Pattern Recognition 44 (2011) 1057–1067

Contents lists available at ScienceDirect

Pattern Recognition

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

Recent advances in graph-based pattern recognition with applications in document analysis

Horst Bunke, Kaspar Riesen *

Institute of Computer Science and Applied Mathematics, University of Bern, Neubrückstrasse 10, CH-3012 Bern, Switzerland

ARTICLE INFO

Article history:
Received 17 December 2009
Received in revised form
18 October 2010
Accepted 21 November 2010

Keywords:
Graph-based representation
Graph kernel
Graph embedding
Graph classification

ABSTRACT

Graphs are a powerful and popular representation formalism in pattern recognition. Particularly in the field of document analysis they have found widespread application. From the formal point of view, however, graphs are quite limited in the sense that the majority of mathematical operations needed to build common algorithms, such as classifiers or clustering schemes, are not defined. Consequently, we observe a severe lack of algorithmic procedures that can directly be applied to graphs. There exists recent work, however, aimed at overcoming these limitations. The present paper first provides a review of the use of graph representations in document analysis. Then we discuss a number of novel approaches suitable for making tools from statistical pattern recognition available to graphs. These novel approaches include graph kernels and graph embedding. With several experiments, using different data sets from the field of document analysis, we show that the new methods have great potential to outperform traditional procedures applied to graph representations.

1. Introduction

The question how to represent objects in a formal way is a key issue in the whole discipline of pattern recognition and in particular in document analysis. There are two major ways to tackle this crucial problem, viz. the *statistical* and the *structural* approach. In the statistical approach, objects are represented by feature vectors. That is, an object is formally represented as a vector $\mathbf{x}=(x_1,\ldots,x_n)\in\mathbb{R}^n$ of n measurements. Representing objects or patterns by feature vectors $\mathbf{x}\in\mathbb{R}^n$ offers a number of useful properties, in particular, the mathematical wealth of operations available in a vector space. That is, computing the sum, the product, the mean, or the distance of two entities is well defined in vector spaces, and moreover, can be efficiently accomplished. 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 pattern recognition, document analysis, and related fields [1].

However, the use of feature vectors implicates two limitations. First, as vectors always represent a predefined set of features, all vectors in a given application have to preserve the same length regardless of the size or complexity of the corresponding objects. Second, there is no direct possibility to describe binary relationships that might exist among different parts of an object. These two drawbacks are severe, particularly when the patterns under consideration are characterized by complex structural relationships rather than the statistical distribution of a fixed set of pattern features [2].

The structural approach is based on symbolic data structures, such as *strings*, *trees*, or *graphs*, out of which graphs are the most general one. In fact, from an algorithmic perspective both strings and trees are simple instances of graphs. A string is a graph in which each node represents one symbol, and two consecutive symbols are connected by an edge. A tree is a graph in which any two nodes are connected by exactly one path. In Fig. 1 an example of a string, a tree, and a graph are illustrated. Although focusing on graphs in this article, the reader should keep in mind that strings and trees are always included as special cases.

The above-mentioned drawbacks of feature vectors, namely the size constraint and the lacking ability of representing relationships, can be overcome by graph-based representations [3]. In fact, graphs are not only able to describe properties of an object, but also binary relationships among different parts of the underlying object by means of edges. Note that these relationships can be of various nature (spatial, temporal, conceptual, etc.). Moreover, graphs are not constrained to a fixed size, i.e. the number of nodes and edges is not limited a priori and can be adapted to the size and the complexity of each individual object under consideration.

One drawback of graphs, when compared to feature vectors, is the significant increase of the 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

* Corresponding author.
E-mail addresses: bunke@iam.unibe.ch (H. Bunke), riesen@iam.unibe.ch (K. Riesen).

0031-3203/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.patcog.2010.11.015

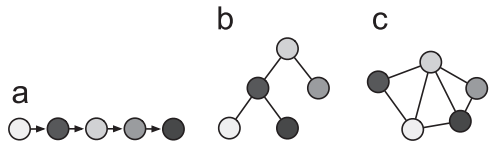

Fig. 1. Symbolic data structures. (a) String, (b) tree, (c) graph.

Table 1
Feature vectors and graphs have complementary properties.

	Vectors	Graphs
Representational power	–	+
Available tools	+	–

two graphs for isomorphism, only exponential algorithms are known today. Another serious limitation in the use of graphs arises from the fact that there is little mathematical structure in the domain of graphs. For example, computing the (weighted) sum or the product of a pair of entities (which are elementary operations required in many standard algorithms) is not possible in the domain of graphs, or is at least not defined in a standardized way. Due to these general problems in the graph domain, we observe a lack of algorithmic tools for graph-based pattern recognition and document analysis.

The objective of the present article is twofold. First, it reviews recent work in graph-based pattern recognition, including graph kernels and a novel graph embedding approach proposed by the authors recently. This recent approach addresses the question of how to combine the complementary properties of feature vectors and graphs (cf. Table 1) in order to get the best out of both worlds. Various aspects of this new graph embedding approach have been published in a number of individual papers. The present article provides a unified and consolidated view of this new approach and shows how it makes an important contribution towards bridging the gap between structural pattern recognition (with its versatility in object description) and the statistical approach (with its numerous algorithms and tools at hand). The second major goal of this article is to provide a review of previous work on graphs in document analysis. Moreover, in an experimental evaluation we show how graph kernels and the novel embedding framework can be used to improve the performance of various tasks in document analysis.

2. Graphs in document analysis—previous work

Due to the ability of graphs to represent properties of entities and binary relations at the same time, a growing interest in graph-based representation can be observed [3]. That is, graphs have found widespread applications in pattern recognition and document analysis. This section provides a review of some important contributions to the field of document analysis that use graph representation. For a more comprehensive and general review of work done in document analysis during the last decades we refer to [4–7].

An area where graph-based representations have been intensively used is graphics recognition [8–14]. In [8] graphs are used for building a model-based scheme for recognizing hand-drawn symbols in schematic diagrams. The observed symbol drawing of one-pixel width is converted into a graph such that endpoints, junctions, and crossings are represented by nodes attributed with the number of neighbors and the angles between incident edges. The edges represent connecting lines in the drawing attributed with the length and curvature of the respective line. In the recognition phase, a small number of candidate graphs are selected first by means of geometrical constraints. A distance measure between graphs extracted from the drawing and candidate graphs from a database is defined. This distance measure is based on the correspondences found between components of the two graphs under consideration. Finally, the classification of the unknown symbol is carried out using a nearest-neighbor classifier.

In [9] the task to be solved also consists of recognizing symbols in hand-drawn diagrams. However, these drawings are represented by region adjacency graphs, where nodes represent closed regions in the drawing and edges represent the adjacency relation between these regions. The nodes are attributed with a boundary string resulting from polygonal approximation of the region. In order to measure the similarity of two regions, string matching techniques are applied to the related boundary strings. By means of an error-tolerant subgraph isomorphism, instances of the symbol can then be found in the large drawing image. This work is remarkable in the sense that it combines string matching (at the local level) with graph matching (at the global level).

The authors of [10] also address the recognition of symbols in a diagram using error-tolerant subgraph isomorphism. Here the nodes represent line segments without attributes and the edges connect two line segments that have a common endpoint. Edges are attributed with the angle between the corresponding line segments. As a crucial extension to other works, the framework in [10] supports dynamic learning of new symbols using some simple meta rules.

The detection of component parts in CAD images of mechanical drawings is addressed in [11]. This problem is tackled by means of a novel subgraph matching algorithm which exploits semantic information about nodes while ignoring the information about the topology of the graphs to be matched.

In [12] graph structures representing technical drawings are classified into two categories depending on whether the structure they represent consists of prototype patterns or repetitive patterns. Depending on this classification, the symbol recognition problem is formulated either in terms of graph matching or graph parsing. Since some symbols include both types of structures, the main contribution of this work is to propose a combined strategy of matching and parsing of graphs.

An early approach to circuit diagram analysis was proposed in [13]. This approach is based on the idea of modeling both individual symbols and complete circuit diagrams by a programmed graph grammar [14]. By applying the grammar rules in the reverse mode, the difficult problem of graph parsing can be avoided and a compact logical description is obtained from a given circuit diagram. As it has been demonstrated in practical experiments, significant amounts of errors and distortions can be dealt with by suitably chosen grammar rules.

Handwriting recognition and machine printed character recognition are further areas of research where graphs have gained noticeable attention [15–20]. In [15] a two-level hierarchical graph representation is proposed in order to represent handwritten Chinese characters. At the lower level, the nodes of the graphs represent the strokes of a character attributed with their respective orientation, while the edges indicate relations between strokes attributed with the type of junction. In the higher level representation, graphs are used to represent connected components with their spatial relations. For classification, error-tolerant graph matching is carried out mapping a given candidate graph to model graphs. Due to the hierarchy in the graph representation, a combinatorial explosion in the graph matching process can be avoided and an efficient recognition system is obtained.

Chinese character recognition is also addressed in [16]. In this work each character class is described by a constrained graph model. The nodes in these graphs represent sampling points on the

strokes, while connection and position constraints are represented as edges. Based on this model, character recognition is formulated as a constrained optimization problem using a relaxation matching algorithm. In [17] a similar approach is presented. Yet, this work uses self-organising Hopfield neural networks in order to solve the graph matching problem.

The task of recognizing machine printed distorted characters in text lines is addressed in [18,19]. The characters to be recognized are modeled by graphs where the nodes correspond to branches, ending and turning points as well as sharp corners, and the edges represent strokes. Complete text lines are recognized by means of error-tolerant subgraph isomorphisms from character prototype graphs to text line graphs. The advantage of such a graph-based text line recognition procedure, compared to feature-based methods, is twofold. First, it makes optical character recognition independent of segmentation, and second, it is able to handle both touching and broken characters.

Online handwriting recognition using graph-based shape representation has been also proposed [20]. In this work the authors turn to the problem of recognizing handwritten script generated by writing on an electronic tablet with a stylus. Given the (x,y)-coordinates of the pen tip, so called *structural properties* (SPs) are computed. These SPs are interior, end, bump, cross, dot, and various other characteristic shape points of the trace of a handwritten text. That is, these SPs represent the smallest units which may be viewed as building blocks of a larger shape (e.g. a handwritten character). Handwritten characters can thus be represented as a graph where the nodes represent the SPs in a character which are connected by an edge if two SPs occur in sequential order along the time axis. For final character recognition any graph matching technique can be employed.

Next, we like to mention the fields of table recognition, layout analysis, and page classification where graphs have also been used in several works [21–26]. In [21] a bottom-up approach for identifying and recognizing tables within a document is proposed. This approach transforms the document image into a layout graph whose nodes and edges represent document entities and their relations. Using a set of rules based on a priori document knowledge, this graph is rewritten such that it provides both logical and layout views of the document content.

In [22] hierarchical graphs are employed to represent tables. In this representation, the nodes of the graphs represent tables, rows, columns, or cells (including their content) while edges are used to model relations between various entities in a table. The graphs are transformed into a vector of real numbers (so-called *graph probes*), such that the similarity between graphs can be approximated using some measure of similarity between the two corresponding vectors.

A general representation of tabular documents containing both structure and layout information is proposed in [23]. The structure analysis system introduced in this paper is based on a graph grammar which represents document structure knowledge. By using grammar notation, one can easily modify the format of the underlying table and simultaneously keep its consistency.

In [24] a graph matching approach to label logical components of a document page is proposed. The framework is able to learn a model for a document class, use this model to label document images, and adaptively improve the model with error feedback.

A method to recognize layout structures of various kinds of tabular document images is proposed in [25]. In order to manage the relationships among different classes of layout structures, a classification tree is employed. The proposed recognition system operates in two modes, viz. layout knowledge acquisition and layout structure recognition. In the former mode, table-form document images are distinguished according to the classification tree, while in the latter mode individual item fields in the table-form document images are extracted and classified.

Page classification using tree representations is addressed in [26]. By means of tree grammars the training set of known pages is automatically expanded. The rationale for this expansion is to model variations in the page tree which might occur due to different segmentations. The pages are eventually classified using a nearest-neighbor classifier in conjunction with tree edit distance [27].

Another field of research where graphs have been studied with emerging interest is that of content mining in web documents [28–30]. Often, mining of graph data refers to the process of extracting useful knowledge from the underlying data, which are represented as a large database graph. Typically, the extracted knowledge mined from the database graph is also a graph, which may be, for instance, a subgraph of the underlying database graph [28]. In [29,30] the authors turn to the classification of web documents using a graph-based model instead of the traditional feature-based representation. The classification task is carried out using a nearest neighbor classifier which is able to clearly outperform the traditional methods in terms of accuracy without any significant loss of computational efficiency.

3. Recent developments in graph-based pattern recognition

The major drawbacks of graphs, when compared to feature vectors, are the significantly increased complexity of many algorithms as well as the lack of mathematical structure in their domain. For a long period of time the traditional approach to pattern recognition was to give up the universality of graphs in favor of more efficient feature vector representations, even in problem domains where graphs would be naturally the method of choice. However, after decades of focusing on independent and identically distributed representation formalisms [1], more and more effort is now rendered on structured data. In fact, the intensive study of problems where the objects under consideration consist of interrelated entities has emerged rapidly in the last decade [3,28,31,32]. The reason for this emerging trend can be found in recent developments in graph-based pattern recognition such as *graph kernels* (reviewed in Section 3.1) and *graph embeddings* (reviewed in Section 3.2).

3.1. Graph kernels

During the past decade kernel methods have become one of the most rapidly emerging sub-fields in intelligent information processing. The reason for this development is twofold. First, kernel methods allow one to extend basic linear algorithms to complex non-linear ones in a unified and elegant manner. Hence, by means of kernel methods the issue of non-linear regularities in the data is inherently coped with. There is theoretical evidence that under some conditions kernel methods are more appropriate for difficult pattern recognition tasks than traditional methods [33]. Second, kernel theory makes standard algorithms (originally developed for vectorial representation) applicable to more complex data structures such as strings, trees, or graphs. That is, the concept of kernel machines can be extended from the domain of vectors to structural domains.

Definition 1 (*Graph kernel*). Let $\mathcal{G}$ be a (finite or infinite) set of graphs. Function $\kappa : \mathcal{G} \times \mathcal{G} \rightarrow \mathbb{R}$ is called a graph kernel if there exists a possibly infinite-dimensional Hilbert space $\mathcal{F}$ and a mapping $\phi : \mathcal{G} \rightarrow \mathcal{F}$ such that

$$\kappa(g,g') = \langle \phi(g), \phi(g') \rangle,$$

for all $g,g' \in \mathcal{G}$ where $\langle \cdot, \cdot \rangle$ denotes a dot product in $\mathcal{F}$.

According to this definition, every graph kernel κ can be thought of as a dot product $\langle \cdot, \cdot \rangle$ in some (implicitly existing) feature space $\mathcal{F}$. In other words, instead of mapping graphs from $\mathcal{G}$ to the

feature space $\mathcal{F}$ and computing their dot product there, one can simply evaluate the value of the kernel function in $\mathcal{G}$ [32].

Consider now an algorithm formulated entirely in terms of dot products. Such algorithms are commonly referred to as *kernel machines*.[1] Clearly, any kernel machine can be turned into an alternative algorithm by merely replacing the dot product $\langle \cdot, \cdot \rangle$ by a valid kernel $\kappa(\cdot,\cdot)$. This procedure is commonly referred to as the *kernel trick* [34,35]. It allows one to cope with non-linear problems in an elegant and efficient manner.

The kernel trick has a huge impact in practice and allows us to compute geometrical properties of graphs $g,g' \in \mathcal{G}$. From a higher level perspective, the kernel trick allows one to run algorithms (kernel machines) in implicitly existing feature spaces $\mathcal{F}$ without computing the mapping $\phi : \mathcal{G} \to \mathcal{F}$ and even without knowing $\mathcal{F}$. Consequently, kernels provide us with an implicit embedding of the symbolic data space – e.g. the graph domain – into an inner product space. That is, graph kernels offer an elegant way to overcome the problems arising from the lack of mathematical structure in the domain of graphs.

While it might be cumbersome to define mathematical operations in some graph domain $\mathcal{G}$, a kernel $\kappa : \mathcal{G} \times \mathcal{G} \to \mathbb{R}$ might be much easier to obtain.[2] Clearly, by means of graph kernels one can benefit from both the high representational power of graphs and the large repository of algorithmic tools available for feature vector representations. A number of graph kernels have been proposed in the literature [32,36,37]. In the present section three classes of graph kernels are briefly discussed.

3.1.1. Diffusion kernels

A first class of graph kernels is given by *diffusion kernels*. The kernels of this class are defined with respect to a base similarity measure which is used to construct a kernel matrix $\mathbf{K} = (\kappa_{ij})_{N \times N}$ [38–42]. This base similarity measure only needs to satisfy the condition of symmetry to guarantee that the resulting kernel matrix is positive definite. Obviously, the diffusion kernel can be defined for any kind of objects and particularly for graphs. Assume that a graph set $\{g_1, \ldots, g_N\} \subseteq \mathcal{G}$, a decay factor $0 < \lambda < 1$, and a similarity measure $s : \mathcal{G} \times \mathcal{G} \to \mathbb{R}$ are given. The $N \times N$ matrix $\mathbf{S} = (s_{ij})_{N \times N}$ of pairwise similarities s_{ij} can be turned into a positive definite kernel matrix $\mathbf{K} = (\kappa_{ij})_{N \times N}$ through the *exponential diffusion kernel* [38] defined by

$$\mathbf{K} = \sum_{k=0}^{\infty} \frac{1}{k!} \lambda^k \mathbf{S}^k = \exp(\lambda \mathbf{S})$$

or the *von Neumann diffusion kernel* [39] defined by

$$\mathbf{K} = \sum_{k=0}^{\infty} \lambda^k \mathbf{S}^k.$$

The decay factor λ assures that the weighting factor λ^k will be negligibly small for sufficiently large k. Therefore, only the first t addends in the diffusion kernel sums have to be evaluated in practice. Of course, any graph (dis)similarity measure can be used to build a diffusion kernel for graphs.

3.1.2. Convolution kernel

A seminal contribution to the field of graph kernel is the work on *convolution kernels*, which provides a general framework for dealing with complex objects [43,44]. Convolution kernels infer the similarity of composite objects from the similarity of their parts. The rationale behind this approach is that a similarity function might more easily be defined or more efficiently be computed for smaller parts rather than for the whole composite object. Given the similarities between the simpler parts of the underlying objects, a convolution operation is eventually applied in order to turn them into a kernel function.

Clearly, graphs are complex composite objects as they consist of nodes and edges. The concept of decomposing a graph g into its parts is mathematically denoted by a relation R, where $R(g_1,\ldots,g_d,g)$ represents the decomposition of g into parts $(g_1,\ldots,g_d)$. By $R^{-1}(g) = \{(g_1,\ldots,g_d) : R(g_1,\ldots,g_d,g)\}$ we denote the set of decompositions of graph $g \in \mathcal{G}$. For a simple example, assume that the set of all decompositions of a graph $g \in \mathcal{G}$ is defined by $R^{-1}(g) = V$, where V denotes the set of nodes of g. Hence, all of g's nodes are a valid decomposition of g. For the definition of the convolution kernel, a kernel function κ_i is required for each part of a decomposition $\{g_i\}_{1 \le i \le d}$. For instance, if $R^{-1}(g) = V$, a kernel function measuring the similarity of the involved nodes could be employed for κ_i. The convolution kernel function for graphs $g,g' \in \mathcal{G}$ can then be written as

$$\kappa(g,g') = \sum_{\substack{(g_1,\ldots,g_d) \in R^{-1}(g) \\ (g'_1,\ldots,g'_d) \in R^{-1}(g')}} \prod_{i=1}^{d} \kappa_i(g_i,g'_i).$$

Hence, this graph kernel derives the similarity between two graphs g and g' from the sum, over all decompositions, of the similarity product of the parts of g and g' [32]. The ANOVA kernel [45], for instance, is a particular convolution kernel, which uses a subset of the components of a composite object for comparison.

3.1.3. Walk kernel

A third class of graph kernels is based on the analysis of *random walks* in graphs. These kernels measure the similarity of two graphs by the number of random walks in both graphs that have all or some labels in common [46–50]. In [48] an important result is reported. It is shown that the number of matching walks in two graphs g and g' can be computed by means of the *direct product graph* $g \times g'$, without the need to explicitly enumerate the walks. This allows us to consider random walks of arbitrary length.

The direct product graph, by definition, identifies the compatible nodes and edges in the two graphs. Given a weighting parameter $\lambda \ge 0$, one can derive a kernel function for graphs $g,g' \in \mathcal{G}$ from the adjacency matrix $\mathbf{A}_\times$ of their product graph $(g \times g')$ by defining

$$\kappa(g,g') = \sum_{i,j=1}^{|V_\times|} \left[\sum_{n=0}^{\infty} \lambda^n \mathbf{A}_\times^n \right]_{ij}.$$

With a weighting factor $\lambda < 1$ it is assured that the contribution of $\lambda^n \mathbf{A}_\times^n$ to the overall sum will be negligibly small for sufficiently large n. Therefore, only the first t terms in the random walk kernel sums have to be evaluated.

In order to handle continuous labels, the random walk kernel has been extended in [46]. This extension allows one to also take non-identically labeled walks into account. In [47] two kernels, the so-called all-path and the shortest-path kernel, are introduced. These kernels consider paths between two pairs of nodes in a graph and use the similarity of two pairs of paths in order to derive the final kernel value. Another kernel that considers paths and label sequences encountered along the paths in two graphs is described in [51]. The problem of tottering is addressed in [52]. Tottering is the phenomenon that, in a walk, a node may be revisited immediately after it has been left. In order to prevent tottering, the random walk transition probability model is appropriately modified in [52].

3.2. Graph embedding

Another approach to overcoming the lack of algorithmic tools for graphs is graph embedding in vector spaces. The objective of graph embeddings is similar to that of graph kernels, i.e. we want to benefit

[1] Prominent examples of kernel machines are support vector machine, principal component analysis, and k-means clustering.

[2] Note that a kernel function is a measure of similarity satisfying the conditions of symmetry and positive definiteness.

from both the universality of graphs for pattern representation and the computational convenience of vectors for pattern recognition. In contrast to graph kernels, however, graph embedding procedures result in an explicit embedding of graphs from some graph domain $\mathcal{G}$ in a real vector space $\mathbb{R}^n$. Formally, a graph embedding is a function $\varphi : \mathcal{G} \to \mathbb{R}^n$ mapping graphs from arbitrary graph domains to a vector space. Based on the resulting graph maps, the considered pattern recognition or document analysis task is eventually carried out. Hence, the whole arsenal of algorithmic tools readily available for vectorial data can be applied to graphs (more exactly to graph maps $\varphi(g) \in \mathbb{R}^n$).

A prominent class of graph embedding is based on spectral methods [53–58]. Spectral graph theory is concerned with understanding how the structural properties of graphs can be characterized using eigenvectors of the adjacency or Laplacian matrix [56]. Although graph spectralization exhibits interesting properties which can be used for vector space embedding of graphs, this approach remains somewhat limited. For instance, spectral methods are not fully able to cope with larger amounts of noise. This stems from the fact that the eigendecomposition is very sensitive towards structural errors, such as missing or spurious nodes. Moreover, spectral methods are applicable to unlabeled graphs or labeled graphs with severely constrained label alphabets only, making this approach only to a limited extent applicable to graphs extracted from real-world data.

3.2.1. Graph embedding procedure using dissimilarities

Recently, a new class of graph embedding procedures has been proposed which can be applied to both directed and undirected graphs, as well as to graphs with arbitrary labels on their nodes and/or edges [59]. Furthermore, this graph embedding framework is distinguished by its ability to handle structural errors.

The idea of this approach to graph embedding stems from the seminal work done by Duin and Pekalska [60] where dissimilarities for pattern representation were proposed for the first time. Later this method was extended so as to map string representations into vector spaces [61]. Recently, the methods described in [60,61] were generalized and substantially extended to the domain of graphs [59]. The key idea of this approach is to use the distances of an input graph g to a number of training graphs, termed *prototype graphs*, as a vectorial description of g.

Assume we have a set of sample graphs, $\mathcal{T} = \{g_1, \ldots, g_N\}$ from some graph domain $\mathcal{G}$ and an arbitrary graph dissimilarity measure $d : \mathcal{G} \times \mathcal{G} \to \mathbb{R}$. Note that $\mathcal{T}$ can be any kind of graph set. However, for the sake of convenience we assume in the following that $\mathcal{T}$ is a given training set of graphs. After selecting a set of prototypical graphs $\mathcal{P} \subseteq \mathcal{T}$, we compute the dissimilarity of a given input graph g to each prototype graph $p_i \in \mathcal{P}$. Note that g can be an element of $\mathcal{T}$ or any other graph set $\mathcal{S}$. Given n prototypes, i.e. $\mathcal{P} = \{p_1, \ldots, p_n\}$, this procedure leads to n dissimilarities, $d_1 = d(g,p_1), \ldots, d_n = d(g,p_n)$, which can be arranged in an n-dimensional vector $(d_1, \ldots, d_n)$.

Definition 2 (*Graph embedding*). Let us assume a graph domain $\mathcal{G}$ is given. If $\mathcal{T} = \{g_1, \ldots, g_N\} \subseteq \mathcal{G}$ is a training set with N graphs and $\mathcal{P} = \{p_1, \ldots, p_n\} \subseteq \mathcal{T}$ is a prototype set with n graphs, the mapping

$$\varphi_n^{\mathcal{P}} : \mathcal{G} \to \mathbb{R}^n$$

is defined as the function

$$\varphi_n^{\mathcal{P}}(g) = (d(g,p_1), \ldots, d(g,p_n)),$$

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

Obviously, by means of this definition we obtain a vector space where each axis corresponds to a prototype graph $p_i \in \mathcal{P}$ and the coordinate values of an embedded graph g are the distances of g to the elements in $\mathcal{P}$. In this way we can transform any graph g from the training set $\mathcal{T}$ as well as any other graph set $\mathcal{S}$ (for instance a validation or a test set of a classification problem), into a vector of real numbers.

After two graphs, g and g', have been mapped to the vector space, we observe the following relationship[3]:

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

$$\leq (n \cdot d(g,g')^2)^{1/2} = \sqrt{n} \cdot d(g,g'). \qquad (2)$$

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 (Eq. (1)). 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, the upper bound of the Euclidean distance of a pair of graph maps $\varphi(g)$ and $\varphi(g')$ is given by $\sqrt{n} \cdot d(g,g')$ (Eq. (2)). Obviously, defining the embedding procedure given in Definition 2 as

$$\varphi_n^{\mathcal{P}}(g) = (d(g,p_1)/q, \ldots, d(g,p_n)/q),$$

with $q = \sqrt{n}$, the upper bound on the Euclidean distance of a pair of graph maps $\varphi(g)$ and $\varphi(g')$ amounts to $d(g,g')$. By means of the normalization by q, the influence of the number of prototypes is reduced. Hence, regardless of the number of prototypes, it is guaranteed that distances in the resulting embedding space are bounded by the original graph distance $d(g,g')$.

3.2.2. Graph edit distance

The embedding procedure proposed in [59] makes use of graph edit distance.[4] 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.

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. 2 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 $\Upsilon(g_1,g_2)$, 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.

[3] For the purpose of simplicity we write $\varphi(g)$ for $\varphi_n^{\mathcal{P}}(g)$. Furthermore, $\langle \cdot, \cdot \rangle : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ denotes the scalar product of two vectors.

[4] Note, however, that any other graph dissimilarity measure can be used as well.

Fig. 2. A possible edit path between graph g_1 and g_2 (node labels are represented by different shades of grey).

Definition 3 (*Graph edit distance*). Let g_1 be the source graph and g_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.

Using graph edit distance in the proposed embedding framework 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, due to the flexibility of graph edit distance, a high degree of robustness against various graph distortions can be expected. Hence, in contrast with other graph embedding techniques (e.g. spectral methods) this embedding approach is characterized by a high degree of flexibility in the kind of underlying graphs and a high tolerance to structural errors in the graph matching process.

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 [62]. A major drawback of those procedures is their computational complexity, which is exponential in the number of nodes of the involved graphs. Consequently, the complexity of the proposed graph embedding is exponential as well. However, to render graph edit distance computation less computationally demanding, there exist efficient approximation algorithms with polynomial runtime complexity [63–65]. Hence, given n predefined prototypes the embedding of one particular graph is established by means of n distance computations with polynomial time.

3.2.3. Relation to kernel methods

Dissimilarity embedding is closely related to kernel methods. In the kernel approach the patterns are described by means of pairwise kernel functions, and in the dissimilarity approach they are represented by pairwise dissimilarities. However, there is also one fundamental difference between kernels and dissimilarity embeddings. In the former method, the kernel values are interpreted as dot products in some implicit feature space. By means of kernel machines, the pattern recognition task is eventually carried out in this kernel feature space. In the latter approach, the set of dissimilarities is interpreted as a vectorial description of the pattern under consideration. Hence, no implicit feature space, but an explicit dissimilarity space is obtained.

Although conceptually different, the embedding paradigm established by the mapping $\varphi_n^{\mathcal{P}} : \mathcal{G} \to \mathbb{R}^n$ constitutes a foundation for a novel class of graph kernels. One can define a valid graph kernel κ based on the graph embedding $\varphi_n^{\mathcal{P}} : \mathcal{G} \to \mathbb{R}^n$ by computing the standard dot product of two graph maps in the resulting vector space[5]

$$\kappa_{\langle\rangle}(g,g') = \langle \varphi_n^{\mathcal{P}}(g), \varphi_n^{\mathcal{P}}(g') \rangle.$$

Of course, not only the standard dot product can be used but any valid kernel function defined for vectors. For instance an RBF kernel function

$$\kappa_{RBF}(g,g') = \exp(-\gamma \|\varphi_n^{\mathcal{P}}(g) - \varphi_n^{\mathcal{P}}(g')\|^2)$$

with $\gamma > 0$ can thus be applied to graph maps. We refer to this procedure as *graph embedding kernel* using some specific vector kernel function.

In a recent book, graph kernels were proposed that directly use graph edit distances [32]. This approach turns an existing dissimilarity measure (e.g. graph edit distance) into a similarity measure by mapping low distance values to high similarity values and vice versa. To this end monotonically decreasing transformations are used. Given the edit distance $d(g,g')$ of two graphs g and g', the similarity kernel is defined, for instance, as $\kappa(g,g') = \exp(-\gamma d(g,g'))$, with $\gamma > 0$. Note that this kernel function is not positive definite in general. However, there is theoretical evidence that using kernel machines in conjunction with indefinite kernels may be reasonable if some conditions are fulfilled [67]. We refer to this method as *similarity kernel*, or *sim* for short, in the following.

3.2.4. The problem of prototype selection

The selection of the n prototypes $\mathcal{P} = \{p_1, \ldots, p_n\}$ is a critical issue in graph embedding since not only the prototypes $p_i \in \mathcal{P}$ themselves but also their number n affect the resulting graph mapping $\varphi_n^{\mathcal{P}}(\cdot)$ and thus the performance of the corresponding pattern recognition algorithm. A good selection of n prototypes seems to be crucial to succeed with the algorithm in the embedding vector space. A first and very simple solution might be to use all available training graphs from $\mathcal{T}$ as prototypes. Yet, two severe shortcomings arise with such a plain approach. First, the dimensionality of the resulting vector space is equal to the size N of the training set $\mathcal{T}$. Consequently, if the training set is large, the mapping results in (possibly too) high dimensional feature vectors disabling efficient computations. Second, the presence of similar prototypes as well as outlier graphs in the training set $\mathcal{T}$ is likely. Therefore, redundant and noisy or irrelevant information will be captured in the graph maps which in turn may harm the performance of the algorithms applied subsequently.

The selection of prototypes for graph embedding has been addressed in various papers [59,68,69]. In [59], for instance, a number of *prototype selection methods* are discussed. These selection strategies use some heuristics based on the underlying dissimilarities in the original graph domain. The basic idea of these approaches is to select prototypes from $\mathcal{T}$ that best reflect the distribution of the training set $\mathcal{T}$ or cover a predefined region of $\mathcal{T}$. The rationale of this procedure is that capturing distances to significant prototypes from $\mathcal{T}$ leads to meaningful dissimilarity vectors. All of the proposed prototype selection methods can be applied classwise or classindependent. Classindependent selection means that the selection is executed over the whole training set $\mathcal{T}$ to get n prototypes, while in classwise selection the selection is performed individually for each of the classes occurring in $\mathcal{T}$.

The Border prototype selector (bps), for instance, selects prototypes situated at the border of the training set $\mathcal{T}$ [59]. In Fig. 3(a) and (b) the classindependent and the classwise bps is illustrated, respectively.[6]

Another prototype selection strategy is spanning selection (sps). This iterative selection algorithm considers all distances to the

[5] Note that is approach is very similar to the empirical kernel map described in [66] where general similarity measures are turned into kernel functions.

[6] The data underlying these illustrative examples are two-dimensional vectors obtained through multidimensional scaling applied to the original graph edit distances from the Letter data set (this data set is used in the experimental evaluation and is described in detail in Section 4).

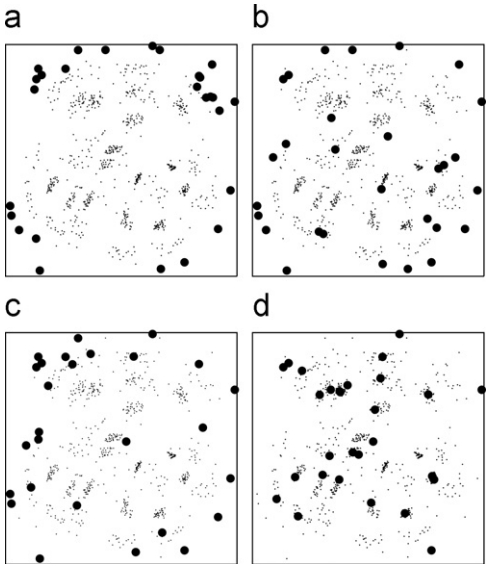

Fig. 3. Illustration of the different prototype selectors applied to the training set $\mathcal{T}$ of the Letter data set. The number of prototypes is defined by $n=30$. The prototypes selected by the respective algorithms are represented by heavy dots. (a) bps, (b) bps-c, (c) sps, (d) k-cps.

prototypes selected before. The first prototype is the set median graph $median(\mathcal{T})$ [70]. Each additional prototype is the graph furthest away from the already selected prototype graphs.

The k-centers prototype selector (k-cps) is a third strategy for prototype selection which is based on k-medians clustering [71]. The prototypes are given by the set medians of the n disjoint clusters provided through k-medians clustering of the training set $\mathcal{T}$. In contrast to sps, we note that k-cps avoids selecting prototypes from the border by focusing on graphs that are in the center of densely populated areas (cf. Fig. 3(c) and (d) for a comparison between sps and k-cps).

Another solution to the problem of noisy and redundant embedding vectors with too high dimensionality is offered by the following procedure. Rather than selecting the prototypes beforehand, the embedding is carried out first and then the problem of prototype selection is reduced to a feature subset selection problem. That is, we define $\mathcal{P}=\mathcal{T}$ and use all available elements from the training set for graph embedding. Next, a large number of different feature selection strategies can be applied to the resulting large scale vectors eliminating redundancies and noise, finding good features, and reducing the dimensionality. In [68], for instance, principal component analysis (PCA) and Fisher linear discriminant analysis (LDA) are applied to the vector space embedded graphs. Kernel PCA [72], rather than traditional PCA, is used for feature transformation in [69].

4. Experimental evaluation

In this section we provide the results of an experimental evaluation of the dissimilarity-based graph embedding procedure applied to various data sets from the field of document analysis in the context of classification tasks. 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 traditional techniques.

For graph edit distance computation the suboptimal algorithm introduced in [63] has been used. This graph edit distance algorithm shows superior performance in time and accuracy compared to other suboptimal algorithms. For prototype selection we focus on the three prototype selection methods discussed above (bps, sps, and k-cps—all applied classwise).

The classifier used in the embedding space is the support vector machine (SVM) [33]. 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.

Table 2
Summary of graph data set characteristics, viz. the size of the training (tr), the validation (va) and the test set (te), the number of classes (#classes), the average and maximum number of nodes and edges ($\varnothing$/max$|V|$/$|E|$), and whether the graphs are uniformly distributed over the classes or not (balanced).

Data set	Size (tr, va, te)	#classes	$\varnothing\|V\|$	$\varnothing\|E\|$	max$\|V\|$	max$\|E\|$	Balanced
Letter low	750, 750, 750	15	4.7	3.1	8	6	Y
Letter medium	750, 750, 750	15	4.7	3.2	9	7	Y
Letter high	750, 750, 750	15	4.7	4.5	9	9	Y
Digit	1000, 500, 2000	10	8.9	7.9	17	16	Y
GREC	836, 836, 1,628	22	11.5	12.2	25	30	Y
Webpage	780, 780, 780	20	186.1	104.6	834	596	N

4.1. Graph data sets

For evaluation, six data sets from the IAM graph database repository[7] for graph-based pattern recognition and machine learning are used [73]. In Table 2 a summary of the six graph data set characteristics is given.

The first three graph data sets involves graphs that represent distorted letter drawings (Letter). We consider the 15 capital letters of the Roman alphabet that consist of straight lines only ($A, E, F, \ldots, Z$). For each class, a prototype line drawing is manually constructed. These prototype drawings are then converted into prototype graphs by representing lines by undirected edges and ending points of lines by nodes. Each node is labeled with a two-dimensional attribute giving its position relative to a reference coordinate system. Edges are unlabeled. In order to test classifiers under different conditions, distortions are applied on the prototype graphs with three different levels of strength. This results in three different versions of this database, including graphs with a *low*, *medium* and *high* level of distortion. These three graph data set consist of a training set, a validation set, and a test set of size 750 each. The graphs are uniformly distributed over the 15 classes.

The fourth data set is quite similar to the letter data set and involves graphs that represent handwritten digits (Digit). Rather than using artificial distortions to simulate the variety of different writers, the underlying data represents real world on-line data rendered by humans. In this data set nodes represent line segments of a handwritten digit. More formally, the sequences of (x,y) coordinates are converted into graphs by grouping coordinate points together forming subpaths of similar length. These subpaths are represented by nodes labeled with their starting and ending position relative to a reference coordinate system (i.e. the first and last (x,y) coordinates from the respective subpath). Successive subpaths are connected by undirected and unlabeled edges. Finally, the derived graphs are normalized such that each corresponding digit has equal width and height. For our database a set of totally 3500 digits from the database described in [74] is used. This set is split into a training set of size 1000, a validation set of size 500, and a test set of size 2000. The digit graphs are uniformly distributed over the 10 classes.

[7] www.iam.unibe.ch/fki/databases/iam-graph-database

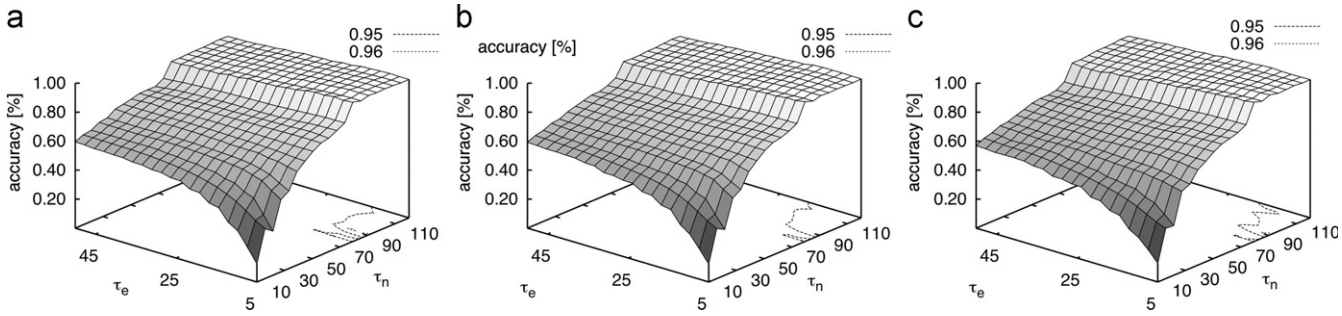

Fig. 4. Validation of the node and edge insertion/deletion cost τ_n and τ_e, and the number of nearest neighbors k on the GREC data set. (a) $k=1$, (b) $k=3$, (c) $k=5$.

The fifth graph data set consists of graphs representing symbols from architectural and electronic drawings (GREC) [75]. The images occur at five different distortion levels. Depending on the distortion level, either erosion, dilation, or other morphological operations are applied. The result is thinned to obtain lines of one pixel width. Finally, graphs are extracted from the resulting denoised images by tracing the lines from end to end and detecting intersections as well as corners. Ending points, corners, intersections and circles are represented by nodes and labeled with a two-dimensional attribute giving their position. The nodes are connected by undirected edges which are labeled as *line* or *arc*. An additional attribute specifies the angle with respect to the horizontal direction or the diameter in case of arcs. From the original GREC database [75], 22 classes are considered. For an adequately sized set, the five graphs per distortion level are individually distorted 30 times to obtain a data set containing 3300 graphs uniformly distributed over the 22 classes. The resulting set is split into a training and a validation set of size 836 each, and a test set of size 1628.

In [29] several methods for creating graphs from webpage documents are introduced. For the graphs included in the sixth data set (Webpage), the following method was applied. First, all words occurring in the webpage document under consideration – except for stop words, which contain only little information – are converted into nodes in the resulting web graph. We attribute each node with the corresponding word and its frequency. Next, different sections of the webpage document are investigated individually. If a word w_i immediately precedes word w_{i+1}, 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 webpage graph. The resulting edge is attributed with the corresponding section label. In our experiments we make use of a data set with documents from 20 categories (*Business*, *Health*, *Politics*, …). The number of documents of each category varies from only 24 (Art) up to about 500 (Health). These web documents were originally hosted at Yahoo as news pages (http://www.yahoo.com). The data set is split into a training, a validation, and a test set of equal size (780).

4.2. Reference systems and experimental setup

As mentioned earlier, in contrast with the high representational power of graphs, there is a lack of general classification algorithms that can be applied to graphs. One of the few classifiers directly applicable to arbitrary graphs is the k-nearest-neighbor classifier (k-NN). 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. This classifier in the graph domain will serve us as our first reference system.

The second reference system is the similarity kernel described in Section 3.2.3 (sim) in conjunction with an SVM. Comparing the performance of this similarity kernel function with the performance of the dissimilarity 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.

In the validation phase, the costs of the edit operations are determined first. For all considered graph data sets but the Webpage data, 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 symmetry, we assume in all experiments identical costs for node deletions and insertions, and for edge deletions and insertions. Hence only two parameters, node deletion/insertion cost τ_n, and edge deletion/insertion cost τ_e, need to be validated. The validation procedure is illustrated in Fig. 4, where the classification results of a k-NN classifier on the validation set as a function of the two edit cost parameters are shown for the GREC data set ($k=1,3,5$). For the webpage data set we followed the cost model used in [29], 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 data set.

For the second reference system (sim) the same edit distances as for the k-NN classifier are used. The weighting parameter C of the SVM, which controls whether the maximization of the margin or the minimization of the error is more important, and the meta parameter γ in the kernel function $\kappa(g,g')=\exp(-\gamma d(g,g'))$ have to be additionally validated for this classifier.

For the task of graph embedding in real vector spaces one additional meta parameter has to be validated, namely the number of prototypes n, i.e. the dimensionality of the resulting vector space. In order to determine suitable values of n, each graph set is embedded in a vector space with all of the prototype selectors described in Section 3.2.4, varying the dimensionality of the target vector space over a certain interval. With the resulting vector sets, which still consist of a training, a validation, 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.[8] In Fig. 5(a) such an SVM parameter validation on the Webpage data set 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 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 Webpage data set in Fig. 5(b) where the accuracies for three classwise prototype selectors (bps-c, sps-c, k-cps-c) and each dimensionality are shown.

By means of this procedure for each prototype selector the number of prototypes n, and the SVM parameters (C,γ) can be determined for every data set on the validation set. The parameter

[8] 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 [34].

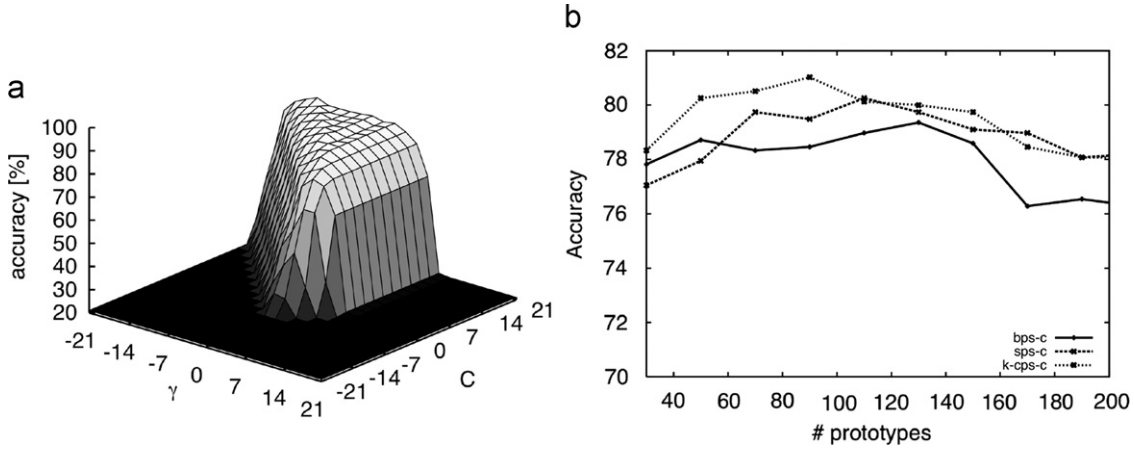

Fig. 5. (a) Validation of the meta parameter tuple (C,γ) for a specific prototype selector and a certain number of prototypes (the parameter values are on a logarithmic scale to the basis 2). (b) Validation of the number of prototypes.

Table 3
Embedding methods vs. reference systems. ①/② Stat. significant improvement over the first/second reference system (k-NN/sim). ❷ Stat. significant deterioration compared to the second reference system (sim).

Data set	Ref. systems		Embedding methods		
	k-NN	sim	sps-c	bps-c	k-cps-c
Letter low	99.3	99.6	99.3	99.3	99.3
Letter medium	94.4	94.9	94.9	94.8	94.7
Letter high	89.1	92.9	92.3 ①	92.9 ①	92.0 ①
Digit	97.4	98.1	98.6 ①②	98.7 ①②	98.7 ①②
GREC	82.2	71.6	92.4 ①②	92.3 ①②	92.4 ①②
Webpage	80.6	82.9	82.7 ①	80.4 ❷	82.3

combination that results in the lowest classification error is finally applied to the independent test set.

4.3. Results and discussion

The results obtained on the test set by means of this procedure are reported for the three prototype selectors in Table 3. Let us first compare the classification accuracies achieved by our embedding framework with the first reference system (k-NN). It clearly turns out that the novel procedure is much more powerful than the traditional k-NN classifier in the graph domain as on all tested applications but one (Letter low) the embedding framework outperforms the first reference system's classification results. Note that on four data sets (Letter high, Digit, GREC, and Webpage) these improvements are statistically significant (using sps-c for prototype selection).

Next we compare the embedding procedure with the similarity kernel (sim). It turns out that the similarity kernel is better on the Letter and Webpage data sets compared to at least one prototype selection method. However, only one of these deteriorations is statistically significant. Moreover, the embedding framework outperforms the similarity kernel on the Digit and GREC data sets. Note that all of these improvements are 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 the strength of the kernel classifier.

For a more detailed analysis of the proposed approach including more thorough descriptions of the achieved results, other tasks than classification (e.g. clustering) as well as experimental evaluations on other data sets (not from document analysis) we refer to [59,68,69,76].

5. Conclusions

Classification is a common task in document analysis and related fields. Examples of classification problems in document analysis can be found in graphics recognition, optical character recognition, page classification, and many other domains. A vast number of algorithms for classification have been designed for object representations given in terms of feature vectors.

After relying for several decades mostly on object representation in terms of feature vectors, recently, a growing interest in graph-based object representations has emerged. 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. Due to their power and flexibility graphs have found widespread applications in graphics recognition, data and web content mining, handwriting and machine printed character recognition, table recognition, layout analysis, page classification, and other subfields of document analysis.

One of the major drawbacks of graph-based representations is, however, that there is only little mathematical structure in the graph domain. In contrast to vectors, most of the basic mathematical operations required for many standard pattern recognition algorithms, including classification and clustering, do not exist for graphs. Consequently, we observe a severe lack of algorithmic tools in the domain of graphs. Traditionally, tasks such as classification and clustering of graphs are solved by defining an appropriate distance measure and then using an algorithm that is based on distances exclusively (e.g. a nearest-neighbor classifier).

In the present paper some recent approaches to implicit and explicit graph embedding are reviewed. Particularly, this article describes a novel approach to graph embedding using dissimilarities. This embedding procedure explicitly makes use of graph edit distance and can therefore deal with various kind of graphs (labeled, unlabeled, directed, undirected, etc.). The basic idea of the embedding method is to describe a graph by means of n dissimilarities to a predefined set of graphs termed prototypes. That is, a graph g is mapped explicitly to the n-dimensional real space $\mathbb{R}^n$ by arranging the edit distances of g to all of the n prototypes as a vector. By means of this procedure both statistical classifiers and vector kernels can be applied to the resulting graph maps.

The task of prototype selection is an issue in the proposed graph embedding framework. Several possible solutions have been

proposed in the literature. Three possible algorithms from the category of heuristic prototype selection are described in detail in the present paper. Note that none of them performs generally the best. That is, the quality of a particular selection strategy depends on the underlying data set.

In the experimental evaluation, involving several data sets with diverse properties from different subfields of document analysis, the proposed graph embedding framework is compared to a nearest-neighbor classifier and another kernel-based classifier. We empirically confirm a high degree of robustness and flexibility of the proposed approach in this paper and related publications. According to these results, one should consider embedding or kernelization as a potentially useful alternative to traditional approaches to graph-based document analysis.

References

[1] R. Duda, P. Hart, D. Stork, Pattern Classification, second ed., Wiley-Interscience, 2000.
[2] H. Bunke, A. Sanfeliu (Eds.), Syntactic and Structural Pattern Recognition, World Scientific, 1990.
[3] D. Conte, P. Foggia, C. Sansone, M. Vento, Thirty years of graph matching in pattern recognition, International Journal of Pattern Recognition and Artificial Intelligence 18 (3) (2004) 265–298.
[4] G. Nagy, Twenty years of document image analysis in PAMI, IEEE Transactions on Pattern Analysis and Machine Intelligence 22 (1) (2000) 38–62.
[5] M. Cheriet, N. Kharma, C. Liu, C. Suen, Character Recognition Systems: A Guide for Students and Practitioners, Wiley-Interscience, 2007.
[6] B.B. Chaudhuri (Ed.), Digital Document Processing, Springer, 2007.
[7] S. Marinai, H. Fujisawa (Eds.), Machine Learning in Document Analysis and Recognition, Springer, 2008.
[8] S. Lee, J. Kim, A. Groen, Translation-, rotation-, and scale-invariant recognition of hand-drawn symbols in schematic diagrams, International Journal of Pattern Recognition and Artificial Intelligence 4 (1) (1990) 1–25.
[9] J. Lladós, E. Martí, J. Villanueva, Symbol recognition by error-tolerant subgraph matching between region adjacency graphs, IEEE Transactions on Pattern Analysis and Machine Intelligence 23 (10) (2001) 1137–1143.
[10] B. Messmer, H. Bunke, Clustering and error-corrrecting matching of graphs for learning and recognition of symbols in engineering drawings, in: J. Hull, S. Taylor (Eds.), Document Analysis Systems II, World Scientific, 1998, pp. 102–117.
[11] L. Cordella, P. Foggia, C. Sansone, M. Vento, Fast graph matching for detecting CAD image components, Proceedings of the 15th International Conference on Pattern Recognition, vol. 2, 2000, pp. 1038–1041.
[12] J. Lladós, G. Sánchez, Graph matching versus graph parsing in graphics recognition, International Journal of Pattern Recognition and Artificial Intelligence 18 (3) (2004) 455–475.
[13] H. Bunke, On the generative power of sequential and parallel programmed graph grammars, Computing 29 (1982) 89–112.
[14] H. Bunke, Attributed programmed graph grammars and their application to schematic diagram interpretation, IEEE Transactions on Pattern Analysis and Machine Intelligence 4 (6) (1982) 582–674.
[15] S. Lu, Y. Ren, C. Suen, Hierarchical attributed graph representation and recognition of handwritten Chinese characters, Pattern Recognition 24 (7) (1991) 617–632.
[16] X. Huang, J. Gu, Y. Wu, A constrained approach to multifont Chinese character recognition, IEEE Transactions on Pattern Analysis and Machine Intelligence 15 (8) (1993) 838–843.
[17] P. Suganthan, H. Yan, Recognition of handprinted Chinese characters by constrained graph matching, Image and Vision Computing 16 (3) (1998) 191–201.
[18] J. Rocha, T. Pavlidis, A shape analysis model with applications to a character recognition system, IEEE Transactions on Pattern Analysis and Machine Intelligence 16 (4) (1994) 393–404.
[19] J. Rocha, T. Pavlidis, Character recognition without segmentation, IEEE Transactions on Pattern Analysis and Machine Intelligence 17 (9) (1995) 903–909.
[20] V. Chakravarthy, B. Kompella, The shape of handwritten characters, Pattern Recognition Letters 24 (12) (2003) 1901–1913.
[21] M. Rahgozar, Document table recognition by graph rewriting, in: M. Nagl, A. Schürr, M. Münch (Eds.), Applications of Graph Transformations with Industrial Relevance, Lecture Notes in Computer Science, vol. 1779, 2000, pp. 185–197.
[22] D. Lopresti, G. Wilfong, A fast technique for comparing graph representations with applications to performance evaluation, International Journal on Document Analysis and Recognition 6 (4) (2003) 219–229.
[23] A. Amano, A. Naoki, Graph grammar based analysis system of complex table form document, Proceedings of the Seventh International Conference on Document Analysis and Recognition, vol. 2, 2003, pp. 916–921.
[24] J. Liang, D. Doermann, Logical labeling of document images using layout graph matching with adaptive learning, in: D. Lopresti, J. Hu, R. Kashi (Eds.), Proceedings of the Fifth International Workshop on Document Analysis Systems, Lecture Notes in Computer Science, vol. 2423, Springer, 2002, pp. 224–235.
[25] Q.L.T. Watanabe, N. Sugie, Layout recognition of multi-kinds of table-form documents, IEEE Transactions on Pattern Analysis and Machine Intelligence 17 (4) (1995) 432–445.
[26] S. Baldi, S. Marinai, G. Soda, Using tree-grammars for training set expansion in page classification, in: Proceedings of the Seventh International Conference on Document Analysis and Recognition, 2003, pp. 829–833.
[27] S. Selkow, The tree-to-tree editing problem, Information Processing Letters 6 (6) (1977) 184–186.
[28] D. Cook, L. Holder (Eds.), Mining Graph Data, 2007, Wiley-Interscience.
[29] A. Schenker, H. Bunke, M. Last, A. Kandel, Graph-Theoretic Techniques for Web Content Mining, World Scientific, 2005.
[30] A. Schenker, M. Last, H. Bunke, A. Kandel, Classification of web documents using graph matching, International Journal of Pattern Recognition and Artificial Intelligence 18 (3) (2004) 475–496.
[31] H. Bunke, P. Dickinson, M. Kraetzl, W. Wallis, A Graph-Theoretic Approach to Enterprise Network Dynamics, Progress in Computer Science and Applied Logic (PCS), vol. 24, Birkhäuser, 2007.
[32] M. Neuhaus, H. Bunke, Bridging the Gap Between Graph Edit Distance and Kernel Machines, World Scientific, 2007.
[33] V. Vapnik, Statistical Learning Theory, John Wiley, 1998.
[34] J. Shawe-Taylor, N. Cristianini, Kernel Methods for Pattern Analysis, Cambridge University Press, 2004.
[35] B. Schölkopf, A. Smola, Learning with Kernels, MIT Press, 2002.
[36] T. Gärtner, Kernels for Structured Data, World Scientific, 2008.
[37] T. Gärtner, A survey of kernels for structured data, SIGKDD Explorations 5 (1) (2003) 49–58.
[38] R. Kondor, J. Lafferty, Diffusion kernels on graphs and other discrete input spaces, in: Proceedings of the 19th International Conference on Machine Learning, 2002, pp. 315–322.
[39] J. Kandola, J. Shawe-Taylor, N. Cristianini, Learning semantic similarity, Neural Information Processing Systems 15 (2003) 657–664.
[40] A. Smola, R. Kondor, Kernels and regularization on graphs, in: Proceedings of the 16th International Conference on Computational Learning Theory, 2003, pp. 144–158.
[41] J. Lafferty, G. Lebanon, Information diffusion kernels, Advances in Neural Information Processing Systems, vol. 15, MIT Press, 2003, pp. 375–382.
[42] J. Lafferty, G. Lebanon, Diffusion kernels on statistical manifolds, Journal of Machine Learning Research 6 (2005) 129–163.
[43] D. Haussler, Convolution kernels on discrete structures, Technical Report UCSC-CRL-99-10, University of California, Santa Cruz, 1999.
[44] C. Watkins, Dynamic alignment kernels, in: A. Smola, P. Bartlett, B. Schölkopf, D. Schuurmans (Eds.), Advances in Large Margin Classifiers, MIT Press, 2000, pp. 39–50.
[45] C. Watkins, Kernels from matching operations, Technical Report CSD-TR-98-07, Royal Holloway College, 1999.
[46] 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.
[47] K. Borgwardt, H.-P. Kriegel, Shortest-path kernels on graphs, in: Proceedings of the Fifth International Conference on Data Mining, 2005, pp. 74–81.
[48] T. Gärtner, P. Flach, S. Wrobel, On graph kernels: hardness results and efficient alternatives, in: B. Schölkopf, M. Warmuth (Eds.), Proceedings of the 16th Annual Conference on Learning Theory, 2003, pp. 129–143.
[49] H. Kashima, A. Inokuchi, Kernels for graph classification, in: Proceedings of the ICDM Workshop on Active Mining, 2002, pp. 31–36.
[50] H. Kashima, K. Tsuda, A. Inokuchi, Marginalized kernels between labeled graphs, in: Proceedings of the 20th International Conference on Machine Learning, 2003, pp. 321–328.
[51] L. Ralaivola, S. Swamidass, H. Saigo, P. Baldi, Graph kernels for chemical informatics, Neural Networks 18 (8) (2005) 1093–1110.
[52] P. Mahé, N. Ueda, T. Akutsu, Graph kernels for molecular structures—activity relationship analysis with support vector machines, Journal of Chemical Information and Modeling 45 (4) (2005) 939–951.
[53] F. Chung-Graham, Spectral Graph Theory, AMS, 1997.
[54] B. Luo, R. Wilson, E. Hancock, Spectral embedding of graphs, Pattern Recognition 36 (10) (2003) 2213–2223.
[55] T. Caelli, S. Kosinov, Inexact graph matching using eigen-subspace projection clustering, International Journal of Pattern Recognition and Artificial Intelligence 18 (3) (2004) 329–355.
[56] R. Wilson, E. Hancock, B. Luo, Pattern vectors from algebraic graph theory, IEEE Transactions on Pattern Analysis and Machine Intelligence 27 (7) (2005) 1112–1124.
[57] A. Robles-Kelly, E. Hancock, A Riemannian approach to graph embedding, Pattern Recognition 40 (2007) 1024–1056.
[58] B. Luo, E. Hancock, Structural graph matching using the EM algorithm and singular value decomposition, IEEE Transactions on Pattern Analysis and Machine Intelligence 23 (10) (2001) 1120–1136.
[59] K. Riesen, H. Bunke, Graph classification based on vector space embedding, International Journal of Pattern Recognition and Artificial Intelligence 23 (6) (2009) 1053–1081.
[60] E. Pekalska, R. Duin, The Dissimilarity Representation for Pattern Recognition: Foundations and Applications, World Scientific, 2005.

[61] B. Spillmann, M. Neuhaus, H. Bunke, E. Pekalska, R. Duin, Transforming strings to vector spaces using prototype selection, in: D.-Y. Yeung, J. Kwok, A. Fred, F. Roli, D. de Ridder (Eds.), Proceedings of the 11th International Workshop on Structural and Syntactic Pattern Recognition, Lecture Notes in Computer Science, vol. 4109, Springer, 2006, pp. 287–296.
[62] H. Bunke, G. Allermann, Inexact graph matching for structural pattern recognition, Pattern Recognition Letters 1 (1983) 245–253.
[63] K. Riesen, H. Bunke, Approximate graph edit distance computation by means of bipartite graph matching, Image and Vision Computing 27 (4) (2009) 950–959.
[64] S. Sorlin, C. Solnon, Reactive tabu search for measuring graph similarity, in: L. Brun, M. Vento (Eds.), Proceedings of the Fifth International Workshop on Graph-based Representations in Pattern Recognition, Lecture Notes in Computer Science, vol. 3434, Springer, 2005, pp. 172–182.
[65] M. Neuhaus, K. Riesen, H. Bunke, Fast suboptimal algorithms for the computation of graph edit distance, in: D.-Y. Yeung, J. Kwok, A. Fred, F. Roli, D. de Ridder (Eds.), Proceedings of the 11th International Workshop on Structural and Syntactic Pattern Recognition, Lecture Notes in Computer Science, vol. 4109, Springer, 2006, pp. 163–172.
[66] K. Tsuda, Support vector classification with asymmetric kernel function, in: M. Verleysen (Ed.), Proceedings of the Seventh European Symposium on Artificial Neural Networks, 1999, pp. 183–188.
[67] B. Haasdonk, Feature space interpretation of SVMs with indefinite kernels, IEEE Transactions on Pattern Analysis and Machine Intelligence 27 (4) (2005) 482–492.
[68] K. Riesen, H. Bunke, Reducing the dimensionality of dissimilarity space embedding graph kernels, Engineering Applications of Artificial Intelligence 22 (1) (2008) 48–56.
[69] K. Riesen, H. Bunke, Non-linear transformations of vector space embedded graphs, in: A. Juan-Ciscar, G. Sanchez-Albaladejo (Eds.), Pattern Recognition in Information Systems, 2008, pp. 173–186.
[70] X. Jiang, A. Münger, H. Bunke, On median graphs: properties, algorithms, and applications, IEEE Transactions on Pattern Analysis and Machine Intelligence 23 (10) (2001) 1144–1151.
[71] L. Kaufman, P. Rousseeuw, Finding Groups in Data: An Introduction to Cluster Analysis, John Wiley & Sons, 1990.
[72] B. Schölkopf, A. Smola, K.-R. Müller, Nonlinear component analysis as a kernel eigenvalue problem, Neural Computation 10 (1998) 1299–1319.
[73] K. Riesen, H. Bunke, IAM graph database repository for graph based pattern recognition and machine learning, in: N. da Vitoria Lobo et al.(Ed.), Proceedings of the International Workshops on Structural Syntactic and Statistical Pattern Recognition, Lecture Notes in Computer Science, vol. 5342, 2008, pp. 287–297.
[74] E. Alpaydin, F. Alimoglu, Pen-based recognition of handwritten digits, Department of Computer Engineering, Bogazici University, 1998.
[75] P. Dosch, E. Valveny, Report on the second symbol recognition contest, in: W. Liu, J. Lladós (Eds.), Graphics Recognition. Ten Years Review and Future Perspectives. Proceedings of the Sixth International Workshop on Graphics Recognition, Lecture Notes in Computer Science, vol. 3926, Springer, 2005, pp. 381–397.
[76] K. Riesen, H. Bunke, Kernel *k*-means clustering applied to vector space embeddings of graphs, in: L. Prevost, S. Marinai, F. Schwenker (Eds.), Proceedings of the Third IAPR Workshop Artificial Neural Networks in Pattern Recognition, Lecture Notes in Artificial Intelligence, vol. 5064, Springer, 2008, pp. 24–35.

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 to 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 also was Acting President of this organization. Horst Bunke is a Fellow of the IAPR, former Editor-in-Charge of the International Journal of Pattern Recognition and Artificial Intelligence, Editor-in-Chief of the journal 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 International Journal of 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).

He served as a co-chair of the 4th International Conference on Document Analysis and Recognition held in Ulm, Germany, 1997 and as a Track Co-Chair of the 16th and 17th International Conference on Pattern Recognition held in Quebec City, Canada and Cambridge, UK in 2002 and 2004, respectively. Also he was chairman of the IAPR TC2 Workshop on Syntactic and Structural Pattern Recognition held in Bern 1992, a co-chair of the 7th IAPR Workshop on Document Analysis Systems held in Nelson, NZ, 2006, and a co-chair of the 10th International Workshop on Frontiers in Handwriting Recognition, held in La Baule, France, 2006. Horst Bunke was on the program and organization committee of many other 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.

Kaspar Riesen received his M.S. and Ph.D. degrees in Computer Science from the University of Bern, Switzerland, in 2006 and 2009, respectively. Currently he is a researcher 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 30 publications, including six journal papers.