we rather regularize the spectrum of L. The regularized version of L, denoted as $\tilde{L}$, is defined as :

$$\tilde{L} = \sum_{i=1}^{n} r(\lambda_i) u_i u_i^t \tag{8}$$

where $\{\lambda_i, u_i\}$ denotes the eigensystem of L while $r(\cdot)$ is a regularization function. Many regularization functions may be used, however we restricted our choice to Regularized Laplacian ([14]): $r(\lambda) = 1 + \eta\lambda$ where η is a tuning variable. The regularized laplacian $\tilde{L}$ is invertible and its inverse $K = \tilde{L}^{-1}$ is taken as a kernel.

4 Experimental Evaluation

The application used for the comparison between the two approaches is the Predictive Toxicology Challenge.

4.1 The Application Domain and the Test Sets

Nowadays, almost every human being in industrialized societies is exposed to chemical compounds whose long-term effects are not evident. The carcinogenicity of such compounds, for instance, is often unknown. The US National Toxicology Program (NTP) [5] conducted several experimentations on the effects of many chemical compounds on mice and rats, but such empirical results are expensive and very slow to obtain. Hence the necessity to find carcinogenicity models able to generate reliable toxicity predictions for other compounds. The Predictive Toxicology Challenge (PTC) [6] was initiated in this aim. It enabled participants to submit machine learning programs which, from the large training datasets of the NTP, provide carcinogenicity predictions of compounds in test sets. According to the Structure Activity Relationship (SAR) hypothesis, which assume that the activity of a molecule can be entirely determined by its structure, the only informations used for toxicity prediction of a molecule are related to its chemical structure.

Four distinct training and test sets were provided (see Table 1), since mice and rats reactions to carcinogens compounds are significantly different. In this paper we consider all the provided datasets using, for both the methods, the following representation: each chemical compound is represented by a graph $G(V, E, \alpha, \beta)$ where V represents the set of atoms, E is the set of edges representing bonds, α is the nodes labeling function that associates an atom symbol to each node of the graph and β is the edges labeling function that associates a bond multiplicity to each edge of the graph. The separation of the sets in training and test set was made directly by the PTC committee, therefore no cross-validation in training phase is allowed.

Table 1. Number of Chemical Compounds in the sets used for the experimentation. The chemical compounds are separated in 4 sets according to the reactions of: Female Mice (FM), Male Mice (MM), Female Rats (FR) and Male Rats (MR)

Dataset		FM	MM	FR	MR
Training Set	Cancerogenic	78	67	62	70
	Non-Cancerogenic	124	123	140	124
Test Set	Cancerogenic	35	28	35	50
	Non-Cancerogenic	43	36	63	55

[5] National Toxicology Program: http://ntp.niehs.nih.gov/
[6] http://www.predictive-toxicology.org/

4.2 Evaluation and Comparison

In Table 2 we report the comparative results obtained on FM, MM, FR and MR datasets by the two different approaches (the classification by Graph Kernels is made by means of SVM classifier).

The three methods (two methods from the Graph Kernels) have comparable results. However the time required, in the training phase, by Symbolic Learning method is very high.

We can conclude that in problems that have strict time constraints Graph Kernels method is very effective while Symbolic Learning can be used only on smaller problems, or where there are less time constraints. On the other hand Symbolic Learning generates explicit, declarative knowledge on the classes in the form of GARGs that may be understood with little effort by a human expert: a direct inspection of such explicit prototypes can be sufficient to validate them or possibly to delete the badly formed prototypes that could be detrimental for classification performance. The Graph Kernel approach, instead, is able to perform a classification, but not to explain the criteria that guided its choice.

Table 2. Accuracy and Elapsed Time (of the training phase) for the two methods on the FM, MM, FR and MR datasets

Method	FM	MM	FR	MR	Elapsed Time
Symbolic Learning	60.4%	52.1%	62.0%	62.0%	$\sim$ 2000 min
Bag of trails Kernel	61.2%	50.4%	62.8%	65.6%	$\sim$ 2 min
Laplacian Kernel	59.2%	55.2%	57.7%	60.9%	$\sim$ 1 min 30 sec

In conclusion the results confirm that there is no way to decide what is the best method because both approaches have their peculiar virtues and deficiencies.

In our view, the two approaches should not be seen as competitors; instead, only their cooperative integration can provide us with more powerful tools for knowledge exploitation. The first step toward the integration of the two methodologies could be their combined use in a multiexpert system [5]. As widely demonstrated in the literature, an accurate selection of the combining criteria allows us to significantly improve the results obtained by the single experts by retaining their strengths, while overcoming their weaknesses. We made a preliminary evaluation in this direction: the results show that the two systems exhibit a certain degree of orthogonality in their errors: that is, some of the samples misrecognized by one of the algorithms are correctly classified by the other one.

5 Conclusions

In this paper we presented a comparison between two approaches, Graph Kernels and Symbolic Learning, within a classification scheme. Our experiments

indicate that both approaches are comparable in terms of accuracy. Graph kernel methods require less execution times but does not provide an explanation of the classification criteria. Symbolic Learning, on the other hand have the reverse advantages and drawbacks. Both methods are thus clearly complementary.

Future steps, as we discussed in the last section, could be an integration of the two approaches in order to improve the results obtained by the application of the single approaches.

Acknowledgments. We would like to express our thanks to Alice Kijewski and David Lemaresquier for their support in the experimental phase.

References

1. Bunke, H., Riesen, K.: A family of novel graph kernels for structural pattern recognition. In: XII Iberoamerican Congress on Pattern Recognition. pp. 20–31 (2007)
2. Dupé, F.X., Brun, L.: Tree covering within a graph kernel framework for shape classification. In: XV ICIAP (2009)
3. Foggia, P., Genna, R., Vento, M.: Symbolic vs connectionist learning: an experimental comparison in a structured domain. IEEE Transaction on Knowledge and Data Engineering 13–2, 176–195 (2001)
4. Haussler, D.: Convolution kernels on discrete structures. Tech. rep., Department of Computer Science, University of California at Santa Cruz (1999)
5. Iannello, G., Percannella, G., Sansone, C., Soda, P.: On the use of classification reliability for improving performance of the one-per-class decomposition method. Data & Knowledge Engineering 68, 1398–1410 (2009)
6. Kashima, H., Tsuda, K., Inokuchi, A.: Marginalized kernel between labeled graphs. In: In Proc. of the Twentieth International conference on Machine Learning (2003)
7. Mahé, P., Vert, J.P.: Graph kernels based on tree patterns for molecules. Machine Learning 75(1), 3–35 (2008)
8. Michalski, R.: Pattern recognition as rule-guided inductive inference. IEEE Transaction on PAMI 2–4, 349–361 (1980)
9. Neuhaus, M., Bunke, H.: Bridging the Gap Between Graph Edit Distance and Kernel Machines. World Scientific Publishing Co., Inc., River Edge, NJ, USA (2007)
10. Nishida, H.: Shape recognition by integrating structural descriptions and geometrical/statistical transforms. CVIU 64, 248–262 (1996)
11. Riesen, K., Bunke, H.: Approximate graph edit distance computation by means of bipartite graph matching. Image Vision Computing 27(7), 950–959 (2009)
12. Rocha, J., Pavlidis, T.: A shape analysis model with applications to a character recognition system. IEEE Transaction on PAMI 16–4, 393–404 (1994)
13. Shervashidze, N., Vishwanathan, S.V., Petri, T.H., Mehlhorn, K., Borgwardt, K.M.: Efficient graphlet kernels for large graph comparison. In: Twelfth International Conference on Artificial Intelligence and Statistics (2009)
14. Smola, A.J., Kondor, R.I.: Kernels and regularization on graphs. In: XV Annual Conference on Learning Theory. pp. 144–158 (2003)
15. Steinke, F., Schölkopf, B.: Kernels, regularization and differential equations. Pattern Recognition 41(11), 3271 – 3286 (2008)
16. Suard, F., Rakotomamonjy, A., Bensrhair, A.: Kernel on bag of paths for measuring similarity of shapes. In: European Symposium on Artificial Neural Networks. Bruges-Belgique (April 2007)