Available online at www.sciencedirect.com
SCIENCE DIRECT
Pattern Recognition 39 (2006) 1852–1863
PATTERN RECOGNITION
THE JOURNAL OF THE PATTERN RECOGNITION SOCIETY
www.elsevier.com/locate/patcog

# Edit distance-based kernel functions for structural pattern classification

Michel Neuhaus*, Horst Bunke

*Department of Computer Science, University of Bern, Neubrückstrasse 10, CH-3012 Bern, Switzerland*

Received 9 February 2006; accepted 6 April 2006

**Abstract**

A common approach in structural pattern classification is to define a dissimilarity measure on patterns and apply a distance-based nearest-neighbor classifier. In this paper, we introduce an alternative method for classification using kernel functions based on edit distance. The proposed approach is applicable to both string and graph representations of patterns. By means of the kernel functions introduced in this paper, string and graph classification can be performed in an implicit vector space using powerful statistical algorithms. The validity of the kernel method cannot be established for edit distance in general. However, by evaluating theoretical criteria we show that the kernel functions are nevertheless suitable for classification, and experiments on various string and graph datasets clearly demonstrate that nearest-neighbor classifiers can be outperformed by support vector machines using the proposed kernel functions.


*Keywords:* String matching; Graph matching; Edit distance; Kernel methods; Support vector machine

## 1. Introduction

In recent years, a number of pattern recognition problems have successfully been addressed by string and graph matching methods [1–3]. In particular, when the class characteristics are mainly given in terms of structure, strings and attributed graphs are often more adequate for pattern representation than feature vectors. Exact matching procedures, such as methods for graph isomorphism, maximum common subgraph, or the longest common subsequence [4–6], are one way to accomplish structural pattern matching. Although these methods are often mathematically sound in their definition, they require the underlying pattern representation to be very tolerant against noise. For pattern classification problems with non-compact and overlapping classes it is often difficult to come up with such robust representations. It is generally more convenient in such cases to address the classification problem with error-tolerant matching procedures. A popular class of error-tolerant matching methods for both strings and graphs is based on the edit distance [7–9]. The edit distance is a dissimilarity measure on patterns defined in terms of structural distortions. It allows us to compute distances between strings and attributed graphs in an intuitive manner. For the classification, however, one usually has to rely on simple nearest-neighbor-based classifiers.

In order to overcome the limitations imposed by being able to compute distances of patterns only, we propose to apply kernel functions to structural patterns. Kernel methods have seen an increasing amount of interest in the pattern recognition community in recent years [10–12]. On the one hand, the theory of kernel methods is well studied and provides us with a number of convenient results applicable to practical problems [13,14]. On the other hand, for statistical patterns, kernel machines have been successful in outperforming other types of classifiers on standard datasets. Even more relevant to the present paper is that they also allow for an elegant extension of classic pattern recognition algorithms to more complex problems and non-vectorial patterns. In the context of the present paper, this means that kernel functions allow us to apply powerful algorithms from statistical pattern recognition to the domain of strings and graphs. We are thus no longer constrained to nearest-neighbor classifiers for structural pattern recognition.

* Corresponding author. Tel.: +41 3163 18699.
*E-mail addresses:* mneuhaus@iam.unibe.ch (M. Neuhaus), bunke@iam.unibe.ch (H. Bunke).

0031-3203/$30.00 
doi:10.1016/j.patcog.2006.04.012

With the rise of kernel machines in pattern recognition, a number of structural kernels for strings and graphs have been developed. A kernel for text categorization, for instance, encoding the number of occurrences of words in texts [15] has been proposed. In another approach, a kernel on string and graph data has been developed that basically describes structures by means of the substructures they contain [16,17]. Another class of kernel functions defined on graphs are marginalized kernels [18,19], which are derived from random walks on attributed graphs. Recently, a kernel function based on the Schur–Hadamard inner product of attributed graphs has successfully been applied to error-tolerant graph matching [20]. These methods differ from our approach in that they explicitly address the problem of matching string and graph structures. In the proposed method, we leave the structural matching to the error-tolerant edit distance algorithm and use kernel machines to finally carry out the classification.

This paper is organized as follows. In Section 2, the concept of string and graph edit distance is introduced. Section 3 briefly reviews kernel methods in the context of pattern recognition, and in Section 4 the proposed kernel functions are described. We then proceed by presenting the evaluated datasets in Section 5 and give experimental results in Section 6. In Section 7, we offer a few concluding remarks. The present paper generalizes the method proposed in Ref. [21] to strings and graphs and provides a more detailed theoretical analysis and significantly extended experimental results.

## 2. String and graph edit distance

A key step in structural pattern recognition is the representation of patterns by a string or graph data structure. The structural representation should provide for a description of characteristic properties of patterns in view of the considered classification task. The classification problem can then be addressed in the corresponding string or graph space without recourse to the original pattern space [1–3]. The process of comparing strings or graphs is generally referred to as string matching or graph matching. To perform such a structural matching, various formalisms have been proposed, ranging from exact matching to error-tolerant methods based on continuous optimization, quadratic programming, and spectral decomposition of graph matrices [22–25]. One of the most common dissimilarity measures for structural patterns, in the context of classification, is based on the intuitive concept of edit distance [7–9], which has successfully been applied to various real-world problems such as handwritten text recognition [26], shape recognition [27], and fingerprint verification [28]. Edit distance requires the definition of a number of structural edit operations that are used to model variations between patterns. The edit distance of two patterns can then be derived from the best model of their structural difference.

More formally, let us assume that an alphabet $V$ of symbols is given. A *string* $t$ over $V$ is defined as an ordered sequence of symbols from $V$ of finite length, that is,

$$t = t_1 \dots t_n \in V^* = \bigcup_{i=0}^{\infty} V^i \quad \text{where } V^0 = \{\varepsilon\} \text{ and } n \geqslant 0.$$

Here $\varepsilon$ is the empty string, $V^i$ is the set of strings of length $i$ over $V$ and $V^*$ denotes the set of all finite sequences of symbols from $V$. Although a string alphabet may generally contain symbols of any kind, it is often assumed in practice that the underlying alphabet is either a finite set of symbolic characters or a vector space of a fixed dimension. We continue by introducing the notion of edit operations to model string distortions. A standard set of string edit operations consists of an insertion operation $\varepsilon \to q$, inserting a symbol $q$ into a string, a deletion operation $p \to \varepsilon$, removing a symbol $p$ from a string, and a substitution operation $p \to q$, replacing symbol $p$ in a string with symbol $q$. For certain applications, of course, it is more appropriate to use more complex edit operations, taking more than just one or two symbols into account. By successively applying edit operations it is possible to transform any string into any other string. A sequence of edit operations transforming string $t$ into $t'$ is termed as *edit path* from $t$ to $t'$, and $e(t, t')$ denotes the set of all edit paths from $t$ to $t'$. To measure the impact of an edit operation on the structure representing a pattern, we define a function $c$ that assigns a non-negative edit cost $c(w) \in R^+ \cup \{0\}$ to each edit operation $w$. The key idea is that the strength of an edit operation is reflected in its cost, such that low costs correspond to weak operations and high costs to strong operations. The cost of an edit path can then be determined by the sum of its individual edit operation costs. To finally obtain a dissimilarity measure on strings, we define the *string edit distance* $d(t, t')$ of two strings $t$ and $t'$, given edit cost function $c$, by the minimum cost required to edit $t$ into $t'$,

$$d(t, t') = \min_{\substack{(w_1,\dots,w_k) \\ \in e(t,t')}} \sum_{i=1}^{k} c(w_i).$$

Small edit distance values indicate that only a few weak edit operations are needed to model the structural difference of the two strings, while a high edit distance means that a number of strong distortions of the underlying edit operation model have to be applied to edit the first into the second string.

For the case of graph matching, graph edit distance is defined in analogy to string edit distance. Graph data structures allow for a more powerful representation than strings. Graphs consist of nodes and (directed or undirected) edges connecting two nodes. For attributed graphs, nodes and edges may additionally be provided with attributes. While a string can be seen as an ordered sequence of feature vectors (represented by symbols), a graph can similarly be regarded as a set of feature vectors (represented by nodes), augmented

with attributed relations between them (represented by edges). The string edit operation and edit path concepts can directly be extended to the graph domain. The standard set of graph edit operations consists of a node insertion, a node deletion, a node substitution, an edge insertion, an edge deletion, and an edge substitution operation. A node substitution $u \to v$, for instance, replaces node $u$ in a graph with node $v$, and an edge insertion $\varepsilon \to (p, q)$ inserts an edge connecting node $p$ with node $q$ into the graph under consideration. Given an edit cost function, the *graph edit distance* of two graphs is defined completely analogously to the string edit distance, that is, by the minimum cost edit path turning one graph into the other.

The string edit distance computation can be carried out by means of an efficient dynamic programming algorithm [7]. The computational complexity of an edit distance computation of two strings is proportional to the product of the length of the two strings. Hence also for large strings, the edit distance approach is generally feasible. For attributed graphs, however, the computational complexity is exponential in the number of nodes of the graphs. In practice, the graph edit distance can therefore be computed for small graphs only, typically with no more than about 10 nodes. In our experiments, we resort to an approximate edit distance algorithm that has proved both sufficiently accurate and computationally efficient on a number of datasets [29].

## 3. Kernel methods in pattern recognition

In recent years, insights from statistical learning theory [13,14] spawned the application of a whole range of novel algorithms for various pattern recognition tasks. A key result of statistical learning theory is that there exists an elegant solution to the problem of transforming a pattern recognition problem from a low-dimensional to a high-dimensional feature space. If two classes of patterns in a vector space are linearly separable, it is always possible to find an optimal separating hyperplane based on a training set. In cases where the condition of linear separability is not satisfied, Cover's theorem [30] may serve as a motivation to transform the data into another space before classification. The theorem states that a complex pattern classification problem cast non-linearly into a high-dimensional space is more likely to be linearly separable than in the original low-dimensional space. Fortunately for a large number of linear algorithms, it is not necessary to explicitly carry out the possibly computationally inefficient transformation.

Formally, let $\mathcal{X}$ denote the original pattern space and $k : \mathcal{X} \times \mathcal{X} \to R$ be a function mapping pairs of patterns to real numbers. If the function $k$ satisfies the condition of positive definiteness [31], there exists a vector space $\mathcal{H}$ and a mapping from $\mathcal{X}$ into $\mathcal{H}$ such that $k$ acts as a dot product in $\mathcal{H}$ [14]. That is, if we denote the mapping known to exist by $\Phi : \mathcal{X} \to \mathcal{H}$, the function $k$ can be regarded as a shortcut for computing the dot product in $\mathcal{H}$ without requiring to explicitly perform the transformation,

$$k(\mathbf{x}, \mathbf{y}) = \langle \Phi(\mathbf{x}), \Phi(\mathbf{y}) \rangle, \tag{1}$$

where $\mathbf{x}, \mathbf{y} \in \mathcal{X}$ are the original patterns and $\langle \cdot, \cdot \rangle : \mathcal{H} \times \mathcal{H} \to R$ stands for the dot product in $\mathcal{H}$. Such functions $k$ are commonly called *kernel functions*. If an algorithm can be formulated in terms of dot products of transformed vectors $\langle \Phi(\mathbf{x}), \Phi(\mathbf{y}) \rangle$ only, without recourse to the actual vectors $\Phi(\mathbf{x})$ and $\Phi(\mathbf{y})$, then by means of Eq. (1) the computation can directly be performed in the original pattern space $\mathcal{X}$ using the kernel function $k$. If a different kernel function $k'$ is applied, we obtain a variant of the algorithm, the same computation being carried out in a different implicit dot product space.

It has been shown for numerous algorithms that kernel functions can be applied in such a way. Using a maximum margin separating hyperplane classifier with kernels eventually leads to *support vector machines* (SVMs) [11,14]. Mainly due to the work by Vapnik [13], learning and convergence behavior and generalization properties of SVMs are mathematically well founded and well known. A large number of reported applications indicate that SVMs are able to generalize well on unseen data and are not prone to overfitting. Other kernel methods include principal component analysis [14], Fisher discriminant analysis [32], canonical correlation analysis, Gaussian mixture modeling, perceptron algorithms, and many others [10].

The kernel concept is not only applicable to vectorial patterns, but also to structural patterns, where $\mathcal{X}$ represents the space of strings or attributed graphs and the kernel function is defined on pairs of strings or graphs. In both the vectorial and the structural case, various positive-definite kernel functions have been developed [16,18,33,34]. The kernel function we propose in the present paper differs from these kernels in that our aim is not to define a kernel function performing the structural matching, but we rather leave the string or graph matching task to the edit distance algorithm and use the kernel function to carry out the classification afterwards, based on pairwise edit distances of the objects under consideration. Our main objective is to extend the repository of structural classification tools based on edit distance, which has been limited to classifiers of the nearest-neighbor type in the past. The edit distance-based kernel function we propose is described in the following section.

## 4. Edit distance-based kernel functions

Instead of defining a kernel function directly on strings (or graphs), we assume that strings (graphs) are characterized only by their distance to other strings (graphs). That is, after computing edit distances of strings (graphs), we do not take the actual string (graph) structures any further into account. The distance-based kernel function we propose is therefore not only applicable to string and graph data, but to all kinds of pattern representations for which distances

can be defined. In the experiments presented in Section 6, however, all distance matrices have been computed by means of either string edit distance or the approximate graph edit distance algorithm mentioned in Section 2.

Let $\mathcal{X}$ be the pattern space of strings or graphs and $\mathcal{X}^t \subset \mathcal{X}$ denote a training set of patterns. For a fixed pattern $x_0 \in \mathcal{X}^t$, a basic kernel function $k_{x_0} : \mathcal{X} \times \mathcal{X} \to R$ can then be defined by

$$k(x, x') = k_{x_0}(x, x') = \tfrac{1}{2}(d(x, x_0)^2 + d(x_0, x')^2 - d(x, x')^2), \quad (2)$$

where $d(\cdot, \cdot)$ is the (non-negative and symmetric) edit distance of two patterns. This kernel function can be understood as a measure of the squared distance from pattern $x$ to $x_0$ and from $x_0$ to $x'$ in relation to the squared distance from $x$ to $x'$ directly. Provided the kernel is valid, there exists a corresponding dot product space $\mathcal{H}$ where every string or graph $x \in \mathcal{X}$ is represented by a unique vector $\Phi(x) \in \mathcal{H}$ and the dot product operation of vectors is equal to the kernel function.

Given a dot product space, the Euclidean distance of vectors in this space can be defined by the norm of the respective difference vector. In our case, we obtain for two vectors $\Phi(x), \Phi(x') \in \mathcal{H}$

$$\begin{aligned}\|\Phi(x) - \Phi(x')\|_{\mathcal{H}}^2 &= \langle \Phi(x) - \Phi(x'), \Phi(x) - \Phi(x') \rangle \\ &= \langle \Phi(x), \Phi(x) \rangle + \langle \Phi(x'), \Phi(x') \rangle \\ &\quad - 2\langle \Phi(x), \Phi(x') \rangle \\ &= k(x, x) + k(x', x') - 2k(x, x') \\ &= d(x, x')^2 - \tfrac{1}{2}(d(x, x)^2 + d(x', x')^2) \\ &= d(x, x')^2, \end{aligned} \quad (3)$$

using the bilinearity and symmetry of the dot product and assuming symmetry of the edit distance. In other words, we find that the Euclidean distance of vectors in $\mathcal{H}$ is equal to the edit distance of the respective strings or graphs in $\mathcal{X}$. This implies, for example, that a kernel nearest-neighbor classifier in $\mathcal{H}$ will behave exactly like a nearest-neighbor classifier in $\mathcal{X}$. Also, any kernel method evaluating Euclidean distances in $\mathcal{H}$ will in fact evaluate edit distances of strings or graphs. Similarly, the length of a vector $\Phi(x)$ in $\mathcal{H}$ turns out to be equal to the edit distance of string or graph $x$ to $x_0$ in $\mathcal{X}$,

$$\begin{aligned}\|\Phi(x)\|_{\mathcal{H}}^2 &= \langle \Phi(x), \Phi(x) \rangle = d(x, x_0)^2 - \tfrac{1}{2} d(x, x)^2 \\ &= d(x, x_0)^2. \end{aligned} \quad (4)$$

Also, computing angles between strings or graphs based on the dot product in $\mathcal{H}$, we find that the angle $\alpha$ between any pattern $\Phi(x) \in \mathcal{H}$ and $\Phi(x_0)$ is undefined, which follows from

$$\cos\alpha = \frac{\langle \Phi(x), \Phi(x_0) \rangle}{\|\Phi(x)\|_{\mathcal{H}} \|\Phi(x_0)\|_{\mathcal{H}}} \quad \text{and} \quad \|\Phi(x_0)\|_{\mathcal{H}} = 0. \quad (5)$$

Moreover, two strings or graphs $\Phi(x), \Phi(x') \in \mathcal{H}$ are orthogonal if the edit distances $d(x, x_0)$, $d(x_0, x')$, and $d(x, x')$, interpreted as straight line segments in the Euclidean plane, form a right triangle. If $\Phi(x)$ and $\Phi(x')$ are orthogonal, that is, $\Phi(x) \perp \Phi(x')$, then we know from the dot product in $\mathcal{H}$ that $k(x, x') = 0$, and hence we obtain

$$\Phi(x) \perp \Phi(x') \iff d(x, x_0)^2 + d(x_0, x')^2 = d(x, x')^2. \quad (6)$$

From these properties we conclude that the fixed string or graph $x_0$—or vector $\Phi(x_0)$ in $\mathcal{H}$—exhibits characteristics of a zero vector, or element that defines the origin of the considered pattern space. Hence we call pattern $x_0$ a *zero string* or *zero graph*, respectively.

If the kernel function $k_{x_0}$ defined above is valid for any $x_0 \in \mathcal{X}$, more complex kernel functions can easily be obtained by selecting a number of zero elements and combining the resulting kernels. The pointwise sum or product of several positive-definite kernel functions, for instance, is known to be positive definite as well. Based on the function defined in Eq. (2), we therefore propose the *sum kernel function* and *product kernel function* defined by

$$k_I^+(x, x') = \sum_{x_0 \in I} k_{x_0}(x, x'), \quad (7)$$

$$k_I^*(x, x') = \prod_{x_0 \in I} k_{x_0}(x, x'), \quad (8)$$

where $I \subseteq \mathcal{X}^t \subseteq \mathcal{X}$ denotes a set of zero strings or zero graphs from the training set. The product kernel function clearly allows for a more complex reflection of distance relations of the original pattern space in the corresponding dot product space. The sum kernel does not extend the expressiveness of the kernel function, but is less constrained in terms of admissible zero patterns than the original function in Eq. (2). In experiments it can be verified that using more than one zero pattern leads to significantly better results and that the optimal number of zero patterns is relatively small (see Section 6). The product kernel function turns out to perform mostly better than the single kernel function and the sum kernel function.

For the purpose of intuitive interpretation and visualization, we proceed by investigating the proposed kernel functions in a two-dimensional Euclidean space instead of a string or graph space, using the Euclidean distance instead of edit distance to measure the dissimilarity of patterns. To obtain a graphical illustration of the behavior of the kernel functions, we choose a constant zero set of vectors $I$, fix the first kernel argument $x$, and evaluate the kernel functions for various instances of the second argument $x'$. Here, the patterns $x$, $x'$, and the elements from $I$ are simply two-dimensional vectors. The result of such a visualization is shown in Fig. 1. Note that the point at the center marked by a triangle represents the constant vector argument $x$ and the points marked by circles constitute the vectors from the zero set $I$. The brightness of the color at a position $x'$ reflects the value of the kernel function $k_{x_0}(x, x')$ in Fig. 1a, the kernel function $k_I^+(x, x')$ in Fig. 1b, and the kernel function

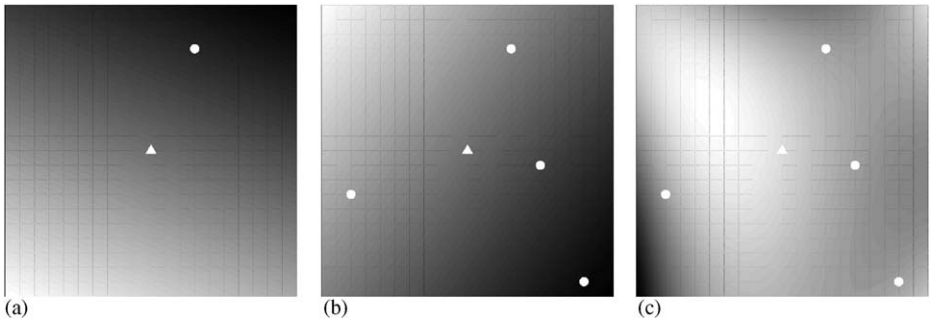


Fig. 1. Illustration of the kernel function using (a) the single kernel function given in Eq. (2), (b) the sum kernel given in Eq. (7), and (c) the product kernel given in Eq. (8).

$k_I^*(x, x')$ in Fig. 1c, respectively; brighter colors indicate higher values of the kernel function. In case of the kernel with a single zero element and the sum kernel, we observe that the resulting similarity function belongs to the class of linear functions of squared distances. The kernel assigns high similarity values to patterns $x'$ that are far away from the zero elements $I$ and simultaneously closer to $x$ than to elements in $I$, which corresponds to the intuitive interpretation of the kernel function mentioned previously. In the product kernel case in Fig. 1c, we find that the kernel function is able to reflect more complex similarity conditions and may therefore be better suited to model complex datasets.

Next, we address the question of how to find a well-performing set of zero patterns. Simple descriptors of the distribution of the zero patterns seem not to correlate with the performance of the kernel function. Although such criteria for the choice of zero sets would be desirable, we employ in our experiments an iterative, greedy selection strategy based on a validation set. We first determine the $n$ patterns $I^{(1)} \subseteq \mathcal{X}^t$ individually best performing as zero element on the validation set. Next, we identify the $n$ best performing pairs of zero patterns $I^{(2)} \subseteq I^{(1)} \times \mathcal{X}^t$. To obtain zero sets of size 3, we compute the $n$ best zero sets $I^{(3)} \subseteq I^{(2)} \times \mathcal{X}^t$. We then proceed similarly to compute zero sets of larger size up to some maximum value $s_{max}$. Eventually, we choose the zero set from $I^{(1)} \cup I^{(2)} \cup I^{(3)} \cup \cdots \cup I^{(s_{max})}$ that best performs on the validation set.

For a kernel function to be applicable in the context of statistical learning theory, it needs to satisfy the property of positive definiteness. The proposed kernel functions given in Eqs. (2), (7), and (8) are all defined with respect to an underlying pattern dissimilarity measure $d : \mathcal{X} \times \mathcal{X} \to R^+ \cup \{0\}$. From positive-definite function theory it follows that if $-d^2$ is conditionally positive definite, the kernel functions in Eqs. (2), (7), and (8) are positive definite and hence valid kernels [31]. For example, the kernel functions resulting from the Euclidean metric and several other distance measures are known to be positive definite. Edit distance measures, however, do not generally satisfy all necessary conditions. The validity of the proposed kernel function thus cannot be established in the general case.The common optimal hyperplane interpretation of SVM training as a process of margin maximization therefore cannot be adopted in our case. However, Haasdonk recently demonstrated [35] that SVM training with kernel functions violating the condition of positive definiteness can be understood, in geometrical terms, as an optimal separation of convex hulls in pseudo-Euclidean spaces. To predict whether or not a non-positive-definite kernel function is applicable to SVMs, some suitability criteria can be derived from the kernel matrix [35]. In our experiments, we actually find that a small number of zero patterns (but more than a single one) combined according to Eqs. (7) and (8) are sufficient for the kernel function to be suitable for SVM learning (see Section 6).

In cases where the kernel function does not satisfy the positive-definiteness condition, it is possible to adjust the resulting kernel matrix such that kernel methods can be applied. If the eigensystem of the kernel matrix is available, for example, reversing negative eigenvalues in the diagonal eigenvalue matrix leads to a valid kernel matrix. Another computationally more efficient approach relies on the Cholesky decomposition of the kernel matrix. Elements of the diagonal Cholesky matrix are made sufficiently positive to yield a positive-definite product matrix. A numerically more stable approach consists in performing a modified Cholesky factorization, where the matrix is updated during computation of the Cholesky factors. In this case, we obtain a positive-definite matrix differing from the original matrix only by a small amount in its diagonal elements. The modified kernel matrix can then be fed into a kernel algorithm instead of the original matrix. Further details can be found in Ref. [36]. It should be noted that the kernel matrices we applied in our experiments did not require such a modification.

## 5. String and graph datasets

In this section, we provide a description of the structural datasets the proposed kernel functions were evaluated on. Some characteristics of the datasets are summarized in Table 1.

Table 1
Number of classes and size of training set, validation set, and test set for various string and graph datasets

| Dataset | Classes | Training | Validation | Testing | Type |
|---|---|---|---|---|---|
| Chicken pieces | 5 | 149 | 149 | 148 | String |
| Toolset | 8 | 16 | 16 | 15 | String |
| Pendigits | 10 | 234 | 234 | 233 | String |
| Chromosomes | 21 | 280 | 280 | 280 | String |
| Manually distorted letters | 15 | 15 | – | 150 | Graph |
| Automatically distorted letters | 15 | 150 | 150 | 150 | Graph |
| Diatom graphs | 22 | 22 | – | 88 | Graph |
| Fingerprint graphs | 5 | 167 | 167 | 166 | Graph |

The chicken pieces dataset [37] consists of a variety of silhouettes of chicken pieces. The contour line of a silhouette is extracted by means of an edge detector. The resulting contour line is then approximated by a sequence of normalized vectors of constant length. From this vector sequence we construct a string consisting of the angles between consecutive vectors, which leads to a rotation-invariant cyclic string of relative angles representing the original chicken piece silhouette. An illustration of the processing steps can be found in Fig. 2a–c. To obtain rotation-invariant edit distances, we compute distances between these strings using an efficient cyclic string edit distance algorithm [38].

The toolset database [39] contains images of tools from eight classes, such as cutter and hammer. After binarization and edge detection, the tools are represented by contour lines. In analogy to the chicken pieces dataset, these contour lines are represented by strings of normalized vectors, and cyclic string edit distances are computed. For an illustration, see Fig. 2d–f.

The Pendigits dataset [40] contains handwritten digits produced by different writers. For our purpose, we convert a subset of the full database into strings by transforming the online handwriting into a sequence of normalized vectors. The standard string edit distance algorithm [7] can then be used to compute distances between handwritten digits. A few handwritten digit examples are provided in Fig. 2g–k.

Unlike the other string datasets, the Copenhagen Chromosome database [41] does not consist of shape descriptions, but of density histograms of chromosome images. A horizontal density profile of a chromosome image is first computed. After discretization, the resulting idealized profile can be encoded as a string. Substitution costs are set equal to the absolute difference of the corresponding discretized density values. An example image of a chromosome, the corresponding raw density profile, and the resulting discretized string are given in Fig. 2l–n.

The letter datasets contain drawings of 15 capital letters that consist of straight lines only. The first letter database includes a set of class prototypes and manually distorted patterns. Because of its small size, this dataset was only split into a training and a test set, and no validation set was used. The training set consists of the class prototypes (one letter prototype per class) and the test set of the manually distorted patterns. For the second letter database, the letter prototypes were repeatedly randomly distorted using an automatic procedure governed by a parameter controlling the strength of the distortions [42]. An illustration of some distorted copies of the same letter is given in Fig. 3a–e. Line drawings are then transformed into attributed graphs by representing end points by nodes and lines connecting end points by edges. Nodes are additionally endowed with an attribute specifying their position. Node substitution costs are defined proportional to the Euclidean distance of the node positions, and edge substitutions involve no costs. Insertions and deletions of nodes and edges have constant costs. An approximate algorithm for the efficient computation of graph edit distance is applied [29]. The manually distorted letter dataset is the only one provided with a predefined set of training patterns, namely the class prototypes. In all other cases, the full dataset has been randomly split into a training set, a validation set, and a test set of about equal size.

The small diatom database contains microscopic images of diatoms. Diatoms are unicellular algae occurring in humid places on earth [43]. Their classification is an important, but difficult task in various disciplines such as environmental monitoring and climate research. The diatom images from the database first undergo a segmentation process. Each segmented image can then be represented by a region adjacency graph. The graph extraction process is illustrated in Fig. 3f–h. For further details, the reader is referred to Ref. [44]. The edit cost function and edit distance algorithm we employ for this data set are analogous to the ones described above for the letter datasets. Similarly to the manually distorted letter data set, no validation set was defined because of the small size of this dataset.

Fingerprint classification is mainly used as a database filtering operation prior to fingerprint matching to reduce the overall running time [45]. The automatic classification of fingerprints is considered a difficult task, as the standard fingerprint classes are known to be strongly overlapping. In this paper, we use a subset of the NIST-4 database of fingerprints [46]. We obtain a graph from a fingerprint image by computing the local ridge orientation at every pixel of the image and representing the resulting orientation vector field by a grid graph [29]. An attribute representing one of eight discrete directions is attached to each edge. Insertions and deletions have constant costs, the substitution of nodes involves no costs, and edge substitution costs are proportional to the directional difference of the corresponding edge attributes. For an illustration of a fingerprint image, its orientation vector field, and the corresponding fingerprint graph refer to Fig. 3i–k.

## 6. Experimental results

In this section, an experimental evaluation of the proposed kernel functions on the string and graph datasets introduced

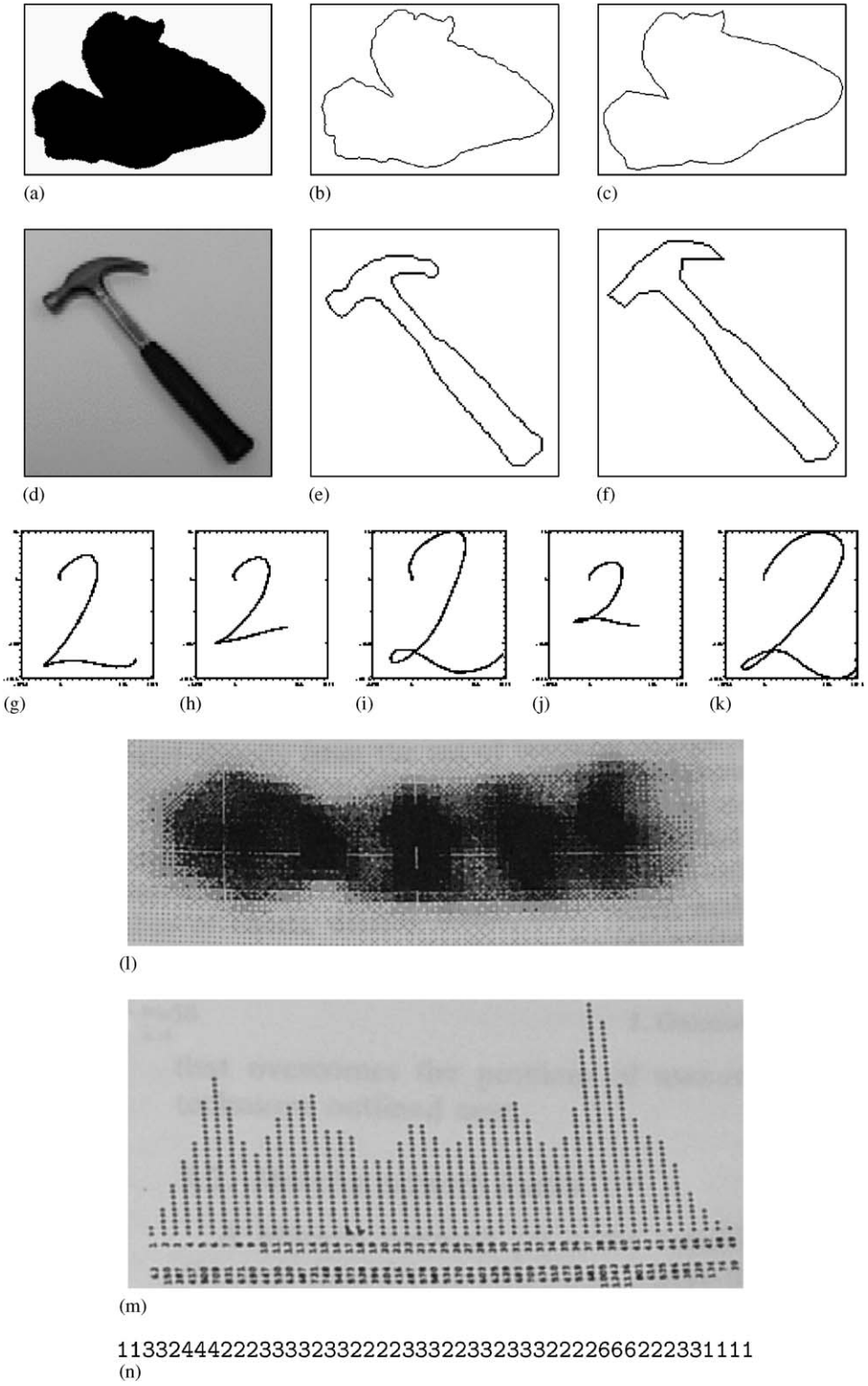


Fig. 2. Chicken pieces dataset: (a) silhouette image; (b) extracted contour line; and (c) normalized string. Tool dataset: (d) tool image; (e) extracted contour line; and (f) normalized string. Pendigits dataset: (g–k) examples of a written digit '2'. Chromosome dataset: (l) chromosome image; (m) raw density profile; and (n) discretized density string.

above is presented. We begin by studying various aspects of the behavior and performance of SVMs compared to nearest-neighbor classifiers. For the training, we use the widespread SVM implementation *libsvm* [47], using an error weighting factor of $C = 1$ throughout our experiments. The number of neighbors $k$ considered by the $k$-nearest-neighbor classifier

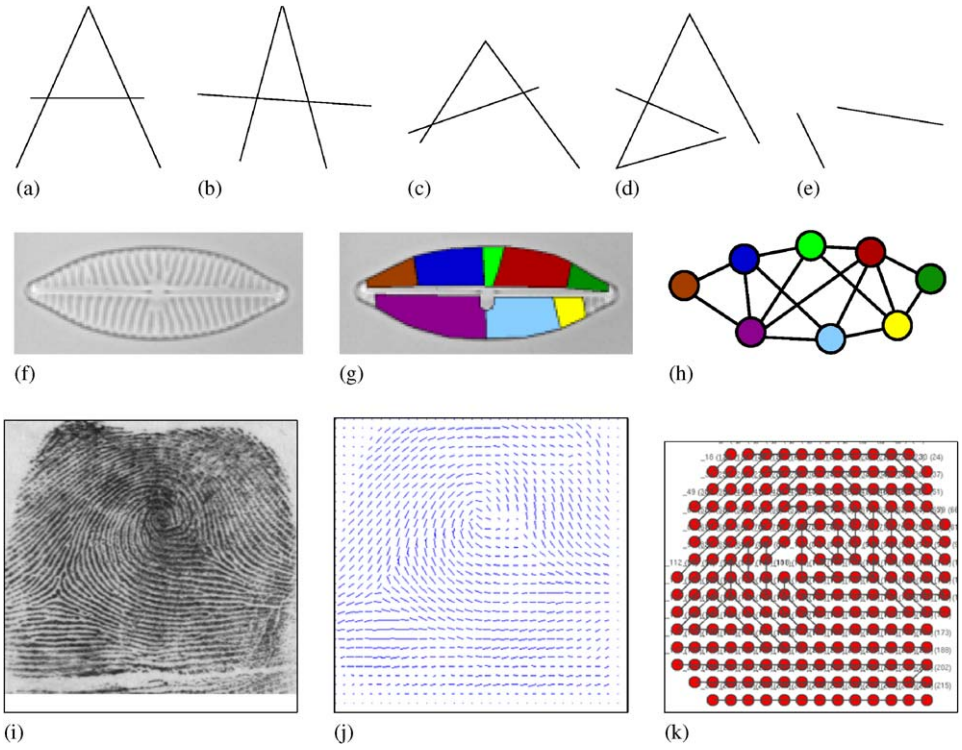


Fig. 3. Letter drawing dataset: (a–e) examples of a distorted letter 'A' drawing. Diatom dataset: (f) diatom image; (g) segmented image; and (h) region adjacency graph. Fingerprint graph dataset: (i) fingerprint image; (j) extracted orientation field; and (k) fingerprint graph.

as well as the various edit distance parameters are optimized on the validation set. A summary and discussion of the classification rates obtained on all datasets follows at the end of this section.

We first investigate the influence of the number of kernel functions combined in Eqs. (7) and (8), that is, the number of zero patterns, on the classification performance. Applying a nearest-neighbor classifier to the manually distorted letter graphs in the graph domain, we obtain a classification rate of 65.3%. Using a single zero graph from the training set to feed the same edit distances into an SVM, we are able to correctly classify all patterns, yielding a classification rate of 100% on the test set. This result is obtained for all but two graphs as zero graph from the training set. The class boundaries learned by the SVM hence constitute a much better decision criterion than that of a nearest-neighbor classifier in the graph domain. In this particular case, to obtain a perfect classification on the test set, it is sufficient to apply a training set consisting of one graph prototype per class using a single zero graph (see Fig. 1a) to an SVM, whereas the nearest-neighbor classifier in the graph domain performs relatively poorly. We conclude from this observation that the kernel function in Eq. (2) does not only exhibit interesting theoretical properties in the related dot product space, but can also extract significant characteristics of a dataset of graphs that facilitate the classification in practice. Note that a validation set is not required in this experiment, for the zero set need not be optimized.

We proceed by studying how the number of zero patterns influences the SVM performance. For this purpose we use the more difficult database of automatically distorted letters (using distortion factor 0.1, which corresponds to Fig. 3b). An SVM is trained on the training set and evaluated on the validation set according to the iterative selection procedure of a zero set described in Section 4. The zero set that results in an optimal classification of the validation set is then applied to the independent test set. The classification rate on the validation set and test set with respect to zero sets of different sizes is illustrated in Fig. 4. The larger the value on the horizontal axis, the more zero graphs are involved. Comparing the SVM performance on the validation set and test set, we find that virtually no overfitting of the validation data occurs. This observation indicates that a validation set approach for optimizing the zero set is suitable for the SVM method. Furthermore, the optimal number of zero graphs is rather small, being no greater than 6. The low classification rates obtained for zero sets of size 30 to 35 and the successive sharp rise of the classification performance indicate that the sequence of selected zero sets does not correlate with a globally optimal one, which is due to the greedy nature of our selection strategy. Nevertheless, the SVM clearly outperforms the nearest-neighbor classifier for most zero sets and in particular for smaller ones. Optimized on the validation set, the SVM achieves a classification rate of 92.7% on the independent test set, whereas the nearest-neighbor classifier reaches a classification rate of 63.3% only.

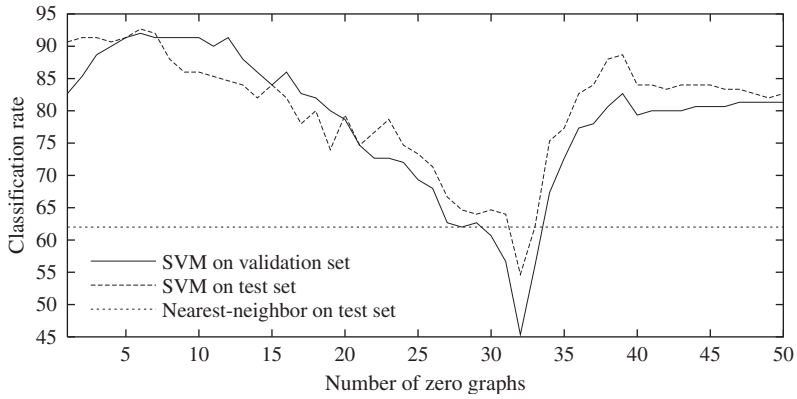


Fig. 4. Performance of an SVM and a nearest-neighbor classifier in the graph space for zero sets of various size.

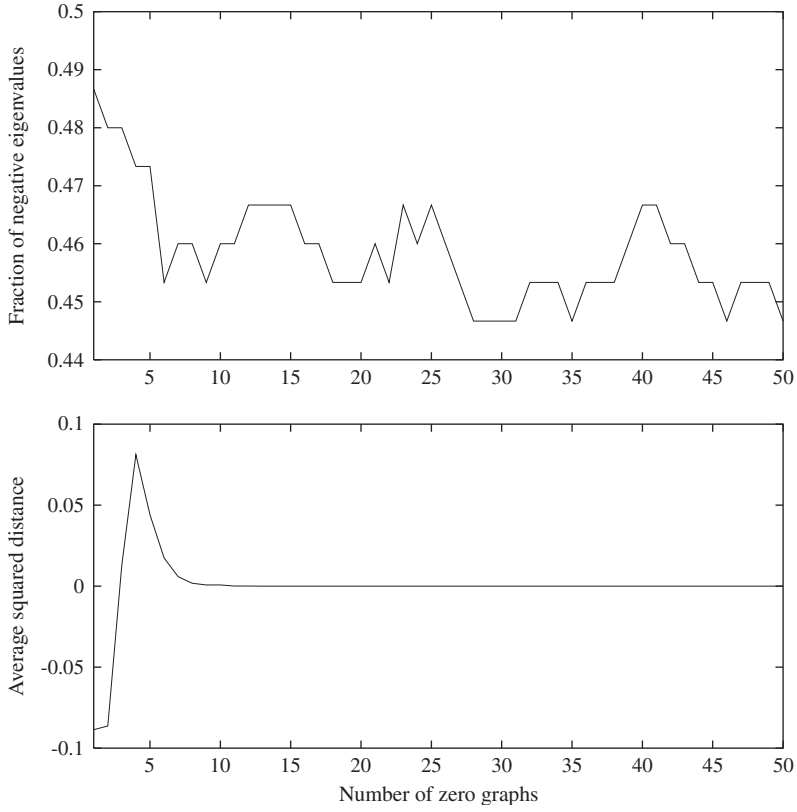


Fig. 5. Criteria for the suitability of an SVM training: fraction of negative eigenvalues (upper) and average squared distance of class means (lower).

The proposed kernel functions do not generally lead to positive-definite kernels. To verify whether it is reasonable to apply non-positive-definite kernels to SVMs, Haasdonk [35] suggests to consider the number of negative eigenvalues of the kernel matrix, since an increasing number of negative eigenvalues makes the separation of the training data more difficult. Even more precise according to Ref. [35] is the constraint that the squared distance of class means should be positive, which is not generally satisfied in pseudo-Euclidean spaces. In Fig. 5, the fraction of negative eigenvalues (which should be small) and the average squared distance of class means (which should be positive) is shown for zero sets of various size, similar to Fig. 4. We derive from these results that it makes sense to use more than one zero pattern. For zero sets of about five patterns, the geometrical interpretation of SVM training as a separation of convex hulls seems to be reasonable according to the criteria evaluated above. The conclusion that a small number of zero patterns is sufficient confirms the empirical results in Fig. 4.

During construction of the letter database, a distortion parameter is used to control the strength of applied distortions (see also Fig. 3a–e). Clearly, the higher the distortion factor, the more difficult is the letter classification problem. To verify that the SVM is not only superior to the nearest-neighbor

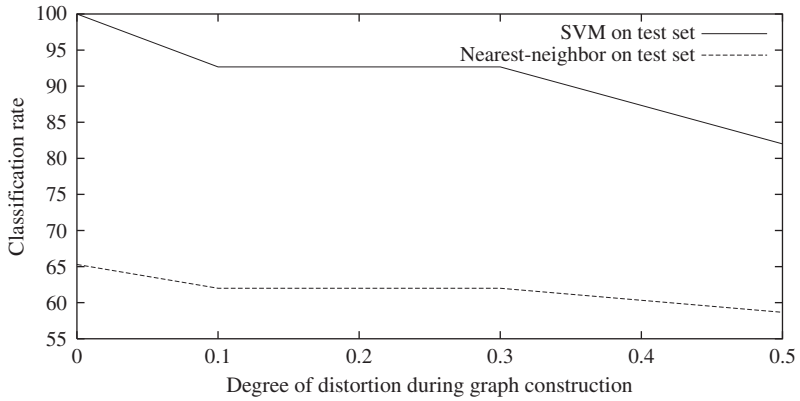


Fig. 6. Performance of an SVM classifier and a nearest-neighbor classifier in the graph space for various distortion degrees.

classifier for weakly distorted letters, we construct a number of letter graph datasets with various degrees of distortion and compute the nearest-neighbor and SVM classification performance. An illustration of the resulting classification rate is provided in Fig. 6. Note that the performance at distortion factor 0 corresponds to the manually constructed database and all other cases correspond to the automatically distorted database. The largest distortion factor considered in Fig. 6 is 0.5, which corresponds to Fig. 3e. As expected, the classification rate becomes worse for stronger distortions. Although the relative performance decrease of the SVM is more severe than that of the nearest-neighbor classifier, the SVM performs clearly better for all distortion degrees. From this result we conclude that the kernel approach is not only applicable to homogeneous classes of graphs, but also to strongly distorted datasets.

The aim of the experiments described so far was to study particular issues, such as influence of zero set size and degree of noise on SVM classification performance, using datasets particularly suited to the considered tasks. Next, we come to an overall comparison of the $k$-nearest-neighbor performance (kNN) in the original string or graph space and the SVM performance including the best number of zero elements and the utilized kernel function. A summary of the results obtained on all datasets described in Section 5 is shown in Table 2. As previously mentioned, the manually distorted letter dataset and the diatom dataset do not involve a validation set because of their small size. For all other datasets, the zero set that performs best on the validation set is identified first and then applied to the independent test set. Hence, for all datasets a computation like the one illustrated in Fig. 4 is performed. The maximum of the validation classification rate determines which zero set is used to compute the test classification rate. In Table 2, we see that SVMs outperform nearest-neighbor classifiers on all datasets. The improvement is statistically significant on all datasets ($\alpha = 0.05$) except the Chromosome dataset and the very small toolset and diatom datasets. Hence, we draw the conclusion that the proposed kernel functions are able to extract characteristics from string and attributed graph datasets that help eliminating misclassifications. For the majority of the datasets, the product kernel leads to the best results. A combination of kernels using either the sum kernel or the product kernel generally results in an improvement of the performance. The optimal size of zero sets is at a rather small average of 4.25 zero elements.

Table 2
Classification rate of a $k$-nearest-neighbor classifier (kNN), a support vector machine (SVM), and the best performing number of zero elements and kernel function

| Dataset | kNN | SVM | Size of $I$ | Kernel |
|---|---|---|---|---|
| Chicken pieces | 74.3 | 81.1[a] | 4 | $k_I^+$ |
| Toolset | 53.3 | 66.7 | 3 | $k_I^*$ |
| Pendigits | 74.2 | 89.7[a] | 4 | $k_I^*$ |
| Chromosomes | 90.7 | 91.1 | 8 | $k_I^+$ |
| Manually distorted letters | 65.3 | 100.0[a] | 1 | $k_I^+$ or $k_I^*$ |
| Automatically distorted letters | 62.0 | 92.7[a] | 7 | $k_I^*$ |
| Diatom graphs | 35.2 | 40.9 | 3 | $k_I^*$ |
| Fingerprint graphs | 41.6 | 53.0[a] | 4 | $k_I^*$ |

[a] Improvement statistically significant ($\alpha = 0.05$).

## 7. Conclusions

Edit distance has been successfully used for a long time in structural pattern recognition to measure the dissimilarity of strings and graphs. Until recently, edit distance-based systems have only been applicable in conjunction with nearest-neighbor classifiers. In this paper we propose a method to overcome this limitation by introducing a class of kernel functions that allow us to apply powerful tools from statistical pattern recognition to the structural domain of strings and graphs. The kernel functions we describe provide a direct link between the structural pattern space and the kernel space in that the Euclidean distance in the kernel space is identical to the edit distance in the pattern space. Unlike other structural kernel methods, we first perform a structural matching by computing the dissimilarity of patterns using

the edit distance. The kernel function is then defined with respect to distances between strings or graphs. The proposed method is applicable to general dissimilarity measures and has been tested using a standard and a cyclic version of the string edit distance algorithm and an approximate graph edit distance algorithm. Although the proposed kernel functions are not generally valid, they perform well on a number of real-world string and graph datasets. In experiments we find that SVMs using the kernel functions are superior to traditional nearest-neighbor classifiers. The results, evaluated on a variety of datasets, suggest that the proposed kernels are in fact applicable to various kinds of string and graph data.

In the future, we intend to verify that the proposed kernel functions can also be applied in conjunction with other statistical data analysis methods, such as principal component analysis, Fisher discriminant analysis, and clustering. We also plan to study whether our kernel method is able to improve classifiers based on tree edit distance [24,48].

## Acknowledgment

This research was supported by the Swiss National Science Foundation NCCR program "Interactive Multimodal Information Management (IM)2" in the Individual Project "Multimedia Information Access and Content Protection". The authors would also like to thank Barbara Spillmann for preparing the string datasets, and Dr. Jens Gregor for making the Chromosome dataset available.

## References

[1] Special Issue on Graph Based Representations, Pattern Recognition Lett. 24(8) (2003) 1033–1122.
[2] Special Section on Graph Algorithms and Computer Vision, IEEE Trans. Pattern Anal. Mach. Intell. 23(10) (2001) 1040–1151.
[3] Special Issue on Graph Matching in Pattern Recognition and Machine Vision, Int. J. Pattern Recognition Artif. Intell. 18(3) (2004) 261–517.
[4] G. Levi, A note on the derivation of maximal common subgraphs of two directed or undirected graphs, Calcolo 9 (1972) 341–354.
[5] J. McGregor, Backtrack search algorithm and the maximal common subgraph problem, Software Pract. Exper. 12 (1) (1982) 23–34.
[6] J. Hunt, T. Szymanski, A fast algorithm for computing longest common subsequences, Commun. ACM 20 (5) (1977) 350–353.
[7] R. Wagner, M. Fischer, The string-to-string correction problem, J. Assoc. Comput. Mach. 21 (1) (1974) 168–173.
[8] A. Sanfeliu, K. Fu, A distance measure between attributed relational graphs for pattern recognition, IEEE Trans. Syst. Man Cybern. 13 (3) (1983) 353–363.
[9] B. Messmer, H. Bunke, A new algorithm for error-tolerant subgraph isomorphism detection, IEEE Trans. Pattern Anal. Mach. Intell. 20 (5) (1998) 493–504.
[10] J. Shawe-Taylor, N. Cristianini, Kernel Methods for Pattern Analysis, Cambridge University Press, Cambridge, 2004.
[11] H. Byun, S. Lee, A survey on pattern recognition applications of support vector machines, Int. J. Pattern Recognition Artif. Intell. 17 (3) (2003) 459–486.
[12] K. Müller, S. Mika, G. Rätsch, K. Tsuda, B. Schölkopf, An introduction to kernel-based learning algorithms, IEEE Trans. Neural Networks 12 (2) (2001) 181–202.
[13] V. Vapnik, Statistical Learning Theory, Wiley, New York, 1998.
[14] B. Schölkopf, A. Smola, Learning with Kernels, MIT Press, Cambridge, MA, 2002.
[15] T. Joachims, Learning to Classify Text Using Support Vector Machines, Kluwer Academic Publisher, Dordrecht, 2002.
[16] C. Watkins, Dynamic alignment kernels, in: A. Smola, P. Bartlett, B. Schölkopf, D. Schuurmans (Eds.), Advances in Large Margin Classifiers, MIT Press, Cambridge, MA, 2000, pp. 39–50.
[17] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, C. Watkins, Text classification using string kernels, J. Mach. Learn. Res. 2 (2002) 419–444.
[18] 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.
[19] T. Gärtner, A survey of kernels for structured data, ACM SIGKDD Explor. 5 (1) (2003) 49–58.
[20] B. Jain, P. Geibel, F. Wysotzki, SVM learning with the Schur–Hadamard inner product for graphs, Neurocomputing 64 (2005) 93–105.
[21] M. Neuhaus, H. Bunke, Edit distance based kernel functions for attributed graph matching, 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, Berlin, 2005, pp. 352–361.
[22] W. Christmas, J. Kittler, M. Petrou, Structural matching in computer vision using probabilistic relaxation, IEEE Trans. Pattern Anal. Mach. Intell. 17 (8) (1995) 749–764.
[23] R. Wilson, E. Hancock, Structural matching by discrete relaxation, IEEE Trans. Pattern Anal. Mach. Intell. 19 (6) (1997) 634–648.
[24] M. Pelillo, K. Siddiqi, S. Zucker, Matching hierarchical structures using association graphs, IEEE Trans. Pattern Anal. Mach. Intell. 21 (11) (1999) 1105–1120.
[25] B. Luo, E. Hancock, Structural graph matching using the EM algorithm and singular value decomposition, IEEE Trans. Pattern Anal. Mach. Intell. 23 (10) (2001) 1120–1136.
[26] G. Seni, V. Kripásundar, R. Srihari, Generalizing edit distance to incorporate domain information: handwritten text recognition as a case study, Pattern Recognition 29 (3) (1996) 405–414.
[27] H. Bunke, U. Bühler, Applications of approximate string matching to 2D shape recognition, Pattern Recognition 26 (12) (1993) 1797–1812.
[28] A. Jain, L. Hong, R. Bolle, On-line fingerprint verification, IEEE Trans. Pattern Anal. Mach. Intell. 19 (4) (1997) 302–314.
[29] M. Neuhaus, H. Bunke, An error-tolerant approximate matching algorithm for attributed planar graphs and its application to fingerprint classification, in: Proceedings of the 10th International Workshop on Structural and Syntactic Pattern Recognition, Lecture Notes in Computer Science, vol. 3138, Springer, Berlin, 2004, pp. 180–189.
[30] T. Cover, Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition, IEEE Trans. Electron. Comput. 14 (1965) 326–334.
[31] C. Berg, J. Christensen, P. Ressel, Harmonic Analysis on Semigroups, Springer, Berlin, 1984.
[32] S. Mika, G. Rätsch, J. Weston, B. Schölkopf, K. Müller, Fisher discriminant analysis with kernels, in: Proceedings of the IEEE Workshop on Neural Networks for Signal Processing, 1999, pp. 41–48.
[33] D. Haussler, Convolutional kernels on discrete structures, Technical Report UCSC-CRL-99-10, University of California, Santa Cruz, 1999.
[34] 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.
[35] B. Haasdonk, Feature space interpretation of SVMs with indefinite kernels, IEEE Trans. Pattern Anal. Mach. Intell. 27 (4) (2005) 482–492.
[36] P. Gill, W. Murray, M. Wright, Practical Optimization, Academic Press, New York, 1981.

[37] G. Andreu, A. Crespo, J. Valiente, Selecting the toroidal self-organizing feature maps (TSOFM) best organized to object recognition, in: Proceedings of the IEEE International Conference on Neural Networks, vol. 2, 1997, pp. 1341–1346.
[38] G. Peris, A. Marzal, Fast cyclic edit distance computation with weighted edit costs in classification, in: Proceedings of the 16th International Conference on Pattern Recognition, vol. 4, 2002, pp. 184–187.
[39] K. Siddiqi, A. Shokoufandeh, S. Dickinson, S. Zucker, Shock graphs and shape matching, Int. J. Comput. Vision 30 (1999) 1–24.
[40] E. Alpaydin, F. Alimoglu, Pen-based recognition of handwritten digits, Department of Computer Engineering, Bogazici University, 1998.
[41] C. Lundsteen, J. Phillip, E. Granum, Quantitative analysis of 6985 digitized trypsin G-banded human metaphase chromosomes, Clin. Genet. 18 (1980) 355–370.
[42] S. Günter, H. Bunke, Self-organizing map for clustering in the graph domain, Pattern Recognition Lett. 23 (4) (2002) 405–417.
[43] H. du Buf, M. Bayer (Eds.), Automatic Diatom Identification, World Scientific, Singapore, 2002.
[44] R. Ambauen, S. Fischer, H. Bunke, Graph edit distance with node splitting and merging and its application to diatom identification, in: Proceedings of the 4th International Workshop on Graph Based Representations in Pattern Recognition, Lecture Notes in Computer Science, vol. 2726, 2003, pp. 95–106.
[45] D. Maltoni, D. Maio, A. Jain, S. Prabhakar, Handbook of Fingerprint Recognition, Springer, Berlin, 2003.
[46] C. Watson, C. Wilson, NIST Special Database 4. Fingerprint Database, National Institute of Standards and Technology, 1992.
[47] C. Chang, C. Lin, LIBSVM: A Library for Support Vector Machines, software available at ⟨http://www.csie.ntu.edu.tw/∼cjlin/libsvm⟩, 2001.
[48] A. Torsello, E. Hancock, Computing approximate tree edit distance using relaxation labeling, Pattern Recognition Lett. 24 (8) (2003) 1089–1097.