Neurocomputing 113 (2013) 177–194

Contents lists available at SciVerse ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Graph matching and clustering using kernel attributes

Miguel Angel Lozano *, Francisco Escolano

Departamento de Ciencia de la Computación e Inteligencia Artificial, Universidad de Alicante, Spain

ARTICLE INFO

Article history:
Received 26 December 2011
Received in revised form
6 November 2012
Accepted 3 January 2013
Communicated by Weifeng Liu
Available online 27 February 2013

Keywords:
Graph algorithms
Pattern matching
Clustering
Computer vision

ABSTRACT

In this paper, we exploit graph kernels for graph matching and clustering. Firstly, we analyze different kinds of graph kernels in order to extract from them attributes to be used as a similarity measure between nodes of non-attributed graphs. Next, such attributes are embedded in a graph-matching cost function, through a probabilistic framework, and we evaluate their performance within a graph-matching algorithm. Secondly, we propose a method for obtaining a representative prototype from a set of graphs, which relies on obtaining all the pairwise matchings between the input graphs, and uses the information provided by graph kernels in order to select the matchings that will be considered for obtaining the prototype. Nodes and edges in such a prototype graph register their frequency of occurrence, so that it can be considered a first-order generative model. The proposed method for building prototypes is efficiently integrated into a central clustering algorithm, which allows us to unsupervisedly learn the class-structure of a given set of graphs, and the prototypes representing each class, thus obtaining a central graph clustering algorithm with the same computational cost than a pairwise one. We successfully apply the proposed methods to structural recognition problems.

1. Introduction

Graph plays an important role in pattern recognition to encode the significant structural/relational information of networks or patterns, yielding useful invariance for object recognition. In computer vision, graphs and the particular case of trees, are commonly used to represent the shape of the objects, such as *shock trees* [1] and *reeb graphs* [2]. Other approaches use graphs to represent the structure of the whole scene [3]. These graph representations can be used to classify the objects found in an image following a structural criteria.

Structural criteria, graph matching, and graph learning have been considered fundamental elements in the setup of the constellation approach to object recognition [4]. Most of the research in this area has been addressed to exploit local statistics, whereas global statistics have been typically confined to the joint Gaussian of features [5]. However, there has been recent interest in modeling and learning structural relationships, such as *tree-structured models* [6,7] and the *k-fans* graph model [8]. In a previous work [9], we address the problem of comparing and classifying proteins with graph-based methods, relying on graphs extracted from protein surfaces that encode the structure of their concave and convex patches.

Our main goal in this paper is to unsupervisedly learn the prototypes describing a set of non-attributed graphs, with no more information than the structure itself. Previous graph clustering approaches have mainly dealt with attributed relational graphs (ARGs), such as random graphs [10–12], the function-described graph (FDG) model [13], and hierarchical random graphs (HRG) [14]. However, when there are no attributes and only structural information is available, these previous approaches cannot be applied.

The clustering problem can be posed either as central or pairwise clustering. Here, we follow central graph clustering [15–17] because of its convenience for developing structural generative models, a task that has been more addressed in the context of trees [18,19] than in the case of general graphs. Structural generative models allow us to generate random observable graphs belonging to a given class, by sampling different parts of the structure. A first-order model only considers nodes and edges individually in the sampling process, usually registering the probability of occurrence of each of them. A n-order model also considers the occurrence frequency of substructures of n nodes.

The key of central graph clustering is to have a strategy for learning automatically the prototype of each class for further analysis. Such task has been addressed by other researchers [18,20,21]. There exist methods for building these representative graphs through either incremental or hierarchical strategies [22,23,9], which use the common labeling between pairs of graphs provided by graph-matching to update the probabilities

* Corresponding author. Tel.: 34 965 90 3400x1248; fax: 34 965 90 3902.
E-mail addresses: malozano@ua.es (M.A. Lozano), sco@dccia.ua.es (F. Escolano).

0925-2312/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.neucom.2013.01.015

of the nodes and edges of the prototypes. In the incremental approach, temporary prototypes are updated sequentially, whereas in the hierarchical approach pairs with minimal distance are successively merged yielding a dendogram.

In this paper, we present an alternative *prototype building* method independent with respect to the order in which the graphs are presented. This approach is neither incremental nor hierarchical, the whole set of graphs is used simultaneously for generating the prototype. It is based on the network of pairwise matchings between all the graphs in the input set, which will be referred as *super-graph*. All the nodes (edges) matched between them correspond to a single node in the prototype. Each node (edge) in the prototype registers its frequency of occurrence, that is, the number of graphs in the input set containing this node (edge), so that it can be considered a first-order generative model. This method relies only on graph matching, so that is can be applied to non-attributed and non-labeled graphs.

However, graph matching is ambiguous (many alternative graph or subgraph isomorphisms may be found) and inaccurate, specially when attributes and labeling are missing. To overcome this problem we endow each node with additional information used to disambiguate possible alternative matchings when building the prototype. Our hypothesis is that a good selection of topological attributes may improve both matching and clustering. We address the key point of extracting attributes for nodes of non-attributed graphs by exploiting recent theoretical results in spectral graph theory [24]: the definition of diffusion kernels on graphs [25] and their generalization to other families of kernels [26]. These latter works have transferred to the discrete domain of graphs the concept of a *kernel*, originally defined in the vector domain [27]. Kernels are key concepts in the context of statistical learning theory [28–30], which capture the structure of a domain by defining a similarity measure between its elements, relying on the inner product of the results of mapping both inputs to a, usually higher dimensional, Hilbert space.

The space of feasible solutions for graph matching is a discrete space, in which each possible solution consists of a permutation matrix that establishes a mapping between nodes of two graphs. We need a cost function to evaluate these possible solutions, and a method to optimize such a function. Although the space of solutions is a discrete space, there exist both discrete and continuous optimization methods for graph matching.

Regarding discrete optimization methods, Ullman presented in [31] an algorithm for obtaining an isomorphism between a graph and a subgraph of another graph. More recently, Cordella et al. proposed in [32] a more efficient algorithm, addressed to be used with bigger graphs. Among these discrete approaches, we also find different studies applying genetic algorithms [33,34], simulated annealing [35] and tabu search [36] to inexact graph matching. Discrete approaches, due to the huge extension of the search space, are less efficient than the continuous ones. For this reason, we will focus on optimizing the results provided by continuous methods.

Continuous methods rely on transforming the discrete search space into a continuous one and then exploiting optimization techniques to find a, typically approximate, solution. One of the first algorithms is Softassign, the well-known graduated assignment method introduced by Gold and Rangarajan [37]. It is a typical energy minimization approach for graph matching, which relies on optimizing a quadratic cost function through a polynomial computational complexity process in order to find a solution. However, it has been reported that the performance of the algorithm decays significantly at mid and high levels of structural corruption, and also that such a decay can be attenuated by optimizing an alternative non-quadratic energy function [38]. Instead of using a non-quadratic cost function, we propose to apply a proper weighting to the quadratic Gold and Rangarajan cost function, by using the information provided by kernels.

An alternative continuous formulation for graph-matching relies on the Motzkin–Strauss theorem [39]. The Motzkin–Strauss cost function is based on the maximum clique formulation. As in the case of Gold and Rangarajan optimization (Softassign), methods like replicator dynamics start at a neutral (barycenter) configuration and progress until stabilization. The main drawback of continuous steepest descent graph-matching methods is that they are very prone to local minima, specially when ambiguity exists. This makes kernel attributes specially useful to improve this kind of methods.

Other approaches for graph matching exploit the spectral information [40] from the graph. The main eigenvector is related to the steady state random walk along the nodes of the graph. Thus, it allows us to serialize the graph as a sequence invariant to permutations in the adjacency matrix. These sequences can be used to compare graphs by means of the string edit distance between them [41]. However, there also exist other approaches that apply edit distance directly to graph structures [42,43]. Such a distance is related to the maximum common subgraph (MCS) [44]. In [45] the graph spectrum is used in order to obtain a set of spectral features that allows us to invariantly identify the graph by a fixed-size feature vector. More recently, in [46] it is proposed another similarity measure between nodes of two graphs relying on their commute times, and in [47] we proposed a graph matching method based on the commute time embedding. Some spectral methods obtain a solution efficiently, but they are too sensitive to the loss of notable (highly connected) nodes [48]. Spectral attributes (extracted from the main eigenvector) are compared in this paper to kernel attributes, to test their performance for identifying node matchings.

The paper is organized as follows. In Section 2, we present some previous definitions and concepts used in the paper. Next, in Section 3, we propose a method for extracting attributes from graph kernels. A probabilistic framework is introduced to test different types of attributes, and it is applied to evaluate the proposed attributes. Next, in Section 4 we show how these attributes can be incorporated into cost functions of existing graph matching methods and how to use the result provided by these methods as a similarity measure between graphs. Then, in Section 5, we present our method for obtaining graph prototypes relying on all the pairwise matchings between the input graphs and their kernel divergences, and we show how this method can be efficiently integrated into a central clustering algorithm. Finally, in Section 6 we perform a set of matching and clustering experiments applied to object recognition with non-attributed graphs, and in Section 7 we present our final conclusions and future work.

2. Definitions and concepts

A non-attributed graph is a 2-tuple $G=(V,E)$, where V is a set of nodes of size $m>0$, and $E\subseteq\{(i,j)|(i,j)\in V\times V,\ i\neq j\}$ is a set of edges.

An attributed graph is a 3-tuple $G=(V,E,\mathcal{A})$ where $\mathcal{A}: V\rightarrow\mathbb{R}^n$ is a function that assigns an attribute (e.g. a feature vector) to each node.

The adjacency matrix of an attributed or non-attributed graph G is defined by

$$A_{ij}=\begin{cases}1 & \text{if } (i,j)\in E,\\ 0 & \text{otherwise}\end{cases}$$

and its diagonal degree matrix is defined by

$$D_{ij}=\begin{cases}\sum_{k=1}^{m}A_{ik} & \text{if } i=j\\ 0 & \text{otherwise.}\end{cases}$$

The Laplacian of G is defined as $L=D-A$, that is,

$$L_{ij}=\begin{cases}-1 & \text{if } (i,j)\in E,\\ D_{ii} & \text{if } i=j,\\ 0 & \text{otherwise}\end{cases}$$

and its normalized Laplacian [24] $\tilde{L}=D^{-1/2}LD^{-1/2}=I-D^{-1/2}XD^{-1/2}$ is defined as

$$\tilde{L}_{ij}=\begin{cases}-D_{ii}^{-1/2}D_{jj}^{-1/2} & \text{if } (i,j)\in E,\\ 1 & \text{if } i=j \text{ and } D_{ii}\neq 0,\\ 0 & \text{otherwise.}\end{cases}$$

The Laplacian and its normalized version can be used to obtain different types of graph kernels. When applied to graphs, kernels provide a similarity measure between vertices of the same graph. Within this type of kernels we find a family of regularization kernels proposed by Smola and Kondor [26], and the particular case of the diffusion kernel, defined by Kondor and Lafferty [25].

2.1. Diffusion kernels

Following [25] the diffusion kernel K associated to a graph G is the result of the exponentiation of its Laplacian matrix L. Using the Taylor expansion at the exponential we have

$$K=e^{-\beta L}=I_m+\beta L+\frac{\beta^2}{2!}L^2+\frac{\beta^3}{3!}L^3+\cdots, \tag{1}$$

where I_m is the $m\times m$ identity matrix, and β is a control parameter that controls the time of the diffusion process (i.e. $\beta=0$ means that no diffusion is performed, in this case the kernel is the identity matrix). Moreover, $e^{-\beta L}$ is the solution of the heat equation [24]

$$\frac{d}{d\beta}K_\beta=-LK_\beta. \tag{2}$$

As L is symmetric, the solution $K=e^{-\beta L}$, the Gram matrix, satisfies the positive semi-definiteness condition for kernels. On behalf of the so-called *kernel trick*, the $m\times m$ matrix K defines a real-valued function between pairs of vertices, and K_{ij} can be interpreted as the inner product of the mappings of both vertices to a Hilbert space [29]. This means that such an inner product encodes the similarity between pairs of vertices in a possibly high-dimensional space. But, from the point of view of discrete structures what is interesting of such similarity is that as L encodes the local structure of V in G, the global structure emerges in K.

More precisely, and due to the fact that K is the solution of the heat equation, the diffusion kernel K is the version for discrete spaces of the Gaussian kernel for $\mathbb{R}^m$ with variance $\sigma^2=2\beta$, that is, the value of K_{ij} decays exponentially with the *distance* between i and j.

This framework is generalized in [26] where a family of graph kernels is proposed in the context of regularization.

2.2. Regularization kernels

These kernels are derived from studying the usefulness of the Laplacian L of a graph (and its normalized version $\tilde{L}$) as a smoothing operator. Considering smoothing operators from a spectral point of view, a family of kernels can be obtained by applying different penalization functions. It can be proved that the inverse of the so-called *regularization matrix* for each element of the family yields a kernel (actually, the diffusion kernel belongs to this family).

The connection of the latter Laplacian matrices with regularization theory is due to the fact that given a real-valued function f defined over the vertices of G, that is, $f:V\to\mathbb{R}$ both, L and $\tilde{L}$ can be seen as discrete differential operators which tend to penalize changes of f between adjacent edges. Considering now f a column vector, that is $f\in\mathbb{R}^m$, the following inner product is a measure of the smoothness of f over the graph G:

$$\langle f,Lf\rangle=f^TLf=\sum_{i=1}^{m}f_i^2D_{ii}-\sum_{(i,j)\in E}2f_if_j=\sum_{(i,j)\in E}(f_i-f_j)^2.$$

An alternative way of formulating regularization is through spectral analysis. In [49], Smola et al. established the connection between regularization, Fourier analysis and kernels in continuous spaces. A smoothness operator in Fourier space can be built by multiplying the Fourier transform by a penalizing function increasing in frequency. As such a multiplication in Fourier space becomes the application of the latter function on the continuous Laplacian operator, an spectral-based regularization operator in graphs comes from

$$\langle f,r(\mathcal{L})f\rangle=f^T\underbrace{\left[\sum_{i=1}^{m}r(\lambda_i)\phi_i\phi_i^T\right]}_{r(\mathcal{L})}f=\sum_{i=1}^{m}\langle f,\phi_i\rangle r(\lambda_i)\langle\phi_i,f\rangle,$$

where $\mathcal{L}$ denotes both L and $\tilde{L}$, $\{\lambda_i,\phi_i\}$ are the eigenvalues and eigenvectors of $\mathcal{L}$, and $r(\lambda_i)$ is a monotone increasing function. Actually, $r^{-1}(\lambda)$, the inverse of such a function is the Fourier transform of the associated kernel in the continuous case, and the discrete regularization kernel K is the inverse (or the pseudo-inverse if necessary) of the so-called *regularization matrix* $r(\mathcal{L})$. Then we have that

$$K=r^{-1}(\mathcal{L})\qquad\text{where}\quad r^{-1}(\mathcal{L})=\sum_{i=1}^{m}r^{-1}(\lambda_i)\phi_i\phi_i^T, \tag{3}$$

considering $0^{-1}\equiv 0$. For instance, in the particular case of the diffusion kernel, which relies on *matrix exponentiation* but not on component-wise exponentiation, we have that

$$K=e^{-\beta\mathcal{L}}=\left(\sum_{i=1}^{m}e^{\beta\lambda}\phi_i\phi_i^T\right)^{-1}=\sum_{i=1}^{m}e^{-\beta\lambda}\phi_i\phi_i^T.$$

In the general case, the relation $K=r^{-1}(\mathcal{L})$ is derived from the fact that given a regularization operator, for instance $M=r(\mathcal{L})$, the matrix K must satisfy the *self-consistency condition* $KMK=K$ to be a kernel, and therefore $K=M^{-1}$ or equal to the pseudo-inverse if M is not invertible. Furthermore, it can be proved [26] that such regularization operator defines a reproducing kernel Hilbert space whose kernel is $K=M^{-1}$. In Table 1 we show several penalization functions and their associated regularization kernels. It is worth to highlight that choosing $\tilde{L}$ yields a spectrum contained in [0,2].

Table 1
Penalization functions and regularization kernels.

$r(\lambda)$	$K=r^{-1}(\mathcal{L})$	Name
$1+\beta\lambda$	$(I+\beta\mathcal{L})^{-1}$	Regularized Laplacian
$e^{\beta\lambda}$	$e^{-\beta\mathcal{L}}$	Diffusion process
$(a-\lambda)^{-p}$	$(aI-\mathcal{L})^p$	p-Step random walk
$(\cos\lambda\pi/4)^{-1}$	$\cos\mathcal{L}\pi/4$	Inverse cosine

3. Kernel attributes

3.1. Extracting node attributes from kernels

From the point of view of random fields, the diffusion kernel K relies on the covariance matrix of a stochastic process in which each vertex has attached a random variable of zero mean and variance σ^2, and each variable sends a small fraction of its value (heat) to its neighbors. In this regard, K_{ij} can be interpreted as the amount of heat accumulated at vertex j after a given amount of time after injecting heat at i and let it diffuse through the edges of the graph. The more *distant* are i and j the less heat we have.

In terms of random walks, K_{ij} can be regarded as the sum of probabilities that a *lazy* random walk takes each path from i to j [25]. Given a pair of vertices, for instance i,j of graph G_X, its kernel K induces the following probability distribution:

$$p_{ij}^X = K_{ij}\left(\sum_{k=1}^{m} K_{ik}\right)^{-1}.$$

A lazy random walk over the undirected graph G, and with parameter β, is a stochastic process which will take each of the edges emanating from i with a fixed probability β and will remain in i with probability $1-\beta D_{ii}$, with β being at most $1/(max_i\, D_{ii})$. From this point of view, the final value of K_{ij} depends on the edge distribution and branching process between i and j. If j is an isolated node, then $K_{ij}=0\ \forall i\neq j$ and $K_{jj}=1$.

The i-th element in the kernel diagonal i (K_{ii}) represents the probability p_{ii}^X that a lazy random walk starting at node i ends at the same node. That is, this value encodes the structure around each node.

Although understanding the role of the diffusion kernel in graphs is intuitive because these kernels are the discrete version of continuous Gaussian kernels, this is not the case for the regularization kernels. When using kernels it is unknown beforehand which is the best choice, so it is necessary to test all the proposed kernels. All these kernels define a dissimilarity measure between nodes, and the self-dissimilarity K_{ii} will be used in all cases as node attribute $\mathcal{A}_i^{kernel}=K_{ii}$.

3.2. Probabilistic framework

Our aim is to obtain from the proposed attributes the probability that two nodes i,j (belonging to graphs G_X and G_Y respectively) match, given the observation of their kernel attributes for both nodes ($\mathcal{A}_i^X=K_{ii}^X$ and $\mathcal{A}_j^Y=K_{jj}^Y$, being K^X and K^Y the kernels of G_X and G_Y respectively). Following Bayes' rule we have:

$$P(match|\mathcal{A}_i^X,\mathcal{A}_j^Y)=\frac{P(\mathcal{A}_i^X,\mathcal{A}_j^Y|match)P(match)}{P(\mathcal{A}_i^X,\mathcal{A}_j^Y)}, \tag{4}$$

where $P(\mathcal{A}_i^X,\mathcal{A}_j^Y|match)$ is the likelihood, $P(match)$ is the prior (probability that any two nodes match), and $P(\mathcal{A}_i^X,\mathcal{A}_j^Y)$ is the probability of observing a given kernel value. We define the prior as follows:

$$P(match)=\frac{(1-r)m}{nm}=\frac{(1-r)}{n}, \tag{5}$$

where m and n are the number of nodes of the graphs being matched, considering that $n<m$, and $r\in[0,1]$ is a parameter that models the node noise fraction between both graphs, that is, the fraction of nodes in the biggest graph not matched with any node of the other graph. This parameter is problem-dependent, and allows us to tackle the amount of noise we expect to find in the input set. From this probability, we define the probability that

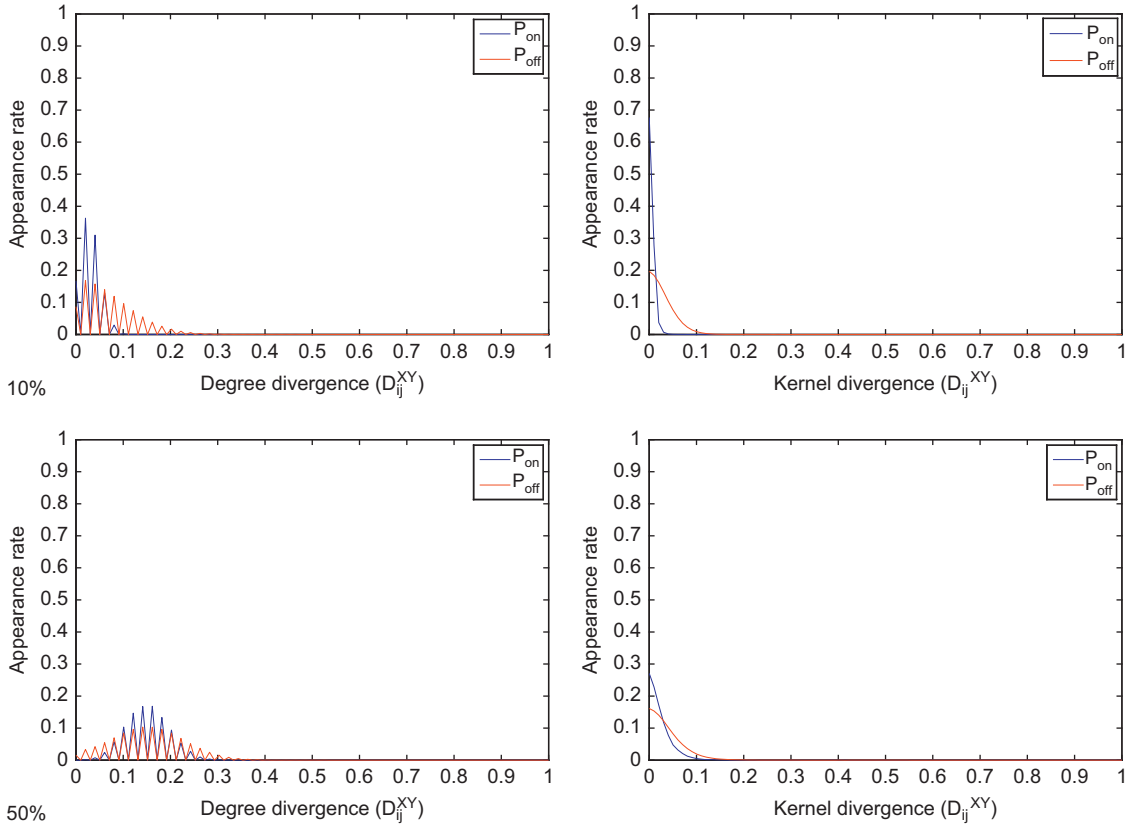

Fig. 1. Examples of P_{on} and P_{off} distributions for randomly generated graphs with 30% edge density. Attributes extracted from degree (left) and regularized Laplacian kernel (right), with 10% (top) and 50% (bottom) node noise.

two nodes do not match as follows:

$$P(\overline{match}) = 1 - \frac{(1-r)}{n} = \frac{n+r-1}{n}. \quad (6)$$

The probability of observing a given pair of attributes can be obtained as

$$P(\mathcal{A}_i^X, \mathcal{A}_j^Y) = P(\mathcal{A}_i^X, \mathcal{A}_j^Y | match)P(match) + P(\mathcal{A}_i^X, \mathcal{A}_j^Y | \overline{match})P(\overline{match}) \quad (7)$$

The likelihoods $P(\mathcal{A}_i^X, \mathcal{A}_j^Y | match)$ and $P(\mathcal{A}_i^X, \mathcal{A}_j^Y | \overline{match})$ could be defined following the empirical Bayesian approach for feature extraction [50] in computer vision, also applied for junction detection [51]. We define the probability distribution P_{on} of observing a particular divergence between the kernel attributes of two nodes, given that these nodes match

$$P(\mathcal{A}_i^X, \mathcal{A}_j^Y | match) = P_{on}(\mathcal{D}_{ij}^{XY}), \quad (8)$$

where $\mathcal{D}_{ij}^{XY}$ indicates the divergence between the attributes $\mathcal{A}_i^X$ and $\mathcal{A}_j^Y$, and it is defined as

$$\mathcal{D}_{ij}^{XY} = |\mathcal{A}_i^X - \mathcal{A}_j^Y|. \quad (9)$$

Similarly, we define P_{off} as the probability distribution of observing a particular divergence between the attributes of two nodes, given that these nodes do not match. Then, we can rewrite the probability of observing a given pair of attributes as

$$P(\mathcal{A}_i^X, \mathcal{A}_j^Y) = P_{on}(\mathcal{D}_{ij}^{XY})P(match) + P_{off}(\mathcal{D}_{ij}^{XY})P(\overline{match}). \quad (10)$$

We can observe several examples of P_{on} and P_{off} distributions in Fig. 1. These distributions have been obtained empirically for different kinds of attributes and different rates of noise. Graphs in our experiments are generated by randomly adding edges until the desired edge density is reached. They can then be corrupted with a given percentage of node or edge noise. Edge noise is introduced by randomly removing the indicated percentage of edges from the original graph, whereas node noise is introduced by randomly removing the indicated percentage of nodes and all the edges connected to them.

If we know before-hand the kind of graphs we may find as input, we can generate P_{on} and P_{off} empirically from a sample of these graphs, in which we have to set manually which is the correct matching. With the latter information we can obtain the posterior probability that two nodes match given the divergence of their attributes as

$$\begin{aligned} P(match | \mathcal{A}_i^X, \mathcal{A}_j^Y) &= \frac{P(\mathcal{A}_i^X, \mathcal{A}_j^Y | match)P(match)}{P(\mathcal{A}_i^X, \mathcal{A}_j^Y)} \\ &= \frac{P_{on}(\mathcal{D}_{ij}^{XY})P(match)}{P_{on}(\mathcal{D}_{ij}^{XY})P(match) + P_{off}(\mathcal{D}_{ij}^{XY})P(\overline{match})} \\ &= \frac{1}{1 + \frac{P_{off}(\mathcal{D}_{ij}^{XY})P(\overline{match})}{P_{on}(\mathcal{D}_{ij}^{XY})P(match)}} \\ &= \frac{1}{1 + \frac{P_{off}(\mathcal{D}_{ij}^{XY})}{P_{on}(\mathcal{D}_{ij}^{XY})}\left(\frac{m}{1-r} - 1\right)}. \end{aligned}$$

If there is not available any knowledge about the expected input graphs, then P_{on} and P_{off} cannot be obtained. In this case, we may suppose a negative exponential distribution decaying with $\mathcal{D}_{ij}^{XY}$ as a standard model

$$P(match | \mathcal{A}_i^X, \mathcal{A}_j^Y) = e^{-\mathcal{D}_{ij}^{XY}}. \quad (11)$$

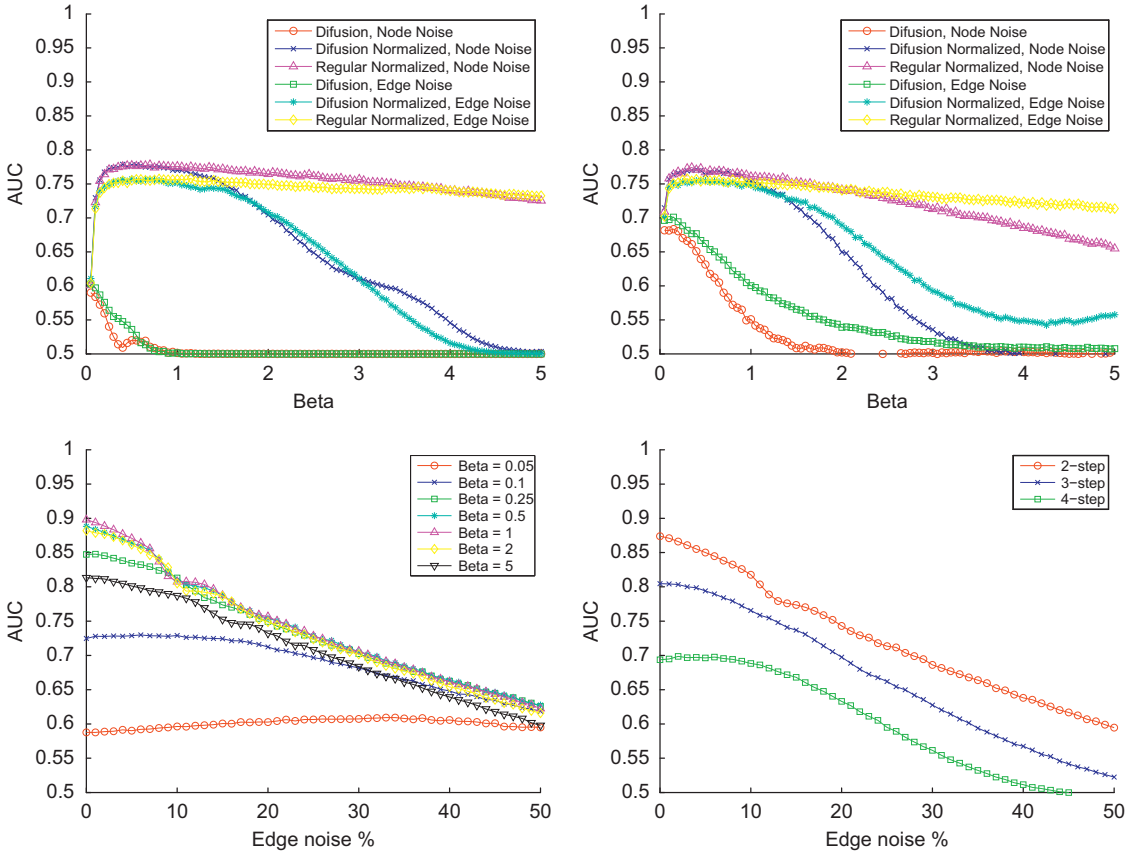

Fig. 2. Top: AUC vs β, for 50-node (left) and 20-node (right) graphs, with 30% edge density and 20% of node and edge noise. Bottom: AUC vs noise for different values of β for the regularized Laplacian kernel (left), and for different values of p for p-step (right), with a 30% of edge density.

This model will be used to introduce the information provided by kernels, and other kind of attributes, into the cost function of different graph matching algorithms. It will also be used in the algorithm for building graph prototypes.

3.3. Evaluating kernel attributes

In order to evaluate the performance of kernel attributes for testing whether two nodes should match we are going to analyze P_{on} and P_{off} distributions, obtained empirically, by means of ROC curves. To simplify, we will only represent the area under the curve (AUC) value. For each experiment we have performed 1000 tests, and we show the average AUC value for each case.

Some of the proposed kernel attributes depend on a parameter, that has to be adjusted to its optimal values. This is the case of diffusion and regularized Laplacian kernels, which depends on a β parameter. Fig. 2 (top) shows AUC vs β parameter, for randomly generated 20-node (left) and 50-node (right) graphs.

Regularization kernels can be obtained from the Laplacian or from its normalized version. It is interesting to highlight that the latter Laplacian performs better in all cases. This is due to the fact that the spectrum of this matrix is bounded. An unbounded spectrum yields high eigenvalues for large graphs, and this causes their corresponding eigenvectors not to be considered in the kernel computation (the higher the eigenvalue, the lower the weight of its corresponding eigenvector, see Eq. (3)). The most stable results are provided by the regularized Laplacian kernel.

We can observe how different β values behave as we increase the amount of edge noise for the regularized Laplacian kernel in Fig. 2 (bottom-left). As noise increases, the results obtained with

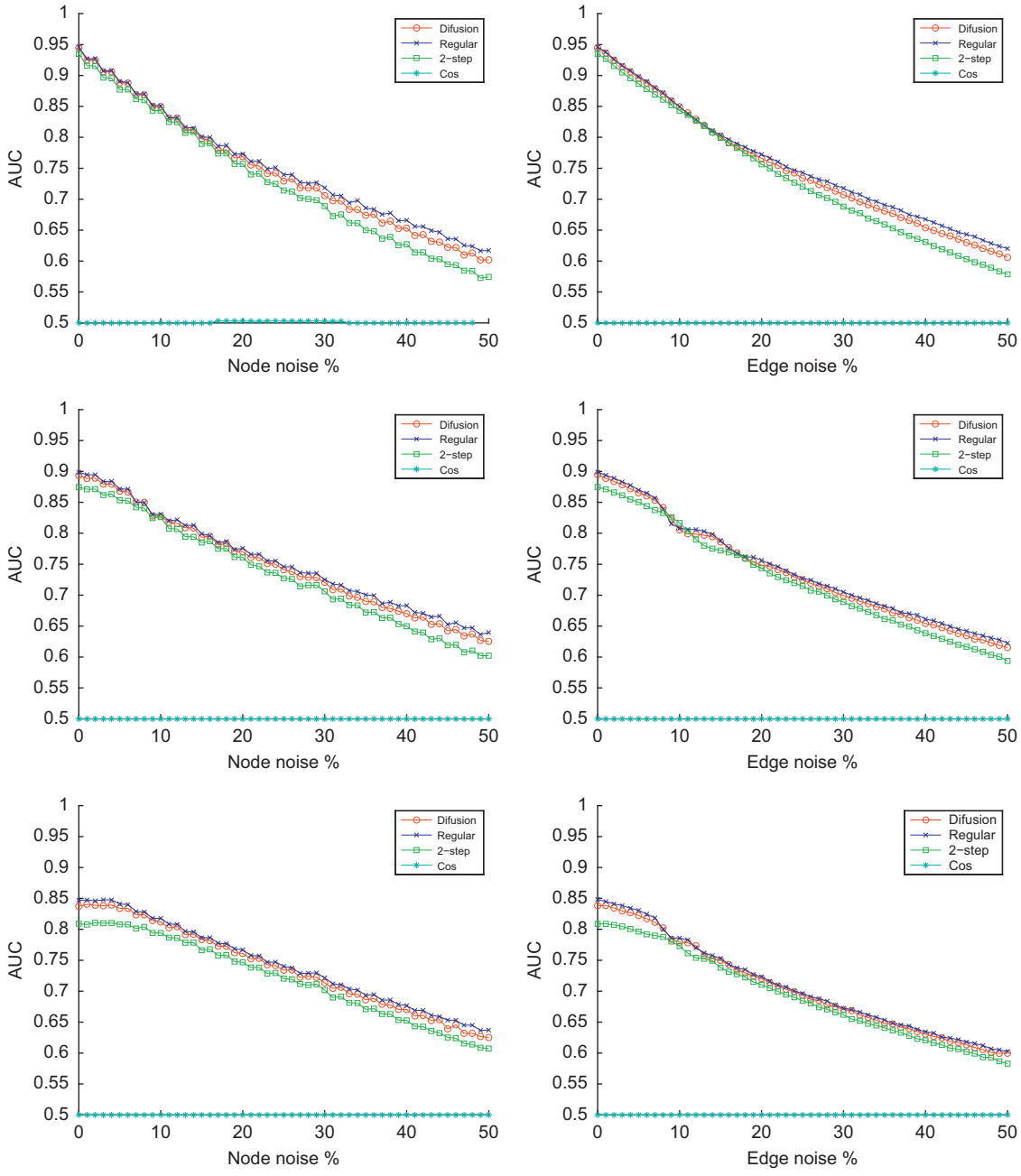

Fig. 3. AUC decay for different types of kernels with 10% (first row), 30% (second row) and 50% (third row) of edge density with node (left column) and edge (right column) noise from 0% to 50%.

different β values tend to converge. These kernels provide optimal results for values of β between 0.5 and 1. We will use always $\beta = 1$ for further experiments.

Other kernel that relies on a parameter is the p-step one. In Fig. 2 (bottom-right) we compare the AUC value for different values of p and noise. We can see that the performance decreases as p is increased. Therefore, from now on we will focus on comparing only the 2-step version of this kernel with the other types of kernels.

In Fig. 3 we show AUC vs noise (from 0% to 50%) for the different kernel types previously defined, and for different rates of edge density (10%, 30%, and 50%). We can see that in all cases the regularized Laplacian kernel yields the best results.

In Fig. 4 we compare the kernels that have provided better results with other types of attributes: normalized degree and spectral attributes extracted from the main eigenvector ϕ of the adjacency matrix X

$$\mathcal{A}_i^{degree} = \frac{1}{m}\sum_{j=1}^{m} X_{ij} \quad \text{and} \quad \mathcal{A}_i^{spectral} = \phi_i^*.$$

Degree information is quite sensitive to noise, because it considers only first-order connections for each node. The spectral information considers global information about the graph, in a similar way as kernels do, and therefore it will be less sensitive to local changes. When we have edge noise the result provided by spectral attributes is similar to the one provided by regularized Laplacian and diffusion kernels. However, when node noise is present, the performance of spectral attributes decays significantly, due to the sensitiveness of the graph spectra to the loss of

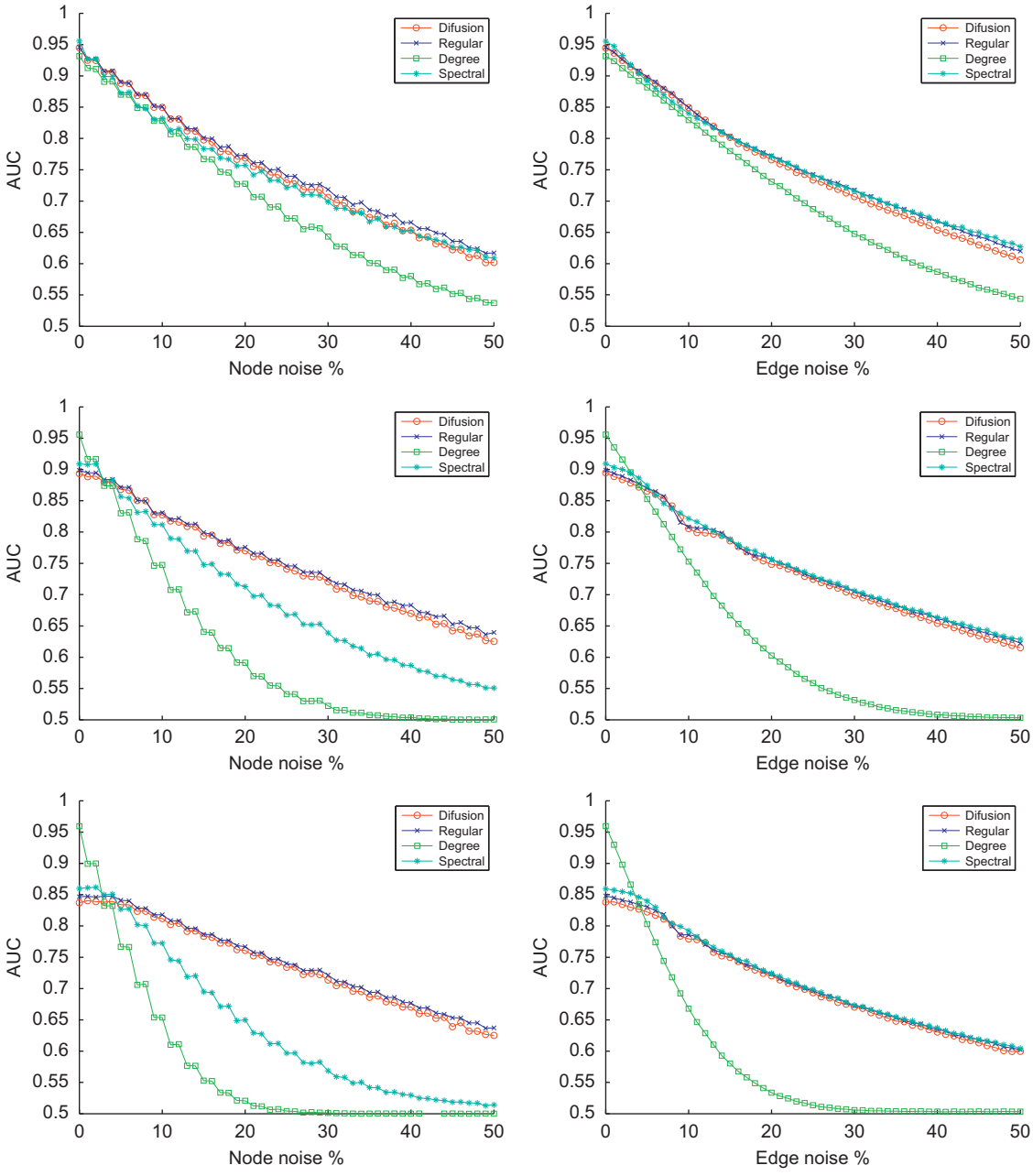

Fig. 4. AUC decay for kernel, spectral and degree attributes with 10% (first row), 30% (second row) and 50% (third row) of edge density with node (left column) and edge (right column) noise from 0% to 50%.

notable nodes [48]. The regularized Laplacian kernel holds its performance greater (or equal) than the other attributes, therefore this will be the most appropriate attribute for node matching.

4. Graph matching with kernel attributes

So far, we have defined and evaluated a set of attributes that measure the similarity between two nodes, helping us to determine if these nodes should be matched. We are interested in obtaining a matching between all the nodes of two graphs, in which the structure of these graphs is preserved. We cannot achieve this by using only these attributes, because although they are able to distinguish between correct and wrong matches in most cases, there also exist a considerable amount of cases in which these attributes fail. Given that these attributes use only local information, when they fail the structure of the graph could be broken.

Therefore, these attributes must be used as support information within a graph matching algorithm that considers the structure of the input graphs. We will show how two well known graph matching algorithms, Softassign [37] (SA) and replicator

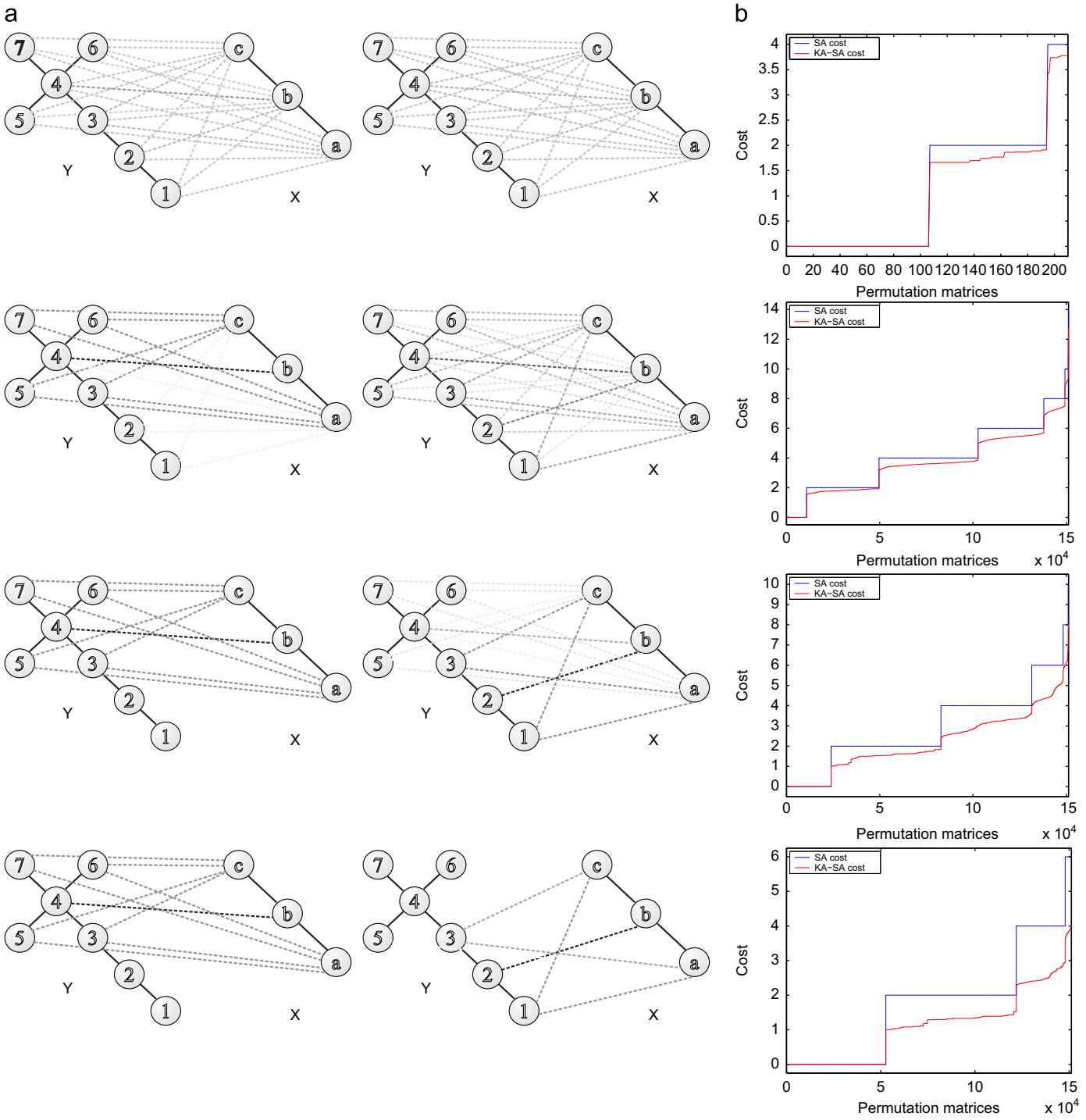

Fig. 5. (a) Progress of the algorithm for a simple matching problem. Matching matrices for different intermediate iterations of Softassign (SA, left) and kernel-attributed Softassign (KA-SA, right). (b) Shape of the classic and the kernel-attributed version of the cost function for the matching between different graphs.

dynamics [39] (RD), could be adapted to tackle kernel information, introducing the kernel attribute divergence in their cost functions.

4.1. Softassign

The graphs to be matched are defined as $G_X=(V_X,E_X)$, with nodes $a\in V_X$ and edges $(a,b)\in E_X$, and $G_Y=(V_Y,E_Y)$, with nodes $i\in V_Y$, edges $(i,j)\in E_Y$, and adjacency matrices X and Y. A feasible solution to the graph matching problem between G_X and G_Y is encoded by a permutation matrix M of size $m\times n$, being $m=|V_X|$ and $n=|V_Y|$, with binary variables

$$M_{ai}=\begin{cases}1 & \text{if } a\in V_X \text{ matches } i\in V_Y,\\ 0 & \text{otherwise.}\end{cases}$$

We will use the Gold and Rangarajan [37] formulation, inserting into its cost function the attributes obtained from kernels. This cost function is defined as

$$F(G_X,G_Y;M)=\frac{1}{2}\sum_{a=1}^{m}\sum_{i=1}^{n}\sum_{b=1}^{m}\sum_{j=1}^{n}M_{ai}M_{bj}C_{aibj}, \tag{12}$$

where usually $C_{aibj}=X_{ab}Y_{ij}$. Considering the attributes defined in the previous section, a simple way to incorporate them into the latter energy function is to define C_{aibj} as

$$C^K_{aibj}=X_{ab}Y_{ij}P(match|\mathcal{A}^X_a,\mathcal{A}^Y_i)P(match|\mathcal{A}^X_b,\mathcal{A}^Y_j), \tag{13}$$

where the attributes $\mathcal{A}^X$ and $\mathcal{A}^Y$ are associated with the kernels from the graphs G_X and G_Y respectively (but it is also possible to use other kinds of attributes, such as degree or spectral information).

The latter definition of C^K_{aibj} ensures that $C^K_{aibj}\le C_{aibj}$, and the equality is only verified when nodes a and i have equal attributes, and the same for nodes b and j. In practice, this weights the matching in such a way that matches with compatible attributes in their opposite vertices are preferred, and otherwise they are underweighted. This version of Softassign with kernel attributes will be referred further on as kernel-attributed Softassign (KA-SA).

To see intuitively the difference between SA and KA-SA, in Fig. 5(a) we show the progress of both algorithms for two example graphs. SA prefers clearly the assignment $(b,4)$ which is consistent with the cardinality heuristic (highly connected vertices in X prefer highly connected vertices in Y). On the other hand, in KA-SA, the assignment $(b,2)$ is clearly preferred and a and c may be assigned either to 1 or 3. The cardinality heuristic is inhibited in favor of a structural compatibility heuristic.

In Fig. 5(b) we compare the shape of SA and KA-SA cost functions. The x-axis represents all the possible permutation matrices (matching solutions), ordered by their cost. We can see that SA has more ambiguous solutions (plateaus) than KA-SA. We can see that as we remove edges, the ambiguity in the classic cost function is increased (plateaus becomes larger), whereas the attributed case yields few ambiguous solutions, which helps the algorithm to achieve the optimal solution.

We can define a symmetric and normalized similarity measure between two graphs relying on our kernel-attributed cost function. Such a measure is defined from the result provided by a graph matching algorithm

$$F_{XY}=\frac{\max_M[F(G_X,G_Y;M)]}{\max[F_{XX},F_{YY}]}, \tag{14}$$

where $F_{XX}=F(G_X,G_X;I_{|V_X|})$, $F_{YY}=F(G_Y,G_Y;I_{|V_Y|})$, being $I_{|V_X|}$ and $I_{|V_Y|}$ the identity matrices defining self-matchings. This results in a normalized measure $F_{XY}=F_{YX}\in[0,1]$. That is, when the compared graphs are isomorphic, this value is 1, and as they become different this value is closer to 0.

4.2. Replicator dynamics

This algorithm relies on building an association graph, which encodes the compatibility between all the possible node correspondences. Given two graphs $G_X=(V_X,E_X)$ and $G_Y=(V_Y,E_Y)$, the size of their association graph is $(mn\times mn)$, being $m=|V_X|$ and $n=|V_Y|$. Each node (a,i) in the association graph represents a matching between $a\in V_X$ and $i\in V_Y$. There exists an edge between two nodes (a,i) and (b,j) if, and only if, these correspondences are compatible. That is, if both $(a,b)\in E_X$ and $(i,j)\in E_Y$, or both $(a,b)\notin E_X$ and $(i,j)\notin E_Y$.

An easy way to apply kernel attributes to this algorithm is by introducing them as edge weights in the association graph

$$A_{(i,h),(j,k)}=\begin{cases}P^{match}_{ih}P^{match}_{jk} & \text{if } i\neq j,h\neq k\\ & \text{and } (i,j)\in E_X\Leftrightarrow(h,k)\in E_Y,\\ 0 & \text{otherwise,}\end{cases}$$

where $P^{match}_{ij}=P(match|\mathcal{A}^{G_X}_i,\mathcal{A}^{G_Y}_j)$ and A is the adjacency matrix of the association graph. This new version of replicator dynamics will be referred as kernel-attributed replicator dynamics (KA-RD).

5. Graph fusion for central graph clustering

In order to apply a central clustering algorithm to the graph domain we need two ingredients: a similarity measure (Eq. (14)) for evaluating whether a graph belongs to a given class and a method for building a prototype from a set of graphs belonging to the same class. In our previous work we proposed an incremental fusion method (IF) [52] for building the prototypes. Given a set with N input graphs, this method needs to solve $N-1$ matching problems to obtain each prototype, that is, $K(N-1)$ in each iteration of a clustering algorithm (considering K classes), which is too computationally expensive.

In this section we propose an alternative kernel-based fusion method (KBF) not depending on the order in which the graphs are fused, which can be integrated efficiently in a central k-means clustering algorithm. This method relies on computing all the pairwise matchings in the input set $(N\times(N-1)/2$ symmetrical matching problems), but they should be obtained only once in the initialization step of the k-means algorithm.

These pairwise matchings form a network consisting of all the nodes in the input set of graphs, in which two nodes are connected if there exists a matching between them. This network will be referred as *super-graph*, and it will be used to build a prototype representing the input set of graphs.

5.1. Building the super-graph

Each graph in the input set $G_i\in S$ is a 2-tuple $G_i=(V_i,E_i)$ where: V_i is the set of nodes and $E_i\subseteq V_i\times V_i$ is the set of edges. The algorithm can be extended to consider also node and edge attributes.

To obtain the super-graph it is necessary to obtain all the matching matrices M^{XY} between all pairs $\langle G_X,G_Y\rangle\in S\times S$ with $X\neq Y$. We use the notation M^{XY}_{ij} to represent the matching between a node i in the graph G_X and a node j in the graph G_Y.

The super-graph G_M encodes the network of all the matchings among the graphs in S, and is defined as a 5-tuple $G_M=(V_M,E_M,\theta,\nu,\xi)$, where

- $V_M=\cup_{i=1}^{|S|}V_i$ is the union of the nodes from all the graphs in S.
- $\theta: V_M\longrightarrow S$, is a function assigning each node in the super-graph with its corresponding graph in the original set.
- $\nu: V_M\longrightarrow\cup_{i=1}^{|S|}V_i$, is a function assigning each node in the super-graph with its corresponding node on the graphs of the original set.

- $E_M = \{(i,j), i,j \in V_M : M_{v_i v_j}^{\theta_i \theta_j} = 1\}$, that is, two nodes will be connected if, and only if, their corresponding nodes in the graphs in the set S are matched.
- $\xi : E_M \longrightarrow \mathbb{R}^+$ is a weighting function for the edges. This weight indicates the compatibility of the matching represented by its corresponding edge. It will be used for dividing the super graph into a set of partitions maximizing its inner compatibility, an will be defined in detail in the next section.

In order to build the super-graph we use the discrete matches $M_{v_i v_j}^{\theta_i \theta_j}$ obtained by any graph matching method (e.g. methods defined in Section 4). These algorithms usually obtain as a result a binary matrix M satisfying the constrain that there are one, and only one, 1 in each row and column (permutation matrix). This 1 indicates that the nodes associated to its corresponding row and column are matched. These matchings are represented by edges in the super-graph. Nevertheless, we have seen that usually there exists ambiguity in the possible solutions to the matching, therefore this implies that there are more than one correct matching matrix. In fact, the Softassign algorithm obtains a real doubly stochastic matrix M (their rows and columns sum 1), then, when there are many ambiguous solutions, the value is spread in many cells in the same row or column. Typically, it is performed a clean-up process over this real matrix, retaining only the greatest values for each row and column and discarding the others. This implies leaving many possible solutions out, that could be as correct as the selected one. In order to avoid this problem, we could use the matching matrix previous to the clean-up process to build the super-graph. In this case we must redefine the criterion for obtaining the super-graph edges

- $E_M = \{(i,j), i,j \in V_M : M_{v_i v_j}^{\theta_i \theta_j} > \epsilon\}$, being ϵ a positive constant close to 0 (in our implementation we have $\epsilon = 0.1$). Thus, we consider that two nodes could match whenever their corresponding cell in the matching matrix has a value clearly greater than 0.

5.2. Super-graph partitions

Our aim is to exploit the information provided by the network of matchings to build the prototype. These matchings, represented as edges in the super-graph, induce a set of disjoint partitions $P_\alpha \subseteq \{i : i \in V_M\}$, whose nodes are connected between them, but they are not connected to nodes belonging to other partitions. We can divide the super-graph into a set of partitions P_S satisfying:

$$\bigcup_{P_\alpha \in P_S} P_\alpha = V_M, \quad \text{and} \quad P_\alpha \cap P_\beta = \emptyset \ \forall P_\alpha, P_\beta \in P_S, \ \alpha \neq \beta. \tag{15}$$

That is, all the nodes in the super-graph belong to one, and only one partition. Each partition should satisfy the following constrain:

$$\forall i \in P_\alpha, \nexists j \in P_\alpha : \theta_i = \theta_j, \ j \neq i, \ \forall P_\alpha \subset V_M. \tag{16}$$

That is, each partition has at most one node coming from each graph in S. In an ideal case, the network of matchings should induce this kind of partitions (e.g. in Fig. 6(a-top) we show an example of fusion for the ideal case). In this case the fusion is easy. Each partition corresponds to a node in the prototype graph.

However, in a real case, due to the matching ambiguity and errors, a partition could have multiple nodes from the same graph. We can see an example in Fig. 6(a-bottom), and a more complex case (considering ambiguous matchings) in Fig. 6(b). We must then decide which matchings are going to be taken into account in order to obtain a set of partitions satisfying the constraint in Eq. (16), and which ones will be discarded. Matchings with a higher value in the matching matrix will be preferred, because the higher this value is, the lower ambiguity has this matching. However, it is common to find many nodes with the same value in the matching matrix. In order to decide which one is preferred, their kernel attributes can be used to disambiguate. Therefore, each edge (i,j) from the super-graph will be weighted by a function ξ, which is defined as

$$\xi(i,j) = \overrightarrow{M}_{v_i v_j}^{\theta_i \theta_j} + \alpha \Phi_{v_i v_j}^{\theta_i \theta_j} \quad \forall (i,j) \in E_M, \tag{17}$$

where α is a small value (i.e. $\alpha \sim 0.01$) and Φ is an affinity measure between matched vertices v_i and v_j. In this case, we define Φ as

$$\Phi_{v_i v_j}^{\theta_i \theta_j} = P(match | K_{v_i}^{\theta_i}, K_{v_j}^{\theta_j}). \tag{18}$$

Being K^{θ_i} the kernel associated to the graph $\theta_i \in S$ containing a vertex v_i (the same holds for K^{θ_j}, from the graph $\theta_j \in S$ with a vertex v_j). Consequently, we have that $K_{v_i}^{\theta_i} = K_{v_i v_i}^{\theta_i}$ is the v_i-th element of the diagonal (similarly $K_{v_j}^{\theta_j} = K_{v_j v_j}^{\theta_j}$).

The latter weights $\xi(i,j)$, which encode structural compatibility, will be used to insert all the edges in E_M into a sorted list $\mathcal{L}_e$. The elements with higher weights will be taken in first place. These edges will be used to build the partitions of the graph, considering the constraints in Eq. (16), that is, within a partition there should be at most only one node from each input graph. For each extracted edge (i,j), there are four possible cases:

- *Neither i nor j are assigned to any partition.* In this case a new partition is created, and both i and j are assigned to it.
- *i is assigned but j is not.* Add j to the partition of i if doing so the constraints are satisfied. Otherwise, add j to a new partition.
- *j is assigned but i is not.* Add i to the partition of j if doing so the constraints are satisfied. Otherwise, add i to a new partition.
- *Both i and j are assigned to a partition.* If both i and j are assigned to the same partition, no action is needed. Otherwise, fuse the partitions of i and j if the final partition satisfies the constraints.

The algorithm to obtain the partitions from the super-graph is detailed in Fig. 7. In this algorithm we introduce a set of auxiliary variables L_i and $O_{\alpha\theta}$. The variables L_i associate each node $i \in V_M$ with its corresponding partition α. The variables $O_{\alpha\theta}$ are binary, and indicate whether a partition α contains a node from an input graph θ or not. These latter variables are addressed to satisfy the constraint in Eq. (16).

5.3. Building the prototype

In a central clustering algorithm, such as k-means, the mean of each class is computed as a weighted sum of all the members of this class. These weights are obtained previously by the algorithm from the similarity between each object and each class. To translate this to the graph domain, we can use the similarity measure defined in Eq. (14) to compute graph weights (probability of belonging to a given class), and a method for building graph prototypes given a weighted set of graphs. We define the prototype as the mixture

$$\overline{G} = \sum_{k=1}^{N} \pi_k G_k = \pi_1 G_1 + \cdots + \pi_N G_N, \tag{19}$$

where $\pi_k G_k$ denotes the weighting of each graph G_k by its probability π_k.

Given a set of partitions P_S satisfying Eqs. (15) and (16), each partition $P_i \in P_S$ corresponds to a node in the prototype. Such a

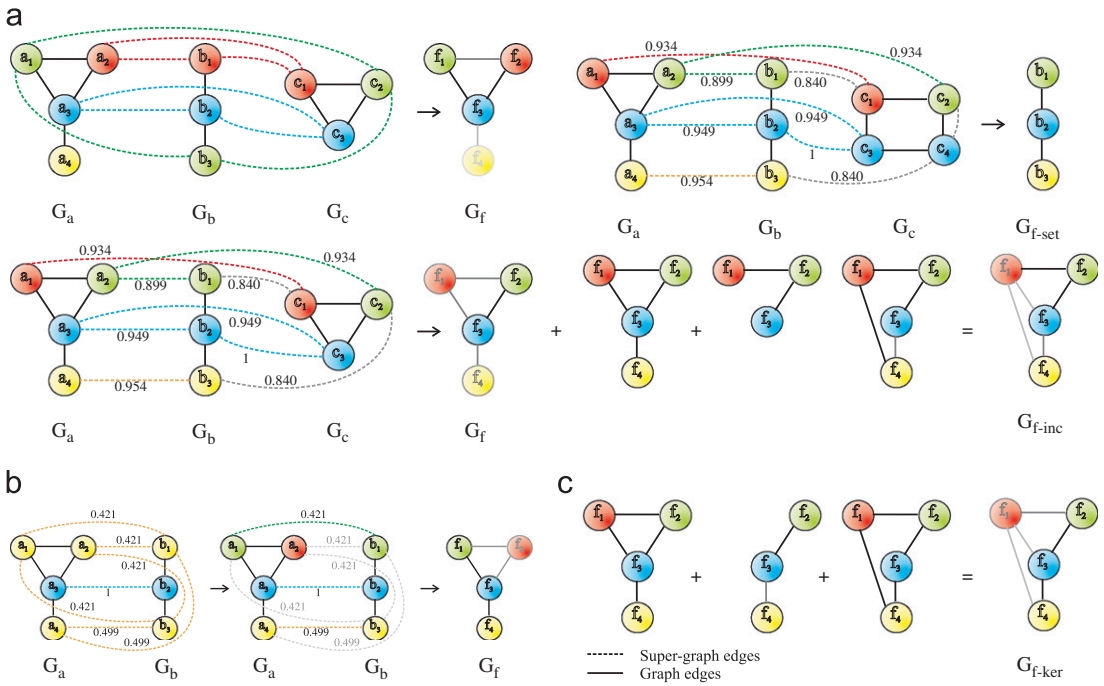

Fig. 6. Illustrating KBF. (a) Prototype building in an ideal case (top), a real one where kernels are needed (bottom). (b) Graph fusion with matching matrices previous to clean-up process: Super-graph with all possible matches (left), selected matches for each partition (center), and final obtained prototype (right). (c) Step-by-step fusion showing partitions and set median graph (SMG, top), and the difference between incremental fusion (IF, middle) and kernel-based fusion (KBF, bottom). Light colors mean low frequencies.

prototype graph is an approximation to the median graph [20] and it is defined by the 5-tuple $\overline{G}=(\overline{V},\overline{E},\mu,\delta,\mathcal{M})$, where:

- $\overline{V}=\{P_i \in P_S\}$ and $\overline{E}=\{(i,j):\exists k\in P_i, l\in P_j | (k,l)\in E^{ij}\}$ where $E^{ij}=\{(k,l):k\in P_i, l\in P_j, \theta_k=\theta_l, (v_k,v_l)\in E^{\theta_k}\equiv E^{\theta_l}\}$.
- $\mu:\overline{V}\longrightarrow[0,1]$ is a function assigning a weight to each node in the prototype. This weight is the probability that its corresponding node P_i belongs to the class represented by the prototype, and it is defined as $\mu_{P_i}=\sum_{k\in P_i}\pi_{\theta_k}$.
- $\delta:\overline{E}\longrightarrow[0,1]$ is a function assigning a weight to each edge in the prototype. This weight is the probability that an edge (i,j) belongs to the prototype, and it is defined as $\delta(i,j)=\sum_{(k,l)\in E^{ij}}\pi_{\theta_k}$. Thus, such weights are defined by integrating the weights of the graphs that the nodes implied in the connections belong to (nodes in P_i connected in the input graphs to nodes in P_j).
- $\mathcal{M}:\overline{V}\times S\longrightarrow \cup_{i=1}^{|S|} V_i$ is a function that registers the matchings between each node P_i in the prototype and its corresponding node in every input graph $X\in S$. It is defined as $\mathcal{M}^{P_i X}=v_k, k\in P_i:\theta_k=X$. Having this mapping we bypass the solving of a graph matching problem between each graph in S and each prototype.

The algorithm for obtaining such a prototype is detailed in Fig. 7. This algorithm receives as input the super-graph G_M and the partitions P_S obtained in the previous steps, besides the set S of input graphs and their weights π. In Fig. 6(c) we illustrate how the prototypes are built, comparing KBF, IF [52] and set median graph (SMG) [20]. SMG selects as prototype the graph in the input set whose sum of distances to all the other graphs in the set is minimal.

It is worth to highlight that the prototype building algorithm is the only task that has to be performed in each clustering iteration. The super-graph and its partitions can be obtained only once at the initialization step. These structures can be reused to build different prototypes in each iteration, by updating the weights π of the input graphs. In k-means K prototypes have to be obtained in each iteration. Each prototype $\overline{G}_k$ $(1\le k\le K)$ is obtained by applying a weight π_{ik} to each graph $G_i\in S$. These weights are obtained from the similarity measure in Eq. (14) between G_i and $\overline{G}_k$, normalizing the results to satisfy $\sum_{i=1}^{N}\pi_{ik}=1, \forall k:1\le k\le K$. An overview workflow of KBS integrated into a k-means algorithm is shown in Fig. 8.

6. Experiments

Up to this moment we have performed several experiments in order to evaluate the attributes extracted from graph kernels. In this section, we are going to test the effectiveness of these attributes for graph matching and clustering by applying the kernel-attributed version of Softassign (KA-SA) and replicator dynamics (KA-RD), and the proposed graph fusion method (KBF) within a k-means clustering algorithm. Kernel attributes are always obtained by applying the standard model defined in Eq. (11) based on the regularized Laplacian kernel, given that this kernel has reported the best results in our previous experiments. We will perform experiments both with randomly generated graphs and with graphs extracted from images (for object recognition).

6.1. Graph matching with randomly generated graphs

In these experiments we will use 50-node graphs with edge densities of 10%, 30% and 50%. For each type of graphs, we will perform some tests increasing the noise level from 0% to 50%, considering both node and edge noise. For each type of graphs and noise we have performed 1000 tests, and in Fig. 9 we show the average obtained results for each case.

We notice that the best results are obtained always by applying the kernel-attributed version of these algorithms. In

Algorithm KBF {
  For each pair of graphs $< G_X, G_Y >$, $G_X, G_Y \in S$
    $M^{XY} \leftarrow GraphMatching(G_X, G_Y)$
  End For
  $G_M \leftarrow BuildSuperGraph(S, M)$
  $P_S \leftarrow ObtainPartitions(G_M)$
  $\bar{G} \leftarrow BuildPrototype(S, G_M, P_S, \pi)$
  Return $\bar{G}$
}

Algorithm Build Super-Graph {
  $\alpha \leftarrow 0.01$, $V_M \leftarrow \emptyset$, $E_M \leftarrow \emptyset$
  For each $G_X \in S$
    For each $i \in V_X$
      $k \leftarrow$ Create new node
      $V_M \leftarrow V_M \cup k$
      $\theta_k \leftarrow G_X$
      $\nu_k \leftarrow i$
  End For
  For each $i \in V_M$
    For each $j \in V_M$
      If $M_{\nu_i\nu_j}^{\theta_i\theta_j} > 0$
        $E_M \leftarrow E_M \cup (i,j)$
        $\xi(i,j) \leftarrow M_{\nu_i\nu_j}^{\theta_i\theta_j} + \alpha\Phi_{\nu_i\nu_j}^{\theta_i\theta_j}$
      End If
    End For
  End For
  $G_M \leftarrow (V_M, E_M, \theta, \nu, \xi)$
  Return G_M
}

Algorithm Obtain Partitions {
  $\mathcal{L}_e \leftarrow Sort_w(\{< i,j,w >: (i,j) \in E_M,\ w = \xi(i,j)\})$
  While $\mathcal{L}_e \neq \emptyset$
    $< i,j,w > \leftarrow$ Remove First($\mathcal{L}_e$)
    Switch
      Case $L_i = \emptyset$, $L_j = \emptyset$
        $l \leftarrow$ new label
        $L_i, L_j \leftarrow l$
        $P_l \leftarrow \{i,j\}$
        $O_l \leftarrow [0]_{1\times|S|}$
        $O_{l\theta_i}, O_{l\theta_j} \leftarrow 1$
      Case $L_i = \emptyset$, $L_j \neq \emptyset$
        If $O_{L_j\theta_i} \neq 1$
          $L_i \leftarrow L_j$
          $P_{L_i} \leftarrow P_{L_i} \cup \{i\}$
          $O_{L_i\theta_i} \leftarrow 1$
        End If
      Case $L_i \neq \emptyset$, $L_j = \emptyset$
        If $O_{L_i\theta_j} \neq 1$
          $L_j \leftarrow L_i$
          $P_{L_j} \leftarrow P_{L_j} \cup \{j\}$
          $O_{L_j\theta_j} \leftarrow 1$
        End If
      Case $L_i \neq \emptyset$, $L_j \neq \emptyset$
        If $L_i \neq L_j$
          If $(O_{L_i} O_{L_j}^T = 0)$
            $O_{L_i} \leftarrow O_{L_i} + O_{L_j}$
            For each $k : L_k = L_j$
              $L_k \leftarrow L_i$
            End For
            $P_{L_i} \leftarrow P_{L_i} \cup P_{L_j}$
            Remove partition P_j
          End If
        End If
  Return P
}

Algorithm Build Prototype {
  $\bar{V} \leftarrow \{P_\alpha, P_\alpha \in P_S\}$, $\bar{E} \leftarrow \emptyset$
  $\mu \leftarrow [0]_{|\bar{V}|}$, $\delta \leftarrow [0]_{|\bar{V}|\times|\bar{V}|}$
  $\mathcal{M} \leftarrow [0]_{|\bar{V}|\times|S|}$
  For each node $i \in V_M$
    $P_i \leftarrow P_\alpha \in P_S, i \in P_\alpha$
    $(V,E) \leftarrow \theta_i$
    For each node $j \in V_M$
      $P_j \leftarrow P_\alpha \in P_S, j \in P_\alpha$
      If $\theta_i = \theta_j$
        If $(\nu_i, \nu_j) \in E$
          $\bar{E} \leftarrow \bar{E} \cup \{(P_i, P_j)\}$
          $\delta_{P_i,P_j} \leftarrow \delta_{P_i,P_j} + \pi_{\theta_i}$
        End If
      End If
    End For
    $\mu_{P_i} \leftarrow \mu_{P_i} + \pi_{\theta_i}$
    $\mathcal{M}^{P_i\theta_i} \leftarrow \nu_i$
  End For
  $\bar{G} \leftarrow (\bar{V}, \bar{E}, \mu, \delta, \mathcal{M})$
  Return $\bar{G}$
}

Fig. 7. Kernel-based fusion (KBF) algorithm.

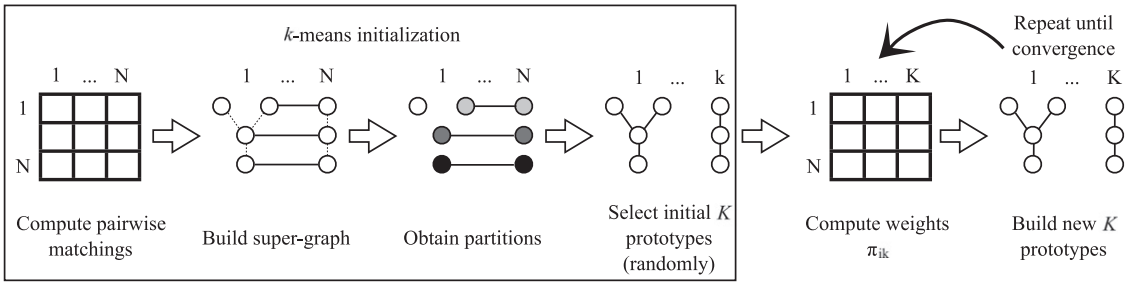

Fig. 8. Overview workflow of a *k*-means graph clustering algorithm with kernel-based fusion (KBF).

low density graphs Softassign performs better. On the contrary, in high density graphs replicator dynamics outperforms Softassign. Nevertheless the kernel-attributed version of each algorithm always outperforms its classical one.

6.2. Graph fusion with randomly generated graphs

In these experiments we will compare three different methods for obtaining the prototype: IF [52], KBF, and SMG [20]. We will

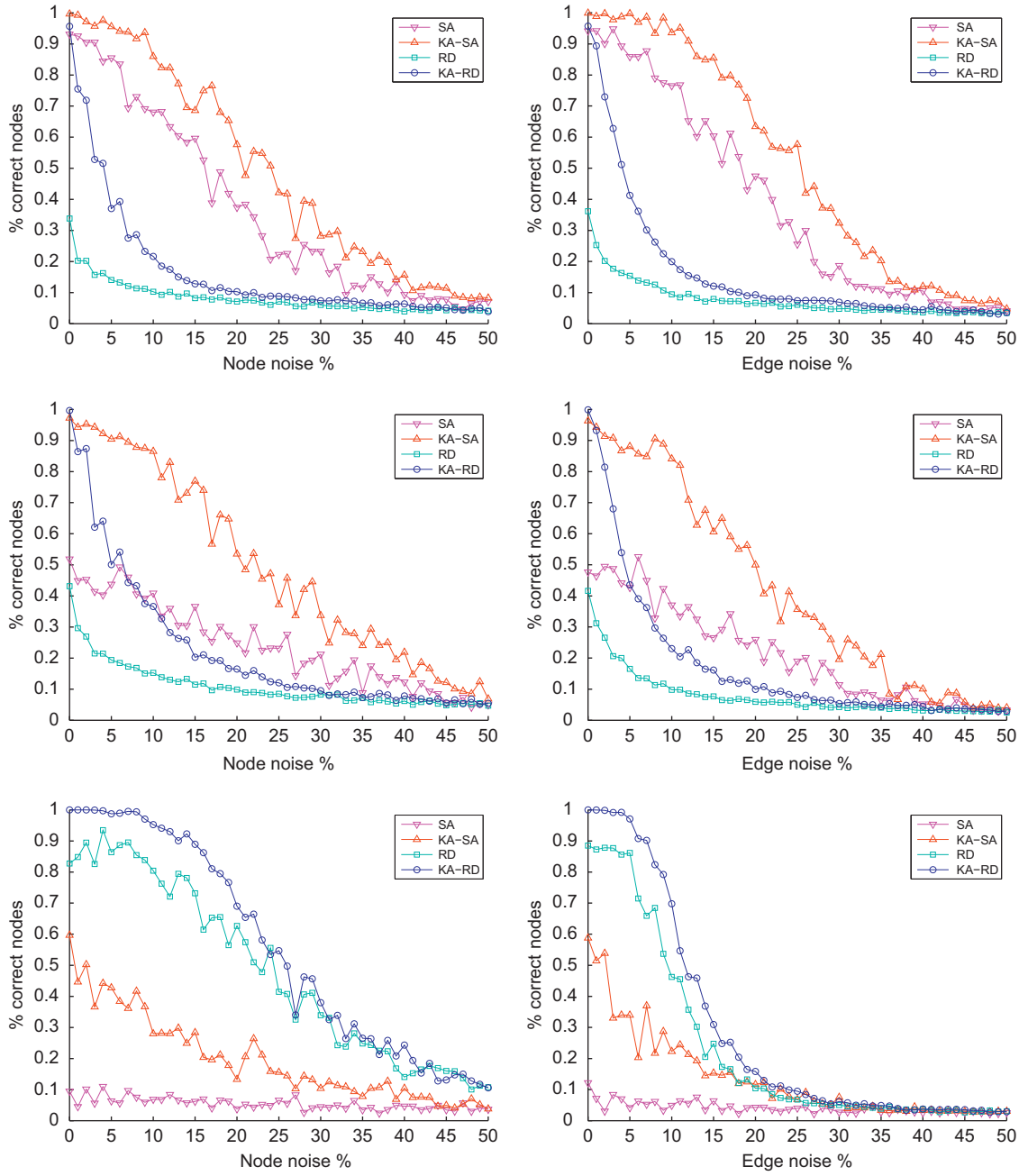

Fig. 9. Softassign (SA), replicator dynamics (RD), kernel-attributed Softassign (KA-SA) and kernel-attributed replicator dynamics (KA-RD) graph matching decay for node (left) and edge (right) noise, for edge densities of: 10% (first row), 30% (second row) and 50% (third row).

measure the quality of the prototypes they provide and their computational cost. As a quality measure we will use the average similarity (Eq. (14)) from the prototype to every graph in the set that this prototype represents. The lower this similarity is, the better this prototype represents its corresponding class.

This experiment (Fig. 10) uses the setting described in Section 6.1. For each kind of graph (edge density and noise fraction), we generate 100 classes containing 10 graphs each one of them. For each class we randomly generate a graph as a base, and then we corrupt this base graph (with node or edge noise) to obtain each one of the 10 graphs belonging to this class. For each kind of graph and for each method we register the average similarity obtained from the 100 performed tests.

We notice that with low levels of noise all the methods for building the prototypes provide similar results, whereas as we increase the level of noise the difference between SMG and KBF is stressed. That is, in compact classes SMG is a good prototype, due to the fact that all the graphs in these classes are similar, and thus any of them may be used as a prototype, whereas in sparser classes this median graph is not representative enough and it is better to use one of the fusion methods. The fusion method providing better results is KBF, but its computational complexity is considerably higher than IF. This latter fusion method provides acceptable results with a lower computational cost.

We have analyzed the execution time for each of these methods with input graphs of different size. In Fig. 11 we show the average execution time vs the number of nodes of each graph (left column) and the number of graphs in each class (right column).

The computational cost is $O(m^4)$ with respect to the number of nodes m in the graphs (due to the matching algorithm). However, the execution time of IF is always much faster than KBF and SMG, because both KBF and SMG need to solve

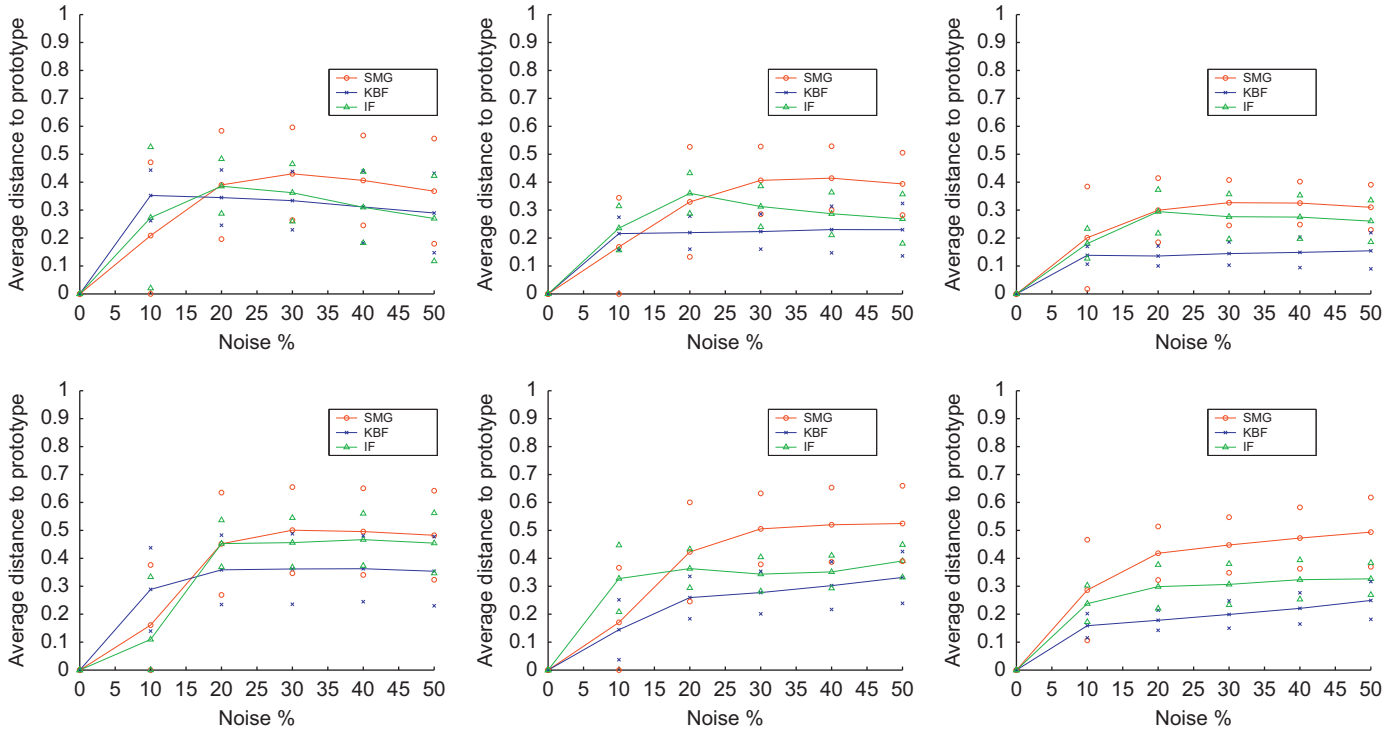

Fig. 10. Robustness of the fusion methods set median graph (SMG), incremental fusion (IF), and kernel-based fusion (KBF) with respect to increasing node (top) and edge (bottom) noise levels, for edge densities of 10% (left column), 30% (middle column), and 50% (right column). Extra dots indicate the standard deviation of the obtained results.

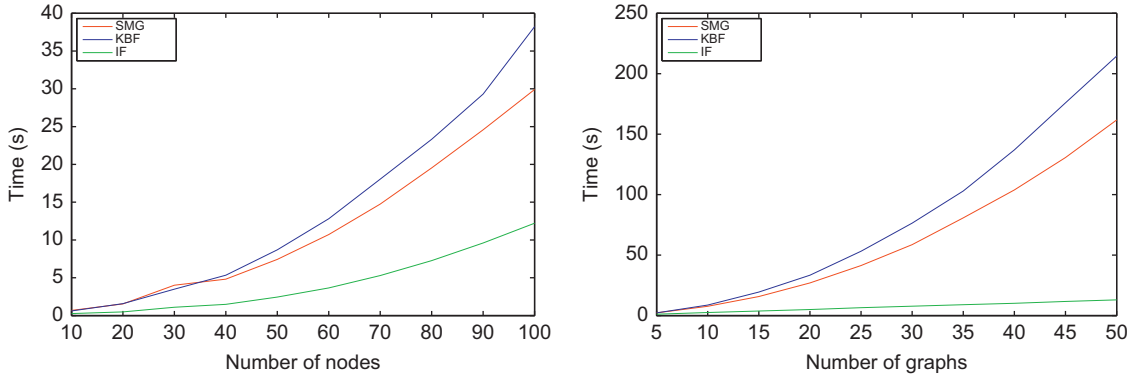

Fig. 11. Time for building the prototypes with set median graph (SMG), incremental fusion (IF), and kernel-based fusion (KBF) as we increase (a) the number of nodes in the input graphs and (b) the number of graphs in each class.

$(N-1)N/2$ graph-matching problems, being N the number of input graphs, whereas IF needs to solve only $(N-1)$ of them, that is, its cost is $N/2$ times lower than the other methods.

We can also see that KBF and SMG have a similar cost (KBF is slightly higher). Hence it follows that almost all the consumed time by KBF is employed in the initial step in which we obtain all the pairwise matchings. On the other hand, we find a sharp increase in the execution time when increasing the number N of graphs to be fused (Fig. 11 (right)). In this case, we are increasing the number of matching problems to solve, linearly in the IF case, and quadratically for the other methods (KBF and SMG). However, KBF not only obtains better results but also more efficient than IF when applied within a central clustering algorithm, as we will see further on.

6.3. Graph matching for non-attributed structural recognition

Once we have experimented with randomly generated graphs, we proceed to perform experiments with real cases. In particular, we are going to use non-attributed graphs extracted from images in order to apply a kernel-attributed graph matching algorithm to object recognition.

In this structural recognition experiment we will use the sequences CMU, MOVI and Chalet. They consist of three sequences of houses, with 10 images each one. Within each sequence, we see the same house from different points of view. We extract a set of salient points from these images, and use them for building a graph through triangulation. In Fig. 12(c) we show some examples of the matchings obtained with both SA and KA-SA. We notice that KA-SA provides a more accurate matching than SA (SA yields a higher number of crossed lines in the matching). The average percentage of nodes correctly matched is 56.73% for SA and 68.30% for KA-SA.

In Fig. 12(b) we show the confusion matrices for all pairs of graphs in the input set, provided by the symmetric and normalized similarity measure defined in Eq. (14). We can observe that KA-SA distinguishes the graphs belonging to the same class from

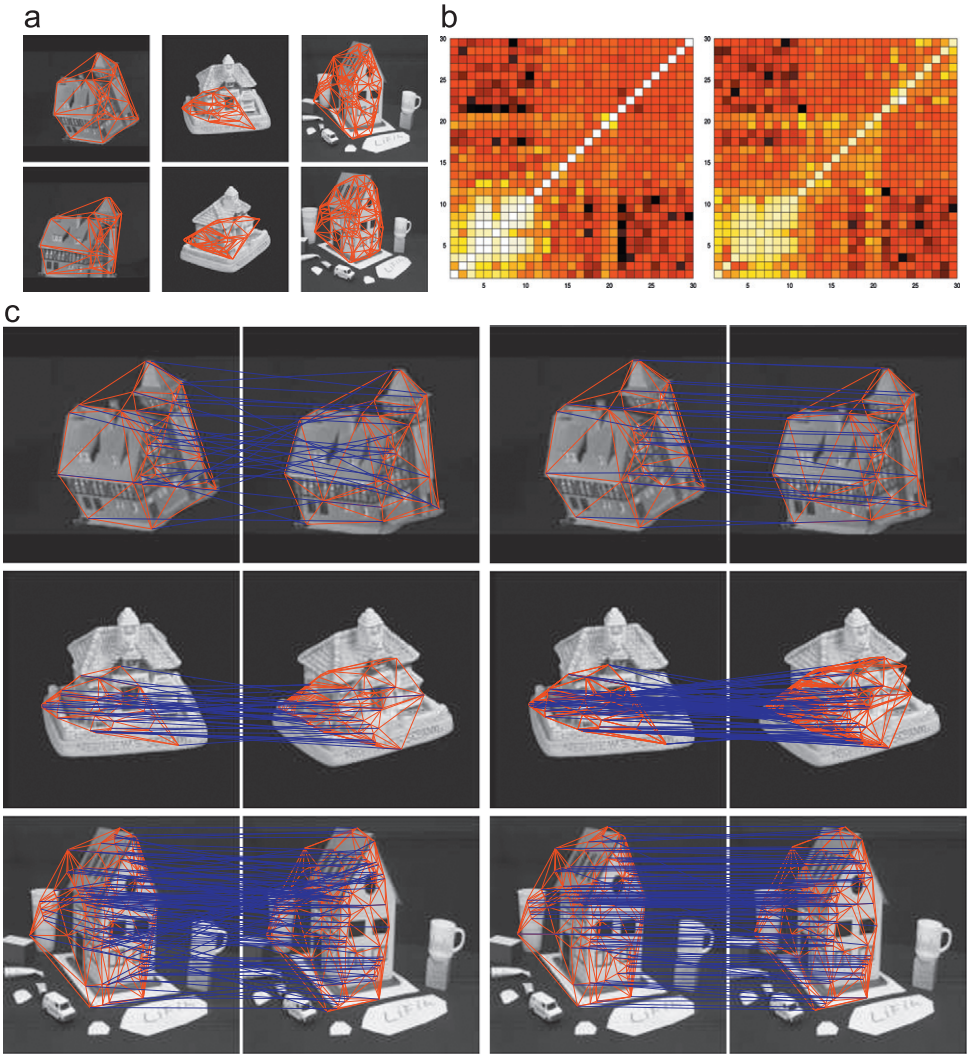

Fig. 12. (a) Images belonging to classes CMU (left), Chalet (center) and MOVI (right); and graphs extracted from them. (b) Confusion matrices between all the images in the three sequences and (c) matching results with unattributed Softassign (SA, left) and kernel-attributed Softassign (KA-SA, right), for different images of CMU (first row), Chalet (second row) and MOVI (third row) sequences.

the graphs of different classes better than SA. With this latter version many graphs in MOVI (21–30) and Chalet (11–20) sequences are totally confused.

6.4. Prototype building for non-attributed structural recognition

Our aim in this experiment is to obtain a graph prototype for each class of images (CMU, Chalet and MOVI). An image will be classified by comparing its graph with each prototype through graph matching and the similarity measure in Eq. (14).

In Fig. 13(d) we show the average distance from each prototype to all the graphs in its class. The CMU sequence provides better results using SMG (graphs in CMU are very similar between them), whereas Chalet and MOVI clearly provide better results with KBF.

In Fig. 13(a–c) we show the similarity between the prototypes and all the input graphs. SMG is able to successfully distinguish the sequence CMU, but in the other two classes it only distinguishes one graph (the one selected as set median graph). In these cases a good prototype cannot be obtained by only selecting a graph from the input set, and KBF provides a better result than the previous methods.

6.5. Clustering experiments

In these experiments we will apply a k-means clustering algorithm both with IF and KBF to the set of graphs of sequences CMU, Chalet and MOVI (30 images altogether), to find the class structure of this set and the prototypes representing these classes.

In Fig. 13(e, f) we show the clustering results. As it occurred in the fusion experiments, we can see that KBF provides better results. However, in this case it also provides a better execution time within the clustering algorithm. This is due to the fact that we can reuse the matchings obtained in the initialization step, and it is not necessary to obtain new correspondences for building the prototypes. Specifically, the clustering process with incremental fusion took 51.23 s, whereas using KBF this time was reduced to 12.57 s. Therefore, in this case it is clear that the most convenient method is KBF.

6.6. Classification experiments

In this experiment the GatorBait database [47] will be used to compare the effectiveness of different graph-matching and prototype building methods for classification tasks. This database consists of 100 shapes representing fishes from 30 different classes (see Fig. 14). Classes correspond to fish genus so the

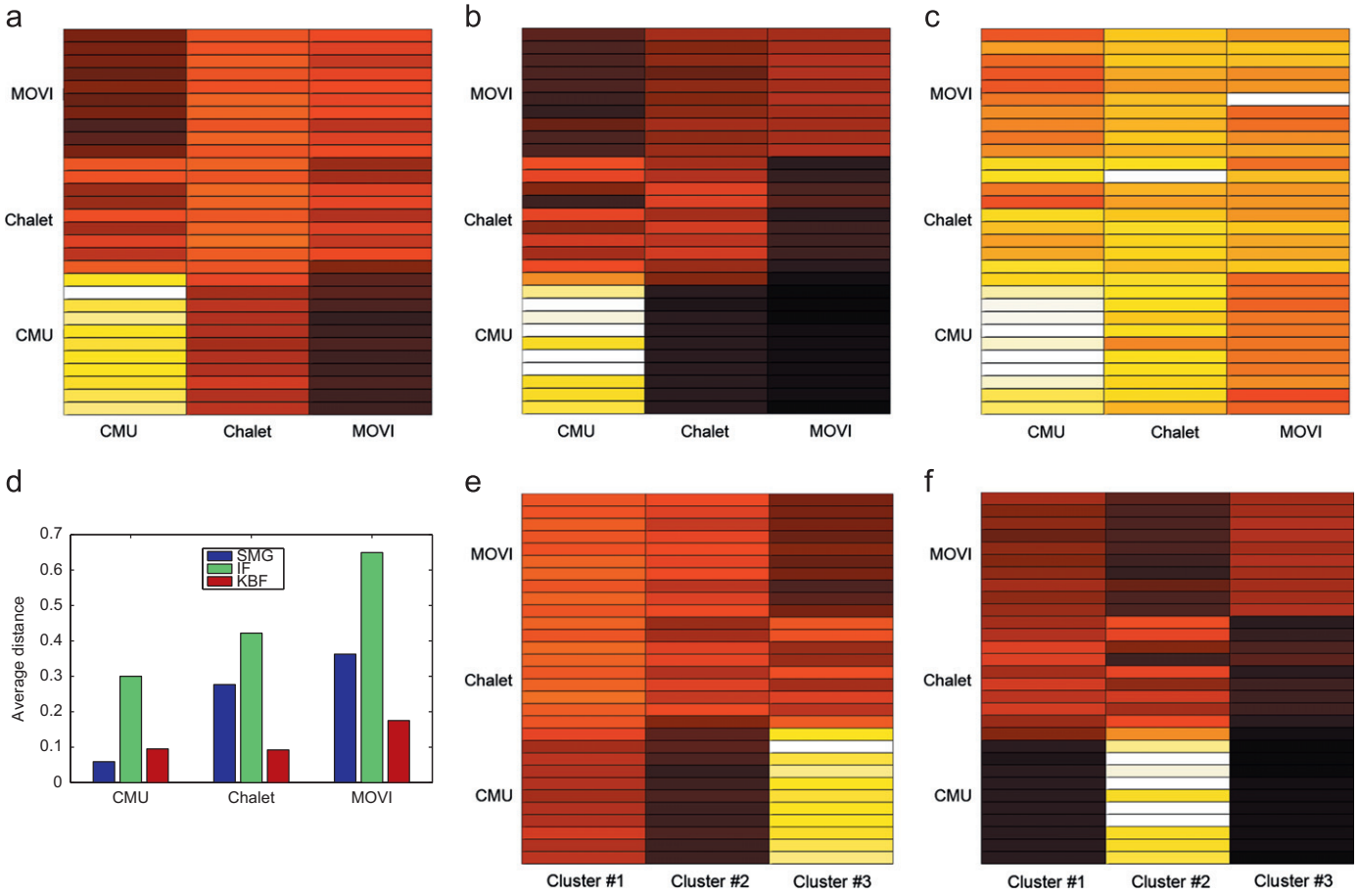

Fig. 13. Confusion matrices between each graph and each prototype of the sequences CMU, Chalet and MOVI. High similarities are shown in light color. (a) Prototypes obtained through incremental fusion (IF). (b) Prototypes obtained through kernel-based fusion (KBF). (c) Set median graph (SMG) of each class. (d) Average distance between the obtained prototypes and the graphs in their corresponding classes. (e) Optimal similarities F_{iz} between the input graphs in the sequences CMU, Chalet and MOVI, and the prototypes obtained through the k-means clustering algorithm using incremental fusion (IF) and (f) kernel-based fusion (KBF).

Table 2
Interclass and intraclass similarities for the GatorBain database with Softassign (SA) and kernel-attributed Softassign (KA-SA).

Graph-matching method	Intraclass similarity		Interclass similarity	
	Mean	Variance	Mean	Variance
SA	0.1704	0.0167	0.0691	0.0002
KA-SA	0.5205	0.0948	0.1030	0.0003

intraclass variability is usually high. Therefore, the database, though having only 100 samples, plays a challenging role in testing graph matching and graph learning algorithms.

In first place, we analyze the intraclass and interclass similarity of all the pairwise matchings. Table 2 shows the average similarity between graphs in the same class (intraclass) and between graphs in different classes (interclass), and their variances. KA-SA algorithm clearly outperforms SA, keeping interclass similarity low, but dramatically increasing intraclass similarity.

Next, different graph-matching and prototype building methods are applied to classify the graphs in this database. First, a prototype is built for each one of the 30 classes with the selected prototype building method. Then, the similarities between all the 100 graphs in the database and the 30 prototypes are obtained by applying the selected graph-matching method and Eq. (14). Table 3 shows the percentage of correctly classified graphs in each case. The obtained results are consistent with our previous graph fusion experiments.

7. Conclusions and future work

In this work we have exploited the information provided by different kind of kernels, in order to transform non-attributed graphs into attributed ones, and thus improve the performance of graph matching algorithms. We have concluded that the regularized Laplacian one is the most convenient kernel in order to distinguish between correct and wrong matches in the general case. We have introduced the information provided by this kernel into existing graph matching algorithms (as an attribute in the Gold and Rangarajan cost function, and as an edge weight in the association graph). Instead of using a non-quadratic cost function, when we weight properly the quadratic Gold and Rangarajan cost function, by using the information provided by kernels, we obtain comparable results to a non-quadratic one. The practical effect of such a weighting is that it provides a good characterization of the local structure for each node, which causes that nodes with similar neighboring structures are matched, in a context in which we deal with a high ambiguity.

We have proposed a new method for building the graph prototypes: kernel-based fusion (KBF), assuming that the input graphs could be non-attributed. We retain the frequency of occurrence of each node (edge) in the input set, so that we can generate new members of a class from its prototype by sampling. The KBF algorithm has a temporal cost $N/2$ times higher than our previous IF [52]. However, this global method, in spite of having a quite higher complexity, also provides notably better results than IF. In addition, it can be integrated efficiently into a central

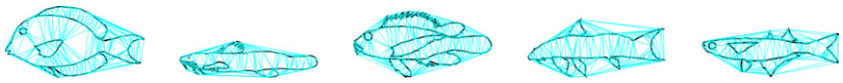

Fig. 14. Some graphs from GatorBait database.

Table 3
Percentage of correctly classified GatorBait database graphs for different graph-matching and prototype buildings methods: Softassign (SA), kernel-attributed Softassign (KA-SA), set median graph (SMG), incremental fusion (IF), and kernel-based fusion (KBF).

Method	SA (%)	KA-SA (%)
SMG	28	32
IF	24	29
KBF	38	44

clustering algorithm, whereas IF cannot. With KBF method we can apply efficiently a central clustering algorithm to the domain of graphs, not needing to solve any matching problem during the execution of each clustering iteration. We only need to approximately solve a fixed number of matching problems in the initialization step. We have seen that KBF inside the clustering algorithm yields better results than IF, and it is also more efficient, allowing us to use a central graph-clustering algorithm whose computational cost is similar to the cost of a pairwise clustering algorithm.

The proposed prototype building method produces a first-order generative model. Our future work includes the definition of a higher order generative model for the graph prototypes. For this purpose we suggest to exploit the super-graph structure, in which we have registered all the relationships between the nodes of the input graphs, by using the Szemerédi regularity lemma [53,54]. On the other hand, we have tested the performance of the kernel matrix values as a similarity measure between the nodes of two graphs. However, we have only applied them as attributes in the cost function of an existing graph matching algorithm. We may suggest to exploit the information provided by the kernel in a different way, through a new matching algorithm.

References

[1] K. Siddiqi, A. Shokoufandeh, S.J. Dickinson, S.W. Zucker, Shock graphs and shape matching, Int. J. Comput. Vision 35 (1) (1999) 13–32.
[2] S. Biasotti, D. Giorgi, M. Spagnuolo, B. Falcidieno, Reeb graphs for shape analysis and applications, Theor. Comput. Sci. 392 (1–3) (2008) 5–22.
[3] M.A. Lozano, F. Escolano, Recognizing indoor images with unsupervised segmentation and graph matching, in: F.J. Garijo, J.C.R. Santos, M. Toro (Eds.), IBERAMIA, Lecture Notes in Computer Science, vol. 2527, Springer, 2002, pp. 933–942.
[4] T.K. Leung, M.C. Burl, P. Perona, Finding faces in cluttered scenes using labeled random graph matching, in: ICCV, IEEE Computer Society, 1995, pp. 637–644.
[5] F.-F. Li, R. Fergus, P. Perona, One-shot learning of object categories, IEEE Trans. Pattern Anal. Mach. Intell. 28 (4) (2006) 594–611.
[6] P. Felzenszwalb, D. Huttenlocher, Pictorial structures for object recognition, Int. J. Comput. Vision 61 (1) (2005) 55–79.
[7] D.J. Crandall, P.F. Felzenszwalb, D.P. Huttenlocher, Spatial priors for part-based recognition using statistical models, in: CVPR (1), IEEE Computer Society, 2005, pp. 10–17.
[8] I. Kokkinos, P. Maragos, A. L. Yuille, Bottom-up & top-down object detection using primal sketch features and graphical models, in: CVPR (2), IEEE Computer Society, 2006, pp. 1893–1900.
[9] M. Lozano, F. Escolano, Protein classification by matching and clustering surface graphs, Pattern Recognition 39 (4) (2006) 539–551.
[10] A. Wong, D. Ghahraman, Random graphs: structural-contextual dichotomy, IEEE Trans. Pattern Anal. Mach. Intell. 2 (4) (1980) 341–348.
[11] A. Wong, M. You, Entropy and distance of random graphs with application to structural pattern recognition, IEEE Trans. Pattern Anal. Mach. Intell. 7 (5) (1985) 599–609.
[12] A. Wong, J. Constant, M. You, Random graphs, Synthactic and Structural Pattern Recognition: Theory and Applications (1990) 197–234.
[13] F. Serratosa, R. Alquézar, A. Sanfeliu, Function-described graphs for modelling objects represented by sets of attributed graphs, Pattern Recognition 23 (3) (2003) 781–798.
[14] H. Kim, J. Kim, Hierarchical random graph representation of handwritten characters and its application to Hangul recognition, Pattern Recognition 34 (2001) 187–201.
[15] H. Bunke, P. Foggia, C. Guidobaldi, M. Vento, Graph clustering using the weighted minimum common supergraph, in: E.R. Hancock, M. Vento (Eds.), Graph Based Representations in Pattern Recognition, 4th IAPR International Workshop, GbRPR 2003, York, UK, June 30–July 2, 2003, Proceedings, Lecture Notes in Computer Science, vol. 2726, Springer, 2003, pp. 235–246.
[16] M. A. Lozano, F. Escolano, Acm attributed graph clustering for learning classes of images, in: E.R. Hancock, M. Vento (Eds.), Graph Based Representations in Pattern Recognition, 4th IAPR International Workshop, GbRPR 2003, York, UK, June 30–July 2, 2003, Proceedings, Lecture Notes in Computer Science, vol. 2726, Springer, 2003, pp. 247–258.
[17] B. Jain, F. Wysotzki, Central clustering of attributed graphs, Mach. Learn. 56 (1–3) (2004) 169–207.
[18] A. Torsello, E. Hancock, Learning shape-classes using a mixture of tree-unions, IEEE Trans. Pattern Anal. Mach. Intell. 28 (6) (2006) 954–967.
[19] A. Torsello, A. Robles-Kelly, E. Hancock, Discovering shape classes using tree edit distance and pairwise clustering, Int. J. Comput. Vision 72 (2007) 259–285.
[20] X. Jiang, A. Münger, H. Bunke, On median graphs: properties, algorithms, and applications, IEEE Trans. Pattern Anal. Mach. Intell. 23 (10) (2001) 1144–1151.
[21] L. Han, E.R. Hancock, R.C. Wilson, Learning generative graph prototypes using simplified von Neumann entropy, in: X. Jiang, M. Ferrer, A. Torsello (Eds.), GbRPR, Lecture Notes in Computer Science, vol. 6658, Springer, 2011, pp. 42–51.
[22] A. Sanfeliu, R. Alquézar, F. Serratosa, Clustering of attributed graphs and unsupervised synthesis of function-described graphs, in: ICPR, IEEE Computer Society, 2000, pp. 6022–6025.
[23] A. Bagdanov, M. Worring, First order gaussian graphs for efficient structure classification, Pattern Recognition 36 (6) (2003) 1311–1324.
[24] F.R.K. Chung, Spectral Graph Theory (CBMS Regional Conference Series in Mathematics, No. 92), American Mathematical Society, 1997.
[25] R. Kondor, J. Lafferty, Diffusion kernels on graphs and other discrete input spaces, in: Proceedings of the Nineteenth International Conference on Machine Learning (ICML 2002), 2002, pp. 315–322.
[26] A. Smola, R. Kondor, Kernels and regularization on graphs, Lecture Notes in Computer Science 2777 (2003) 144–158.
[27] T. Gärtner, A survey of kernels for structured data, ACM SIGKDD Explorations Newsletter 5 (1) (2003) 49–58.
[28] N. Cristianini, J. Shawe-Taylor, An Introduction to Support Vector Machines, Cambridge University Press, 2000.
[29] B. Schölkopf, A. Smola, Learning with Kernels, MIT Press, 2002.
[30] K.-R. Müller, S. Mika, G. Räshc, K. Tsuda, B. Schölkopf, An introduction to kernel-based learning algorithms, IEEE Trans. Neural Networks 12 (2) (2001) 181–201.
[31] J. Ullmann, An algorithm for subgraph isomorphism, J. Assoc. Comput. Mach. 23 (1) (1976) 31–42.
[32] L. Cordella, P. Foggia, C. Sansone, M. Vento, An improved algorithm for matching large graphs, in: Proceedings of the 3th IAPR Int. Workshop on Graph Based Representations in Pattern Recognition, 2001, pp. 149–159.
[33] A. Cross, R. Wilson, E. Hancock, Inexact graph matching with genetic search, Pattern Recognition 30 (6) (1997) 953–970.
[34] R. Myers, E. Hancock, Least-commitment graph matching with genetic algorithms, Pattern Recognition 34 (2001) 375–394.
[35] L. Herault, R. Horaud, F. Veillon, J. Niez, Symbolic image matching by simulated annealing, in: Proceedings of the British Machine Vision Conference, British Machine Vision Association, 1990, pp. 319–324.
[36] M. Williams, R. Wilson, E. Hancock, Deterministic search for relational graph matching, Pattern Recognition 32 (7) (1999) 1255–1271.
[37] S. Gold, A. Rangarajan, A graduated assignement algorithm for graph matching, IEEE Trans. Pattern Anal. Mach. Intell. 18 (4) (1996) 377–388.
[38] A.M. Finch, R.C. Wilson, E.R. Hancock, An energy function and continuous edit process for graph matching, Neural Comput. 10 (7) (1998) 1873–1894.
[39] M. Pelillo, Replicator equations, maximal cliques, and graph isomorphism, Neural Comput. 11 (8) (1999) 1933–1955.
[40] D. Cvetkovic, M. Doob, H. Sachs, Spectra of Graphs: Theory and Application, Academic Press, 1980.
[41] A. Robles-Kelly, E. Hancock, Graph edit distance from spectral seriation, IEEE Trans. Pattern Anal. Mach. Intell. 27 (3) (2005) 365–378.
[42] A. Sanfeliu, K. Fu, A distance measure between attributed relational graphs for pattern recognition, IEEE Trans. Syst. Man Cybern. 13 (3) (1983) 353–362.

[43] H. Bunke, Error correcting graph matching: on the influence of the underlying cost function, IEEE Trans. Pattern Anal. Mach. Intell. 21 (1999) 917–922.
[44] H. Bunke, On a relation between graph edit distance and maximum common subgraph, Pattern Recognition Lett. 18 (8) (1997) 689–694.
[45] B. Luo, R. Wilson, E. Hancock, Spectral embedding of graphs, Pattern Recognition 36 (10) (2003) 2213–2230.
[46] H. Qiu, E. Hancock, Clustering and embedding using commute times, IEEE Trans. Pattern Anal. Mach. Intell. 29 (11) (2007) 1873–1890.
[47] F. Escolano, E. R. Hancock, M. A. Lozano, Graph matching through entropic manifold alignment, in: CVPR, IEEE, 2011, pp. 2417–2424.
[48] S. Umeyama, An eigendecomposition approach to weighted graph matching problems, IEEE Trans. Pattern Anal. Mach. Intell. 10 (5) (1988) 695–703.
[49] A. Smola, B. Schölkopf, K.-R. Müller, The connection between regularization operators and support vector kernels, Neural Networks 11 (1998) 637–649.
[50] S. Konishi, A. Yuille, J. Coughlan, S. Zhu, Statistical edge detection: learning and evaluating edge cues, IEEE Trans. Pattern Anal. Mach. Intell. 25 (1) (2003) 57–74.
[51] M. Cazorla, F. Escolano, Two Bayesian methods for junction classification, IEEE Trans. Image Process. 12 (3) (2003) 317–327.
[52] M.A. Lozano, F. Escolano, Em algorithm for clustering an ensemble of graphs with comb matching, in: A. Rangarajan, M.A.T. Figueiredo, J. Zerubia (Eds.), EMMCVPR, Lecture Notes in Computer Science, vol. 2683, Springer, 2003, pp. 52–67.
[53] E. Szemerédi, Regular partitions of graphs, in: Problèmes combinatoires et théorie des graphes, Colloques Internationaux, CNRS, vol. 260, 1976, pp. 399–401.
[54] A. Sperotto, M. Pelillo, Szemerédi's regularity lemma and its applications to pairwise clustering and segmentation, in: A.L. Yuille, S.C. Zhu, D. Cremers, Y. Wang (Eds.), EMMCVPR, Lecture Notes in Computer Science, vol. 4679, Springer, 2007, pp. 13–27.

Miguel Angel Lozano received his Bachelors Degree in Computer Science from the University of Alicante (Spain), in 2001. Since 2007 he has been a Lecturer in the Department of Computer Science and Artificial Intelligence of the University of Alicante. He has visited Edwin Hancock's Computer Vision & Pattern Recognition Lab at the University of York, and Liisa Holm's Bioinformatics Lab at the University of Helsinki. He is the head of the Mobile Vision Research Lab of the University of Alicante. His research interests include pattern recognition and computer vision.

Francisco Escolano obtained his Bachelors Degree in Computer Science at the Polytechnical University of Valencia (Spain), in 1992 and his Ph Degree in Computer Science at the University of Alicante, in 1997. Since 1998, he has been an Associate Professor in the Computer Science and Artificial Intelligence Department at the University of Alicante. He has been postdoctoral fellow with Dr. Norberto M. Grzywacz at the Biomedical Engineering Department at the University of South California in Los Angeles, and he has also collaborated with Dr. Alan L. Yuille at the Smith-Kettlewell Eye Research Institute of San Francisco. He also visited the Liisa Holm's Bioinformatics Lab at the University of Helsinki. His research interests are focused on the development of efficient and reliable pattern recognition and computer vision algorithms. He is the head of the Robot Vision Group. He co-chaired and organized the 6th TC15-IAPR Workshop on Graph-based Representations in Pattern Recognition (Gbr), and co-chaired the 7th Gbr workshop. He is Vice-president of the TC2 commitee of the IAPR. He has recently published the book "Information Theory in Computer Vision and Pattern Recognition", Springer-London, July 2009, ISBN: 978-1-84882-296-2. In 2010 he co-organized the Joint IAPR International Workshops on Structural and Syntactic Pattern Recognition (SSPR 2010) and Statistical Techniques in Pattern Recognition (SPR 2010). He also co-organized the workshop "Advanced ITinCVPR in a Nutshell" Anand Rangarajan in CVPR'2010. He has recently co-organized with Alan Yuille, Edwin R. Hancock and Anand Rangarajan the "1st IEEE Workshop on Information Theory in Computer Vision and Pattern Recognition" jointly with ICCV 2011.