Exact Median Graph Computation Via Graph Embedding

Miquel Ferrer[1], Ernest Valveny[1], Francesc Serratosa[2], and Horst Bunke[3]

[1] Centre de Visió per Computador, Dep. Ciències de la Computació,
Universitat Autònoma de Barcelona, Edifici O, Campus UAB, 08193 Bellaterra, Spain
{mferrer,ernest}@cvc.uab.cat
[2] Departament d'Enginyeria Informàtica i Matemàtiques,
Universitat Rovira i Virgili, Av. Països Catalans 26, 43007 Tarragona, Spain
francesc.serratosa@urv.cat
[3] Department of Computer Science and Applied Mathematics,
University of Bern, Neubrückstrasse, 14 CH3012 Bern, Switzerland
bunke@iam.unibe.ch

Abstract. Given a set of graphs, the median graph is defined as the graph which has the smallest sum of distances (SOD) to all the graphs in the set. It has been proposed as a tool to obtain the representative of such a set. In spite of its potential applications, the existing algorithms are computationally complex and have a very limited applicability. In this paper we propose a new approach for the exact computation of the median graph based on graph embedding in vector spaces. Graphs are embedded into a vector space and the median is computed in the vector domain. After that, the median graph is recovered from this median vector. Experiments on a synthetic database show that our approach outperforms the previous existing exact algorithms both on computation time and number of SOD computations.

1 Introduction

Given a set of graphs, the median graph [1] is defined as a graph that has the minimum sum of distances (SOD) to all graphs in the set. It can be seen as the representative of the set and, therefore, it has a large number of potential applications including many classical algorithms for learning, clustering and classification. Furthermore, it can be potentially applied to any graph-based algorithm where a representative of a set of graphs is needed. However, the computation of the median graph is exponential both in the number of input graphs and their size [2]. A number of algorithms have been reported in the past to compute the median graph. Two exact approaches have been proposed in [2] and [3]. As the computational cost of these algorithms is very high, a set of approximate algorithms have also been presented in the past based on different approaches such as genetic search [1,2], greedy algorithms [4] and spectral graph theory [5].

In this paper we propose a novel technique for the median graph computation based on graph embedding into vector spaces. It is composed of three main steps:

the embedding of graphs in a vector space; the median vector computation and the recovering of the median graph from this median vector. In all these steps we use the graph edit distance under the assumption it is computed using a particular cost function. The use of this cost function is motivated by the fact that it permits to relate the distance between two graphs with their *mcs* [6] and to dramatically reduce the search space of the median graph as shown in [3]. Differently of the approach presented in [3], where a suboptimal method for median computation in the vector space is applied and the general edit distance is used to obtain the back-transformation, our method yields to obtain exact solutions for the median graph.

We performed a set of experiments using synthetic data and we have compared our approach with all the existing methods for the exact median computation. Results show that our method outperforms the existing methods both in computation time and the number of SOD computations, while obtaining the same or equivalent medians in terms of their SOD. With these results at hand, the use of the median graph in real (although limited) applications becomes a realistic choice.

The rest of this paper is organized as follows. In the next section we define the basic concepts and we introduce the notation we will use later in the paper. Then, in Section 3 we introduce in detail the concept of the median graph. In Section 4 the proposed method for the median computation is described. Section 5 reports a number of experiments and present results achieved with our method. Finally, in Section 6 we draw some conclusions and we point out to possible future work.

2 Basic Definitions

2.1 Graph and Subgraph

Definition 1 (Graph). *Given L, a finite alphabet of labels for nodes and edges, a graph g is defined by the four-tuple $g = (V, E, \mu, \nu)$ where V is a finite set of nodes, $E \subseteq V \times V$ is the set of edges, μ is the node labeling function ($\mu : V \longrightarrow L$) and ν is the edge labeling function ($\nu : V \times V \longrightarrow L$). The number of nodes of a graph g is denoted by $|g|$.*

Notice that L is not constrained in any way. It can be defined as a vector space (i.e. $L = \mathbb{R}^n$) or simply as a set of discrete labels (i.e. $L = \{\Delta, \Sigma, \Psi, \cdots\}$).

Definition 2 (Subgraph). *Let $g_1 = (V_1, E_1, \mu_1, \nu_1)$, and $g_2 = (V_2, E_2, \mu_2, \nu_2)$ be two graphs. The graph g_1 is a subgraph of g_2, denoted by $g_1 \subseteq g_2$ if: $V_1 \subseteq V_2$, $E_1 = E_2 \cap (V_1 \times V_1)$, $\mu_1(u) = \mu_2(u)$ for all $u \in V_1$ and $\nu_1(e) = \nu_2(e)$ for all $e \in E_1$.*

From Definition 2 it follows that, given a graph $g = (V, E, \mu, \nu)$, a subset $V' \subseteq V$ of its vertices uniquely defines a subgraph, called the subgraph *induced* by V'. That is, an induced subgraph of g can be obtained by removing some of its nodes $(V - V')$ and all their adjacent edges.

2.2 Maximum Common Subgraph and Minimum Common Supergraph

Definition 3 (Maximum Common Subgraph (mcs)). *Let $g_1 = (V_1, E_1, \mu_1, \nu_1)$ and $g_2 = (V_2, E_2, \mu_2, \nu_2)$ be two graphs. A graph g is called a common subgraph (cs) of g_1 and g_2 if there exists a subgraph isomorphism from g to g_1 and from g to g_2. A common subgraph of g_1 and g_2 is called maximum common subgraph (mcs) if there exists no other common subgraph of g_1 and g_2 with more nodes than g.*

Definition 4 (Minimum Common Supergraph (MCS)). *Let $g_1=(V_1, E_1, \mu_1, \nu_1)$ and $g_2 = (V_2, E_2, \mu_2, \nu_2)$ be two graphs. A graph g is called a common supergraph (CS) of g_1 and g_2 if there exists a subgraph isomorphism from g_1 to g and from g_2 to g. A common supergraph of g_1 and g_2 is called minimum common supergraph (MCS) if there exists no other common supergraph of g_1 and g_2 with less nodes than g.*

Definition 5 (Minimum Common Supergraph of a Set of Graphs). *Let $S = \{g_1, g_2, ..., g_n\}$ be a set of graphs. A graph $g_M(S)$ (also denoted by $MCS(S)$) is called a minimum common supergraph of S if $\{g_1, g_2, \cdots, g_n\}$ are subgraphs of $g_M(S)$ and there is no other common supergraph of $\{g_1, g_2, \cdots, g_n\}$ with less nodes than $g_M(S)$.*

2.3 Graph Edit Distance

The graph edit distance [7,8], has been shown as one of the most widely used methods to compute the dissimilarity between two graphs.

The basic idea behind the graph edit distance is to define the dissimilarity of two graphs as the minimum amount of distortion required to transform one graph into the other. To this end, a number of distortion or edit operations e, consisting of the insertion, deletion and substitution of both nodes and edges are defined. Given these edit operations, for every pair of graphs, g_1 and g_2, there exists a sequence of edit operations, or edit path $p(g_1, g_2) = (e_1, \ldots, e_k)$ (where each e_i denotes an edit operation) that transforms g_1 into g_2. In general, several edit paths exist between two given graphs. This set of edit paths is denoted by $\wp(g_1, g_2)$. To quantitatively evaluate which edit path is the best, edit costs are introduced through a cost function. The basic idea is to assign a penalty cost c to each edit operation according to the amount of distortion it introduces in the transformation. The edit distance d between two graphs g_1 and g_2, denoted by $d(g_1, g_2)$, is the minimum cost edit path that transforms one graph into the other.

Graph Edit Distance and the Maximum Common Subgraph. In this work, we will use the graph edit distance under a particular cost function [6], where deletions and insertions of nodes have always a cost of 1, deletions and insertions of edges have always a cost of 0, and node and edge substitutions take the values of 0 or ∞ depending on whether the substitution is identical or not, respectively. Under this cost function, the graph edit distance between two graphs $d(g_1, g_2)$ is related to their mcs as follows:

$$d(g_1, g_2) = |g_1| + |g_2| - 2\,|mcs(g_1, g_2)| \quad (1)$$

Although the definition of the cost function might seem quite simple, this result demonstrates the intuitive idea that the more two graphs have in common, the lower is their distance. Several other distances related to the mcs have been proposed, and summarized in [9]. In the rest of the paper we will assume, that the distance between two graphs is computed according to the Eq. (1).

3 Generalized Median Graph

Definition 6 (Generalized Median Graph). *Let U be the set of graphs that can be constructed using labels from L. Given $S = \{g_1, g_2, ..., g_n\} \subseteq U$, the* **generalized median graph** *$\bar{g}$ of S is defined as:*

$$\bar{g} = arg \min_{g \in U} \sum_{g_i \in S} d(g, g_i) \quad (2)$$

That is, $\bar{g}$ is a graph $g \in U$ that minimizes the sum of distances (SOD) to all the graphs in S. Notice that $\bar{g}$ is usually not a member of S, and in general more than one generalized median graph may exist for a given set S.

Two important aspects in the median graph computation are the distance computation and the search space. In the general case, the search space becomes exponential with respect to the sum of nodes of the graphs in S. Nevertheless, in [3], it is shown that under the particular cost function introduced in Section 2.3,

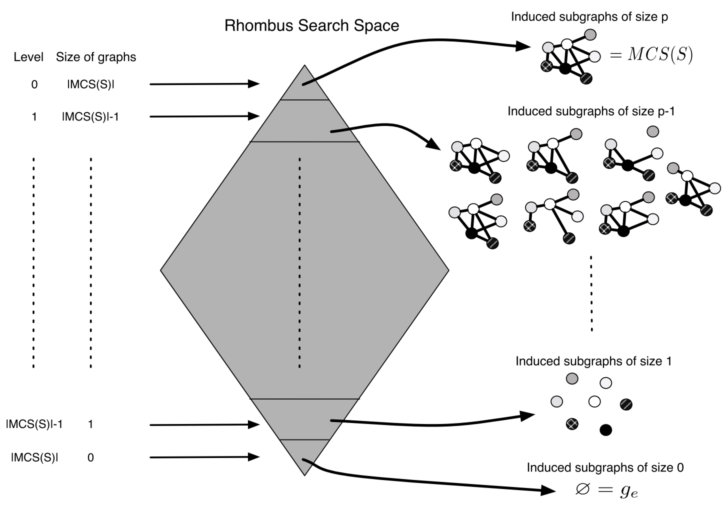

Rhombus Search Space
Level
Size of graphs
0
IMCS(S)I
1
IMCS(S)I-1
IMCS(S)I-1
1
IMCS(S)I
0
Induced subgraphs of size p
= MCS(S)
Induced subgraphs of size p-1
Induced subgraphs of size 1
Induced subgraphs of size 0
∅ = g_e

Fig. 1. Review of rhombus search space

the search space of the median graph can be drastically reduced and is composed only of the induced subgraphs of $g_M(S)$. Figure 1 shows this search space (also called Rhombus). At the top of the Rhombus there is the minimum common supergraph of S, $MCS(S)$. At the bottom, there is the empty graph $g_e = \phi$. Inside the rhombus there are all the possible induced subgraphs of $MCS(S)$. Each graph in S is located in this search space. In the following we will assume that we are working in this search space and therefore the term $MCS(S)$ have to be computed.

4 Exact Median Graph Via Graph Embedding

In this section we present a new embedding technique for the median graph computation (we will assume that a set of n graphs $S = \{g_1, g_2, \ldots, g_n\}$ for the median graph computation is given), that is composed of three main steps:

- **1. Graph Embedding in a Vector Space:** Each graph in S becomes a point in an n-dimensional space. The graph edit distance is used for this embedding.
- **2. Median Vector Computation:** The median vector is computed using the points of the first step.
- **3. Going back to the Graph Domain:** The median vector is converted back to a graph, which is taken as the median graph of S.

4.1 Graph Embedding in Vector Spaces

In order to perform the embedding step, we will use a novel procedure proposed in [10], briefly described in the following.

Assume we have a set of training graphs $T = \{g_1, g_2, \ldots, g_n\}$ and a graph similarity measure $d(g_i, g_j)$. Then, a set $P = \{p_1, \ldots, p_m\} \subseteq T$ of m prototypes is selected from T (with $m \leq n$). After that, the similarity between a given graph of $g \in T$ and every prototype $p \in P$ is computed. This leads to m dissimilarity values, $d_1, \ldots, d_m$ where $d_k = d(g, p_k)$. These dissimilarities can be arranged in a vector $(d_1, \ldots, d_m)$. In this way, we can transform any graph of the training set T into an m-dimensional vector using the prototype set P.

We perform the graph embedding step according to this definition, but we let the training set T and the prototype set P be the same, i.e, the set S for which the median graph is to be computed. So, we compute the distance between every pair of graphs in the set S. These distances are arranged in a distance matrix. Each row/column of the matrix can be seen as an n-dimensional vector representing one of the graphs in S. Figure 2 illustrates this procedure. In this example, it is assumed that the first row in the matrix corresponds to the distances of the black graph in the set to all other graphs in S.

At the end, each graph has a corresponding point (n-dimensional vector) in the vector space. What is important to remark here is the meaning of each

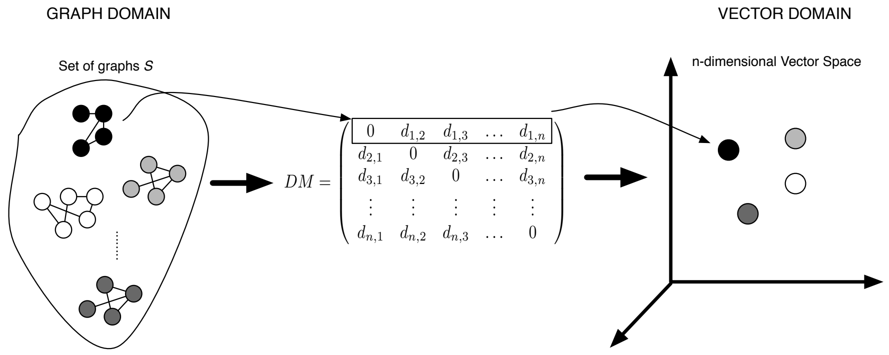


Fig. 2. Illustration of the first step (graph embedding)

position in this vector. If a vector $\boldsymbol{v}_i$ corresponds to the graph $g_i \in S$, then the coordinate j (with $j = 1 \ldots n$) of this vector is the distance from the graph g_i to the graph g_j, that is $d(g_i, g_j)$.

4.2 Median Vector Computation

The median vector is computed using the points obtained in the first step. Although we are in a Euclidean space, the nature of the cost function, in which only nodes insertions and deletions, each with cost 1 will occur [6], allows us to assume a L_1 metric can be used to compute the median vector. Using this metric, the median vector can be computed by means of the *Manhattan median*:

Definition 7 (Manhattan Median). *Given a set $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m\}$ of m points with $\boldsymbol{x}_i \in \mathbb{R}^n$ for $i = 1 \ldots m$, the Manhattan median is defined as*

$$Manhattan\ median = arg \min_{y \in \mathbb{R}^n} \sum_{i=1}^{m} |\boldsymbol{x}_i - \boldsymbol{y}| \tag{3}$$

where $|\boldsymbol{x}_i - \boldsymbol{y}|$ denotes the Manhattan distance between the points $\boldsymbol{x}_i, \boldsymbol{y} \in \mathbb{R}^n$.

The *Manhattan median* [11] is simply the median of each coordinate of the vector, which makes the *Manhattan median* computation really simple.

4.3 Back to the Graph Domain

The median vector is used to go back to the graph domain and obtain the median graph. Every component of the median vector represents the distance between the median graph and one of the graphs in the set S (see Figure 3). In this example, the median graph would have a distance of 2 to the graph $g_1 \in S$, a distance of 1 to the graph $g_2 \in S$, and so on.

Knowing the distance from the median to one graph in S, we can draw an interval in the Rhombus search space around this graph according to the distance.

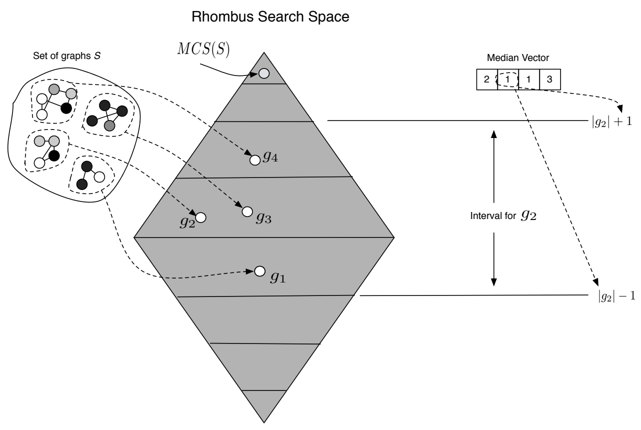


Fig. 3. Interpretation of the median vector

Since in the distance we are using only insertions and deletions of nodes, and these operations have a cost of 1, a distance k from the median to the graph g_i will imply that we would have to take into account graphs around g_i with size $|g_i| \pm k$, since a distance k will add (or remove) at most k nodes to g_i. Figure 3 shows a simple situation where the set S is composed of 4 graphs. For clarity, in this example only the interval for g_2 is shown.

Once the intervals are set, we choose only the smallest interval and its corresponding graph (this corresponds to the graphs 2 or 3 in Figure 3). The median is chosen as the graph with minimum SOD only in this interval. With this approach the search space is reduced to only this interval (grey part in Fig. 4).

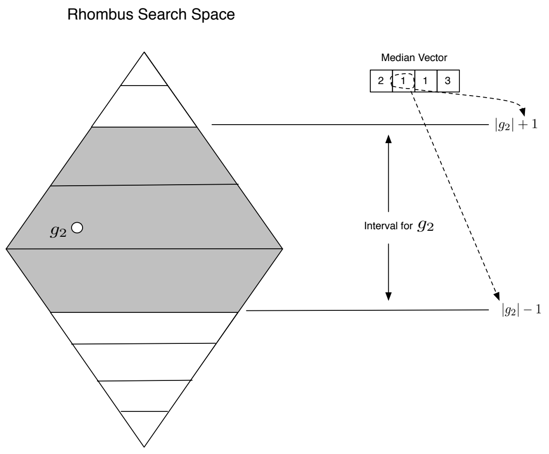
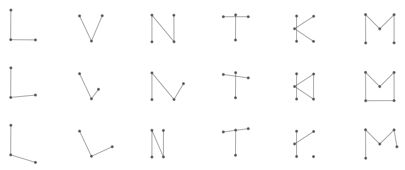


Fig. 4. Explored part of the search space (grey part)

5 Experimental Setup

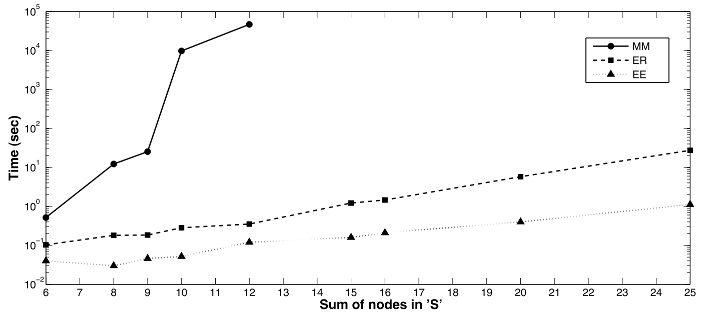
In this section, we will compare our algorithm (referred as EE) with the multimatch algorithm (referred as MM) and the exact algorithm presented in [3] (referred as ER). The experiments consisted of the computation of several generalized median graphs using different number of graphs in each computation. For each median, the elapsed time and the number of SOD computations needed to compute it were recorded. In these experiments, the graph dataset was composed of graphs that represent capital letters [10]. From the original set composed by 15 letters, we only used a subset of 6 letters L, V, N, T, K and M. Then, from each original model, we manually generated 4 distorted instances. Therefore, our dataset is composed of 30 elements and 6 classes. Each class represents a letter and contains 5 different instances of each letter. Table 5 shows the original letters in the first row and the four distorted letters in the rest of the rows. Notice that, in the distorted letters, some lines have been erased or moved their position, but the number of terminal point has been kept unchanged. The letters are represented by graphs as follows. The straight lines are represented by edges and the terminal points of the lines by the nodes. Nodes are labelled by a two-dimensional attribute that represents the position (x,y) of the terminal point. Edges have no attributes.

Table 1. Original models (first row), some distorted models (other rows)

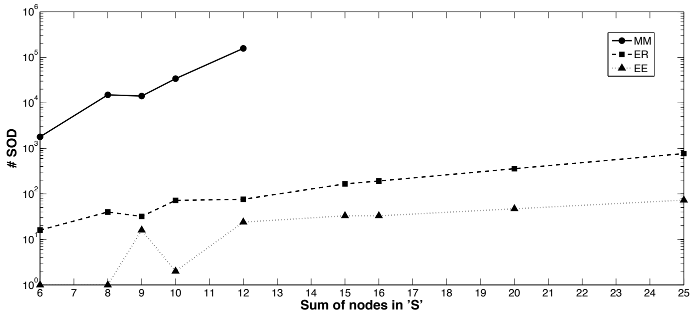


The results for both the computation time and the SOD computations required as a function of the total number of nodes in S are shown in Figures 5(a) and 5(b), respectively. Notice that the results are the mean values obtained for all sets S having the same number of nodes. Although the median vector computation yields the optimal median, the back-transformation is not proven to really give us a true median graph. However, in all our experiments we obtained optimal medians.

First of all it is important to remark that, due to the high computational requirements, the MM approach could only be applied to sets of graphs whose sum of nodes is 12, while the ER and EE approaches could be applied to all the sets. In addition, both figures show a substantial improvement of the EE and ER algorithms with respect to the MM approach. It is important to remark that the EE approach clearly outperforms the ER algorithm. The results show that while for the ER algorithm, the computation time for sets of graphs with 25 nodes is around 18 seconds, the EE algorithm only needs less than 3 seconds. Similar results can be observed for the number of SOD computations. For the



(a)



(b)

Fig. 5. Computation time (a) and number of SOD computations (b) as a function of the total number of nodes of the graphs in S

ER algorithm this number grows up to 500 SOD computations (for sets of graphs with 25 nodes), while for the EE algorithm it is only 75 in the same case.

As a final conclusion, we can say that with the embedding approach for the median graph computation we are able to obtain exact solutions for the median graph with relatively large sets of graphs with a low computation time. This suggests this method may extend the exact median graph computation to real (although limited) applications.

6 Conclusions

The median graph has been shown as a good choice to obtain a representative of a set of graphs, which has many potential applications in pattern recognition and related areas. Nevertheless, it suffers from a large complexity.

In this paper we have presented a new technique for the exact median graph computation based on graph embedding into vector spaces. This procedure is composed of three steps: the embedding of graphs into a real vector space, where each graph becomes a point in this space; the median vector computation using these points; the recovering of the median graph from the median vector. Although this technique can be used in many ways, we have particularized it to the exact median graph computation assuming the distance between graphs is carried out using the graph edit distance and a particular cost function. Under these conditions, the search space for the median graph computation can be drastically reduced and this technique is used to obtain exact solutions for the median graph.

Experiments made on a synthetic database show that our method clearly outperforms the previous existing methods for the exact median graph computation both in the computation time and the number of graph distances needed to obtain the median.

The proposed technique has permit us to extend the median graph computation to more realistic (although limited) sets of graphs. This situation opens the door to the application of the median graph to more complex problems, such as classification or clustering tasks.

References

1. Jiang, X., Münger, A., Bunke, H.: On median graphs: Properties, algorithms, and applications. IEEE Trans. Pattern Anal. Mach. Intell. 23(10), 1144–1151 (2001)
2. Münger, A.: Synthesis of prototype graphs from sample graphs. Diploma Thesis, University of Bern (in German) (1998)
3. Ferrer, M.: Theory and algorithms on the median graph. application to graph-based classification and clustering. PhD Thesis, Universitat Autònoma de Barcelona (2007)
4. Hlaoui, A., Wang, S.: Median graph computation for graph clustering. Soft Comput. 10(1), 47–53 (2006)
5. White, D., Wilson, R.C.: Mixing spectral representations of graphs. In: 18th International Conference on Pattern Recognition (ICPR 2006), Hong Kong, China, August 20-24, pp. 140–144. IEEE Computer Society, Los Alamitos (2006)
6. Bunke, H.: On a relation between graph edit distance and maximum common subgraph. Pattern Recognition Letters 18(8), 689–694 (1997)
7. Sanfeliu, A., Fu, K.: A distance measure between attributed relational graphs for pattern recognition. IEEE Transactions on Systems, Man and Cybernetics 13(3), 353–362 (1983)
8. Bunke, H., Allerman, G.: Inexact graph matching for structural pattern recognition. Pattern Recognition Letters 1(4), 245–253 (1983)
9. Schenker, A., Bunke, H., Last, M., Kandel, A.: Graph-Theoretic Techniques for Web Content Mining (Machine Perception and Artificial Intelligence) (Series in Machine Perception and Artificial Intelligence). World Scientific Publishing Co., River Edge (2005)
10. Riesen, K., Neuhaus, M., Bunke, H.: Graph embedding in vector spaces by means of prototype selection. In: Escolano, F., Vento, M. (eds.) GbRPR 2007. LNCS, vol. 4538, pp. 383–393. Springer, Heidelberg (2007)
11. Krause, E.F.: Taxicab Geometry: An Adventure in Non-Euclidean Geometry. Dover, New York (1986)