International Journal of Pattern Recognition and Artificial Intelligence


# EMBEDDING OF GRAPHS WITH DISCRETE ATTRIBUTES VIA LABEL FREQUENCIES

JAUME GIBERT*
*École Nationale Supérieure d'Ingénieurs de Caen, ENSICAEN*
*Université de Caen Basse-Normandie, 6 Boulevard Maréchal Juin*
*14050 Caen, France*
*jaume.gibert@ensicaen.fr*

ERNEST VALVENY
*Computer Vision Center, Computer Science Department*
*Universitat Autònoma de Barcelona, Edifici O - Campus UAB*
*08193 Bellaterra, Spain*
*ernest@cvc.uab.es*

HORST BUNKE
*Institute of Computer Science and Applied Mathematics*
*University of Bern, Neubrückstrasse 10*
*CH-3012 Bern, Switzerland*
*bunke@iam.unibe.ch*

Graph-based representations of patterns are very flexible and powerful, but they are not easily processed due to the lack of learning algorithms in the domain of graphs. Embedding a graph into a vector space solves this problem since graphs are turned into feature vectors and thus all the statistical learning machinery becomes available for graph input patterns. In this work we present a new way of embedding discrete attributed graphs into vector spaces using node and edge label frequencies. The methodology is experimentally tested on graph classification problems, using patterns of different nature, and it is shown to be competitive to state-of-the-art classification algorithms for graphs, while being computationally much more efficient.

*Keywords*: Discrete Attributed Graphs; Graph Embedding; Graph Classification.

## 1. Introduction

Pattern recognition algorithms are usually designed for feature vector representations of data. Formally, a given family of objects —or more generally, patterns— is represented in terms of numerical vectors the components of which are measurements on the underlying objects. Such measurements are of crucial importance to the problem being solved, and a proper representation in terms of good features

*This work was done while Jaume Gibert was pursuing his Ph.D. title in the Computer Vision Center of the Universitat Autònoma de Barcelona, Spain.

is a key step in successfully solving pattern recognition problems. Industrial quality control, biometric person identification, optical character recognition and many other tasks are prominent examples that have been tackled with feature vector representations[10].

The main characteristic of feature vectors that makes them widely used is the fact that vectorial data are easily processed. Many operations on vectors are easily accomplished due to the flexibility of vectorial spaces. This situation has led the machine learning community to develop a rich set of data processing algorithms that have been successfully applied to multiple kinds of problems. Particular examples include dimensionality reduction via Linear Discriminant and Principal Component Analysis[10], classification by means of Artificial Neural Networks[2] and Support Vector Machines[47], and clustering using $k$-Means and Gaussian Mixture Models[10].

We should, however, point out a main restriction regarding feature vectors as a representational paradigm of patterns. In general, there is no natural way of representing structural relations between parts of the considered patterns under a vectorial representation. Moreover, statistical feature vectors are restricted to be all of the same dimension no matter the differences in terms of the inherent complexity the underlying objects exhibit. These problems are easily solved by using a representation of patterns in terms of graphs. Graphs can represent both numerical measurements and structural relations between these measurements. Numerical measurements can be represented as labels of nodes and the structural relations —together with some properties of such relations— can be represented by edges between nodes. Graph-based representations for pattern recognition have been recently gaining popularity and now form an important branch of research among the pattern recognition community. An interesting survey on graph-based pattern recognition problems and applications can be found in Ref. 8.

Despite their representational power, graphs also exhibit some drawbacks. Mainly, these drawbacks regard the set of available algorithmic tools for processing them and their computational complexity. As a matter of fact, even simple operations between graphs may be cumbersome to define. Classically, the only existing pattern recognition algorithms that are available for processing instances in the graph domain are classifiers of the $k$-Nearest Neighbor family together with some similarity measure between graphs. In the past years, many efforts have been made to overcome such limitations[6]. Two prominent lines of research are graph kernels and graph embedding into vector spaces.

Graph kernels is an important research line that aims at bridging the gap between the structural formalism of graphs and vector-based machine learning algorithms. By defining kernel functions for graphs one is directly capable of applying kernel machines in the domain of graphs. Kernel machines are learning algorithms that are defined in terms of scalar products between input patterns. Mercer's theorem[47] assures that any kernel function is in fact equivalent to a scalar product in some Hilbert space and thus any learning algorithm that is defined in terms of scalar products can be kernelized. By defining a graph kernel one can also use this

property and make kernel machines applicable to the graph domain. Graph kernels have been intensively investigated so far[14]. Diffusion kernels are those kernels that are based on exponentiating a similarity matrix between graphs[29,27]. Convolution kernels build similarity measures between graphs by convoluting similarity measures between smaller parts of the graphs which, in general, are easier to compute[21,52]. Finally, walk kernels are the family of kernel functions that compute the similarity between two input graphs by looking for the common random walks the involved graphs share[25,4,15]. Recently, in Ref. 50, a comprehensive work has been published where a unification of several graph kernels is proposed.

The other prominent line of research is graph embedding into vector spaces. They provide an interesting way of making applicable any learning algorithm originally designed for feature vectors to graph-based representations. The main idea is to define a way to map every graph in a given dataset into a point in a vector space. Several ways of assigning pattern vectors to graph-based representations are known from the literature. A very well known example is the one in Ref. 30. The authors propose a way of assigning a feature vector to every graph by making an eigen-decomposition of its adjacency matrix. From the leading eigen-modes of the adjacency matrix, several unary and binary features are extracted and arranged as the components of a feature vector. For instance, features are defined in terms of leading eigenvalues or inter-mode distances. Another example of graph embedding is the one proposed in Ref. 54. In this case, the authors use the Laplacian matrix and its eigen-decomposition to extract features for a vector representation of graphs. By sampling elementary symmetric polynomials on the eigen-modes of the Laplacian matrices, this approach permits to extract node invariant features for the vectorial representation of graphs. A recent work following this line of research is the one in Ref. 41. The Ihara Zeta function is a model that characterizes a graph in terms of its prime cycles. In Ref. 41, it is used together with the coefficients of a polynomial expansion to properly characterize graphs in a vector form. Another notable work is the one proposed in Ref. 44. Based on the dissimilarity representation formalism[39,40], vectorial representations of graphs are extracted by computing the edit distance of a given graph to a set of graph prototypes. The distances of a graph to these prototypes constitute the components of the final vectorial representation. In fact, this methodology leads to high classification rates on various diverse datasets of graphs. Another family of embedding methods are the so-called isometric embeddings. Given a dissimilarity matrix of graphs, points in a vector space might be built so that they respect as much as possible the original distances in the graph space. A prominent example can be found in Ref. 23, where a PCA-inspired algorithm is applied to a given dissimilarity matrix based on constant shifting embedding. Finally, in the field of chemoinformatics, we also encounter the so-called fingerprint characterization of molecules[16]. Particular molecular substructures (subgraphs) are counted in each molecule and a feature vector is built based on the occurrence of each of these substructures. For instance, in Refs. 28, 22 the authors compute vectorial features based on particular paths or cycles that occur

in every molecule.

As already stated, in contrast to graph kernels which make applicable only kernel machines to the graph domain, graph embeddings make any learning algorithm applicable to graph-based pattern representations. The main drawback that graph embedding methodologies exhibit is, however, their high computational complexity. In some cases, an eigen-decomposition of a matrix has to be performed for every graph. In others, several edit distances to a set of prototypes have to be computed. Finally, in the case of the fingerprint characterization, several subgraph structures need to be sought in every graph which constitutes a hard graph matching problem. In this work we aim at overcoming these limitations and present an efficient graph embedding methodology for graphs with discrete attributes. It is essentially an arrangement of the information stored in the nodes and the edges of the graphs in a vectorial form. This embedding is conceptually simple and has a very low computational complexity. We define a vectorial representation for a discrete attributed graph in terms of frequencies of its node and edge labels. In particular, the vectorial representation we propose has a natural link to the above mentioned random walk family of kernels and the so-called fingerprint characterization of molecules. The proposed features might be understood as 0-length and 1-length walks in the graphs. Random walk kernels provide an implicit formulation of walks of unrestricted length while fingerprint methodologies build up a pattern vector for graphs counting occurrences of particular substructures. The relation between our proposal and these approaches is clear since we only consider a subset of walks in the graphs and these are also counts of simple substructures. More details on these connections as well as a discussion of the advantages of our proposal with respect to these systems will be given in Section 3.4.

The remainder of this paper is structured as follows. In the next section we introduce the basic notation. We also present in detail graph edit distance and a corresponding graph kernel that will serve us as a reference system. In Section 3, we provide a formal description of the proposed embedding methodology and discuss the existing connections with other graph characterization formalisms. In Section 4, we present the experimental part of this work. Finally, Section 5 concludes the article by summarizing the main contributions and discussing open issues.

## 2. Graph Matching

In this section, we recall some concepts related to graphs and graph matching and establish the notation that is going to be used on this paper from now on.

### 2.1. *Basics Concepts*

**Definition 1. (Graph)** A graph $g$ is a four-tuple $g = (V, E, \mu, \nu)$, where $V$ is a non-empty set of nodes, $E \subseteq V \times V$ is the set of edges, $\mu : V \longrightarrow L_V$ is the node labelling function and $\nu : E \longrightarrow L_E$ is the edge labelling function. $L_V$ and $L_E$ are the corresponding sets of labels for the nodes and edges, respectively.

An edge $e \in E$ in a graph $g$ is given by its source and its target nodes, $e = (u, v)$. A graph is called undirected if for each edge $(u, v) \in E$ there exists the edge $(v, u) \in E$. A walk of length $n$ is a sequence of $n + 1$ nodes $(v_1, \ldots, v_{n+1})$, where $v_j \in V, \forall\, j \in \{1, \ldots, n + 1\}$ and $(v_i, v_{i+1}) \in E, \forall\, i \in \{1, \ldots, n\}$. The sets of labels can be of any type, thus allowing for a wide spectrum of graph representations. For instance, the set of node and edge labels can be finite alphabets, sets of numbers, or even sets of vectors. A special labelling function is the one that maps every node (or edge) to the same label, called the null label $\epsilon$. In this situation the graph is said to be node unattributed (or edge unattributed). When one of the labelling functions is of this special type, we can omit it in the definition of the graph. For instance, an edge unattributed graph is defined by $g = (V, E, \mu)$. In this paper, we will mainly work with undirected graphs whose node and edge labelling sets are always finite discrete alphabets: $L_V = \{\alpha, \beta, \gamma, \ldots, \omega\}$ and $L_E = \{a, b, c, \ldots, z\}$.

### 2.2. *Exact and Inexact Graph Matching*

Graph matching is the process by which one is capable to tell how similar or dissimilar two graphs are. Graph matching techniques can be split into two main categories, namely, exact and inexact graph matching. Exact graph matching aims at deciding whether there is a one-to-one map from the nodes of one graph to the nodes of the other graph respecting the topology and the labelling characteristics of the involved graphs. Such a map is called a graph isomorphism and, if there exists a graph isomorphism, the two corresponding graphs are called isomorphic. In the context of exact graph matching, similarity measures can be defined by means of maximum common subgraph and minimum common supergraph[7,13,51]. However, such approaches are often not applicable to pattern recognition problems since the extraction of graphs from patterns is usually a noisy procedure that leads to errors and distortions. Thus, more general algorithms that are capable of coping with structural deformations and labelling errors are needed.

This leads to the definition of the second category of graph matching algorithms, which consists of the so-called error-tolerant, or inexact, methods. Several error-tolerant graph matching algorithms have been proposed in the literature[8]. Because of its broad applicability, we describe in the next section graph edit distance as the main paradigm of error-tolerant graph matching[5,46,45]. In the experimental part of this work, we will use it as a reference method to compare our approach with.

### 2.3. *Graph Edit Distance*

The main idea of graph edit distance is to define a dissimilarity measure between two given graphs by taking into account the minimum number of structural transformations that are needed to convert one graph into the other. Basic edit operations are defined in terms of insertion, deletion and substitution of nodes and edges.

More precisely, given two graphs $g_1$ and $g_2$, a sequence of edit operations $e_1, e_2, \ldots, e_n$ is sought, consisting of the deletion of some nodes and edges of the first

graph $g_1$, substituting some other nodes and edges of $g_1$ by those of $g_2$, and finally inserting the nodes and edges that are needed to obtain the second graph $g_2$. Such a sequence of operations is called an edit path and there is obviously an infinite number of edit paths that transform graph $g_1$ into graph $g_2$. For instance, one could always remove all nodes and edges of the first graph and then insert all nodes and edges of the second graph. This procedure is in fact an edit path transforming $g_1$ into $g_2$, but it might not be regarding the actual structural deformations between both graphs in a proper way. In order to find the most appropriate edit path out of all existing ones, edit costs are assigned to each of the edit operations, corresponding to the strength of each operation. Such edits costs are usually given in terms of a cost function. Defining such a function is a key issue in the application of graph edit distance and several ways have been proposed to tackle the problem[35,36]. The basic idea is to define them in such a way that there exists an inexpensive edit path between two similar graphs and an expensive one between two dissimilar graphs. Formally, the edit distance between two graphs is thus defined as the cost of the edit path that has the minimum cost among all the existing paths:

**Definition 2. (Graph Edit Distance)** Given two graphs $g_1 = (V_1, E_1, \mu_1, \nu_1)$ and $g_2 = (V_2, E_2, \mu_2, \nu_2)$, the edit distance between $g_1$ and $g_2$ is defined as

$$d(g_1, g_2) = \underset{e \in \mathcal{E}(g_1, g_2)}{\operatorname{argmin}} \sum_{i=1}^{n} c(e_i) \tag{1}$$

where $e = (e_1, \ldots, e_n)$ is an edit path from the set $\mathcal{E}(g_1, g_2)$ of all existing edit paths from $g_1$ to $g_2$, and $c(e_i)$ is indicates the cost of the edit operation $e_i$.

Computing the edit distance between two graphs is a problem of high computational complexity. Typical solutions are based on exploring all possible mappings from the sets of nodes and edges of one graph to those of the other graph. Such procedures have an exponential complexity in the number of nodes of the graphs and thus they are restricted to graphs with just a small number of nodes. Suboptimal methods for computing the edit distance of graphs have been widely studied and they allow larger graphs as input. Commonly, suboptimal methods are based on local search in the graphs reducing the search space of possible node-to-node mappings[3,48]. In Ref. 24, the distance between graphs with unlabelled edges can be efficiently computed by a linear programming approach. Two modifications of an optimal algorithm are presented in Ref. 38 in order to speed up the edit distance computation. The idea is to split the graphs into smaller subgraphs and transform the problem into that of finding an optimal match between the sets of subgraphs by dynamic programming. Finally, in Ref. 43, a suboptimal algorithm based on bipartite graph matching is presented. The idea is to use Munkres' algorithm[33] to find the optimal assignment of nodes based on a cost matrix regarding all possible substitutions of the local structure of all nodes in the two involved graphs. This suboptimal approach has been proved to obtained results similar to the optimal edit distance while drastically reducing the computation time. All graph edit distance

computations that are reported in this work will be done by using the suboptimal approach of Ref. 43.

### 2.4. *Relation of Graph Edit Distance to Graph Kernels*

In Ref. 37, graph edit distance is used to define several graph kernels. The idea is to convert a dissimilarity measure into a similarity measure by a monotonically decreasing transformation.

**Definition 3. (Similarity Kernel)** Given two graphs $g_1 = (V_1, E_1, \mu_1, \nu_1)$ and $g_2 = (V_2, E_2, \mu_2, \nu_2)$, the similarity kernel based on the edit distance is defined as

$$\kappa_G(g_1, g_2) = \exp(-\gamma \cdot d(g_1, g_2)) \tag{2}$$

where $d(g_1, g_2)$ is the edit distance between graphs $g_1$ and $g_2$, and $\gamma$ is a positive real-valued parameter, $\gamma > 0$.

Under this kernel, very dissimilar graphs (large edit distance) are given a low value, close to zero, while similar graphs (low edit distance) will produce kernel values close to one. This kernel function is, however, not positive definite since graph edit distance is not generally a definite negative distance[53,9]. Nevertheless, as pointed out in Refs. 37 and 20, there is evidence that proves the utility of indefinite kernels in conjunction with kernel machines under certain specific conditions. This kernel function will also be used as a reference system.

## 3. Embedding of Graphs via Label Frequencies

In this section, we give a formal description of our graph embedding procedure. A graph embedding is a mapping from the domain of graphs into a Euclidean vector space, $\phi : \mathcal{G} \to \mathbb{R}^n$. We define the embedding of a graph into a vector space in terms of single and binary relations between node labels.

### 3.1. *Basic Procedure*

Given a graph $g = (V, E, \mu)$ without edge attributes and with node alphabet $L_V = \{l_1, l_2, \ldots, l_n\}$, a simple vectorial representation of $g$ is, for instance, the one that takes, as each component, the number of times each node label appears in the graph, *i.e.*,

$$\mathbf{x}_g = (\#(l_1, g), \#(l_2, g), \ldots, \#(l_n, g)), \tag{3}$$

where $\#(l_i, g)$ refers to the frequency that $l_i$ happens to be the label of a node in graph $g$. For example, both graphs $g_1$ and $g_2$ in Fig. 1 have node alphabets $L_V = \{A, B, C\}$ and they both have two labels $A$, one $B$ and one $C$. Their respective vectorial representation in this form would be $\mathbf{x}_{g_1} = \mathbf{x}_{g_2} = (2, 1, 1)$.

In the simple example of Fig. 1, one gets the same vectorial representation from both graphs, although the graphs differ in their edge structure. Thus, more

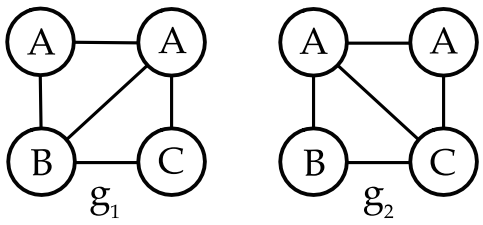



Fig. 1. Two non-isomorphic graphs with the same vectorial representation counting node label appearances.

components should be added to the vectors in order to make this representation more discriminative. This can be achieved by considering not only the labelling frequencies but also the frequencies of the structural links between any two different nodes according to their corresponding attributes. More precisely, the vector representation (3) is enriched by $\mathcal{O}(n^2)$ components of the form

$$\#(l_i \leftrightarrow l_j, g), \tag{4}$$

counting how many edges between every pair of node labels occur in a given graph. With this information at hand, the vectors $\mathbf{x}_{g_1}$ and $\mathbf{x}_{g_2}$ in the example above will no longer be equal because the features $\#(A \leftrightarrow B, g)$ and $\#(A \leftrightarrow C, g)$ are, in fact, different. In the following order,

$$\begin{aligned}\mathbf{x}_g = (&\#(A,g), \#(B,g), \#(C,g),\\ &\#(A \leftrightarrow A, g), \#(A \leftrightarrow B, g), \#(A \leftrightarrow C, g),\\ &\#(B \leftrightarrow B, g), \#(B \leftrightarrow C, g), \#(C \leftrightarrow C, g)),\end{aligned} \tag{5}$$

the vectors $\mathbf{x}_{g_1}$ and $\mathbf{x}_{g_2}$ become

$$\begin{aligned}\mathbf{x}_{g_1} &= (2,1,1,1,2,1,0,1,0),\\ \mathbf{x}_{g_2} &= (2,1,1,1,1,2,0,1,0).\end{aligned}$$

Obviously, these two vectors are now a more proper representation of the graphs in Fig. 1.

### 3.2. *Adding Edge Attributes*

The example given above is built on graphs whose edges are unattributed. Let us now assume that graphs have discrete labels on their edges and that the edge alphabet is $L_E = \{a, b, \ldots, z\}$. In this situation, we should take into account those edges that are attributed with different labels but link nodes with the same attributes. In the example of Fig. 2, both graphs have only two nodes and one edge. The topology and their node labels are the same, but the edge labels differ.

Consequently, the relation $\#(A \leftrightarrow B, g)$ between nodes with labels $A$ and $B$ should be redefined so as to distinguish between different labels. The edge on the


Fig. 2. Two edges with different attributes joining equally labelled nodes.

left graph of Fig. 2 will count as an appearance of the relation between nodes with labels $A$ and $B$, having edge label $a$, this is

$$\#\left([A \leftrightarrow B]_a, g\right),$$

while the edge on the right graph will count for the relation between nodes with label $A$ and $B$, having edge label $b$, this is

$$\#\left([A \leftrightarrow B]_b, g\right).$$

### 3.3. *Formal Definition*

To formalize the distinction of edges with different labels, consider a set of graphs $\mathcal{G} = \{g_1, \ldots, g_N\}$, with $g_i = (V_i, E_i, \mu_i, \nu_i)$ being the $i$th graph in the set with labelling alphabet $L_{V_i}$ for the nodes and $L_{E_i}$ for the edges. We assume that all graphs in $\mathcal{G}$ have the same labelling alphabets, this is $L_{V_i} = L_{V_j}$ and $L_{E_i} = L_{E_j}$ for all $i, j \in \{1, \ldots, N\}$. We do not assume, however, that each node and edge label occurs in each graph. Let $L_V = \{\alpha_1, \ldots, \alpha_p\}$ and $L_E = \{\omega_1, \ldots, \omega_q\}$ be the common labelling alphabets.

For each graph $g = (V, E, \mu, \nu) \in \mathcal{G}$, we define $p$ unary features regarding the number of times each label in $L_V$ appears in the graph, this is

$$U_i = \#(\alpha_i, g) = |\,\{v \in V \,|\, \alpha_i = \mu(v)\}\,|. \tag{6}$$

For the edges we will distinguish two different scenarios. We will construct features in the case where edges of the graphs remain unattributed and features in the case of attributed edges. Binary features for edge unattributed graphs are defined by

$$\begin{aligned} B_{ij} &= \#(\alpha_i \leftrightarrow \alpha_j, g) \\ &= |\,\{e = (u, v) \in E \,|\, \alpha_i = \mu(u) \wedge \alpha_j = \mu(v)\}\,|, \end{aligned} \tag{7}$$

and binary features for edge attributed graphs are defined by

$$\begin{aligned} B^k_{ij} &= \#\left([\alpha_i \leftrightarrow \alpha_j]_{\omega_k}, g\right) \\ &= |\,\{e = (u, v) \in E \,|\, \alpha_i = \mu(u) \wedge \alpha_j = \mu(v) \wedge \omega_k = \nu(e)\}\,|. \end{aligned} \tag{8}$$

These two sets of edge features —based on whether we consider the attributes on edges or not— give rise to two different vectorial representations of a given graph from $\mathcal{G}$.

**Definition 4. (Graph Embedding)** For the edge unattributed case, the embedding of a graph $g \in \mathcal{G}$ into a vector space is defined as the vector

$$\varphi_1(g) = (\, U_1, \ldots, U_p, B_{11}, \ldots, B_{ij}, \ldots, B_{pp} \,), \tag{9}$$

and for the edge attributed case as

$$\begin{aligned} \varphi_2(g) = (\, & U_1, \ldots, U_p, \\ & B^1_{11}, \ldots, B^1_{ij}, \ldots, B^1_{pp} \\ & B^2_{11}, \ldots, B^2_{ij}, \ldots, B^2_{pp}, \\ & \vdots \\ & B^q_{11}, \ldots, B^q_{ij}, \ldots, B^q_{pp} \,), \end{aligned} \tag{10}$$

where $1 \leq i \leq j \leq p$, and $U_i$, $B_{ij}$ and $B^k_{ij}$ are defined as in Eq. (6), (7) and (8), respectively.

### 3.4. *On the relation to other graph kernels and embeddings*

As already pointed out in the introductory section of this work, the embedding features that have been presented in the previous sections have a close relation to other existing methodologies for graph characterization. In particular, we find interesting connections to the fingerprint methodologies and to the random walk family of kernels.

**Fingerprint methods** Fingerprint characterizations aim at describing the topological structure of graphs by counting the appearance of particular substructures in every graph. Formally, a graph is encoded using vectorial features which regard the number of times each of these substructures is present in it. Given a set $\mathcal{H} = \{h_i\}_{1 \leq i \leq n}$ of $n$ graph substructures, a graph $g$ is embedded into an $n$ dimensional space by

$$g \longmapsto (\#(h_1, g), \#(h_2, g), \ldots, \#(h_n, g)), \tag{11}$$

where $\#(h_i, g)$ is the number of times the graph substructure $h_i \in \mathcal{H}$ appears in the graph $g$.

It is clear that the embedding methodology proposed in this work is a special case of this way of characterizing graphs in terms of feature vectors. In our framework, the set $\mathcal{H}$ is simply the set of all nodes with a certain label and the set of all node-edge-node walks with all possible label sequences. In the case of molecule characterization —where this approach has shown successful results—, the set of substructures is carefully chosen based on prior chemical knowledge[31]. Such prior knowledge may be available in a chemical task, but might not be at hand in the general case. In our case, we want to be as general as possible and so we avoid the problem of substructure selection by considering all discrete labels and pairs of labels that occur in the graphs. In particular, the finding of these substructures is a

much easier step than the case of more complex substructures, which will constitute a costly graph matching problem.

Indeed, the complexity of building a system in the general fingerprint characterization is high since one has to decide which substructures need to be sought. On top of that, one has to find a suitable number of such substructures. By contrast, the complexity of building a system in our embedding framework is very low since there are no such free meta-parameters. The number of node and edge labels to consider is firmly defined and there are no choices.

**Random walk kernels** Generally speaking, random walk kernels try to infer the similarity between two graphs by considering the amount of random walks the involved graphs share. Formally, graphs are implicitly embedded into an infinite dimensional space where each feature regards the number of times a specific sequence of node and edge labels appear in the graph. The graph similarity is computed by the standard dot product in this implicit Hilbert space.

The idea was originally proposed in Ref. 15, where it is also shown an efficient way to compute such kernel function by means of the direct product graph. The direct product graph $G_\times$ of two graphs $g_1$ and $g_2$ is the graph whose nodes are pairs of nodes of $g_1$ and $g_2$ with the same labels, and edges link nodes in the product graph whenever the corresponding pairs of nodes were also connected in the original graphs. The key idea is to realize that walks in the direct product graph $G_\times$ are common walks between $g_1$ and $g_2$.

Formally, let $A_\times$ be the adjacency matrix of the direct product graph of $g_1$ and $g_2$. The random walk kernel is defined by

$$\kappa_\times(g_1, g_2) = \sum_{i,j=1}^{|\mathcal{V}_\times|} \left[ \sum_{k=0}^{\infty} \lambda_k A_\times^k \right]_{ij} \tag{12}$$

where $\mathcal{V}_\times$ denotes the set of nodes of the direct product graph and $\lambda = \lambda_0, \lambda_1, \ldots$ ($\lambda_k \in \mathbb{R}; \lambda_k \geq 0, \forall k \in \mathbb{N}$) is a sequence of weights. In particular, exponentiating the adjacency matrix to the power of $k$ has an interpretation in terms of how many walks of length $k$ are there between two nodes of the direct product graph.

With respect to our embedding methodology, we build similar sets of features. We count how many nodes are there with a certain label (all possible walks of length 0) and how many edges with a certain label are there between two specific node labels (all possible walks of length 1). Once these features are built, we may compare those for every two graphs doing standard vector operations for comparison.

All in all, the relation of the random walk kernel with the proposed embedding methodology is the fact that we actually build a subset of the infinite set of features that this kernel methodology implicitly builds. The main advantage over it is, however, the fact that we explicitly build these features and so more flexible comparisons can be made than just the standard dot product. Also, the efficiency of the random walk kernel is governed by the infinite sum in Eq. (12), which is

quite more complex than our approach.

**Discussion** The random walk kernel is in fact an infinite dimensional fingerprint characterization of graphs, counting the appearance of all possible label sequences in the graphs. Both methodologies are thus closely related. However, as the selection of substructures in the fingerprint characterization is based on prior knowledge and it is not easy to define in the general case, we will only use the random walk kernel as a reference method to compare our approach with.

Extensions of the random walk kernel exist in the literature. For instance, in Ref. 25, a marginalized version of it is proposed by considering expectation of walks in the graphs instead of just their presence. Authors of Ref. 31 extend this very same idea in order to avoid backtracking walks in graphs. Such extensions usually perform better than the original formulation of the kernel. In any case, we will use the original one since —as we have already discussed— the features we compute are a particular subset of the implicit features of the random walk kernel and thus we understand it is a good trade-off between the family of random walk kernels and the fingerprint methods for graph comparison.

## 4. Experiments

In the experimental evaluation, we aim at discovering the differences between the two vector embedding systems proposed in the last section in the context of various classification tasks, and compare their performance with two reference systems. Results will be given in terms of classification accuracies. We will describe the datasets used in the experiments, explain the experimental setup, and show the results that were obtained.

### 4.1. *Databases*

We have used 6 datasets of graphs for evaluation of the proposed methodology. All graphs employed in this work have discrete attributes. The datasets are split into two main categories regarding the nature of patterns they represent. The first group of graphs represent objects in images and the second one represent molecule structures.

#### 4.1.1. *Object datasets*

We construct graphs with discrete attributes from three publicly available large image databases. All databases represent different objects under a rotating viewpoint. In particular, we used the Columbia Object Image Library (COIL)[34], the Amsterdam Library of Object Images (ALOI)[17], and the Object Databank by Carnegie-Mellon University (ODBK)[49].

The COIL dataset is a collection of images of 100 different objects. Pictures of each object are taken at intervals of 5 degrees of rotation, leading to 72 images per

object and a total of 7200 images. Examples of such images can be found in the top row of Fig. 3. From the 72 images of each object, we have selected 24 for training (one at each 15 degrees of rotation) and from the remaining ones, we randomly selected 5 for validation and 10 for testing. This leads to a training set of 2400, a validation set of 500, and a test set of 1000 images.

The ALOI dataset is a generalization of the COIL dataset. Images of 1000 objects are acquired by changing the illumination and the view pose of the objects several times, leading to a total amount of 110,250 images. In the middle row of Fig. 3, some examples can be found. To keep the computational time within reasonable limits, we only use 50 randomly picked objects from the 1000 categories in this dataset. For the 72 different images of an object at 5 degrees of rotation, we proceed as for the COIL dataset: we keep 24 images (one at each 15 degrees of rotation) for training and, from the remaining ones, we randomly select 5 for validation and 10 for testing. The training set thus consists of 1200, the validation set of 250, and the test set of 500 images.

Finally, the ODBK dataset is rather different from COIL and ALOI. It is a collection of 209 different object models that have been rendered with photo-realistic quality using 14 different viewpoints. The bottom row of Fig. 3 shows some examples. We select 100 of the 209 objects. Out of the 14 viewing points of each object model, 12 are views arising from 30 degrees rotation and 2 are the top and the bottom views. We only keep the 12 rotated viewpoints, 6 of which are used for training (one at each 60 degrees of rotation) and the remaining six are randomly put either in the validation or in the test set. We eventually have a training set of 600 images, and a validation and test sets of 300 images each.

The graph extraction process is the same for the three datasets and it is illustrated in Fig. 4. Given the original image (Fig. 4(a)), we first segment it (Fig. 4(b)) using the graph-based image segmentation approach[a] described in Ref. 12. We crop the meaningless regions by convoluting the image with a mask (Fig. 4(c)) that distinguishes the object from the background. The mask is constructed by first thresholding the grey-level image to remove the background and then closing possible holes in the object by mathematical morphology operations. We finally obtain a segmented version of the original object (Fig. 4(d)).

From the segmented object images we extract discrete attributed graphs in the following way. Each region in the image is assigned to one of the eleven colors of the color naming theory. In Ref. 1 the authors proposed a model by which an RGB value is assigned to one of the eleven basic colors[b]. We label each region in our segmented objected by the color naming output of this model when providing the mean RGB value of all pixels in the region. By doing so, our node labelling alphabet

[a]The source code of a C++ implementation is publicly available at the author's web page: `http://people.cs.uchicago.edu/~pff/segment/`

[b]A Matlab implementation is also publicly available in the author's web page: `http://www.cat.uab.cat/Publications/2008/BVB08/`

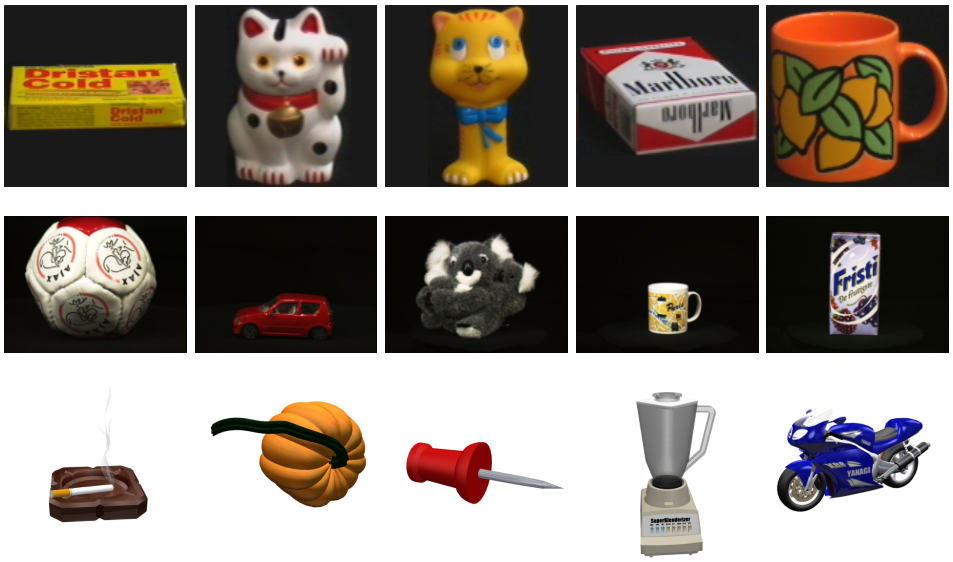


Fig. 3. Examples of images in the object datasets.

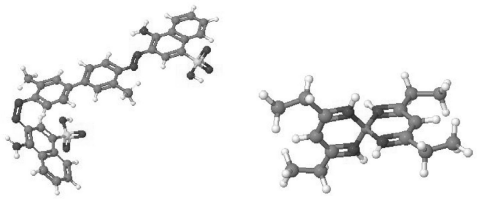
(a) Original image (b) Segmented image (c) Image mask (d) Result

Fig. 4. Object segmentation for graph extraction.

will be

$$L_V = \{\, Black,\, Blue,\, Brown,\, Green,\, Grey,\, Orange,\\ Pink,\, Purple,\, Red,\, White,\, Yellow \,\}. \tag{13}$$

Regarding the edges of the graphs, we link all adjacent regions, this is, we put an edge between every two regions whose borders have neighbouring pixels. In order to label these edges, we compute the length —in pixels— of the common border between two adjacent regions. Such length is normalized by the sum of the lengths of all common borders in the image. After such normalization, the interval $[a, b] \in \mathbb{R}$, where $a$ and $b$ are the shortest and the longest normalized common borders in the graph respectively, is further discretized into three equal bins regarding the amount of pixels they share. The edge of those regions that share a short border —this is,



(a) Active (b) Non-active

Fig. 5. Examples of molecules from the AIDS dataset.

those in the first bin of the discretization— will be labelled by *Short*, those falling in the second bin by *Medium* and those in the third by *Long*. The edge alphabet is thus

$$L_E = \{\, Short,\, Medium,\, Long \,\}. \tag{14}$$

4.1.2. *Molecule datasets*

The second group of datasets is formed by three molecule datasets. Molecule compounds are certain biological structures that are easily represented by graphs. Atoms in the molecules are represented by nodes whose labels are the corresponding atomic elements. Edges represent the covalent bonds and they are labelled with the corresponding bond type.

The first dataset of molecules is the AIDS database from the IAM Graph Database Repository[42]. Graphs are constructed from the AIDS Antiviral Screen Database of Active Compounds[11]. They consist of molecules that are either positive or negative against HIV activity. In Fig. 5 an example of a molecule of each class is shown. For training and validation purposes, 150 molecules are used in each set, while 1500 are used for testing. In this dataset, 21 different atomic elements happen to be the label of a node in the training set. Thus the size of the node label alphabet is 21. For the edges, only three labels are there: single, double, and triple bonds.

The second dataset of molecules is also taken from the IAM Graph Database Repository[42]. The Mutagenicity dataset (MUTAG) is composed of molecules that are divided into two classes, *mutagen* and *non-mutagen*. The mutagenicity of a molecule is a biological property that restricts its potential to become a commercial drug[26]. In this dataset, 1500 molecules are used for training, 500 for validation and 2337 for testing. Moreover, 13 different atomic elements constitute the node label alphabet, while again there are three different types of atomic bonds represented by the edges labels.

Finally, the last molecule dataset is the Monoamine Oxidase dataset (MAO), which includes molecules that are either active or inactive with respect to inhibiting the monoamine oxidase[32]. Depending on their response, one can pharmacologically

Table 1. Characteristics of the different datasets. Size of the training (tr), validation (va) and test (te) sets, the number of classes for each dataset (#classes), the average number of nodes and edges ($\pm|V|/\pm|E|$), the maximum number of nodes and edges ($\max|V|/\max|E|$) and the size of the node ($|L_V|$) and the edge ($|L_E|$) label alphabets.

| Dataset | Size (tr, va, te) | #classes | $\pm\|V\|$ | $\max\|V\|$ | $\pm\|E\|$ | $\max\|E\|$ | $\|L_V\|$ | $\|L_E\|$ |
|---|---|---|---|---|---|---|---|---|
| COIL | 2400, 500, 1000 | 100 | 29.0 | 68 | 64.8 | 171 | 11 | 3 |
| ALOI | 1200, 250, 500 | 50 | 22.9 | 78 | 49.6 | 204 | 11 | 3 |
| ODBK | 600, 300, 300 | 100 | 15.2 | 62 | 30.8 | 172 | 11 | 3 |
| AIDS | 250, 250, 1500 | 2 | 15.7 | 95 | 16.2 | 103 | 21 | 3 |
| MUTAG | 1500, 500, 2337 | 2 | 30.3 | 417 | 30.8 | 112 | 13 | 3 |
| MAO | 24, 22, 22 | 2 | 18.3 | 27 | 19.6 | 29 | 3 | 3 |

treat them as antidepressant drugs. This dataset is rather small when compared to the previous ones and there is no natural training/test set separation of the data. The dataset has 38 active and 30 inactive molecules. In order to be able to use the same evaluation protocols as for all other datasets, we have created 20 splits in which we create a training set of 24 molecules (14 active, 10 non-active) and a validation and test sets of 22 molecules each (12 active, 10 non-active). The experimentation will be carried on each of these 20 splits and we will provide average results over them. Only three atomic elements (carbon, nitrogen and oxygen) appear as node labels in molecules, and also three kinds of atomic bonds are represented as edge labels.

As a summary, in Table 1 we show the main characteristics of each dataset that is used in this work.

### 4.2. *Experimental Setup and Validation of parameters*

As already mentioned, we want to compare the two vectorial representations of graphs that are proposed in Section 3. We will do so by measuring the recognition rates of two different classifiers, a $k$NN and an SVM classifiers, both working on the extracted feature vectors. We will compare the results with two reference systems working directly on the original graph representations. In particular, a $k$NN and an SVM classifers will be build upon graph edit distance and the random walk kernel.

*Graph edit distance*

The first reference system is a $k$-Nearest Neighbor ($k$NN) classifier on graph edit distances. The graph edit distance is computed using the suboptimal approach described in Ref. 43. Edit cost functions are defined as follows for the two different types of datasets (objects and molecules). Node insertions and deletions have a fixed cost $c(\epsilon \rightarrow u) = c(u \rightarrow \epsilon) = \tau_n$. Edge insertions and deletions have also a fixed cost $c(\epsilon \rightarrow e) = c(e \rightarrow \epsilon) = \tau_e$. The parameters $\tau_n$ and $\tau_e$ are properly optimized using the validation sets.

For node substitutions we define the cost equal to 0 when the involved nodes

have the same label, and equal to $2 \cdot \tau_n$ when the involved nodes have different labels:

$$c(u \to v) = \begin{cases} 0, \text{ if } \mu(u) = \mu(v), \\ 2 \cdot \tau_n, \text{ otherwise.} \end{cases} \tag{15}$$

For edge substitutions in the set of object datasets, we assume a cost that is analogous to that of node substitutions, *i.e.* we define a cost of $2 \cdot \tau_e$ when labels are different and a cost equal to 0 otherwise. In the case of the molecules datasets, we assume that edge substitutions are free of cost (this is justified by the findings reported in Ref. 44, which indicate that edge substitution costs have almost null impact on the final edit distance).

There is a third parameter $\alpha \in (0, 1)$ governing the amount of weight that one assigns to node and edge operations. Node costs are multiplied by $\alpha$ and edge costs by $1 - \alpha$.

Thus, for each dataset, we have to optimize the triplet of parameters $(\tau_n, \tau_e, \alpha)$, plus the number of neighbours $k$ in the $k$NN classifier. To do so, we initially compute the distance of all graphs in the validation set to all graphs in the training set. We run these computations using several values in the triplet of parameters to validate. Out of all distance matrices that are obtained, we perform a $k$NN classification for several values of the parameter $k$. Those four parameters $(\tau_n, \tau_e, \alpha)$ and $k$ that give the best recognition rate are eventually used on the test set. Distances of all graphs in the test set are computed to all graphs in the training set using the optimal triplet of parameter values and then $k$NN is applied using the optimal value of $k$.

The second reference system is a Support Vector Machine (SVM) classifier in conjunction with the graph kernel defined in Section 2.4. In this case, we do not validate the costs of the edit distance again but we use those parameter values that give an optimal performance for the $k$NN classifier. As we train an SVM model, besides the $\gamma$ parameter in the kernel function, we need to validate the cost parameter $C$ of the SVM. Several values of the couple $(\gamma, C)$ are used to train SVM models. Those parameter values that give the best performance on the validation set are eventually used on the test set.

*Random walk kernel*

For the case of the random walk kernel, also a $k$NN and an SVM classifiers will be built. SVMs are trained by computing all pairwise kernel values using Eq. (12). In particular, the $\lambda_k$ parameters are set to be $\gamma^k$ for $\gamma \in (0, 1)$. This way short walks in the graphs are given more impact than longer ones. In this case, the infinite sum that defines the kernel can be computed[14,15] by

$$\lim_{k \to \infty} \sum_{i=0}^{k} \gamma^k A_\times^k = (I - \gamma A_\times)^{-1}. \tag{16}$$

Parameter $\gamma$, as well as the cost parameter $C$ in the SVM procedure are validated

by using several values of the pair $(\gamma, C)$, and check for those that obtain a best recognition rates in the validation set. Such a pair will be eventually used in the test sets.

Let $\phi(g_1)$ and $\phi(g_2)$ be the infinite dimensional feature vectors that the random walk kernel implicitly builds for graphs $g_1$ and $g_2$, respectively. We have the following relation

$$\begin{aligned} d^2_{\times}(g_1, g_2) = ||\phi(g_1) - \phi(g_2)||^2 &= \langle \phi(g_1), \phi(g_1) \rangle + \langle \phi(g_2), \phi(g_2) \rangle - 2\langle \phi(g_1), \phi(g_2) \rangle \\ &= \kappa_{\times}(g_1, g_1) + \kappa_{\times}(g_2, g_2) - 2\kappa_{\times}(g_1, g_2). \end{aligned} \tag{17}$$

This is, the Euclidean distance between the implicitly computed vectors can be computed in terms of the kernel values. We use this relation in order to compute distance values between pairs of graphs. A $k$NN classifier can thus be applicable. Parameter $\gamma$ in the random walk kernel computation and parameter $k$ in the $k$NN procedure are validated as in the previous cases.

*Embedding approaches*

To evaluate the performance of the systems proposed in this work, we also use the $k$NN and SVM classifiers. This way, we can directly assess the impact of the proposed graph embedding regardless of the classification strategy. With respect to the $k$NN classifier, we use it in conjunction with two different distances, namely, the Euclidean distance and the $\chi^2$ distance. The Euclidean distance is used so a more comprehensive analysis of the behaviour of the methods can be made since the random walk kernel is also evaluated using an Euclidean measure between the corresponding feature vectors. We recall here that

$$d_{L_2}(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}. \tag{18}$$

for two vectors $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ in $\mathbb{R}^n$. Being our embedding representations histogram-based we also use the $\chi^2$ distance since it is a commonly used metric for this type of feature vectors[55]. It is defined by

$$d_{\chi^2}(x, y) = \frac{1}{2} \sum_{i=1}^{n} \frac{(x_i - y_i)^2}{(x_i + y_i)}. \tag{19}$$

Distances between the vectors in the validation set to those in the training set are computed using Eqs. (18) and (19). The only parameter that needs to be validated is the $k$ parameter in the $k$NN classifier.

With respect to the SVM classifier, we use the standard linear kernel $\langle \cdot, \cdot \rangle$ and a $\chi^2$ kernel for the training step. The linear kernel —or dot product— is used since it is the one that is actually used between the corresponding feature vectors in the random walk approach. Since we explicitly have the features, we may compute

more appropriate kernel functions as the $\chi^2$ kernel. It is defined analogously to the similarity kernel (Section 2.4) and also to the radial basis function kernel[c]:

$$\kappa_{\chi^2}(x, y) = \exp(-\gamma \cdot d_{\chi^2}(x, y)), \tag{20}$$

for two vectors $x, y \in \mathbb{R}^n$ and $\gamma > 0$. For the linear kernel only the cost parameter in the SVM model is validated. For the $\chi^2$ one, this parameter is validated together with the $\gamma$ one.

We recall here that the evaluation of the reference and the proposed systems on the MAO dataset is done using exactly the same protocol as describe above but repeating the experiments on 20 different splits of the data. We will report average results on the test set.

### 4.3. *Results on the Test Set*

In this section we report the results on all datasets using the protocols explained in the previous section. In particular, we will first discuss the runtime of all the algorithms compared, and then present the classification results.

*Computational times*

Before comparing the quantitative results, we want to put special emphasis on the computational time that all of the involved systems need at the testing phase. We do not consider the time required for applying the corresponding classifiers as it is the same in all representations. We just consider the time required for the distance and kernel computations and the construction of the embedding representations of the graphs. For a test graph to be evaluated under the first reference system, we need to compute the graph edit distance to all graphs in the training set. Using these distances $k$NN can be applied. For SVM using the similarity kernel on graphs (Eq. (2)), in addition to the computation of edit distances, we have to exponentiate them to obtain the kernel values (Eq. (2)). In the case of the random walk kernel, for the SVM classifier we first need to compute each kernel value between a test graph and all train graphs (Eqs. (12) and (16)). For the $k$NN classifier, distances between the same graphs are computed using Eq. (17). Regarding the embedding methods, the first step is always to construct the vector representation. Afterwards, in order to apply the $k$NN classifier, the Euclidean and $\chi^2$ distances of a test element to all training elements have to be computed. In the case of the SVM classifiers, there is a final step in which kernel values are computed. For both the reference systems and the proposed ones, the SVM classifiers actually only need the kernel values of each test element to the support vectors. Nevertheless, we show all computations for a more consistent comparison with $k$NN.

[c]A more consistent choice for the $\chi^2$ kernel with respect to the underlying metric would be $\kappa_{\chi^2}(x, y) = d_{\chi^2}(x, z) + d_{\chi^2}(y, z) - d_{\chi^2}(x, y)$, for any $z \in \mathbb{R}^n$. However, the described formulation provides more favorable results since the $\gamma$ parameter allows to adapt to each specific problem.

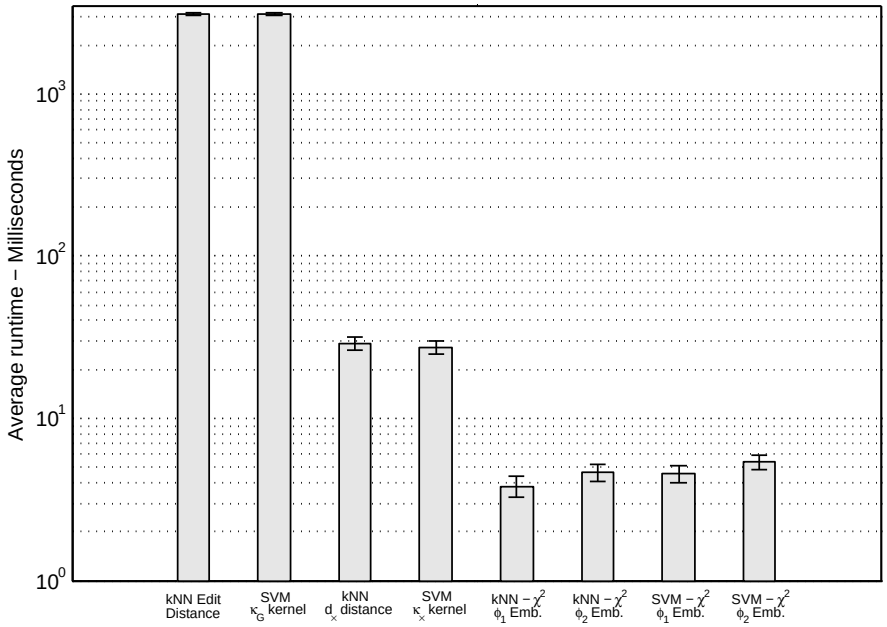



Fig. 6. ODBK dataset. Average times in milliseconds (vertical axis, logarithmic scale) of the test stage of both the reference and the proposed systems. The length of the ticks on the top of the bars show the variance around the mean values.

In Fig. 6, we show results on the ODBK dataset. In particular, for the embedding representations we only show the results for the $\chi^2$-based classifiers, since the runtimes of the Euclidean distance and the linear kernel are comparable. ODBK is the dataset with the smallest average number of nodes per graph, and so it is the most favorable case for edit distance and random walk kernel computations. In any case, results have shown that the relative behavior of the graph-based versus the embedded vector representations is basically the same independently of the dataset and the size of the graphs.

One can observe that the proposed systems are several orders of magnitude faster when compared to the reference systems. Even in the most favorable case for the reference systems, the proposed embedding systems are about $10^3$ times faster with respect to edit distance and 10 times with respect to the random walk approaches. In all other datasets, we observe a similar behavior. Apart from that, validating the parameters of the reference systems is indeed a hard task. Several parameters are there such as node and edge costs and the weighting factor in the edit distance or the variable balancing the effect of the length of walks in the random walk approach. In the proposed embedding methodology, one has no parameters to tune at the training stage and so the validation is done much more efficiently. Regarding a comparison between the two proposed representations, $\varphi_1$ clearly shows

Table 2. $k$NN results on the test set. Accuracy rates are given in %.

| | Reference Systems | | Embedding Systems | | | |
|---|---|---|---|---|---|---|
| | | | Euclidean distance | | $\chi^2$ distance | |
| Dataset | Graph Edit Distance | Random walk distance $d_\times$ | $\varphi_1$ Emb. | $\varphi_2$ Emb. | $\varphi_1$ Emb. | $\varphi_2$ Emb. |
| COIL | 77.0 | 74.9 | 79.9 | 74.0 | 88.1 | 83.1 |
| ALOI | 85.8 | 84.2 | 87.8 | 85.4 | 92.2 | 90.0 |
| ODBK | 70.0 | 68.3 | 69.6 | 66.6 | 76.6 | 73.6 |
| AIDS | 94.9 | 86.4 | 99.2 | 99.2 | 98.4 | 98.4 |
| MUTAG | 66.9 | — | 71.4 | 71.3 | 73.6 | 74.9 |
| MAO | 82.2 | 80.6 | 89.7 | 86.3 | 84.3 | 82.2 |

— Results not available due to memory requirements.

Table 3. SVM results on the test set. Accuracy rates are given in %.

| | Reference Systems | | Embedding Systems | | | |
|---|---|---|---|---|---|---|
| | | | Linear kernel | | $\chi^2$ kernel | |
| Dataset | Dissimilarity kernel $\kappa_G$ | Random walk kernel $\kappa_\times$ | $\varphi_1$ Emb. | $\varphi_2$ Emb. | $\varphi_1$ Emb. | $\varphi_2$ Emb. |
| COIL | 85.7 | 82.8 | 85.2 | 84.5 | 90.7 | 89.7 |
| ALOI | 91.0 | 87.6 | 92.8 | 92.2 | 95.2 | 93.2 |
| ODBK | 80.0 | 76.0 | 74.6 | 77.0 | 79.6 | 78.0 |
| AIDS | 97.0 | 90.8 | 99.6 | 99.5 | 99.4 | 99.4 |
| MUTAG | 68.6 | — | 71.9 | 73.4 | 74.9 | 76.5 |
| MAO | 84.7 | 82.3 | 75.2 | 72.7 | 90.6 | 87.7 |

— Results not available due to memory requirements.

faster results than $\varphi_2$ due to its lower dimensionality.

*Classification performances*

Next, we consider the recognition performance of all systems. In Tables 2 and 3 the recognition rates of the $k$NN and the SVM classifiers are shown, respectively. We shall first discuss our embedding representations. We notice that SVM results outperform those for the $k$NN classifiers in all cases (also for the reference systems) except for the linear SVMs of the MAO dataset. This result is most likely due to the strength of the SVM classifier with respect to the $k$NN. We also observe that the $\chi^2$ distance and the $\chi^2$ kernel usually provide more successful results than the Euclidean distance and the linear kernel in the corresponding classifiers. In fact, the $\chi^2$ measure (Eqs. 19 and 20) disregards small differences on those features with large values. This seems a convenient way to proceed with our vector representations of graphs. In particular, we should understand that, between two graphs, a difference of 10 nodes with a specific label ought to be more meaningful whenever one graph has none of these nodes than when it has a large number of them.

Another interesting observation is that, except for very few cases, the performance of the embedding representation in which edge labels are not taken into account ($\varphi_1$) is equal to, or outperforms, that of the representation in which this particular information is taken into account ($\varphi_2$). Theoretically, one would expect

it the other way around since more information is being included in the feature vector representation when edge labels are taken into account. However, by considering such features, the structural information of the graphs falls apart, and the relevant information is divided into many isolated pieces that eventually may lead to a lower performance. This behaviour may also be explained by the larger dimensionality of the $\varphi_2$ representation with respect to $\varphi_1$ because larger vectors are more prone to overfitting. In any case, the fact that edge labels are not helping in the classification task under this methodology is in compliance with the observation already mentioned regarding the null impact of edge costs in the graph edit distance computation for molecules[44]. Anyway, this situation is beneficial for the proposed methodology in the sense that the resulting feature vectors of the embedding systems have less components (less features are taken into account) and thus the complexity of the classifiers that are eventually used is drastically reduced.

Regarding a comparison of the proposed embedding methodology with the reference systems, we see how the proposed graph embedding procedures, when using the Euclidean measures, are at the same levels of classification performance as the random walk. Such a behavior is expected since, as we have discussed, our features are a subset of the features the random walk kernel implicitly computes. In any case, whenever the $\chi^2$ metrics are used, the proposed embedding methodologies provide more successful results, suggesting that the explicit manipulation of the features is indeed an interesting approach to tackle this problem. Regarding a comparison with graph edit distance, in almost all cases, we also observe we obtain an improvement with respect to it.

Even in case no significant improvement over the reference systems is achieved, the proposed approach still seems to be a better option since the computational complexity is significantly lower than that of the reference systems. Computing the suboptimal solution of the edit distance[43] between two input graphs is a procedure that requires a cubic number of operations in the number of nodes of the involved graphs. Computing the random walk kernel requires the inversion of a matrix which in the worst case has dimension $n^2 \times n^2$, where $n$ is the number of nodes of the involved graphs. In particular, for the MUTAG dataset whose average number of nodes is the highest, memory requirements would not let compute such kernels. In order to classify any graph in the test set, these computational procedures have to be done against all elements in the training set, and this certainly requires much more operations than just arranging the graphs in the vectorial form that is proposed in this work and then computing the distances among them.

The proposed embedding systems are based on counting appearances of node labels in the input graph and appearances of specific edges between them. It seems that their simplicity, both conceptually and computationally, and their good performance on classification problems involving datasets of graphs with discrete attributes on nodes and edges make them an attractive new tool for graph-based pattern recognition problems.

## 5. Conclusions

Graph representations are an elegant and powerful tool for pattern representation. They offer the possibility of representing not only numerical features of the underlying patterns, but also structural relations among parts of these patterns. The complexity of this representational paradigm is, however, responsible for making the treatment and processing of graphs a hard problem.

A solution to this problem is provided by graph embedding in vector spaces. By mapping every graph in a given dataset to a feature vector, all machine learning algorithms that were originally designed for vector-based representations happen to become applicable to graph input patterns. In this work we have presented a way of embedding a set of graphs with discrete attributes into vector spaces by counting the appearance of the node and edge labels. We have distinguished among two different representations of graphs depending on whether we take care of edge labels or not. These methodologies are conceptually straightforward and can be regarded as a redistribution of the information inherent to the nodes and edges of a graph into a vectorial form. They are, thus, computationally very efficient. In particular, we have discussed the connections of the embedding methodology we propose with the fingerprint characterization of molecular structures and the random walk family of graph kernels. It turns out that ours are particular cases of these methodologies but are defined in such a way that we are able to represent graphs with feature vectors in a more efficient way.

The proposed embedding systems are specifically suitable for graphs with discrete labelling alphabets. We have tested the proposed methodology using several datasets of graphs representing object images and molecular compounds. In the case of the object images, segmented regions are labelled with a color name and edges with discretized lengths of the border of adjacent regions. Molecules are easily represented by nodes with atomic elements as labels and edges with the corresponding covalent bond. The experimental part of this work compares the embedding representation of graphs with two classifiers in the graph domain, namely, a the edit distance of graphs and the classical formulation of the random walk kernel. The results indicate that this methodology is a valuable approach to enrich the repertoire of processing tools for graph-based pattern recognition.

This work is part of a more general study in which not only graphs with discrete attributes are considered. In Refs. 18 and 19, the same authors propose a related methodology for embedding a set of graphs with continuous attributes by counting appearances of a set of node label representatives. While the work reported in Refs. 18 and 19 is of rather preliminary nature, and deals with a different class of graphs, the current paper provides a much more comprehensive and in-depth treatment, and features a significantly extended set of experiments. Future work will focus on extending the methodology and experiments reported in Refs. 18 and 19 in a similar fashion.

## References

1. R. Benavente, M. Vanrell, R. Baldrich, Parametric fuzzy sets for automatic color naming, *J. Optical Society of America A*, **25**(10) (2008) 2582-2593.
2. C. Bishop, *Neural Networks for Pattern Recognition* (Oxford University Press, 1995).
3. M.C. Boeres, C.C. Ribeiro, I. Bloch, A randomized heuristic for scene recognition by graph matching, *Proc. 3rd Workshop on Efficient and Experimental Algorithms*, eds. C.C. Ribeiro, S.L. Martins, LNCS 3059 (Springer, 2004), pp. 100-113.
4. K. Borgwardt, C. Ong, S. Schönauer, S. Vishwanathan, A. Smola, H.P. Kriegel, Protein function prediction via graph kernels, *Bioinformatics* **21**(1) (2005) 47-56.
5. H. Bunke, G. Allermann, Inexact graph matching for structural pattern recognition, *Patt. Recogn. Lett.* **1** (1983) 245-253.
6. H. Bunke, K. Riesen, Towards the unification of structural and statistical pattern recognition, *Patt. Recogn. Lett.*, **33**(7) (2012) 811-825.
7. H. Bunke, K. Shearer, A graph distance metric based on the maximal common subgraph, *Patt. Recogn. Lett.* **19**(3) (1998) 255-259.
8. D. Conte, P. Foggia, C. Sansone, M. Vento, Thirty years of graph matching in pattern recognition, *Int. J. Patt. Recogn. Artif. Intell.* **18**(3) (2004) 265-298.
9. F.C. Curriero, On the Use of Non-Euclidean Distance Measures in Geostatistics, *Math. Geo.* **38**(8) (2006) 907-926.
10. R. Duda, P. Hart, D. Stork, *Pattern Classification* (Wiley-Interscience, 2nd edition, 2000).
11. Development Therapeutics Program DTP, AIDS antiviral screen (2004), `http://dtp.nci.nih.gov/docs/aids/aids data.html`
12. P.F. Felzenszwalb, D.P. Huttenlocher, Efficient graph-based image segmentation, *Int. J. Comp. Vision* **59**(2) (2004) 167-181.
13. M.L. Fernandez, G. Valiente, A graph distance metric combining maximum common subgraph and minimum common supergraph, *Patt. Recogn. Lett.* **22**(6-7) (2001) 753-758.
14. T. Gärtner, *Kernels for Structured Data* (World Scientific, 2008).
15. T. Gärtner, P. Flach, S. Wrobel, On graph kernels: hardness results and efficient alternatives, *Proc. 16th Annual Conference on Learning Theory*, eds. B. Schölkopf and M. Warmuth (2003), pp. 129-143.
16. J. Gasteiger, T. Engel, *Chemoinformatics: a Textbook* (Wiley-VCH, 2003).
17. J.M. Geusebroek, G.J. Burghouts, A.W.M. Smeulders, The Amsterdam library of object images, *Int. J. Comp. Vision* **61**(1) (2005) 103-112.
18. J. Gibert, E. Valveny, H. Bunke, Vocabulary selection for graph of words embedding, *Proc. 5th Iberian Conference on Pattern Recognition and Image Analysis*, eds. J. Vitrià, J.M. Sanches, M. Hernández, LNCS 6669 (Springer, 2011), pp. 216-223.
19. J. Gibert, E. Valveny, H. Bunke, Dimensionality reduction for graph of words embedding, *Proc. 8th Int. Workshop on Graph-Based Representations for Pattern Recognition*, eds. X. Jiang, M. Ferrer, A. Torsello, LNCS 6658 (Springer, 2011), pp. 22-31.
20. B. Haasdonk, Feature space interpretation of SVMs with indefinite kernels, *IEEE Trans. Patt. Anal. Mach. Intell.* **27**(4) (2005) 482-492.
21. D. Haussler, Convolution kernels on discrete structures, *Technical Report UCSC-CRL-99-10*, University of California, Santa Cruz (1999).
22. A. Inokuchi, T. Washio, H. Motoda, An apriori-based algorithm for mining frequent substructures from graph data, *Proc. 4th Principles of Data Mining and Knowledge Discovery*, eds. D.A. Zighed, J. Komorowski, J. Zytkow, LNAI 1910 (Springer, 2000), pp. 13-32.
23. S. Jouili, S. Tabbone, Graph Embedding using constant shift embedding, *Proc. In-*

*ternational Conference on Pattern Recognition*, eds. D. Ünay, Z. Çataltepe, S. Aksoy, LNCS 6388 (Springer, 2010), pp. 83-92.
24. D. Justice, A. Hero, A binary linear programming formulation of the graph edit distance, *IEEE Trans. Patt. Anal. Mach. Intell.* **28**(8) (2006) 1200-1214.
25. H. Kashima, K. Tsuda, A. Inokuchi, Marginalized kernels between labelled graphs, *Proc. 20th Int. Conf. Machine Learning* (2003), pp. 321-328.
26. J. Kazius, R. McGuire, R. Bursi, Derivation and validation of toxicophores for mutagenicity prediction, *J. Med. Chemistry* **48**(1) (2005) 312-320.
27. R. Kondor, J. Lafferty, Diffusion kernels on graphs and other discrete input spaces, *Proc. 19th Int. Conf. Machine Learning* (2002), pp. 315-322.
28. S. Kramer, L. De Raedt, Feature construction with version spaces for biochemical application, *Proc. 18th Int. Conf. Machine Learning* (2001), pp. 258-265.
29. J. Lafferty, G. Lebanon, Information diffusion kernels, *Advances in Neural Information Processing Systems* (MIT Press, 2003), pp. 375-382.
30. B. Luo, R.C. Wilson, E.R. Hancock, Spectral embedding of graphs, *Patt. Recogn.* **36**(10) (2003) 2213-2230.
31. P. Mahé, N. Ueda, T. Akutsu,J.-L. Perret, J.-P. Vert, Graph Kernels for Molecular Structure-Activity Relationship Analysis with Support Vector Machines, *J. of Chem. Inform. and Mod.* (2005) 939-951.
32. MAO dataset, `http://iapt-tc15.greyc.fr/links.html#chemistry`
33. J. Munkres, Algorithms for the assignment and transportation problems, *J. Society for Industrial and Applied Mathematics* **5** (1957) 32-38.
34. S. Nene, S. Nayar, H. Murase, Columbia Object Image Library: COIL-100, *Technical report*, Department of Computer Science, Columbia University, New York (1996).
35. M. Neuhaus, H. Bunke, Self-organizing maps for learning the edit costs in graph matching, *IEEE Tran. Syst. Man and Cybern. (Part B)* **35**(3) (2005) 503-514.
36. M. Neuhaus, H. Bunke, Automatic learing of cost functions for graph edit distance, *Inf. Sci.* **177**(1) (2007) 239-247.
37. M. Neuhaus, H. Bunke, *Bridging the Gap between Graph Edit Distance and Kernel Machines* (World Scientific, 2007).
38. M. Neuhaus, K. Riesen, H. Bunke, Fast suboptimal algorithms for the computation of graph edit distance, *Proc. 11th Int. Workshop on Structural and Syntactic Pattern Recognition*, eds. D.Y. Yeung, J.T. Kwok, A. Fred, F. Roli, D. de Ridder, LNCS 4109 (Springer, 2006), pp. 163-172.
39. E. Pekalska, R. Duin, *The Dissimilarity Representation for Pattern Recognition: Foundations and Applications* (World Scientific, 2005).
40. E. Pekalska, R. Duin, P. Paclick, Prototype selection for dissimilarity-based classifiers, *Patt. Recogn.* **39**(2) (2006) 189-208.
41. P. Ren, R. Wilson, E. Hancock, Graph characterization via Ihara coefficients, *IEEE Trans. Neural Networks* **22**(2) (2011) 233-245.
42. K. Riesen, H. Bunke, IAM Graph database repository for graph based pattern recognition and machine learning, *Proc. Int. Workshops on Structural, Syntactic and Statistical Pattern Recognition*, eds. N. da Vitoria Lobo et al., LNCS 5342 (Springer, 2008), pp. 287-297.
43. K. Riesen, H. Bunke, Approximate graph edit distance computation by means of bipartite graph matching, *Image and Vision Comp.* **27**(4) (2009) 950-959.
44. K. Riesen, H. Bunke, *Graph Classification and Clustering Based on Vector Space Embedding* (World Scientific, 2010).
45. A. Robles-Kelly, E.R. Hancock, Graph edit distance from spectral seriation, *IEEE Trans. Patt. Anal. Mach. Intell.* **27**(3) (2005) 365-378.

46. A. Sanfeliu, K.S. Fu, A distance measure between attributed relational graphs for pattern recognition, *IEEE Tran. Syst. Man and Cybern. (Part B)* **13**(3) (1983) 353-363.
47. B. Schölkopf, A.J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond* (MIT Press, 2002).
48. S. Sorlin, C. Solnon, Reactive tabu search for measuring graph similarity, *Proc. 5th Int. Workshop on Graph-Based Representations for Pattern Recognition*, eds. L. Brun, M. Vento, LNCS 3434 (Springer, 2005), pp. 172-182.
49. M.J. Tarr, *The object databank*, `http://www.cnbc.cmu.edu/tarrlab/stimuli/objects/index.html`
50. S. Vishwanathan, N.N. Schraudolph, R. Kondor, K. Borgwardt, Graph kernels, *J. Mach. Learn. Research*, **11** (2010) 1201-1242.
51. W.D. Wallis, P. Shoubridge, M. Kraetzl, D. Ray, Graph distances using graph union, *Patt. Recogn. Lett.* **22**(6) (2001) 701-704.
52. C. Watkins, Dynamic alignment kernels, *Advances in Large Margin Classifiers* (MIT Press, 2000), pp. 39-50.
53. J.H. Wells, L.R. Williams, *Embeddings and extensions in analysis* (Springer-Verlag, New York, 1975)
54. R. Wilson, E. Hancock, B. Luo, Pattern vectors from algebraic graph theory, *IEEE Trans. Patt. Anal. Mach. Intell.* **27**(7) (2005) 1112-1124.
55. J. Zhang, M. Marszalek, S. Lazebnik, C. Schmid, Local features and kernels for classification of texture and object categories: a comprehensive study, *Int. J. Comp. Vision* **73**(2) (2007) 213-238.