Graph Embedding in Vector Spaces by Means of Prototype Selection

Kaspar Riesen, Michel Neuhaus, and Horst Bunke

Department of Computer Science, University of Bern,
Neubrückstrasse 10, CH-3012 Bern, Switzerland
{riesen,mneuhaus,bunke}@iam.unibe.ch

Abstract. The field of statistical pattern recognition is characterized by the use of feature vectors for pattern representation, while strings or, more generally, graphs are prevailing in structural pattern recognition. In this paper we aim at bridging the gap between the domain of feature based and graph based object representation. We propose a general approach for transforming graphs into n-dimensional real vector spaces by means of prototype selection and graph edit distance computation. This method establishes the access to the wide range of procedures based on feature vectors without loosing the representational power of graphs. Through various experimental results we show that the proposed method, using graph embedding and classification in a vector space, outperforms the tradional approach based on k-nearest neighbor classification in the graph domain.

1 Introduction

The field of pattern recognition can be divided into two sub-fields, namely the statistical and the structural approach. In statistical pattern recognition, patterns are represented by feature vectors $(x_1, \ldots, x_n) \in \mathbb{R}^n$. The recognition process is based on the assumption that patterns of the same class are located in a compact region of $\mathbb{R}^n$. In recent years a huge amount of methods for clustering and classification of patterns represented by feature vectors have been proposed, such as k-means clustering, Bayes classifier, neural network, support vector machine, and many more. Object representations given in terms of feature vectors have a number of useful properties. For example, object similarity, or distance, can easily be computed by means of Euclidean distance. Computing the sum or weighted sum of two objects represented by vectors is straightforward, too. Yet, graph-based representations, which are used in the field of structural pattern recognition, have a number of advantages over feature vectors. Graphs are much more powerful and flexible than vectors, as feature vectors provide no direct possibility to describe structural relations in the patterns under consideration. Furthermore, vectors are constrained to a predefined length, which has to be preserved for all patterns encountered in a particular application. Obviously, graphs have a higher representational power than feature vectors. On the other

F. Escolano and M. Vento (Eds.): GbRPR 2007, LNCS 4538, pp. 383–393, 2007.

hand, a major drawback of graph representations is their lack of suitable methods for clustering and classification. This is mainly due to the fact that some of the basic operations needed in clustering and classification are not available for graphs. Hence, up to a few exceptions [1] classification of patterns represented by graphs is more or less limited to nearest-neighbor classifiers using some graph distance measure.

In this paper we describe a general method that aims at preserving the best of both approaches, that is the high representational power given by graphs and the large amount of algorithms for clustering and classification in feature vector spaces. Our approach is based on graph embedding in an n-dimensional feature vector space by means of prototype selection and edit distance computation. Originally, this idea was proposed in order to map patterns into dissimilarity spaces [2,3]. Later it was extended so as to map string representations into vector spaces [4]. In the current paper we go one step further and generalize the methods described in [4] to the domain of graphs. The key-idea of our approach is to use the distances of an input graph to a number of training graphs as vectorial description of the graph. Consequently, any statistical pattern recognition method will be applicable to such a pattern representation. In the remainder of this paper we will use the term *graph embedding* for the task of mapping graphs from the graph space into a vector space.

2 Graph Edit Distance

In contrast to statistical pattern recognition, where patterns are described by vectors, graphs do not offer a straightforward distance model like the Euclidean distance. However, a common approach to define a distance model for graphs is given by *graph edit distance*, which is one of the most flexible graph distance measures that is applicable to various kinds of graphs [5,6]. The key idea of graph edit distance is to define the dissimilarity, or distance, of graphs by the amount of distortion that is needed to transform one graph into another. These distortions are 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 is to delete some nodes and edges from g_1, relabel some of the remaining nodes and edges (substitutions) and possibly insert some nodes and edges, such that g_1 is finally transformed into g_2. A sequence of edit operations that transforms g_1 into g_2 is called an *edit path* between g_1 and g_2. One can introduce cost functions for each edit operation measuring the strength of the given distortion. The idea of such cost functions is that one can define whether or not an edit operation represents a strong modification of the graph. Hence, between two structurally similar graphs, there exists an inexpensive edit path, representing low cost operations, while for structurally 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. The edit distance can be computed, for example, by a tree search algorithm [5,7]. Typically, the edit distance is used to classify an input graph by computing its distance to a number of training

graphs and feeding the resulting distance values into a nearest-neighbor classifier. In our approach we make use of edit distances to construct a vectorial description of a given graph.

3 Graph Embedding by Means of Prototype Selection

The idea of embedding a population of graphs in an m-dimensional real vector space is motivated through the lack of suitable classification and clustering algorithms in the graph domain. An approach to graph embedding has been proposed in [8]. This method is based on algebraic graph theory and utilizes spectral matrix decomposition. In our approach we will explicitly make use of graph edit distance. Hence, we can easily deal with various kinds of graphs (labelled, unlabelled, directed, undirected, etc.) and utilize domains specific knowledge in defining the dissimilarity of nodes and edges through edit costs. Thus a high degree of robustness against various graph distortions can be achieved. The idea underlying our method was originally developed for the problem of embedding sets of feature vectors in a dissimilarity space [2,3,9,10]. In this paper we introduce an extension of this method to the domain of graphs. Assume we have a labeled set of training graphs, $T = \{g_1, \ldots, g_n\}$, and a dissimilarity measure $d(g_i, g_j)$. After having selected a set $P = \{p_1, \ldots, p_m\}$ of $m < n$ prototypes from T, we compute the dissimilarity of a graph $g \in T$ to each prototype $p \in P$. This leads to m dissimilarities, $d_1 = d(g, p_1), \ldots, d_m = d(g, p_m)$, which can be interpreted as an m-dimensional vector $(d_1, \ldots, d_m)$. In this way we can transform any graph from the training set, as well as any other graph from a validation or testing set, into a vector of real numbers. Note that whenever a graph from the training set, which has been choosen as a prototype before, is transformed into a vector $\mathbf{x} = (x_1, \ldots, x_m)$ one of the vector components is zero. Formally, if $T = \{g_1, \ldots, g_n\}$ is a training set of graphs and $P = \{p_1, \ldots, p_m\} \subseteq T$ is a set of prototypes, the mapping $t_m^P : T \to \mathbb{R}^m$ is defined as a function $t_m^P(g) \mapsto (d(g, p_1), \ldots, d(g, p_m))$ where $d(g, p_i)$ is a dissimilarity measure — in our case graph edit distance — between the graph g and the i-th prototype.

3.1 Prototype Selectors

The first problem to be solved is an appropriate choice of the prototype set $P = \{p_1, \ldots, p_m\}$. A good selection of m prototypes seems to be crucial to succeed with the classification algorithm in the feature vector space. Intuitively, the prototypes should mirror the distribution of the graphs in T as well as possible. That means protoypes should avoid redundancies in terms of selection of similar graphs, and prototypes should include as much information as possible. Hence, they should be uniformely distributed over the whole set of patterns. In this section we discuss five different algorithms for the task of prototype selection, one randomized method and four deterministic algorithms. Assume there are k different classes $c_1, \ldots, c_k$ represented in the training set T. We distinguish between class-wise and class-independent selection, that is to say that

the selection can be executed over the whole training set T to get m prototypes, or the selection can be performed individually for each of the k different classes $c_1, \ldots, c_k$. In the latter case, l_i prototypes are selected independently for each class c_i such that $\sum_{i=1}^{k} l_i = m$. Whether the class-wise or class-independent method is more convenient depends on a number of factors, including the size of T, the structure of the graphs in T, whether or not classes are balanced, and the application. The five prototype selectors used in this paper are described below.

- Centers. The Centers prototype selector selects prototypes situated in the center of the graph set T. Assume that the set median graph of a set S, $median(S)$, is the most central graph. That is, the set median graph $median(S) \in S$ is the graph whose sum of distances to all other graphs $g_i \in S$ is minimal: $median(S) = \text{argmin}_{g_1 \in S} \sum_{g_2 \in S} d(g_1, g_2)$. Then the set of prototypes $P = \{p_1, \ldots, p_m\}$ is iteratively constructed as follows:

$$P_i = \begin{cases} \varnothing & \text{if } i = 0 \\ P_{i-1} \cup \{g_i\} & \text{if } 0 < i \leqslant m \text{ , where } g_i = \text{median}(T \setminus P_{i-1}). \end{cases}$$

  It seems that the Centers prototype selector is not a very appropriate idea for selecting m prototypes, because it neither avoids redundancies nor distributes the prototypes uniformely. Nevertheless, we mention it here for the purpose of completeness. To obtain a better distribution, one can apply the Centers prototype selector also class-wise (CentersC). In this case, supposably, the prototypes mirror the given distribution better than the class-independent version.
- Random. A random selection of m prototypes from T is performed. Of course, the Random prototype selector can be applied class-independent or class-wise (RandomC). RandomC provides a random selection of l_i prototypes per class c_i.
- Spanning. A set of prototypes, P, is selected by the Spanning prototype selector by means of the following iterative procedure. The first prototype selected is the set median graph. Each additional prototype selected by the spanning prototype selector is the graph the furthest away from the already selected prototype graphs.

$$P_i = \begin{cases} median(T) & \text{if } i = 1 \\ P_{i-1} \cup \{p_i\} & \text{if } 1 < i \leqslant m \text{ , where } p_i = \underset{g \in T \setminus P_{i-1}}{\text{argmax}} \min_{p \in P_{i-1}} d(g, p). \end{cases}$$

  The Spanning prototype selector can be applied class-independent or class-wise (SpanningC).
- k-Centers. The k-Centers prototype selector tries to choose m graphs from T so that they are evenly distributed with respect to the dissimilarity information given by d. The algorithm proceeds as follows:
  1. Select an initial set of m prototypes: $P_0 := \{p_1, \ldots, p_m\}$. One can choose the initial prototypes randomly or by a more sophisticated procedure, for example, the above-mentioned spanning prototype selector.

2. Construct m sets S_i where each set consists of one prototype: $S_1 = \{p_1\}, \ldots, S_m = \{p_m\}$ For each graph $g \in T \setminus P$ find its nearest neighbor $p_i \in P$ and add the graph under consideration to the set S_i corresponding to prototype p_i. This step results in m disjoint sets with $T = \bigcup_{1 \leqslant i \leqslant m} S_i$.
3. For each set S_i find its center c_i, that is the graph for which the maximum distance to all other objects in S_i is minimum.
4. For each center c_i, if $c_i \neq p_i$, replace p_i by c_i in S_i. If any replacements is done, return to step 2, otherwise stop.

The procedure stops when no more changes in the sets S_i occur. The prototypes are given by the centers of the m disjoint sets. The k-Centers prototype selector can be applied class-independent as well as class-wise (k-CentersC). Note that this prototype selector is similar to k-means clustering [11].

– Targetsphere. The Targetsphere prototype selector first looks for the center graph g_c in T. The center graph is the graph for which the maximum distance to all other graphs in T is minimum. After finding the center graph, the graph the furthest away from g_c, i.e. the graph $g_f \in T$ whose distance to g_c is maximum, is located. Both graphs (g_c and g_f) are selected as prototypes. The distance $d_{max} = d(g_c, g_f)$ is then divided in $m-1$ partitions with $interval = \frac{d_{max}}{m-1}$. The $m-2$ graphs that are located the nearest to the interval borders in terms of edit distance are selected as prototypes:

$$P_i = \begin{cases} \{g_c, g_f\} & \text{if } i = 0 \\ P_{i-1} \cup \{g_i\} & \text{if } 0 < i \leqslant m-2 \text{ , where } g_i = \underset{g \in T \setminus P_{i-1}}{\operatorname{argmin}} |d(g, g_c) - i \cdot interval|. \end{cases}$$

The Targetsphere prototype selector can be applied class-independent as well as class-wise (TargetsphereC).

Of course, one can imagine other techniques and strategies for prototype selection. The intention of all methods remains the same — finding a good selection of m prototypes that lead to a good performance of the resulting classifier in the vector space.

4 The Classification Problem

4.1 Classification in Graph Spaces — k-NN Classifier

The traditional approach to addressing the classification problem in a graph space is to apply a k-nearest-neighbor classifier (k-NN) in conjunction with edit distance. Given a labeled set of training graphs, an unknown graph is assigned to the class that occurs most frequently among the k nearest graphs (in terms of edit distance) from the training set. Formally, let us assume that a pattern space X, a space of class labels Y, and a labeled training set of patterns $\{(x_i, y_i)\}_{i=1,\ldots,m} \subseteq X \times Y$ is given. If $\{(x_{(1)}, y_{(1)}), \ldots, (x_{(k)}, y_{(k)})\} \subseteq \{x_i, y_i)\}_{i=1,\ldots,m}$ are the k patterns that have the smallest distance $d(x, x_{(i)})$ to a test pattern x, the k-NN classifier $f : X \to Y$ can be defined by

$$f(x) = \underset{y \in Y}{\operatorname{argmax}} |\{(x_{(i)}, y_{(i)}) : y_{(i)} = y\}|$$

If $k = 1$ the k-NN classifiers decision is based on just one element from the training set, no matter if this element is an outlier or a true class representative. Obviously, a choice of parameter $k > 1$ reduces the influence of outliers by evaluating which class occurs most frequently in a neighborhood around the test pattern. This classifier in the graph domain will serve us as a reference system.

4.2 Classification in Vector Spaces — Support Vector Machine Classifier

As already pointed out, in vector spaces a large amount of methods for pattern classification exist. Besides the k-NN classifier, one can choose among more sophisticated algorithms, such as neural network, Bayes classifier, decision tree classifier, support vector machine, and others. Pattern classification by means of support vector machines (SVMs) has become very popular recently. In the present paper we want to compare the classification accuracy achieved by k-NN classifiers in the graph domain with k-NN classifiers and SVM in vector spaces. It is our objective to find out if we can outperform the classification accuracy obtained by k-NN classifiers in the graph space by classifiers relying on vectorial representations after embedding the graphs in an m-dimensional vector space, especially by SVM. For the sake of completeness, we give a brief overview of SVMs below. For a more thorough introduction we refer the reader to [12,13,14]

The basic idea of SVM is to separate classes of patterns by hyperplanes. Assume a pattern space $X = \mathbb{R}^n$, two classes $Y = \{-1, +1\}$ and a labeled training set $\{(x_i, y_i)\}_{i=1,\ldots,m} \subseteq X \times Y$ are given, and the two classes are linearly seperable. Then, there exist infinitely many possible separating hyperplanes that correctly classify the training data. The fact that all patterns are classified correctly can be written as $y_i \cdot (\langle w, x_i \rangle + b) > 0$ for $i = 1, \ldots, m$, with parameters $w \in \mathbb{R}^n$ and $b \in \mathbb{R}$. Intuitively, one would choose a hyperplane such that its distance to the clostest pattern of either class is maximal. Such hyperplanes are commonly called maximum-margin hyperplanes. The sum of the distances from the hyperplane to the closest pattern of each class is commonly termed *margin*. The maximum-margin hyperplane is expected to perform best on an independent test set. Since multiplying the parameters w and b with a constant does not change the hyperplane, one can rescale them such that $\min_{i=1,\ldots,m} |\langle w, x_i \rangle + b| = 1$. This rescaled hyperplane is called to be in *canonical form*. Assume a hyperplane in canonical form is given. Then it is obvious that $margin = \frac{2}{||w||}$. Hence, the smaller the length of the weight vector $||w||$, the larger is the margin. Since we are looking for an optimal hyperplane, which maximizes *margin*, we have to find parameters w and b such that

- $margin = \frac{2}{||w||}$ is maximum
- subject to $y_i \cdot (\langle w, x_i \rangle + b) \geqslant 1$ for $i = 1, \ldots, n$.

The first line corresponds to the maximization of *margin*, while the second line assures that all training samples are classified without error. In a more realistic scenario the two classes would not be linearly seperable anymore. Thus the

second condition does not hold. In order to handle linearly non-seperable data, so called *slack variables* ξ are used. Whenever a training element x_i is misclassified by the hyperplane, the corresponding slack variable ξ_i is greater than zero. Thus we have to find parameters w and b such that

- $margin = \frac{||w||^2}{2} + C \sum_{i=1}^{m} \xi_i$ is minimum
- subject to $y_i \cdot (\langle w, x_i \rangle + b) \geqslant 1 - \xi_i$ for $i = 1, \ldots, m$.

Minimization of $\frac{||w||^2}{2}$ is equivalent to the maximization of *margin*. Quantity $C \geqslant 0$ denotes a regularization parameter to control whether the maximization of *margin* or the minimization of the sum of errors is more important. Finally, we have to deal with general non-linear classification problems, where a linear hyperplane will not work any longer. Fortunately, one can show that we can transform a linear to a non-linear classifier by only substituting the original dot product with a specific kernel function K [13,14]. Hence, by changing the dot product to a kernel function we can get different non-linear classifiers. In our experiments we used the following kernel functions:

- **Linear:** $K(u, v) = u' \cdot v$
- **Polynomial:** $K(u, v) = (\gamma \cdot u' \cdot v)^d$
- **Radial Basis Function (rbf):** $K(u, v) = exp(-\gamma \cdot ||u - v||^2)$

where $\gamma \in \mathbb{R}$ and $d \in \mathbb{N}$.

5 Experimental Results

As main contribution of this paper, we have introduced a general approach for transforming graphs into n-dimensional real vector spaces by means of prototype selection and graph edit distance. It is furthermore our intention to demonstrate that certain classification tasks can be better solved with methods that use the resulting vectorial patterns rather than the original graph representation. Hence, the reference system in our experiments is given by a k-NN classifier in the graph domain, while the proposed statistical classifiers in real vector spaces are given by different SVMs. Note that k-NN classifiers are the only classifiers that can be directly applied in the original graph domain. In each of our experiments we make use of three disjoint graph sets, viz. *validation set*, *test set* and *training set*. The validation set is used to determine optimal parameter values for graph embedding and classification. The embedding parameters consist of the number of prototypes, i.e. the dimensionality of the resulting feature vector space, and the best performing embedding method, while the parameters for classification consist of parameter k for the nearest neighbor classifier and the different parameters for the SVMs, i.e. C, γ and d. That is, for each embedding method and dimensionality an individual SVM is trained. The parameter values, the embedding method, and the dimensionality that result in the lowest classification error on the validation set are then applied to the independent test set.

5.1 Letter Database

The first database used in the experiments consists of graphs representing distorted letter drawings. In this experiment we consider the 15 capital letters of the Roman alphabet that consists of straight lines only (*A, E, F, ...)*. For each class, a prototype line drawing is manually constructed. To obtain aribtrarily large sample sets of drawings with arbitrarily strong distortions, distortion operators are applied to the prototype line drawings. This results in randomly shifted, removed, and added lines. These drawings are then converted into graphs in a simple manner by representing lines by edges and ending points of lines by nodes. Each node is labeled with a two-dimensional attribute giving its position. The graph database used in our experiments is composed of a training set, a validation set, and a test set, each of size 750. In Table 1 we give the classification accuracy of a k-nearest neighbor classifier in the graph space, a k-nearest neighbor classifier in the feature vector space, and the 3 different SVM-classifiers. The best accuracy on the validation set with all SVMs is achieved by the k-CentersC prototype selector (Distortions 0.1 and 0.5) and the TargetsphereC prototype selector (Distortions 0.3, 0.7 and 0.9). Therefore, these prototype selectors have been used for all SVMs when classifying the test set. It turns out that classification accuracy can be improved by all considered SVMs on all distortion levels. Note that 9 out of 15 improvements are statistically significant.

Table 1. Letter Database: Classification accuracy in the graph and vector space

	Ref. System	Embedding classifiers			
Distortion	k-NN (graph)	k-NN (vector)	SVM (lin)	SVM (poly)	SVM (rbf)
0.1	98.27	98.53	98.93	98.40	98.53
0.3	97.60	97.47	98.53 ○	98.53 ○	98.80 ○
0.5	94.00	93.60	97.07 ○	97.20 ○	96.93 ○
0.7	94.27	92.53 ●	95.33	95.47	95.47
0.9	90.13	91.20	92.93 ○	92.93 ○	92.93 ○

○ Statistically significantly better than the reference system ($\alpha = 0.05$).
● Statistically significantly worse than the reference system ($\alpha = 0.05$).

5.2 Real World Data

For a more thorough evaluation of the proposed methods we additonally use three real world data sets. First we apply the proposed method to the problem of image classification. Images are converted into graphs by segmenting them into regions, eliminating regions that are irrelevant for classification, and representing the remaining regions by nodes and the adjacency of regions by edges [15]. The Le Saux image database consists of five classes (*city, countryside, people, snowy, streets*) and is split into a training set, a validation set and a test set of size 54 each. The best accuracy on the validation set with all SVMs is achieved by the Targetsphere prototype selector. The classification accuracies obtained by the different methods are given in the first row of Table 2. We note that the SVM

Table 2. Fingerprint-, Image- and Molecules Database: Classification accuracy in the graph and vector space

	Ref. System	Embedding classifiers			
Database	k-NN (graph)	k-NN (vector)	SVM (lin)	SVM (poly)	SVM (rbf)
Le Saux	57.4	48.2	59.3	57.4	64.8
NIST-4	82.6	81.8	85.4 ○	82.4	85.0 ○
Molecules	97.1	95.9 ●	97.7	98.2 ○	98.1 ○

○ Statistically significantly better than the reference system ($\alpha = 0.05$).
● Statistically significantly worse than the reference system ($\alpha = 0.05$).

with polynomial kernel results in the same error rate as the reference system, while the other two kernels lead to lower error rates. Although the SVM with the rbf kernel function improves the accuracy by 7.4%, this improvement is not statistically significant. This is due to the small size of Le Saux database.

The second real world dataset is given by the NIST-4 fingerprint database [16]. We construct graphs from fingerprint images by extracting characteristic regions in fingerprints and converting the results into attributed graphs [17]. We use a validation set of size 300 and a test and training set of size 500 each. In this experiment we address the 4-class problem (*arch, left-loop, right-loop, whorl*). Validation of parameter values needed by linear and polynomial SVM prove to be difficult. With many parameter value combinations, both SVMs are not able to terminate the optimization in a reasonable time. Thus, the optimization of the parameter values is based on a subset of all parameter combinations only. Nevertheless, the linear and rbf kernel SVMs achieve statistically significantly better results than the reference system. These results are achieved by the TargetsphereC prototype selector. The number of choosen prototypes per class is proportional to the respective class size.[1]

Finally, we apply the proposed method of graph embedding and subsequent SVM classification to the problem of molecule classification. To this end, we construct graphs from the AIDS Antiviral Screen Database of Active Compounds [18]. Our molecule database consists of two classes (*active, inactive*), which represent molecules with activity against HIV or not. We use a validation set of size 250, a test set of size 1500 and training set of size 250. Thus, there are 2000 elements totally (1600 inactive elements and 400 active elements). The molecules are converted into graphs in a straightforward manner by representing atoms as nodes and the covalent bonds as edges. Nodes are labeled with the number of the corresponding chemical symbol and edges by the valence of the linkage. The results achieved on this database are shown in the third row of Table 2. Although the accuracy of the reference system in the graph domain is quite high, it can be improved by graph embedding and SVM classification. In two out of three cases, the improvement is statistically significant. On this data set the Spanning prototype selector obtains the best result on the validation set and is therefore used on the test set.

[1] In contrast with other databases, the individual classes are of different size.

6 Conclusions

Although graphs have a higher representational power than feature vectors, there is a lack of suitable methods for pattern classification using graph representations. By contrast, a large number of methods for classification have been proposed for object representations given in terms of feature vectors. In this paper, we propose a general approach for bridging the gap between structural and statistical pattern recognition. The idea is to map graphs to an n-dimensional real vector space by means of prototype selection and graph edit distance. To this end, we discuss different prototype selectors with the objective of finding a good distribution of these prototypes. With the proposed graph embedding a large number of different methods from statistical pattern recognition become available to graph representations. It has been our intention in this paper to improve the accuracy achieved by nearest neighbor classifiers in the graph domain by classifiers operating on vectorial representations. We used SVMs as a popular method from statistical pattern recognition and showed that this approach outperforms the nearest neighbor classifiers in the graph domain. From the results of our experiments, one can conclude that the classification accuracy can be statistically significantly enhanced by most SVMs and different prototype selection algorithms. In our future work we will study additional statistical classifiers and try to further improve the recognition accuracy by using classifier ensembles.

Acknowledgements

This work has been supported by the Swiss National Science Foundation (Project 200021-113198/1). Furthermore, we would like to thank R. Duin for valuable discussions and hints regarding our embedding methods. Finally, we thank B. Le Saux for making the Le Saux database available to us.

References

1. Bianchini, M., Gori, M., Sarti, L., Scarselli, F.: Recursive processing of cyclic graphs. IEEE Transactions on Neural Networks 17(1), 10–18 (2006)
2. Pekalska, E., Duin, R., Paclik, P.: Prototype selection for dissimilarity-based classifiers. Pattern Recognition 39(2), 189–208 (2006)
3. Duin, R., Pekalska, E.: The Dissimilarity Representations for Pattern Recognition: Foundations and Applications. World Scientific, Singapore (2005)
4. Spillmann, B., Neuhaus, M., Bunke, H., Pekalska, E., Duin, R.: Transforming strings to vector spaces using prototype selection. In: Yeung, D.-Y., Kwok, J.T., Fred, A., Roli, F., de Ridder, D. (eds.) Structural, Syntactic, and Statistical Pattern Recognition. LNCS, vol. 4109, pp. 287–296. Springer, Heidelberg (2006)
5. Bunke, H., Allermann, G.: Inexact graph matching for structural pattern recognition. Pattern Recognition Letters 1, 245–253 (1983)
6. Sanfeliu, A., Fu, K.S.: A distance measure between attributed relational graphs for pattern recognition. IEEE Transactions on Systems, Man, and Cybernetics (Part B) 13(3), 353–363 (1983)

7. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions of Systems, Science, and Cybernetics 4(2), 100–107 (1968)
8. Wilson, R.C., Hancock, E.R., Luo, B.: Pattern vectors from algebraic graph theory. IEEE Trans. on Pattern Analysis ans Machine Intelligence 27(7), 1112–1124 (2005)
9. Hjaltason, G., Samet, H.: Properties of embedding methods for similarity searching in metric spaces. IEEE Trans. on Pattern Analysis ans Machine Intelligence 25(5), 530–549 (2003)
10. Roth, V., Laubm, J., Kawanabe, M., Buhmann, J.: Optimal cluster preserving embedding of nonmetric proximity data. IEEE Trans. on Pattern Analysis ans Machine Intelligence, vol. 15(12) (2003)
11. MacQueen, J.: Some methods for classification and analysis of multivariant observations. In: Proc. 5th. Berkeley Symp University of California Press 1, pp. 281–297 (1966)
12. Burges, C.: A tutorial on support vector machines for pattern recognition. Data. Mining and Knowledge Discovery 2(2), 121–167 (1998)
13. Shawe-Taylor, J., Cristianini, N.: Kernel Methods for Pattern Analysis. Cambridge University Press, Cambridge (2004)
14. Schölkopf, B., Smola, A.: Learning with Kernels. MIT Press, Cambridge, MA (2002)
15. Le Saux, B., Bunke, H.: Feature selection for graph-based image classifiers. In: Marques, J.S., de la Blanca, N.P., Pina, P. (eds.) IbPRIA 2005. LNCS, vol. 3523, pp. 147–154. Springer, Heidelberg (2005)
16. Watson, C.I., Wilson, C.L.: NIST special database 4, fingerprint database. National Institute of Standards and Technology, March (1992)
17. Neuhaus, M., Bunke, H.: A graph matching based approach to fingerprint classification using directional variance. In: Kanade, T., Jain, A., Ratha, N.K. (eds.) AVBPA 2005. LNCS, vol. 3546, pp. 191–200. Springer, Heidelberg (2005)
18. Development Therapeutics Program DTP. Aids antiviral screen (2004), http://dtp.nci.nih.gov/docs/aids/aids_data.html.