International Journal of Pattern Recognition
and Artificial Intelligence
Vol. 23, No. 6 (2009) 1053–1081
© World Scientific Publishing Company

GRAPH CLASSIFICATION BASED ON VECTOR SPACE EMBEDDING

KASPAR RIESEN* and HORST BUNKE†

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

Graphs provide us with a powerful and flexible representation formalism for pattern classification. Many classification algorithms have been proposed in the literature. However, the vast majority of these algorithms rely on vectorial data descriptions and cannot directly be applied to graphs. Recently, a growing interest in graph kernel methods can be observed. Graph kernels aim at bridging the gap between the high representational power and flexibility of graphs and the large amount of algorithms available for object representations in terms of feature vectors. In the present paper, we propose an approach transforming graphs into n-dimensional real vectors by means of prototype selection and graph edit distance computation. This approach allows one to build graph kernels in a straightforward way. It is not only applicable to graphs, but also to other kind of symbolic data in conjunction with any kind of dissimilarity measure. Thus it is characterized by a high degree of flexibility. With several experimental results, we prove the robustness and flexibility of our new method and show that our approach outperforms other graph classification methods on several graph data sets of diverse nature.

Keywords: Graph matching; graph kernel; graph embedding; prototype selection.

1. Introduction

The classification of objects is a common task in intelligent information processing. Classification refers to the process of assigning an unknown input object to one out of a given set of classes. Examples of classification problems can be found in biometric person identification, optical character recognition, medical diagnosis, and many other domains. A vast number of algorithms for classification have been proposed in the literature.[14] Almost all of these algorithms have been designed for object representations given in terms of feature vectors. That is, an object is formally represented as a point in an n-dimensional feature space and a classifier attempts to partition the feature space into disjoint subspaces, such that each subspace corresponds to exactly one class. Object representations given in terms of feature vectors have a number of useful properties. For example, object similarity, or distance, can be easily computed by means of the Euclidean distance or other distance measures in the n-dimensional real space $\mathbb{R}^n$. Similarly, dimensionality reduction methods, such as principal component analysis, linear discriminant analysis, and others can be readily applied.[14] The convenience and low computational complexity of algorithms that use feature vectors as their input have eventually resulted in a rich repository of algorithmic tools for classification.

Recently, however, a growing interest in graph-based object representations for classification has emerged.[11] As a matter of fact, object representations by means of graphs have a number of advantages over feature vectors. First, graphs are able to represent not only the values of object properties, i.e. features, but can be used to explicitly model relations that exist between different parts of an object. Moreover, graphs do not suffer from the constraint of fixed dimensionality. That is, the number of nodes and edges in a graph is not limited *a priori* and depends on the size and the complexity of the actual object to be modeled. A survey on the use of graph representations in the field of pattern recognition can be found in Ref. 11.

One drawback of graphs, when compared to feature vectors, is the significantly increased complexity of many algorithms. For example, the comparison of two feature vectors for identity can be accomplished in linear time with respect to the length of the two vectors. For the analogous operation on general graphs, i.e. testing two graphs for isomorphism, only exponential algorithms are known today. Another serious limitation in the use of graphs for object classification arises from the fact that there is little mathematical structure in the domain of graphs. For example, computing the sum, the weighted sum, or the product of a pair of entities (which are elementary operations, required in many algorithms) is not possible in the domain of graphs. Despite little mathematical foundation in the domain of graphs, a number of different graph matching approaches have been proposed in the literature. For a review of graph matching methods and applications, we refer to Ref. 11.

A promising direction to overcome the lack of algorithmic tools for graph classification is graph embedding in the real vector space. Basically, an embedding of graphs into vector spaces establishes the access to the rich repository of algorithmic tools for pattern analysis. In Ref. 31, for instance, features derived from the eigendecomposition of graphs are studied. Another idea deals with string edit distance applied to the eigensystem of graphs.[57] This procedure results in distances between graphs which are used to embed the graphs into a vector space by means of multidimensional scaling. In Ref. 58, the authors turn to the spectral decomposition of the Laplacian matrix. They show how the elements of the spectral matrix for the Laplacian can be used to construct symmetric polynomials. In order to encode graphs as vectors, the coefficients of these polynomials are used as graph features. Another approach for graph embedding has been proposed in Ref. 44. The authors use the relationship between the Laplace–Beltrami operator and the graph Laplacian to embed a graph onto a Riemannian manifold.

The present paper considers a new class of graph embedding procedures which are based on prototype selection and graph edit distance computation. This idea is somewhat related to the empirical kernel map described in Ref. 48. However, our approach is primarily based on the idea proposed in Refs. 39 and 38 where feature vectors are mapped into dissimilarity spaces. Later, the method proposed in Refs. 39 and 38 was extended so as to map string representations into vector spaces.[51] In the current paper, we go one step further and generalize the methods described in Ref. 51 to the domain of graphs. The key idea of this approach is to use the distances of an input graph to a number of training graphs, termed prototype graphs, as a vectorial description of the graph. That is, we use the dissimilarity representation for pattern recognition rather than the original graph based representation.

Another idea to overcome the lack of algorithmic tools for graph classification, which is closely related to graph embedding procedures, is kernel methods. In fact, the idea of kernel methods was introduced into pattern recognition long time ago, viz. in 1964 by the authors of Ref. 1. Almost 20 years later, this idea was picked up by the authors of Ref. 6, who combined it with large margin classifiers, leading to support vector machines and the (re)introduction of kernel theory in the fields of pattern recognition, machine learning and data mining.[6,53,48,49] In recent years, kernel methods have become one of the most rapidly emerging sub-fields in intelligent information processing. The vast majority of work on kernel methods is concerned with transforming a given feature space into another one of higher dimensionality without computing the transformation explicitly for each individual feature vector. As a fundamental extension, the existence of kernels for symbolic data structures, especially for graphs, has been shown. By means of suitable kernel functions, graphs can be implicitly mapped into vector spaces. Consequently, a large class of kernel machines for classification, most of them originally developed for feature vectors, become applicable to graphs. Hence by means of kernel functions, one can benefit from both the high representational power of graphs and the large repository of algorithmic tools available for feature vector representations of objects. A number of graph kernels have been proposed in the literature. For an early survey, see Ref. 17.

The graph embedding procedure proposed in this paper can be also seen as a foundation for a novel class of graph kernels. However, in contrast to some other kernel methods, the approach proposed in this paper results in an explicit embedding of the considered graphs in a vector space. Hence, not only scalar products, but individual graph maps are available in the target space. We observe that this approach is more general than other graph kernels for a number of reasons. First, as the map of each graph in the target vector space is explicitly computed, not only kernel machines, but also other non-kernelizable algorithms can be applied to the resulting vector representation. Secondly, there are almost no restrictions on the type of graphs the proposed method can deal with. It can be applied to directed or undirected graphs, and to graphs without or with labels on their nodes and/or edges. In case there are labels on the nodes and/or edges, these labels can be of any nature, for example, they can be elements from a finite or infinite set of discrete symbols, the set of integer numbers, real numbers, or real vectors. Thirdly, our method is versatile in the sense that it is possible to integrate domain specific knowledge about object similarity, if available, when defining the costs of the elementary edit operations. (However, automatic procedures for learning the edit costs from a set of sample graphs are available as well.[34,35]) The versatility and flexibility of the proposed approach will be experimentally verified on a number of graph datasets from diverse application fields.

A preliminary version of the current paper appeared in Ref. 42. The current paper has been significantly extended with respect to the underlying methodology and the experimental evaluation. That is, besides some technical extensions of the proposed methodology, our novel approach to graph embedding is now set into the context of graph kernels. Furthermore, an analysis is provided that shows how the Euclidean distances between maps of graphs in the vector space are related to the corresponding edit distances in the graph domain. Also, the number of datasets where our embedding procedure is tested on is considerably increased and results of two additional reference systems, a similarity kernel and the widely used random walk kernel, are added. Finally, a detailed discussion and results about the validation of the meta parameters of our approach are provided.

The rest of this paper is organized as follows. In the next section, we define basic concepts and introduce our notation. Then, in Sec. 3, the proposed approach for graph embedding in real vector spaces is described. In Sec. 4, we report a number of experiments and present results achieved with our new embedding method. Also, a comparison with three reference systems is made. Finally, in Sec. 5, we draw conclusions and provide suggestions for future work.

2. Basic Concepts and Notation

In this section, we introduce our terminology and some concepts used in this paper.

2.1. *Basic terminology*

Generally, a graph g is defined by a finite set of nodes V and a finite set of edges E and their corresponding labeling functions. Let L be a finite or infinite set of labels for nodes and edges.

Definition 1. (Graph) A graph g is defined by the four-tuple $g = (V, E, \mu, \nu)$, where V is the finite set of nodes, $E \subseteq V \times V$ is the set of edges, $\mu : V \to L$ is the node labeling function, and $\nu : E \to L$ is the edge labeling function.

The definition given above allows us to handle arbitrary graphs with unconstrained labeling functions. For example, the label alphabet can be given by the set of integers, the vector space $\mathbb{R}^n$, or a set of symbolic labels $L = \{\alpha, \beta, \gamma, \ldots\}$. Moreover, unlabeled graphs can be handled by assigning the same label l to all nodes and edges. Edges are defined by pairs of nodes (u, v), where $u \in V$ denotes the source node and $v \in V$ the target node of a directed edge. Undirected graphs can be modeled by inserting a reverse edge $(v, u) \in E$ for each edge $(u, v) \in E$ with $\nu(u, v) = \nu(v, u)$.

2.2. *Graph matching*

Graph matching refers to the task of evaluating the similarity of graphs. There are two major versions of this task, namely exact and error-tolerant graph matching. The aim of exact graph matching is to determine whether two graphs or parts of them are identical in terms of structure and labels. Compared to vectors, where the determination of identical parts is trivial, exact matching of graphs is substantially more difficult.[32,52,30] Generally, there is no canonical ordering for nodes and edges in a graph, which makes the comparison of two sets of nodes and edges quite complex.

Based on the exact graph matching paradigms of maximum common subgraph and minimum common supergraph, a few graph similarity measures have been proposed.[9,54,15] However, the main restriction of exact graph matching and related similarity measures is the requirement that a significant part of the topology together with the corresponding node and edge labels in two graphs have to be identical for obtaining a high similarity value. In fact, this is not realistic for many applications. Especially if the node or edge label alphabet L is given by the n-dimensional vector space $\mathbb{R}^n$, the exact matching paradigm is too restrictive. In order to make graph matching better applicable to real world problems, several error-tolerant, or inexact, graph matching methods have been proposed.[11] One of the most flexible methods for error-tolerant graph matching is based on the edit distance of graphs.[8,46,43] This paradigm is used in the present paper and described in detail in the next subsection.

2.3. *Graph edit distance*

The key idea of graph edit distance is to define the dissimilarity, or distance, of graphs by the minimum amount of distortion that is needed to transform one graph into another. A standard set of distortion operations is given by *insertions*, *deletions*, and *substitutions* of nodes and edges. Other operations, such as *merging* and *splitting* of nodes,[2] can be useful in certain applications but are not considered in this paper.

Given two graphs, the source graph g_1 and the target graph g_2, the idea of graph edit distance is to delete some nodes and edges from g_1, relabel (substitute) some of the remaining nodes and edges, and insert some nodes and edges in g_2, such that g_1 is finally transformed into g_2. A sequence of edit operations $e_1, \ldots, e_k$ that transform g_1 into g_2 is called an *edit path* between g_1 and g_2. In Fig. 1, an example of an edit path between two graphs g_1 and g_2 is given. This edit path consists of three edge deletions, one node deletion, one node insertion, two edge insertions, and two node substitutions. Obviously, for every pair of graphs (g_1, g_2), there exist a number of different edit paths transforming g_1 into g_2. Let $\Upsilon(g_1, g_2)$ denote the set of all such edit paths. To find the most suitable edit path out of

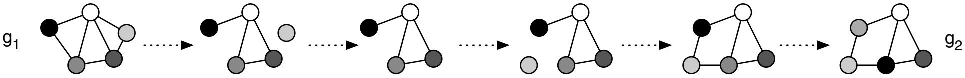


Fig. 1. A possible edit path between graphs g_1 and g_2 (node labels are represented by different shades of grey).

$\Upsilon(g_1, g_2)$, one introduces a cost for each edit operation, measuring the strength of the corresponding operation. The idea of such cost functions is to define whether or not an edit operation represents a strong modification of the graph. Hence, between two similar graphs, there should exist an inexpensive edit path, representing low cost operations, while for different graphs, an edit path with high costs is needed. Consequently, the *edit distance* of two graphs is defined by the minimum cost edit path between two graphs.

Definition 2. (Graph Edit Distance) Let $g_1 = (V_1, E_1, \mu_1, \nu_1)$ be the source graph and $g_2 = (V_2, E_2, \mu_2, \nu_2)$ be the target graph. The graph edit distance between g_1 and g_2 is defined by

$$d(g_1, g_2) = \min_{(e_1,\ldots,e_k)\in\Upsilon(g_1,g_2)} \sum_{i=1}^{k} c(e_i),$$

where $\Upsilon(g_1, g_2)$ denotes the set of edit paths transforming g_1 into g_2, and c denotes the edit cost function measuring the strength $c(e_i)$ of edit operation e_i.

Optimal algorithms for computing the edit distance of graphs g_1 and g_2 are typically based on combinatorial search procedures that explore the space of all possible mappings of the nodes and edges of g_1 to the nodes and edges of g_2.[8] A major drawback of those procedures is their computational complexity, which is exponential in the number of nodes of the involved graphs. Consequently, the application of optimal algorithms for edit distance computations is limited to graphs of rather small size in practice. To render graph edit distance computation less computationally demanding, a number of suboptimal methods have been proposed. In some approaches, the basic idea is to perform a local search to solve the graph matching problem, that is, to optimize local criteria instead of global, or optimal ones.[36,3,50] In Ref. 25, a linear programming method for computing the edit distance of graphs with unlabeled edges is proposed. The method can be used to derive lower and upper edit distance bounds in polynomial time. Two fast but suboptimal algorithms for graph edit distance computation are proposed in Ref. 37. The authors propose simple variants of a standard edit distance algorithm that make the computation substantially faster. In Ref. 41, another new efficient algorithm for graph edit distance computation is introduced. The method is based on a fast suboptimal bipartite optimization procedure mapping nodes, or edges, of one graph to nodes, or edges, of another graph.

3. Graph Embedding by Means of Prototype Selection

The idea underlying our method for graph embedding was originally developed for the problem of embedding sets of feature vectors in a dissimilarity space.[39,38] In Ref. 38, the authors claim that a notion of proximity is more fundamental than that of a feature or a class. Furthermore, it is pointed out that in the case of structural data (like graphs), the extraction of numerical features is difficult or even intractable, while proximity can directly be derived from the data using an adequate dissimilarity model. In this paper, we introduce an extension of the idea of dissimilarity representation to the domain of graphs.

3.1. *General embedding procedure*

Assume we have a labeled set of sample graphs, $T = \{g_1, \ldots, g_n\}$, and a graph dissimilarity measure $d(g_i, g_j)$. Note that T can be any kind of graph set and $d(g_i, g_j)$ can be any kind of dissimilarity measure. The samples in T can even be synthesized graphs. However, for convenience, we define T as a training set of existing graphs and use the edit distance as dissimilarity measure. After having selected a set $P = \{p_1, \ldots, p_m\}$ of $m \leq n$ prototypes from T, we compute the dissimilarity of a given input graph g to each prototype $p \in P$. Note that g can be an element of T or any other graph set. This leads to m dissimilarities, $d_1 = d(g, p_1), \ldots, d_m = d(g, p_m)$, which can be arranged in an m-dimensional vector $(d_1, \ldots, d_m)$. In this way, we can transform any graph from the training as well as any other graph set (for instance a validation or a test set of a classification problem), into a vector of real numbers.

Definition 3. (Graph Embedding) Let us assume a graph domain G with graphs g_i is given. If $T = \{g_1, \ldots, g_n\} \subseteq G$ is a training set of graphs and $P = \{p_1, \ldots, p_m\} \subseteq T$ is a set of prototypes, the mapping

$$\varphi_m^P : G \to \mathbb{R}^m$$

is defined as the function

$$\varphi_m^P(g) = (d(g, p_1), \ldots, d(g, p_m)),$$

where $d(g, p_i)$ is any graph dissimilarity measure between graph g and the ith prototype.

Obviously, by means of this definition, we obtain a vector space where each axis corresponds to a prototype $p_i \in T$ and the coordinate values of the embedded graph g are the distances of g to the elements in P. Note that our embedding procedure has a close relation to Lipschitz embeddings where a coordinate space is defined such that each axis corresponds to a reference set P_i drawn from T.[24,7,22] In contrast to our approach, however, which is defined with respect to a single prototype set P, Lipschitz embeddings are defined in terms of m prototype sets $P_1, \ldots, P_m$. Clearly, for this approach to work properly, the distance function d has to be extended (e.g. by defining $d(g, P_i) = \min_{p \in P_i}\{d(g, p)\}$).

The embedding procedure proposed in this paper makes use of graph edit distance. This allows us to deal with a large class of graphs (directed, undirected, unlabeled, node and/or edge labels from any finite or infinite domain). Furthermore, a high degree of robustness against various graph distortions can be expected. Since the computation of graph edit distance is exponential in the number of nodes for general graphs, the complexity of this graph embedding is exponential as well. However, as mentioned in Sec. 2, there exist a number of efficient approximation algorithms for graph edit distance computation. For instance, the method described in Ref. 41 has cubic time complexity. Consequently, given m predefined prototypes, the embedding of one particular graph is established by means of m distance computations with polynomial time.

Based on the mapping φ_m^P, one can define a valid graph kernel κ by computing the standard dot product of two graph maps in the resulting vector space

$$\kappa_{\langle\rangle}(g_i, g_j) = \langle \varphi_m^P(g_i), \varphi_m^P(g_j) \rangle$$

Of course, not only the standard dot product can be used but any valid kernel function defined for vectors, e.g. a RBF kernel function

$$\kappa_{\mathrm{RBF}}(g_i, g_j) = \exp(-\gamma \|\varphi_m^P(g_i) - \varphi_m^P(g_j)\|^2)$$

where $\gamma > 0$.

This embedding graph kernel is somewhat similar in spirit to the subgraph kernel proposed in Ref. 40. However, rather than using a potentially infinite set of subgraphs, we restrict our considerations to a finite number of prototypes, i.e. dimensions. Furthermore, rather than counting the number of occurrences of a particular subgraph within some host graph, we employ the edit distance in order to capture graph similarity. Obviously, large values of the kernel function occur when both g_i and g_j are similar to the same prototype graphs. Clearly, whenever both g_i and g_j are similar to the same prototype graphs, we may conclude that g_i and g_j are similar to each other. Using the graph edit distance rather than counting the number of occurrences of certain subgraphs allows us to flexibly deal with various kinds of distortions that may affect the underlying graphs.

After two graphs, g and g', have been mapped to the vector space, we observe the following relationship[a]:

$$\begin{aligned}\|\varphi(g) - \varphi(g')\| &= (\langle \varphi(g), \varphi(g)\rangle + \langle \varphi(g'), \varphi(g')\rangle - 2\langle \varphi(g), \varphi(g')\rangle)^{\frac{1}{2}} \\ &= \left(\sum_{i=1}^{m} d(g, p_i)^2 + \sum_{i=1}^{m} d(g', p_i)^2 - 2\sum_{i=1}^{m} d(g, p_i) d(g', p_i)\right)^{\frac{1}{2}} \\ &= \left(\sum_{i=1}^{m} (d(g, p_i) - d(g', p_i))^2\right)^{\frac{1}{2}} \qquad (1)\end{aligned}$$

[a] For the purpose of simplicity, we write $\varphi(g)$ for $\varphi_m^P(g)$.

Hence, the Euclidean distance of a pair of graphs g and g' in the vector space is equal to the square root of the sum of squared differences between the edit distances of g and g' to the prototype graphs. Given d is a metric, the more similar g and g' are, the smaller will be the terms $(d(g, p_i) - d(g', p_i))^2$ and, consequently, the smaller will be their Euclidean distance in the vector space. Moreover, due to the triangle inequality $(|d(g, p_i) - d(g', p_i)| \leq d(g, g'))$, we have

$$\begin{aligned} \|\varphi(g) - \varphi(g')\| &= \left(\sum_{i=1}^{m} (d(g, p_i) - d(g', p_i))^2 \right)^{\frac{1}{2}} \\ &\leq (m \cdot d(g, g')^2)^{\frac{1}{2}} \\ &= \sqrt{m} \cdot d(g, g') \end{aligned}$$

That is, the upper bound of the Euclidean distance of a pair of graphs g and g' is given by $\sqrt{m} \cdot d(g, g')$. Obviously, defining the embedding procedure given in Def. 3 as $\varphi_m^P(g) = (d(g, p_1)/q, \ldots, d(g, p_m)/q)$ with $q = \sqrt{m}$ leads to an upper bound independent of the number of prototypes m.

The selection of the m prototypes is a critical issue since not only the prototypes themselves but also their number affect the resulting vectors and the performance of the corresponding kernel machine. Thus, a good selection of m prototypes seems to be crucial to succeed with the classification algorithm in the feature vector space. Intuitively, prototypes should not be redundant, i.e. we should avoid selecting similar graphs, and prototypes should include as much information as possible, i.e. they should be uniformly distributed over the whole set of patterns.[b]

3.2. *Prototype selection*

Different prototype selection methods have been proposed in the literature in conjunction with k-nearest-neighbor classifiers.[20,45,56] These prototype selection methods aim at overcoming the three major drawbacks of k-nearest neighbor classifiers, viz. large storage requirements, large computational effort for distance evaluations, and sensitivity to outliers. However, in the present paper, the objective of prototype selection is different. We use the selected prototypes as reference points in the underlying graph domain. These reference graphs then directly serve us to transform graphs into real vectors.

In order to define the prototype set P and eventually map graphs into the n-dimensional real space, some special graphs, termed *set median graph*,[23] *set center graph*, and *set marginal graph*, will be needed. They are introduced below.

Definition 4. (Set Median, Center, and Marginal Graph) Given a set G of graphs, the set median graph[23] $g_{mdn} \in G$ is defined as

$$\text{median}(G) = g_{mdn} = \arg \min_{g_1 \in G} \sum_{g_2 \in G} d(g_1, g_2)$$

[b] The feasibility of these two requirements can also be concluded from Eq. (1).

The set median graph is the graph whose sum of edit distances to all other graphs $g_i \in G$ is minimal. Intuitively, the set median graph is located in the center of a given graph set. A slightly different concept is given by the set center graph which is defined as

$$\text{center}(G) = g_c = \arg\min_{g_1 \in S} \max_{g_2 \in g} d(g_1, g_2)$$

Obviously, the set center graph is the graph for which the maximum distance to all other graphs in G is minimum. In contrast to those graphs located in the center of a given set, the set marginal graph $g_{mrg} \in G$ is located at the border of a given graph set. Formally,

$$\text{marginal}(G) = g_{mrg} = \arg\max_{g_1 \in G} \sum_{g_2 \in G} d(g_1, g_2)$$

The set marginal graph is thus the graph whose sum of distances to all other graphs $g_i \in G$ is maximal.

Assume there are k different classes of graphs $C_1, \ldots, C_k$ represented in the training set T. We distinguish between class-wise and class-independent prototype selection. Class-independent selection means that the selection is executed over the whole training set T to get m prototypes, while in class-wise selection, the selection is performed individually for each of the k different classes $C_1, \ldots, C_k$. In the latter case, l_i prototypes are selected independently for each class C_i such that $\sum_{i=1}^{k} l_i = m$. Whether the class-wise or class-independent method is more convenient depends on a number of factors, including the size of T, the distribution of the graphs in T, whether or not classes are balanced, and the application.

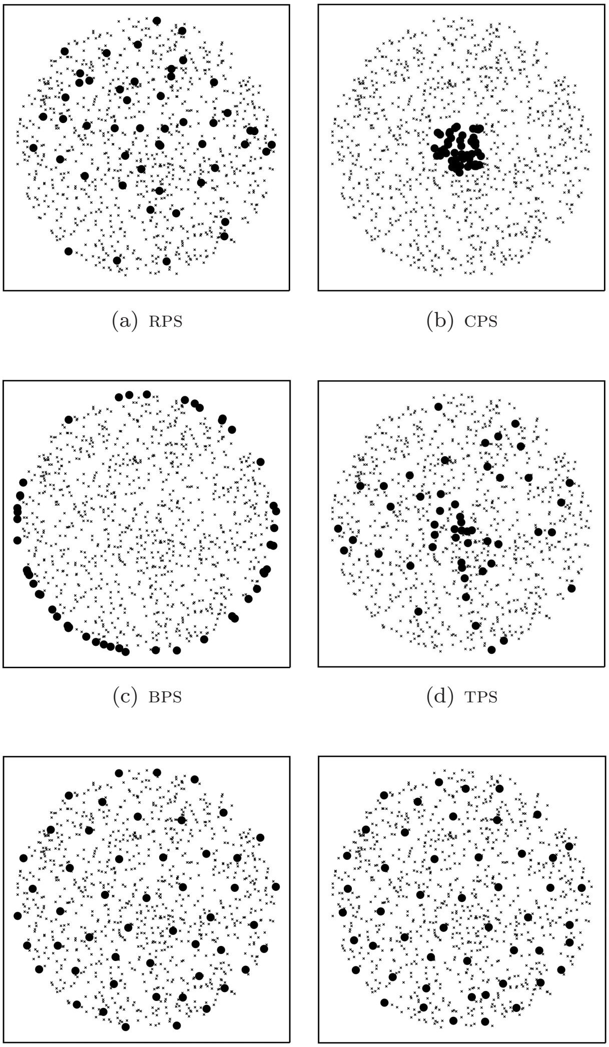
The prototype selection algorithms used in this paper are described below (see also Ref. 39). In order to illustrate the behavior of the different prototype selectors, we provide a simple illustrative example. The data underlying this example are two-dimensional random vectors that are uniformly distributed inside a circle in the plane. The proposed prototype selectors are applied to this artificial vector set. Note that in order to visualize the prototype selection, the algorithms are applied to vectors and not to graphs.

Random. A random selection of m prototypes from T is performed. Of course, the Random prototype selector (rps) can be applied class-independent or class-wise (rps-c). rps-c provides a random selection of l_i prototypes per class C_i. In Fig. 2(a), the results of random selection from the random data set are given.

Center. The Center prototype selector (cps) selects prototypes situated in, or near, the center of the training set T. The set of prototypes $P = \{p_1, \ldots, p_m\}$ is iteratively constructed as follows, starting with the median graph of T:

$$P_i = \begin{cases} \varnothing, & \text{if } i = 0 \\ P_{i-1} \cup \{g_i\}, & \text{if } 0 < i \leq m \end{cases}$$

where $g_i = \text{median}(T \backslash P_{i-1})$.



(a) RPS (b) CPS

(c) BPS (d) TPS

(e) SPS (f) k-CPS

Fig. 2. Illustration of the different prototype selectors.

In Fig. 2(b), an example result returned by the Center prototype selector is given. It seems that the Center prototype selector is not very appropriate for selecting m prototypes. Obviously, both requirements, avoiding redundancy and uniform distribution, are not satisfied. Nevertheless, we mention this prototype selector here for the purpose of completeness. To obtain a better distribution, one can apply the Center prototype selector class-wise (CPS-C). In this case, supposably, the prototypes mirror the given distribution better than the class-independent version.

Border. The Border prototype selector (bps) selects prototypes situated at the border of the training set T. The set of prototypes $P = \{p_1, \ldots, p_m\}$ is iteratively constructed as follows, starting with the marginal graph of T:

$$P_i = \begin{cases} \varnothing, & \text{if } i = 0 \\ P_{i-1} \cup \{g_i\}, & \text{if } 0 < i \leq m \end{cases}$$

where $g_i = \text{marginal}(T \backslash P_{i-1})$.

In Fig. 2(c), the Border prototype selection is illustrated. In contrast to the Center prototype selector, where many prototypes are structurally similar, this kind of selection avoids redundancies much better. However, there are no prototypes selected from the center of the set. Obviously, the requirement of uniform distribution is not satisfied. Furthermore, one has to expect that bps possibly selects outliers. The Border prototype selector can be applied class-independent and class-wise (bps-c).

Targetsphere. The idea of this prototype selector is to distribute the prototypes from the center to the border as uniformly as possible. The Targetsphere prototype selector tps first determines the center graph g_c in T. After the center graph has been found, the graph furthest away from g_c, i.e. the graph $g_f \in T$ whose distance to g_c is maximum, is located. Both graphs, g_c and g_f, are selected as prototypes. The distance from g_c to g_f is referred to as $d_{\max}$, i.e. $d_{\max} = d(g_c, g_f)$. The interval $[0, d_{\max}]$ is then divided into $m - 1$ equidistant subintervals of width $w = \frac{d_{\max}}{m-1}$. The $m - 2$ graphs for which the corresponding distances to the center graph g_c are located nearest to the interval borders in terms of edit distance are selected as prototypes:

$$P_i = \begin{cases} \{g_c, g_f\}, & \text{if } i = 0 \\ P_{i-1} \cup \{g_i\}, & \text{if } 0 < i \leq m-2 \end{cases}$$

where $g_i = \arg\min_{g \in T \backslash P_{i-1}} |d(g, g_c) - i \cdot w|$.

In Fig. 2(d), the selection obtained by tps is illustrated. The selection of the prototypes considers the distance to the center graph only. That is, the procedure selects m prototypes uniformly distributed in the interval $[0, d_{\max}]$. The individual distances to the prototypes selected before are not considered. The Targetsphere prototype selector can be applied class-independent as well as class-wise (tps-c).

Spanning. The Spanning prototype selector sps considers all distances to the prototypes selected before. The first prototype is the set median graph. Each additional prototype selected by the spanning prototype selector is the graph furthest away from the already selected prototype graphs.

$$P_i = \begin{cases} \text{median}(T), & \text{if } i = 1 \\ P_{i-1} \cup \{p_i\}, & \text{if } 1 < i \leq m \end{cases}$$

where $p_i = \arg\max_{g \in T \backslash P_{i-1}} \min_{p \in P_{i-1}} d(g, p)$.

In Fig. 2(e), the SPS is illustrated. In contrast to the TPS, where only the distances to the center graph are considered during the selection, the SPS takes all distances to the already selected prototypes into account and tries to cover the whole training set as uniformly as possible. The SPANNING prototype selector can be applied class-independent or class-wise (SPS-C).

k-CENTERS. The k-CENTERS prototype selector k-CPS tries to choose m graphs from T so that they are evenly distributed with respect to the dissimilarity information given by d. The algorithm proceeds according to the following procedure:

(1) Select an initial set of m prototypes: $P_0 = \{p_1, \ldots, p_m\}$. One can choose the initial prototypes randomly or by a more sophisticated procedure, for example, the spanning prototype selector mentioned above.
(2) Construct m sets S_i where each set consists of one prototype: $S_1 = \{p_1\}, \ldots, S_m = \{p_m\}$. For each graph $g \in T \backslash P$ find its nearest neighbor $p_i \in P$ and add the graph under consideration to the set S_i corresponding to prototype p_i. This step results in m disjoint sets with $T = \bigcup_{1 \leq i \leq m} S_i$.
(3) For each set S_i find its center c_i, that is, the graph for which the maximum distance to all other objects in S_i is minimum.
(4) For each center c_i, if $c_i \neq p_i$, replace p_i by c_i in S_i. If any replacement is done, return to step 2, otherwise stop.

The procedure stops when no more changes in the sets S_i occur. The prototypes are given by the centers of the m disjoint sets. The k-CENTERS prototype selector can be applied class-independent as well as class-wise (k-CPS-C). Note that this prototype selector is similar to k-medoids clustering.[28]

In Fig. 2(f), this kind of prototype selection is illustrated. Although Fig. 2(f) looks quite similar to Fig. 2(e) at first glance, we note that the k-CPS avoids selecting prototypes from the border by focusing on graphs that are in the center of densely populated areas.

4. Experimental Results

In this section, we provide the results of an experimental evaluation of the proposed embedding procedure. We aim at empirically confirming that the method of prototype based graph embedding and subsequent classification in real vector spaces is applicable to different graph classification problems and matches, or even surpasses, the performance of comparable techniques. For graph edit distance computation, the suboptimal algorithm introduced in Ref. 41 has been used. This graph edit distance algorithm shows superior performance in time and accuracy compared to other suboptimal algorithms. The classifier used in the vector space is the SVM.[53] Of course, any other classifier could be used for this purpose as well. However, we feel that the SVM is particularly suitable because of its theoretical advantages and its superior performance that has been empirically confirmed in many practical classification problems.

The remainder of this section is organized as follows. Next, we give an overview of the graph databases that we use in our experiments. Then, in Sec. 4.2, we describe the reference systems used to compare the performance of the proposed graph embedding procedure with subsequent SVM classification. The experimental setup and validation of the meta-parameters are presented in Sec. 4.3. Finally, the results achieved with the different methods on all data sets are given in Sec. 4.4.

4.1. *Data sets*

The pattern classification tasks considered in this paper include the recognition of line drawings (letters at three distortion levels and technical symbols), scene images, webpages, fingerprints, and molecules, involving a total of eight different graph data sets.

Letter Database. The first graph dataset used in our experiments involves graphs that represent distorted letter drawings. We consider the 15 capital letters of the Roman alphabet that consist of straight lines only $(A, E, F, H, I, K, L, M, N, T, V, W, X, Y, Z)$. For each class, a prototype line drawing is manually constructed. These prototype drawings are then converted into prototype graphs by representing lines by edges and ending points of lines by nodes. Each node is labeled with a two-dimensional attribute giving its position. To obtain large sample sets of graphs with arbitrarily strong distortions, distortion operators are applied to the prototype graphs. This results in randomly translated, deleted, and inserted nodes and edges.

The graph database used in our experiments consists of a training set and a validation set of size 750 each, and a test set of size 1500. In order to test the classifiers under different conditions, the distortions are applied on the prototype graphs with three different levels of strength, viz. *low*, *medium* and *high*. Hence, our experimental data set comprises 9000 graphs altogether. In Fig. 3, the prototype graph and a graph instance for each distortion level representing the letters A, M,

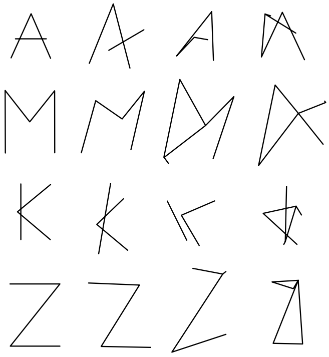


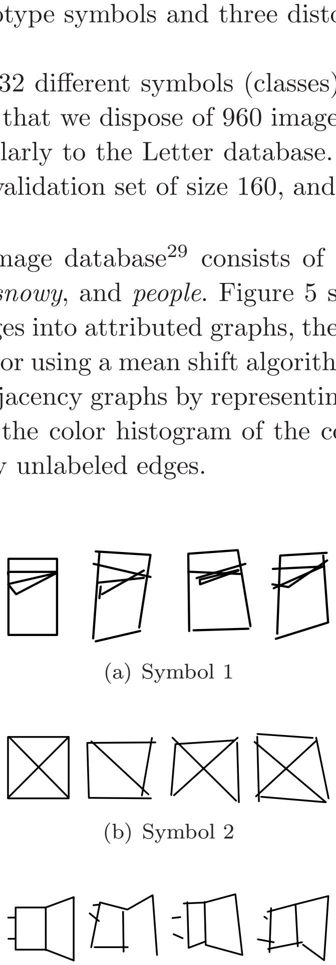
Fig. 3. Instances of letters A, M, K, and Z: Original and distortion levels *low*, *medium* and *high* (from left to right).

K, and Z are illustrated. Note that letters that underwent distortions of *medium* and *high* strength are difficult to recognize even for a human observer.

GREC Database. The GREC database used in our experiments consists of a subset of the symbol database underlying the GREC 2005 contest.[12] The images represent symbols from architecture, electronics, and other technical fields. All images consist of straight lines only. In order to get a large sample set with arbitrarily strong distortions, different distortion operators are applied to the original images. First, all ending and junction points are moved randomly within a circle of radius r. Note that whenever a junction point is moved, all lines connected to it are moved as well. Next, some of the ending points are moved individually, such that the connectivity among the segments may be lost. Finally, some of the lines are deleted. In Fig. 4, three ideal prototype symbols and three distorted versions of each ideal prototype are shown.

Our subset consists of 32 different symbols (classes). Each symbol is distorted 30 times individually such that we dispose of 960 images totally. These images are converted into graphs similarly to the Letter database. The dataset is split into a training set of size 320, a validation set of size 160, and a test set of size 480.

Image Database. The image database[29] consists of images from four different classes, *city*, *countryside*, *snowy*, and *people*. Figure 5 shows an example image of each class. To convert images into attributed graphs, the images are segmented into regions of homogeneous color using a mean shift algorithm.[10] Segmented images are transformed into region adjacency graphs by representing regions by nodes, labeled with attributes specifying the color histogram of the corresponding segment, and the adjacency of regions by unlabeled edges.

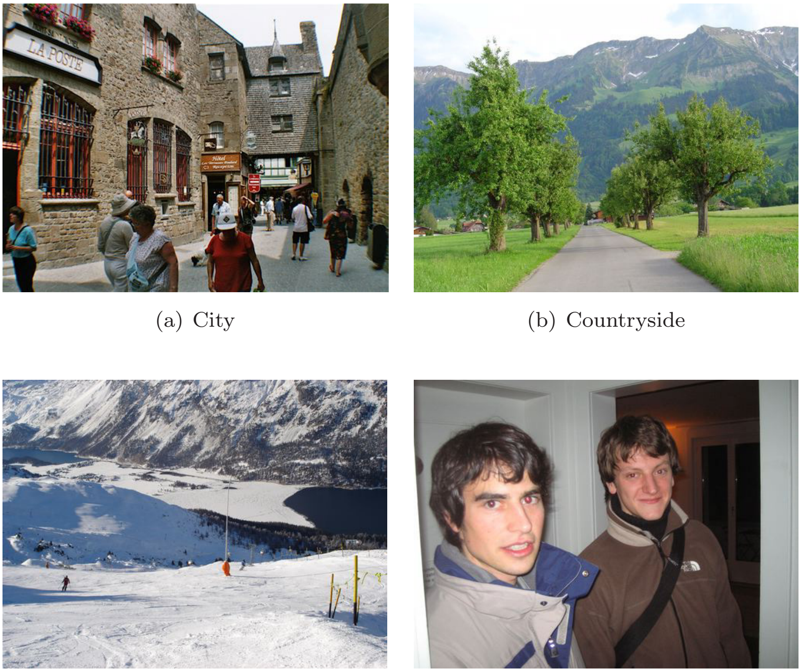


(a) Symbol 1

(b) Symbol 2

(c) Symbol 3

Fig. 4. Three prototype symbols of the GREC 2005 database (left) with some distorted instances.



(a) City

(b) Countryside

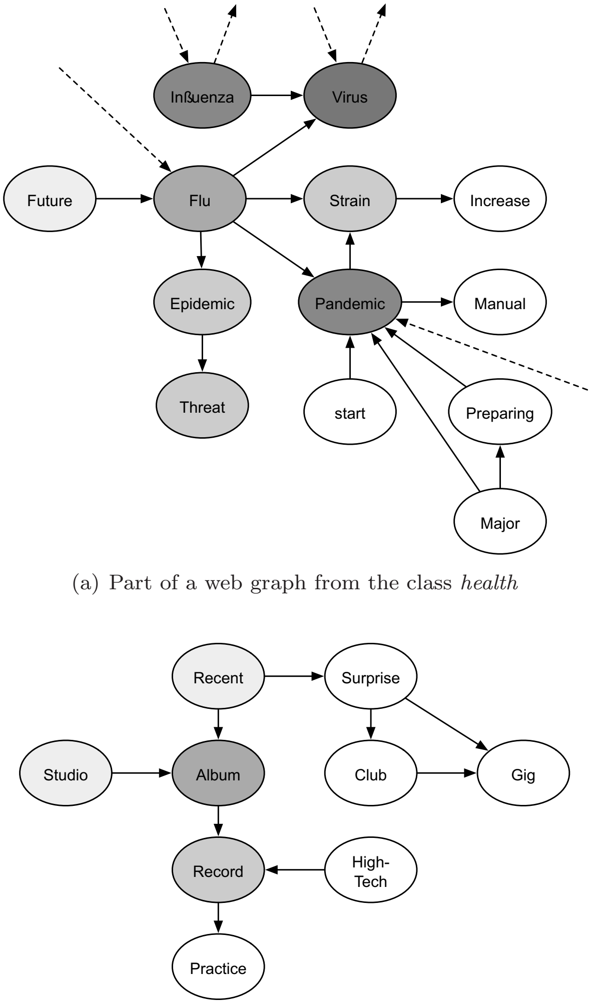
(c) Snowy

(d) People

Fig. 5. Image example from all classes.

This database consists of 1200 images totally, i.e. 300 images per class. The database is split into a training set of size 400, a validation set of size 200, and a test set of size 600.

Web Database. In Ref. 47, several methods for creating graphs from web documents are introduced. For the graphs used in our experiments, the following method was applied. First, all words occurring in the web document – except for stop words, which contain only little information – are converted into nodes in the resulting web graph. We attribute the nodes with the corresponding word and its frequency, i.e. even if a word appears more than once in the same web document, we create only one unique node for it and store its total frequency as an additional node attribute. Next, different sections of the web document are investigated individually. These sections are *title*, which contains the text related to the document's title, *link*, which is a text in a clickable hyperlink, and *text*, which comprises any of the readable text in the web document. If a word w_i immediately precedes word w_{i+1} in any of the sections *title*, *link*, or *text*, a directed edge from the node corresponding to word w_i to the node corresponding to the word w_{i+1} is inserted in our web graph. The resulting edge is attributed with the appropriate section label. Finally, only the most frequently used words (nodes) are kept in the graph and the terms are conflated to the most frequently occurring form. In Fig. 6, two examples of



(a) Part of a web graph from the class *health*

(b) Part of a web graph from the class *music*

Fig. 6. Two examples of partial webgraphs.

webgraphs are shown. Note that these are only part of the graph representing the whole document and different shades of grey represent different frequencies of the corresponding nodes.

In our experiments, we make use of a dataset that consists of 2340 documents from 20 categories (*Business*, *Health*, *Politics*, *Sports*, *Technology*, *Entertainment*, *Art*, *Cable*, *Culture*, *Film*, *Industry*, *Media*, *Multimedia*, *Music*, *Online*, *People*, *Review*, *Stage*, *Television*, and *Variety*). The last 14 categories are subcategories related to entertainment. These web documents were originally hosted at Yahoo as news pages (http://www.yahoo.com). The database is split into a training, a validation, and a test set of equal size (780).

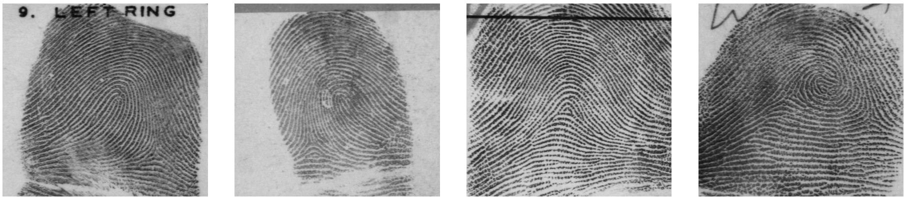
Fingerprint Database. Fingerprints are converted into graphs by filtering the images and extracting regions that are relevant for classification. In order to obtain graphs from fingerprint images, these extracted regions are binarized and a noise removal and thinning procedure is applied. This results in a skeletonized representation of the extracted regions. Ending points and bifurcation points of the skeletonized regions are represented by nodes. Additional nodes are inserted in regular intervals between ending points and bifurcation points. Finally, undirected edges are inserted to link nodes that are directly connected through a ridge in the skeleton. The edges are attributed with an angle denoting the orientation of the edge with respect to the horizontal direction. The procedure of converting fingerprint images into attributed graphs is described in more detail in Ref. 33.

The fingerprint database used in our experiments is based on the NIST-4 reference database of fingerprints.[55] It consists of a validation set of size 300, a test and training set of size 500 each. Thus, there are 1300 fingerprint images totally out of the four classes *arch*, *left*, *right*, and *whorl* from the Galton–Henry classification system.[21] Note that in our experiments, only the four-class problem of fingerprint classification is considered, i.e. the fifth class *tented arch* is merged with the class *arch*. For examples of these fingerprint classes, see Fig. 7.

Molecule Database. The molecule dataset consists of graphs representing molecular compounds. We construct graphs from the AIDS Antiviral Screen Database of Active Compounds.[13] Our molecule database consists of two classes (*active*, *inactive*), which represent molecules with activity against HIV or not. The molecules are converted into graphs in a straightforward manner by representing atoms as nodes and the covalent bonds as edges. Nodes are labeled with the number of the corresponding chemical symbol and edges by the valence of the linkage. In Fig. 8, some molecular compounds of both classes are illustrated. Note that different shades of grey represent different chemical symbols, i.e. node labels.

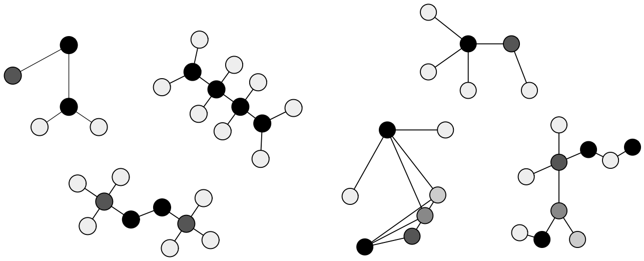
We use a training and validation set of size 250 each, and a test set of size 1500. Thus, there are 2000 elements totally (1600 inactive elements and 400 active elements).

Note that our graph sets can be divided into two categories, viz. continuously labeled graphs (Letter, GREC, Image, and Fingerprint databases), and graphs



(a) Left (b) Right (c) Arch (d) Whorl

Fig. 7. Fingerprint examples from the four classes.



(a) Inactive compounds (b) Active compounds

Fig. 8. Some molecule examples of both classes (a) inactive and (b) active.

labeled with discrete symbols (Web and Molecule databases). The proposed embedding kernel makes use of graph edit distance, which can handle both types of graphs.

4.2. *Reference systems*

In contrast with the high representational power of graphs, we observe a lack of general classification algorithms that can be applied in the graph domain. One of the few classifiers directly applicable to arbitrary graphs is the k-nearest-neighbor classifier (k-NN) in conjunction with graph edit distance. Given a labeled set of training graphs, an unknown graph is assigned to the class that occurs most frequently among the k nearest graphs (in terms of edit distance) from the training set. The decision boundary of this classifier is a piecewise linear function which makes it very flexible. If $k = 1$, the k-NN classifier's decision is based on just one element from the training set, no matter if this element is an outlier or a true class representative. Obviously, a choice of parameter $k > 1$ reduces the influence of outliers by evaluating which class occurs most frequently in a neighborhood around the test pattern. This classifier in the graph domain will serve us as our first reference system.

The second reference system is a trivial similarity kernel in conjunction with an SVM. This approach basically turns the existing dissimilarity measure (graph edit distance) into a similarity measure by mapping low distance values to high similarity values and vice versa. To this end, we use a simple monotonically decreasing transformation. Given the edit distance $d(g, g')$ of two graphs g and g', the similarity kernel is defined as $\kappa(g, g') = -d(g, g')^2$. In Ref. 19, this similarity kernel is explicitly suggested for classifying distance-based data. Note that this kernel function is not positive definite and is therefore not a valid kernel. However, there is theoretical evidence that using kernel machines in conjunction with indefinite kernels may be reasonable if some conditions are fulfilled.[36,19] Comparing the performance of this trivial kernel function with the performance of the prototype based embedding procedure offers us the possibility to understand whether the power of our system is primarily due to the embedding process or to the strength of the kernel classifier.

The third reference system is given by a random walk kernel. The basic idea of this type of graph kernel is to define graph similarity by comparing random walks in two graphs.[16,26,27,18,5,4] For continuously labeled graphs, the kernel actually used as the third reference system is the one proposed in Ref. 5. This kernel is quite flexible and can handle noisy data. For the graph sets labeled with symbolic information, the kernel proposed in Ref. 18 is used as the third reference system.

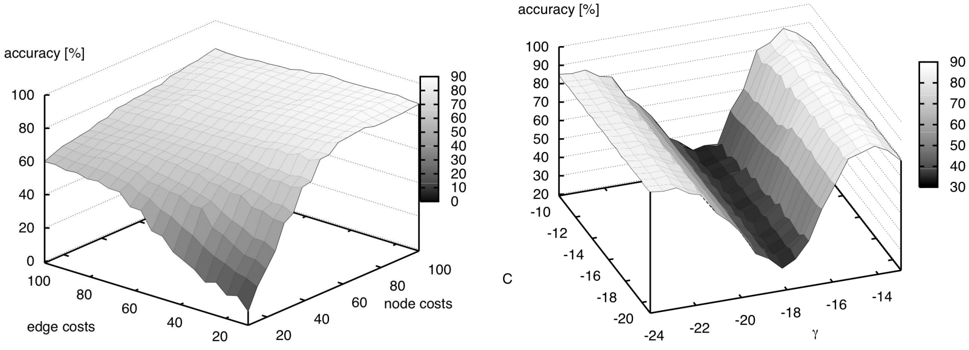
4.3. *Experimental setup and validation of the meta parameters*

In our experiments, we divide each database into three disjoint subsets, viz. the training, the validation, and the test set. The training set is used to construct and train the classifier. If an SVM is used, the support vectors and the hyperplanes separating the classes are derived from the training set. If a nearest neighbor classifier is applied, the graphs from the training set are used as labeled prototypes to find the nearest neighbors. The validation set is used to determine those meta parameters that cannot directly be obtained from the training procedure. We run the classifier with different meta parameters and report the accuracies achieved on the validation set. The meta parameter values that lead to the best performance on the validation set are finally applied to the independent test set and the resulting accuracies are reported. Examples of meta parameters are the cost function c for graph edit distance, the number of patterns k considered by the nearest neighbor classifier, the best performing embedding algorithm, and the dimensionality of the target vector space m.

In the validation phase, the costs of the edit operations are determined first. For all considered graph data sets but the web database, node and edge labels are integer numbers, real numbers, or real vectors. Here, the substitution cost of a pair of labels is given by a distance measure (e.g. Euclidean distance), and only the deletion and insertion costs have to be determined. For the sake of simplicity, we assumed in all experiments identical costs for node deletions and node insertions, and for edge deletions and insertions. Hence only two parameters, node deletion/insertion cost, and edge deletion/insertion cost, need to be validated. For the web database, we followed the cost model used in Ref. 47, i.e. node substitutions are not admissible and the costs of all other edit operations are set to an arbitrary constant. Consequently, no edit cost validation is needed for this dataset.

The validation procedure is illustrated in Fig. 9(a), where the classification results on the validation set as a function of the two edit cost parameters are shown for the GREC symbol database. Finally, the parameter pair that leads to the highest classification accuracy on the validation set is used on the independent test set to perform nearest-neighbor classification in the graph domain (first reference system) and for graph embedding in real vector spaces.

For the second reference system, the trivial similarity kernel in conjunction with an SVM, the same edit distances as for the k-NN classifier are used. Hence, the weighting parameter C, which controls whether the maximization of the margin or the minimization of the error is more important, is the only additional parameter that has to be validated.



(a) Graph edit cost validation

(b) Validation of SVM parameters

Fig. 9. Parameter validation on the GREC database.

For the third reference system, the SVM with random walk kernel, two or three meta parameters have to be validated, depending on the type of random walk kernel actually used. The kernel in Ref. 18 is characterized by the ability of computing the number of matching walks in two graphs g_1 and g_2 by means of the direct product graph of g_1 and g_2. To this end, one has to sum up powers of the adjacency matrix of the direct product graph weighted with a meta parameter λ. Hence, the weighting parameter λ is the first parameter that has to be found by means of the validation set. Since we make use of an SVM, the weighting parameter C has to be also validated. If the method described in Ref. 5 is used, a third parameter value has to be determined. In order to measure the similarity of continuously labeled nodes and edges, we make use of an RBF kernel function $k(\mathbf{x}, \mathbf{x}') = \exp(-\gamma\|\mathbf{x} - \mathbf{x}'\|^2)$, where $\gamma > 0$ denotes the third meta parameter.

For the task of graph embedding in real vector spaces, two meta parameters have to be validated, namely, the prototype selection method and the number of prototypes selected, i.e. the dimensionality of the target feature space. In order to determine suitable values of these two meta parameters, each graph set is embedded in a vector space with all of the prototype selectors described in Sec. 3.2, varying the dimensionality of the target vector space over a certain interval. With the resulting vector sets, which still consist of a validation, a training, and a test set, an SVM is trained. We make use of an SVM with RBF-kernel where besides the weighting parameter C, the meta parameter γ in the kernel function has to be optimized.[c] In Fig. 9(b), such an SVM parameter validation on the GREC database is illustrated.

The SVM optimization is performed on a validation set for every possible dimension of the target space and every prototype selection method. Thus, for each

[c] We make use of the RBF kernel since it is the most widely used kernel and it has been extensively studied in various fields of pattern recognition.[49] In fact, unfortunately it is not always possible to make the right choice of a kernel *a priori*. Note that in Ref. 42, also other kernel functions are applied to the embedded graphs, viz. a linear and a polynomial kernel. However, it turns out that for this particular problem, the RBF-kernel performs generally the best.

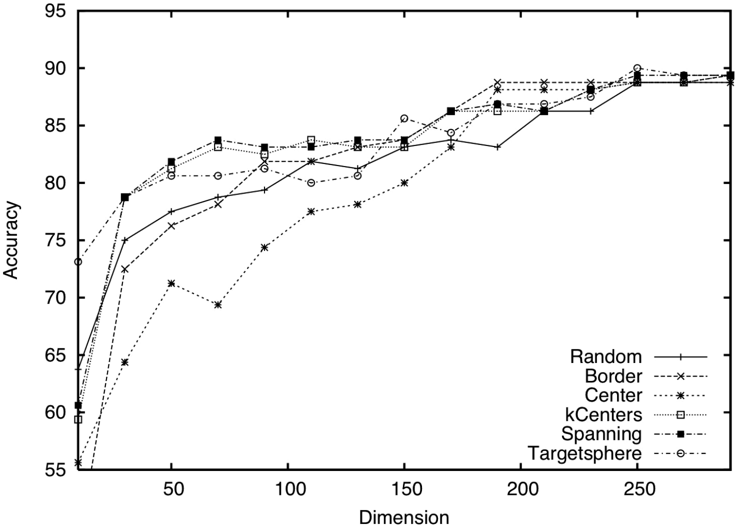


Fig. 10. Validation of prototype selector and dimensionality of the target vector space on the GREC database.

embedding procedure, the classification accuracy can be regarded as a function of the dimensionality and the prototype selector. This final optimization is illustrated on the GREC database in Fig. 10 where the accuracies for each prototype selector and each dimensionality are shown.

By means of the procedure described before, the globally best performing embedding method, the number of prototypes m, and the SVM parameters (C, γ) can be determined for every dataset on the validation set. The parameter combination that results in the lowest classification error is finally applied to the independent test set. The results obtained on the test set by means of this procedure are reported and discussed in the next subsection.

4.4. *Results and discussion*

In Table 1, the classification accuracies on all databases achieved by the k-NN classifier (first reference system), the similarity kernel (second reference system), the random walk kernel (third reference system), and our novel approach are provided. Furthermore, for each data set, the optimum number of prototypes, m, and the specific prototype selector, PS, as found on the validation set are indicated. Let us first compare the classification accuracies achieved by the embedding kernel with the first reference system. It clearly turns out that the novel procedure is much more powerful than the traditional k-NN in the graph domain as on all tested applications, the embedding kernel outperforms the first reference system's classification results. Note that seven out of eight improvements are statistically significant.

Next, we compare the embedding kernel with the similarity kernel. It turns out that the similarity kernel is better on three databases (Letter Low, GREC and

Table 1. Classification accuracies.

	Reference Systems			Embedding Classifiers		
Database	k-NN	Sim. Kernel	Walk Kernel	Embedding Kernel	m	PS
Letter Low	91.07	92.13	89.93	91.80 ③	225	k-CPS
Letter Med	76.80	80.33	73.47	81.80 ①②③	450	TPS
Letter High	61.60	71.67	61.20	74.27 ①②③	270	k-CPS
GREC	86.04	91.86	89.79	89.17 ①	250	TPS
Image	59.50	66.17	77.00	75.50 ①②	40	CPS
Webgraphs	77.44	84.87	82.18	82.44 ①❷	130	k-CPS
Fingerprints	82.60	79.40	83.60	85.00 ①②	300	TPS-C
Molecules	97.13	93.47	98.07	98.13 ①②	150	SPS

Z-test for testing statistical significance ($\alpha = 0.05$):

①/②/③ Stat. significant improvement over the first/second/third reference system.

❶/❷/❸ Stat. significant deterioration compared to the first/second/third reference system.

Webgraphs). However, the embedding kernel outperforms the similarity kernel on five other applications. Note that of all the improvements, only one of the deteriorations as statistically significant. From these findings, we can conclude that the power of our novel approach primarily results from the embedding process itself and not from to the strength of the kernel classifier.

Finally, we observe that the random walk kernel is better than the proposed procedure on two databases (GREC and Image). However, the embedding kernel outperforms the random walk kernel on six other databases. Note that three out of six improvements are statistically significant. The random walk kernel seems to have problems with the Letter databases where its performance is even lower than that of the first reference system (in two out of three cases even with statistical significance). Hence, the embedding kernel achieves not only better accuracies on most databases but also provides us with a higher degree of robustness under a broad spectrum of different classification tasks.

By means of the proposed graph embedding procedure, we are able to transform arbitrary graphs into vectors of real numbers. It is obvious that graph embedding in real vector spaces can lead to a loss of information. That is, regardless of how elaborate the prototype selection procedure is, the representational power of graphs cannot be completely maintained. In fact, applying a classifier of the nearest-neighbor type to the resulting vectorial descriptions potentially leads to a deterioration of the classification accuracy. This finding was reported in Ref. 42 for some of the datasets used in the present paper. However, by means of graph embedding, we gain the possibility to access a wide spectrum of classification algorithms for vectorial data (e.g. SVM). Regarding the results of Table 1, one can conclude that the loss of information can be more than compensated by an SVM, which is not directly applicable to the original graph data.

For a more detailed analysis of the proposed approach, it is useful to look at the prototype selectors. First of all, there is no globally best prototype selector for all databases. The k-Centers and the Targetsphere prototype selector are used three times, while the Spanning and the Center prototype selector are used one time each. Note furthermore that the class-wise selection is able to outperform the class-independent selection only on one database. One can conclude that it remains difficult to predict which of the prototype selectors will perform best on a specific graph set. However, one tendency can be clearly observed. Seven out of the eight best prototype selectors try to distribute the prototypes uniformly over the whole graph set. Note that this result reflects the intuitive observation that the prototypes should avoid redundancies and include as much information as possible.

The optimal number m of prototypes selected, i.e. the dimensionality of the target vector space, is also of interest. On the average, there are about 44% of all training graphs selected as prototypes. However, similarly to the prototype selection method, the optimal dimensionality of the resulting vector space does not follow a global rule. It varies from only 10% of all training graphs on the image database to about 78% of all training graphs on the GREC database. Of course, from the computational complexity point of view, a smaller number of prototypes is preferable, because the smaller the number of prototypes is, the fewer graph edit distance computations have to be performed for transforming graphs into real vectors.

5. Conclusions

For objects given in terms of feature vectors, a rich repository of algorithmic tools for classification has been developed over the past years. Graphs are a versatile alternative to feature vectors, and are known to be a powerful and flexible representation formalism. The representational power of graphs is due to their ability to represent not only feature values but also relationships among different parts of an object, and their flexibility comes from the fact there are no size or labeling constraints that constrain the representation of a given object. However, graph based object representation suffers from the lack of mathematical structure in the graph domain. In contrast to vectors, most of the basic mathematical operations required for classification do not exist for graphs. Graph kernels, a relatively novel approach for graph based classification, offer an elegant solution to this problem as they provide us with an embedding of the domain of graphs into an inner product space. The key idea of such kernels is that rather than defining handcrafted mathematical operations in the original graph domain, the graphs are mapped into a feature vector space where all the required mathematical operations are readily available. Hence, the basic limitation of graph representations, i.e. the lack of algorithmic tools in the graph domain due to its little mathematical structure, can be overcome.

In the present paper, a novel approach to graph embedding using prototypes and dissimilarities, is proposed. Our embedding procedure explicitly makes use of graph edit distance and can therefore deal with various kinds of graphs (labeled, unlabeled, directed, undirected, etc.). Furthermore, one can utilize domain specific knowledge in defining costs for edit operations adapted to the underlying node and edge labels. The basic idea of the embedding method is to describe a graph by means of m dissimilarities to a predefined set of graphs termed prototypes. That is, a graph g is mapped explicitly to the m-dimensional real space $\mathbb{R}^m$ by arranging the edit distances of g to all of the m prototypes as a vector. By means of this procedure, one obtains not only pairwise scalar products in an implicit kernel feature space but also the individual maps of each graph. Consequently, not only kernel machines but also non-kernelizable algorithms can be applied to the resulting maps.

In several experiments, a high degree of robustness and flexibility of the proposed approach is empirically verified. On eight graph sets, the classification accuracy of the proposed embedding kernel is compared to a k-nearest neighbor classifier in the original graph domain (first reference system), a similarity kernel (second reference system), and a random walk kernel in an implicit kernel feature space (third reference system). Our approach outperforms the first reference system on all data sets and the second and third reference systems on most datasets (five out of eight and six out of eight, respectively). Moreover, our approach is characterized by a high degree of flexibility, i.e. the proposed kernel achieves good classification accuracies on all of our databases.

The question which of the prototype selectors is globally best cannot definitely be answered. On eight datasets, four different selection methods are superior to the others. There is a clear tendency that prototype selectors that distribute the graphs more or less uniformly over the whole graph set lead to higher recognition rates. Consequently, in case there is no possibility to do independent validation experiments, a selector among the three methods k-Centers, Spanning, or Targetsphere might be a good choice.

There are a number of issues to be investigated in future research. For example, there seems to be room for developing and investigating additional prototype selectors. Moreover, there might be possibilities for weighting the individual prototypes according to the information given by the prototype selection method (e.g. the cluster size in k-Centers). That is, a feature of the embedded graph, i.e. the distance to a prototype, could be weighted according to the importance of this particular prototype. Taking a more fundamental approach, the problem of prototype selection could be viewed as a feature selection problem, for which many methods are available. Also, the integration of different prototype selectors into a multiple classifier system could be a rewarding avenue to be explored. Furthermore, all experiments described in this paper are based on SVMs. It would be interesting to conduct similar experiments with other classifiers. Also, the use of the proposed kernel function in conjunction with clustering algorithms seems to be an attractive topic for further investigation. Finally, in this paper, we have used only one graph distance measure, i.e. the edit distance. It would be interesting to evaluate the proposed embedding method in conjunction with other types of dissimilarity functions.

Acknowledgments

This work has been supported by the Swiss National Science Foundation (Project 200021-113198/1). Furthermore, we would like to thank R. Duin and E. Pekalska for valuable discussions and hints regarding our embedding methods. We thank B. Le Saux for making the Image database and the corresponding applications available to us. Finally, we thank M. Ferrer for his help with the distortion operators on the GREC symbol database, and M. Neuhaus for useful hints and discussion.

References

1. M. Aizerman, E. Braverman and L. Rozonoer, Theoretical foundations of potential function method in pattern recognition learning, *Autom. Rem. Contr.* **25** (1964) 821–837.
2. R. Ambauen, S. Fischer and H. Bunke, Graph edit distance with node splitting and merging and its application to diatom identification, *Proc. 4th Int. Workshop on Graph Based Representations in Pattern Recognition*, eds. E. Hancock and M. Vento, LNCS 2726 (Springer, 2003), pp. 95–106.
3. M. C. Boeres, C. C. Ribeiro and I. Bloch, A randomized heuristic for scene recognition by graph matching, *Proc. 3rd Workshop on Efficient and Experimental Algorithms*, eds. C. C. Ribeiro and S. L. Martins, LNCS 3059 (Springer, 2004), pp. 100–113.
4. K. Borgwardt and H.-P. Kriegel, Shortest-path kernels on graphs, *Proc. 5th Int. Conf. on Data Mining* (2005), pp. 74–81.
5. K. Borgwardt, C. Ong, S. Schönauer, S. Vishwanathan, A. Smola and H.-P. Kriegel, Protein function prediction via graph kernels, *Bioinformatics* **21**(1) (2005) 47–56.
6. B. E. Boser, I. Guyon and V. Vapnik, A training algorithm for optimal margin classifiers, *Computational Learning Theory* (1992), pp. 144–152.
7. J. Bourgain, On Lipschitz embedding of finite metric spaces in Hilbert spaces, *Israel J. Math.* **52**(1–2) (1985) 46–52.
8. H. Bunke and G. Allermann, Inexact graph matching for structural pattern recognition, *Patt. Recogn. Lett.* **1** (1983) 245–253.
9. H. Bunke and K. Shearer, A graph distance metric based on the maximal common subgraph, *Patt. Recogn. Lett.* **19**(3) (1998) 255–259.
10. D. Comaniciu and P. Meer, Robust analysis of feature spaces: Color image segmentation, *IEEE Conf. Comp. Vision and Pattern Recognition* (1997), pp. 750–755.
11. D. Conte, P. Foggia, C. Sansone and M. Vento, Thirty years of graph matching in pattern recognition, *Int. J. Patt. Recogn. Artif. Intell.* **18**(3) (2004) 265–298.
12. Ph. Dosch and E. Valveny, Report on the second symbol recognition contest, *Graphics Recognition. Ten Years Review and Future Perspectives. Proc. 6th Int. Workshop on Graphics Recognition (GREC'05)*, eds. L. Wenyin and J. Lladós, LNCS 3926 (Springer, 2005), pp. 381–397.
13. Development Therapeutics Program DTP, AIDS antiviral screen, 2004, http://dtp.nci.nih.gov/docs/aids/aids_data.html.
14. R. Duda, P. Hart and D. Stork, *Pattern Classification* (Wiley-Interscience, 2nd edition, 2000).
15. M.-L. Fernandez and G. Valiente, A graph distance metric combining maximum common subgraph and minimum common supergraph, *Patt. Recogn. Lett.* **22**(6–7) (2001) 753–758.
16. T. Gärtner, Exponential and geometric kernels for graphs, *NIPS Workshop on Unreal Data: Principles of Modeling Nonvectorial Data* (2002).

17. T. Gärtner, A survey of kernels for structured data, *SIGKDD Explorations* **5**(1) (2003) 49–58.
18. T. Gärtner, P. Flach and S. Wrobel, On graph kernels: Hardness results and efficient alternatives, *Proc. 16th Annual Conf. Learning Theory*, eds. B. Schölkopf and M. Warmuth (2003), pp. 129–143.
19. B. Haasdonk, Feature space interpretation of SVMs with indefinite kernels, *IEEE Trans. Patt. Anal. Mach. Intell.* **27**(4) (2005) 482–492.
20. P. E. Hart, The condensed nearest neighbor rule, *IEEE Trans. Inform. Th.* **14**(3) (1968) 515–516.
21. E. Henry, *Classification and Uses of Finger Prints* (Routledge, London, 1900).
22. G. Hjaltason and H. Samet, Properties of embedding methods for similarity searching in metric spaces, *IEEE Trans. Patt. Anal. Mach. Intell.* **25**(5) (2003) 530–549.
23. X. Jiang, A. Münger and H. Bunke, On median graphs: Properties, algorithms, and applications, *IEEE Tran. Patt. Anal. Mach. Intell.* **23**(10) (2001) 1144–1151.
24. W. Johnson and J. Lindenstrauss, Extensions of Lipschitz mappings into Hilbert space, *Contemp. Math.* **26** (1984) 189–206.
25. D. Justice and A. Hero, A binary linear programming formulation of the graph edit distance, *IEEE Trans. Patt. Anal. Mach. Intell.* **28**(8) (2006) 1200–1214.
26. H. Kashima and A. Inokuchi, Kernels for graph classification, *Proc. ICDM Workshop on Active Mining* (2002), pp. 31–36.
27. H. Kashima, K. Tsuda and A. Inokuchi, Marginalized kernels between labeled graphs, *Proc. 20th Int. Conf. Machine Learning* (2003), pp. 321–328.
28. L. Kaufman and P. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis* (John Wiley & Sons, 1990).
29. B. Le Saux and H. Bunke, Feature selection for graph-based image classifiers, *Proc. 2nd Iberian Conf. Pattern Recognition and Image Analysis, Part II*, eds. J. Marques, N. Perez de Blanca and P. Pina, LNCS 3523 (Springer, 2005), pp. 147–154.
30. G. Levi, A note on the derivation of maximal common subgraphs of two directed or undirected graphs, *Calcolo* **9** (1972) 341–354.
31. B. Luo, R. Wilson and E. R. Hancock, Spectral embedding of graphs, *Patt. Recogn.* **36**(10) (2003) 2213–2223.
32. B. D. McKay, Practical graph isomorphism, *Congressus Numerantium* **30** (1981) 45–87.
33. M. Neuhaus and H. Bunke, A graph matching based approach to fingerprint classification using directional variance, *Proc. 5th Int. Conf. Audio- and Video-Based Biometric Person Authentication*, eds. T. Kanade, A. Jain and N. K. Ratha, LNCS 3546 (Springer, 2005), pp. 191–200.
34. M. Neuhaus and H. Bunke, Self-organizing maps for learning the edit costs in graph matching, *IEEE Trans. Syst. Man Cybern. (Part B)* **35**(3) (2005) 503–514.
35. M. Neuhaus and H. Bunke, Automatic learning of cost functions for graph edit distance, *Inform. Sci.* **177**(1) (2007) 239–247.
36. M. Neuhaus and H. Bunke, *Bridging the Gap Between Graph Edit Distance and Kernel Machines* (World Scientific, 2007).
37. M. Neuhaus, K. Riesen and 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 and D. de Ridder, LNCS 4109 (Springer, 2006), pp. 163–172.
38. E. Pekalska and R. Duin, *The Dissimilarity Representation for Pattern Recognition: Foundations and Applications* (World Scientific, 2005).

39. E. Pekalska, R. Duin and P. Paclik, Prototype selection for dissimilarity-based classifiers, *Patt. Recogn.* **39**(2) (2006) 189–208.
40. J. Ramon and T. Gärtner, Expressivity versus efficiency of graph kernels, *Proc. First International Workshop on Mining Graphs, Trees and Sequences* (2003), pp. 65–74.
41. K. Riesen, M. Neuhaus and H. Bunke, Bipartite graph matching for computing the edit distance of graphs, *Proc. 6th Int. Workshop on Graph Based Representations in Pattern Recognition*, eds. F. Escolano and M. Vento, LNCS 4538 (2007), pp. 1–12.
42. K. Riesen, M. Neuhaus and H. Bunke, Graph embedding in vector spaces by means of prototype selection, *Proc. 6th Int. Workshop on Graph Based Representations in Pattern Recognition*, eds. F. Escolano and M. Vento, LNCS 4538 (2007), pp. 383–393.
43. A. Robles-Kelly and E. R. Hancock, Graph edit distance from spectral seriation, *IEEE Trans. Patt. Anal. Mach. Intell.* **27**(3) (2005) 365–378.
44. A. Robles-Kelly and E. R. Hancock, A Riemannian approach to graph embedding, *Patt. Recogn.* **40** (2007) 1024–1056.
45. J. S. Sanchez, F. Pla and F. J. Ferri, Prototype selection for the nearest neighbour rule through proximity graphs, *Patt. Recogn. Lett.* **18**(6) (1997) 507–513.
46. A. Sanfeliu and 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. A. Schenker, H. Bunke, M. Last and A. Kandel, *Graph-Theoretic Techniques for Web Content Mining* (World Scientific, 2005).
48. B. Schölkopf and A. Smola, *Learning with Kernels* (MIT Press, 2002).
49. J. Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis* (Cambridge University Press, 2004).
50. S. Sorlin and C. Solnon, Reactive tabu search for measuring graph similarity, *Proc. 5th Int. Workshop on Graph-Based Representations in Pattern Recognition*, eds. L. Brun and M. Vento, LNCS 3434 (Springer, 2005), pp. 172–182.
51. B. Spillmann, M. Neuhaus, H. Bunke, E. Pekalska and R. Duin, Transforming strings to vector spaces using prototype selection, *Proc. 11th Int. Workshop on Strucural and Syntactic Pattern Recognition*, eds. D.-Y. Yeung, J. T. Kwok, A. Fred, F. Roli and D. de Ridder, LNCS 4109 (Springer, 2006), pp. 287–296.
52. J. R. Ullman, An algorithm for subgraph isomorphism, *J. Assoc. Comput. Mach.* **23**(1) (1976) 31–42.
53. V. Vapnik, *Statistical Learning Theory* (John Wiley, 1998).
54. W. D. Wallis, P. Shoubridge, M. Kraetzl and D. Ray, Graph distances using graph union, *Patt. Recogn. Lett.* **22**(6) (2001) 701–704.
55. C. I. Watson and C. L. Wilson, *NIST Special Database 4, Fingerprint Database*, National Institute of Standards and Technology, March 1992.
56. D. R. Wilson and T. R. Martinez, Reduction techniques for instance-based learning algorithms, *Mach. Learn.* **38**(3) (2000) 257–286.
57. R. Wilson and E. R. Hancock, Levenshtein distance for graph spectral features, *Proc. 17th Int. Conf. Pattern Recognition*, eds. J. Kittler, M. Petrou and M. Nixon, Vol. 2 (2004), pp. 489–492.
58. R. C. Wilson, E. R. Hancock and B. Luo, Pattern vectors from algebraic graph theory, *IEEE Trans. Patt. Anal. Mach. Intell.* **27**(7) (2005) 1112–1124.

Kaspar Riesen received his M.S. degree in computer science from the University of Bern, Switzerland, in 2006. Currently he is a Ph.D. student and lecture assistant in the research group of Computer Vision and Artificial Intelligence at the University of Bern, Switzerland.

His research interests include structural pattern recognition, and in particular, graph embeddings in real vector spaces. He has more than 25 publications, including five journal papers.

Horst Bunke received his M.S. and Ph.D. degrees in computer science from the University of Erlangen, Germany. In 1984, he joined the University of Bern, Switzerland, where he is a professor in the Computer Science Department. He was Department Chairman from 1992–1996, Dean of the Faculty of Science from 1997 to 1998, and a member of the Executive Committee of the Faculty of Science from 2001 to 2003.

From 1998 to 2000, Horst Bunke served as 1st Vice-President of the International Association for Pattern Recognition (IAPR). In 2000 he was also the Acting President of this organization. Horst Bunke is a Fellow of the IAPR, former Editor-in-Charge of the *Int. J. Pattern Recognition and Artificial Intelligence*, Editor-in-Chief of the *J. Electronic Letters of Computer Vision and Image Analysis*, Editor-in-Chief of the book series on *Machine Perception and Artificial Intelligence* by World Scientific Publ. Co., Advisory Editor of *Pattern Recognition*, Associate Editor of *Acta Cybernetica and Frontiers of Computer Science* in China, and Former Associate Editor of the *Int. J. Document Analysis and Recognition, and Pattern Analysis and Applications*.

Horst Bunke received an honorary doctor degree from the University of Szeged, Hungary, and held visiting positions at the IBM Los Angeles Scientific Center (1989), the University of Szeged, Hungary (1991), the University of South Florida at Tampa (1991, 1996, 1998–2006), the University of Nevada at Las Vegas (1994), Kagawa University, Takamatsu, Japan (1995), Curtin University, Perth, Australia (1999), and Australian National University, Canberra (2005).

Horst Bunke was on the program and organization committee of many conferences and served as a referee for numerous journals and scientific organizations. He is on the Scientific Advisory Board of the German Research Center for Artificial Intelligence (DFKI). Horst Bunke has more than 550 publications, including 36 authored, co-authored, edited or co-edited books and special editions of journals.