# Median Graph Computation by Means of Graph Embedding into Vector Spaces

**Miquel Ferrer, Itziar Bardají, Ernest Valveny, Dimosthenis Karatzas, and Horst Bunke**

## 1 Introduction

In pattern recognition [8, 14], a key issue to be addressed when designing a system is how to represent input patterns. Feature vectors is a common option. That is, a set of numerical features describing relevant properties of the pattern are computed and arranged in a vector form. The main advantages of this kind of representation are computational simplicity and a well sound mathematical foundation. Thus, a large number of operations are available to work with vectors and a large repository of algorithms for pattern analysis and classification exist. However, the simple structure of feature vectors might not be the best option for complex patterns where nonnumerical features or relations between different parts of the pattern become relevant.

In this context, graphs comprise an attractive alternative to represent complex and structured objects. One of the main advantages of graphs over feature vectors is that they can explicitly model the relations between the different parts of the

M. Ferrer (✉)
DAMA-UPC, Universitat Politècnica de Catalunya - BarcelonaTech, Barcelona, Spain
e-mail: mferrer@ac.upc.edu

I. Bardají
Institut de Robòtica i Informàtica Industrial, CSIC-UPC, Universitat Politècnica de Catalunya - BarcelonaTech, Barcelona, Spain
e-mail: ibardaji@iri.upc.edu

E. Valveny • D. Karatzas
Centre de Visió per Computador, Universitat Autònoma de Barcelona, Bellaterra, Spain
e-mail: ernest@cvc.uab.cat; dimos@cvc.uab.cat

H. Bunke
NICTA, Vistoria Research Laboratory, The University of Melbourne, Parkville, Victoria, Australia
e-mail: bunke@iam.unibe.ch

Y. Fu and Y. Ma (eds.), *Graph Embedding for Pattern Analysis*,
DOI 10.1007/978-1-4614-4457-2_3,

object, whereas feature vectors are only able to describe an object as an aggregation of numerical properties. In addition, graphs permit to associate any kind of label (not only numbers) to both edges and nodes. Furthermore, the dimensionality of graphs, i.e., the number of nodes and edges, can be different for every object. Thus, the more complex an object is, the larger the number of nodes and edges used to represent it can be. However, the main drawback of using graphs arises from the difficulty and complexity of computational manipulation. Even the simple task of comparing two graphs, which is commonly referred to as graph matching, turns out to be very complex under some conditions [5]. In addition, most mathematical operations required in many learning and classification algorithms are not possible in the graph domain.

In order to overcome these limitations graph embedding techniques have gained popularity recently. They can combine the strengths of both domains, that is the high representational power of graphs together with all the mathematical foundation and algorithms available for the feature vectors. Graph embedding [20] aims to convert graphs into real vectors and then operate in the associated vector space. Thus, it emerges as a powerful way to provide graph-based representations with access to the rich repository of algorithmic tools available in statistical pattern analysis [7,16]. To this end, different graph embedding procedures have been proposed in the literature so far. Some of them [4, 25, 33, 39, 42, 46] are based on the spectral graph theory. Graphs are converted into a vector representation using some spectral features extracted from the adjacency or the Laplacian matrix of a graph. Another family of graph embedding approaches is based on the similarity between graphs. For instance, in [22] graph features are extracted out of the dissimilarity matrix among a set of graphs. Alternatively, another embedding inspired in the work proposed in [31] is presented in [38]. Given a set of some a priori selected graph prototypes each point is embedded into a vector space by taking the distance (in this case the graph edit distance is used) to all these prototypes. The basic intuition of this work is that the description of the regularities in observations of classes and objects is the basis to perform pattern classification. Thus, assuming that the prototypes have been chosen appropriately, each class will form a compact zone in the vector space. Finally, there is a family of embedding approaches based on computing frequencies of certain substructures of the graphs [15, 24].

Graph embedding permits to go from the graph domain to the vector domain. The reverse problem, going from the vector space back to the graph space, is not so easy since it implies recovering the structural information that is usually lost with the embedding. The ability to translate a vectorial result calculated in the embedding space back to a graph is a condition sine qua non for linking statistical and structural pattern recognition through graph embedding.

Formally, the median graph [21] is defined as the graph that has the minimum sum of distances (SOD) to all graphs in a given set. Thus, it can be taken as the representative of the set and, therefore, it has a large number of potential applications including classical algorithms for learning, clustering, and classification that are normally used in the vector domain. As a matter of fact, 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 [21]. The only exact algorithm proposed up to now [27] is based on an $A^*$ algorithm using a data structure called multimatch. As the computational cost of this algorithm is very high, a set of approximate algorithms have also been presented in the past based on different approaches such as genetic search [21,27], greedy algorithms [19], and spectral graph theory [11,45]. However, all these algorithms can only be applied to restricted sets of graphs, regarding either the type or the size of the graphs.

Graph embedding can help in finding more efficient methods to compute the median graph and has already been explored in the past [12, 13]. The hypothesis underlying these works is that embedding the graphs into vectors and computing the median in the vector space can lead to a corresponding graph in the graph space is a good approximation of the median graph. Therefore, this procedure relies on some method to reconstruct the graph corresponding to the median vector. In these previous works [12, 13] an approximate reconstruction is obtained using some heuristics that permit to recover a graph based on the concept of the weighted mean of a pair of graphs using the median vector and the graphs corresponding to the 2 or 3 closest points to it.

In this chapter we present a generic procedure to convert a point in a vector space into a graph, given that we have a set of $n$ graphs that have been previously embedded in the vector space (with $n$ being the dimension of the vector space) and that the point we want to convert lies inside the convex hull of such embedded vectors, which is for example the case with the median and barycenter of the set of points. The basic idea of this approach is as follows. Given $n$ graphs mapped to their corresponding points in the $n$-dimensional real space and a point inside the convex hull of such set a of points, we iteratively project the point into subspaces of lower dimensionality until a projected point is obtained lying on a line that connects the maps of two graphs of the given set. The graph corresponding to this point can be approximately reconstructed by means of the weighted mean. Next, we recursively consider all other projected points obtained before in higher dimensional spaces and apply the same reconstruction principle until the graph corresponding to the desired point is obtained. Ideally, this procedure would permit to recover the graph that corresponds exactly to the input point. However, due to the complexity of graph matching problems, we are forced to use approximate algorithms at different steps and therefore we will only be able to obtain partial approximations of the input point.

We show the application of this procedure to the computation of the median graph. As in the previous works, the median graph is obtained after embedding a set of graphs into a vector space, computing the median of such a set in the vector domain and then recovering the graph corresponding to median vector. For this last step, the proposed generic procedure is used. The application of this procedure raises some considerations about the order in which graphs have to be taken along the procedure. In this sense, we analyze four additional variations of the method which take into account different sorting schemes of the original set of graphs. These variations can help to understand the influence of the approximations assumed

across the procedure. It is also important to remark that this generic framework could be potentially used in conjunction with any embedding technique and with any method to compute the representative of the set in the vector space.

In order to test the feasibility of applying this procedure to the computation of the median graph, we evaluate the final SOD of the median graph obtained to all the graphs of the set as a measure of the quality of median graph. We have also made clustering experiments on three different graph databases, one semi-artificial and two containing real-world data. In these clustering experiments, the proposed algorithms for the median graph computation are used to obtain the centers of the clusters. The underlying graphs have no constraints regarding the number of nodes and edges. The results are evaluated, according to four different clustering measures, namely, the Rand index, the Dunn index, the bipartite index, and the mutual information index. We will show that the clusters obtained using the proposed method are better than those using the set median graph or previous approaches also based on graph embedding.

The rest of this chapter is organized as follows. In the next section we define the basic concepts and we introduce the notation we will use later in the chapter. Then, in Sect. 3 the proposed generic method for recovering a graph from a point in the vector space is described. After that, Sect. 4 presents the practical implementation of the proposed generic framework for the computation of the median graph. Section 5 reports a number of experiments and presents the results achieved with our method. Also a comparison with several reference systems is provided. Finally, in Sect. 6 we draw some conclusions and we point out to possible future work.

## 2 Basic Concepts

This section introduces the basic terminology and notation we will use throughout the chapter.

### *2.1 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, $\mu : V \longrightarrow L$ is the node labeling function, and $\nu : V \times V \longrightarrow L$ is the edge labeling function. The alphabet of labels is not constrained in any way. For example, $L$ can be defined as a vector space (i.e., $L = \mathbb{R}^n$) or simply as a set of discrete labels (i.e., $L = \{\Delta, \Sigma, \Psi, \ldots\}$). Edges are defined as ordered pairs of nodes, i.e.,, an edge is defined by $(u, v)$ where $u, v \in V$. The edges are directed in the sense that if the edge is defined as $(u, v)$ then $u \in V$ is the source node and $v \in V$ is the target node.

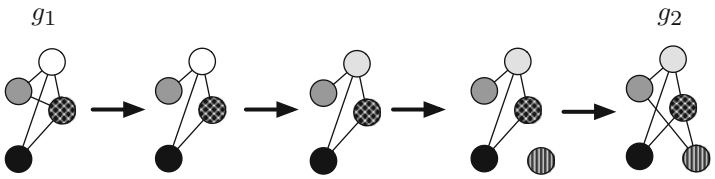


**Fig. 1** A possible edit path between two graphs $g_1$ and $g_2$. Note that node labels are indicated by different colors

## 2.2 Graph Edit Distance

The process of evaluating the structural similarity of two graphs is commonly referred to as graph matching. This issue has been addressed by a large number of works. For an extensive review of different graph matching methods and applications, the reader is referred to [5]. In this work, we will use the graph edit distance [2, 40], one of the most widely used methods to compute the dissimilarity between two graphs.

The basic idea behind the graph edit distance [2, 40] is to define the dissimilarity of two graphs as the minimum amount of change required to transform one graph into the other. To this end, a number of 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 $g_2$ (see Fig. 1 for example). In general, several edit paths may exist between two given graphs. This set of edit paths is denoted by $\wp(g_1, g_2)$. To evaluate which edit path is the best one, edit costs are introduced through a cost function. The basic idea is to assign a cost $c(e)$ to each edit operation according to the amount of distortion it introduces in the transformation. Then, the edit distance between two graphs $g_1$ and $g_2$, denoted by $d(g_1, g_2)$, is the minimum cost edit path over all edit paths that transform $g_1$ into $g_2$:

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

Different optimal and approximate algorithms for the computation of the graph edit distance have been proposed so far. Optimal algorithms are usually based on combinatorial search procedures that explore all the possible mappings of nodes and edges of one graph to the nodes and edges of the second graph [40]. The major drawback of such an approach is its computational complexity, which is exponential in the number of nodes of the involved graphs. As a result, the application of these methods is restricted to graphs of rather small size in practice. As an alternative, a number of suboptimal methods have been proposed to make the graph edit distance less computationally demanding and therefore usable in real applications. Some of

these methods are based on local optimization [28]. A linear programming method to compute the graph edit distance with unlabeled edges is presented in [23]. Such a method can be used to obtain lower and upper edit distance bounds in polynomial time. In [29] simple variants of the standard method are proposed to derive two fast suboptimal algorithms for graph edit distance, which make the computation substantially faster. Finally, a new efficient algorithm is presented based on a fast suboptimal bipartite optimization procedure [36]. We will use these two last approximate methods for the graph-edit distance computation.

In [2] it was shown that $d(g_1, g_2)$ is a metric if the underlying cost function is a metric. Under the approximation algorithms of [29, 36] used in this work, however, the metric property is no longer guaranteed. But this does not have any negative impact on the approach proposed in this work because, firstly, the embedding procedure that maps each graph onto an $n$-dimensional vector can be applied regardless if the underlying distance function is a metric or not [30] and, secondly, after embedding all points, which represent the graph, are located in a Euclidean (and in particular a metric) space.

## 2.3 Weighted Mean of a Pair of Graphs

For the purpose of median graph computation, the weighted mean of a pair of graphs [3] is a crucial tool. For this reason we include its definition in the following.

Let $g$ and $g'$ be two graphs. The *weighted mean* of $g$ and $g'$ is a graph $g''$ such that

$$d(g, g'') = a$$
$$d(g, g') = a + d(g'', g')$$

That is, the graph $g''$ is a graph in between the graphs $g$ and $g'$ along the edit path between them. Furthermore, if the distance between $g$ and $g''$ is $a$ and the distance between $g''$ and $g'$ is $b$, then the distance between $g$ and $g'$ is $a+b$. Figure 2 illustrates this idea.

Observe that $g''$ is not necessarily unique. Consider, for example, a graph $g$ consisting of only a single node with label $A$ and a graph $g'$ consisting of three isolated nodes labeled with $A$, $B$, and $C$, respectively. Assume that the insertion and deletion of a node has a cost equal to 1, regardless of the label of the affected node. Then we have $d(g, g') = 2$. Obviously, for $a = 1$ there exist two 1-mean graphs: $g_1$, which consists of two isolated nodes, one with label $A$ and the other with label $B$, and $g_2$, which also consists of two isolated nodes, one with label $A$ and the other with label $C$. In this work, we will assume that two graphs that are at the same distance from the original graphs are equivalent.

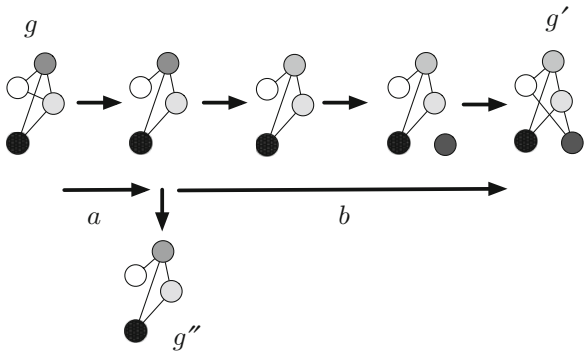


**Fig. 2** Weighted mean of a pair of graphs

## 2.4 Median Graph

Given $L$, a finite alphabet of labels for nodes and edges, let $U$ be the set of all graphs that can be constructed using labels from $L$. Given $S = \{g_1, g_2, \ldots, g_n\} \subseteq U$, the generalized median graph $\bar{g}$ of $S$ is defined as

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

That is, the generalized median graph $\bar{g}$ of $S$ is a graph $g \in U$ that minimizes the SOD (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 $S$. It can be seen as the representative of the set. Consequently, it can be potentially used by any graph-based algorithm where a representative of a set of graphs is needed.

Despite its simple mathematical definition (2), the computation of the median graph is extremely complex. As shown in (2) some distance measure $d(g, g_i)$ between the candidate median $g$ and every graph $g_i \in S$ must be computed. However, since the computation of the graph edit distance is a well-known NP-complete problem, the computation of the generalized median graph can only be done in exponential time, both in the number of graphs in $S$ and their size (even in the special case of strings, the time required is exponential in the number of input strings [18]). As a consequence, in real applications we are forced to use suboptimal methods in order to obtain approximate solutions for the generalized median graph in reasonable time. Such approximate methods [11, 19, 21, 27, 45] apply some heuristics in order to reduce the complexity of the graph edit distance computation and the size of the search space. Two different approaches that use graph embedding have already been explored in the past [12, 13]. The basic idea underlying these works is that embedding the graphs into vectors and computing the median in the vector space can lead to a median vector whose corresponding graph in the graph space can be a good approximation of the median graph. In these previous works [12, 13] an approximate reconstruction is obtained using, respectively, the 2 or 3

closest points to the median vector. We will use these methods (referred as E2P and E3P) as reference embedding methods for comparison later in the experiments.

Another alternative to reduce the computation time is to use the set median graph instead of the generalized median graph. The difference between the two concepts is only the search space where the median is looked for. As it is shown in (2), the search space for the generalized median graph is $U$, i.e., the whole universe of graphs. In contrast, the search space for the set median graph is simply $S$, i.e., the set of graphs in the given set. It makes the computation of set median graph exponential in the size of the graphs, due to the complexity of graph edit distance, but polynomial with respect to the number of graphs in $S$, since it is only necessary to compute pairwise distances between the graphs in the set. The set median graph is usually not the best representative of a set of graphs, but it is often a good starting point towards the search of the generalized median graph. As a matter of fact, we will use the set median graph as a baseline for the experiments presented later.

## 3 From Vectors to Graphs

In this section we will propose a generic procedure to compute the graph corresponding to a point in the vector space associated to a particular graph embedding. For this procedure to be applied we require to know the points corresponding to the embedding of a set $S = \{g_1, g_2, \ldots, g_n\}$ of $n$ graphs, where $n$ is the dimension of the vector space. Theoretically, any embedding could be used, as long as the distance relationships in the graph space are maintained in the vector space. In reality, this depends on the embedding method used, and it is only approximately true.

Once we have the set of $n$ points corresponding to the embedding of $S$, the procedure can be applied to recover any point $M$ lying inside the convex hull defined by these $n$ points. It is based on recursively projecting the point $M$ into hyperplanes of decreasing dimensionality and recovering the graph corresponding to the projected points by means of the weighted mean of a pair of graphs. It is important to note that all geometric operations needed in the reconstruction are carried out in the $n$-dimensional real space using the Euclidean distance. Hence, all the operations take place in a metric space. Thus, if we were able to compute the exact edit distance and the optimal edit path between two graphs, we would be able to obtain the graph that corresponds to the original point $M$. However, we are forced to use several approximations in practice. As a result we will only be able to obtain approximations of the corresponding graph.

Let us introduce some important aspects before explaining the method.

1. Given a set of $n$ linearly independent points in $\mathbb{R}^n$ we can define a hyperplane $H_{n-1}$ of dimensionality *n-1* (e.g. in the case of *n=2*, two points define a unique 1D line, in the case of *n=3*, three points define a unique 2D plane, etc.). See Fig. 3 for example.

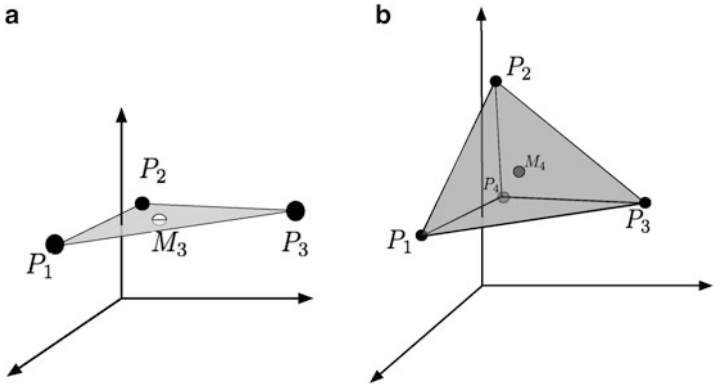


**Fig. 3** (**a**) The 2D-hyperplane $H_2$ is defined by the 3 points $P_i = \{P_1, P_2, P_3\}$. The Euclidean median $M_3$ falls in the 2D space defined by the 3 points and specifically within the triangle (2D simplex) with vertices $P_i$ ($i = 1, \ldots, 3$). (**b**) The 3D-hyperplane $H_3$ is defined by the 4 points $P_i = \{P_1, P_2, P_3, P_4\}$. The Euclidean median $M_4$ falls in the 3D space defined by the 4 points and specifically within the pyramid (3D simplex) with vertices $P_i$ ($i = 1, \ldots, 4$)

2. Assume that we can define a line segment in the vector space that connects two points $P_1$ and $P_2$ corresponding to known graphs $g_1$ and $g_2$, such that the point $M$ in a $2D$ space $M_2$ lies on this line segment. We can then calculate the graph $g_{M_2}$ corresponding to the point $M_2$ as the weighted mean of $g_1$ and $g_2$.[1]

From the second point we can observe that, given $n$ embedded points $\{P_1, P_2, \ldots, P_n\}$ and the original point $M_n$, in order to obtain the graph corresponding to $M_n$, the problem is to find two points in the vector space, whose corresponding graphs are known, such that the median $M_n$ lies on the line defined by these two points. In this way, we can then apply the weighted mean of these two points in order to find the graph corresponding to $M_n$. In the following we will describe how we can obtain such two points and, thus, such a graph. We will illustrate this procedure with the example shown in Fig. 4 with four points. Figure 4a shows the four points $\{P_1, P_2, P_3, P_4\}$ and the original point $M_4$.

Given $P_1, P_2, \ldots, P_n$, we can choose without loss of generality, any one of them, say $P_n$, and create the vector $(\mathbf{P_n} - \mathbf{M_n})$ (vector $(\mathbf{P_4} - \mathbf{M_4})$ in Fig. 4b). This vector will lie fully on the hyperplane $H_{n-1}$ defined by these $n$ points. Then, if we call $H_{n-2}$ the hyperplane of dimensionality $n-2$ defined by the set of the remaining $n-1$ points $\{P_1, P_2, \ldots, P_{n-1}\}$, i.e., all the original points except $P_n$, then the intersection of the line defined by the vector $(\mathbf{P_n} - \mathbf{M_n})$ and the new hyperplane $H_{n-2}$ will be a single point. We will call this new point $M_{n-1}$ ($M_3$ in Fig. 4b which lies on the hyperplane $H_2$ (plane) defined by $P_1$, $P_2$, and $P_3$).

[1] For clarity, in the remainder, we will refer to the projection of $M$ into $n$-dimensional space as $M_n$.

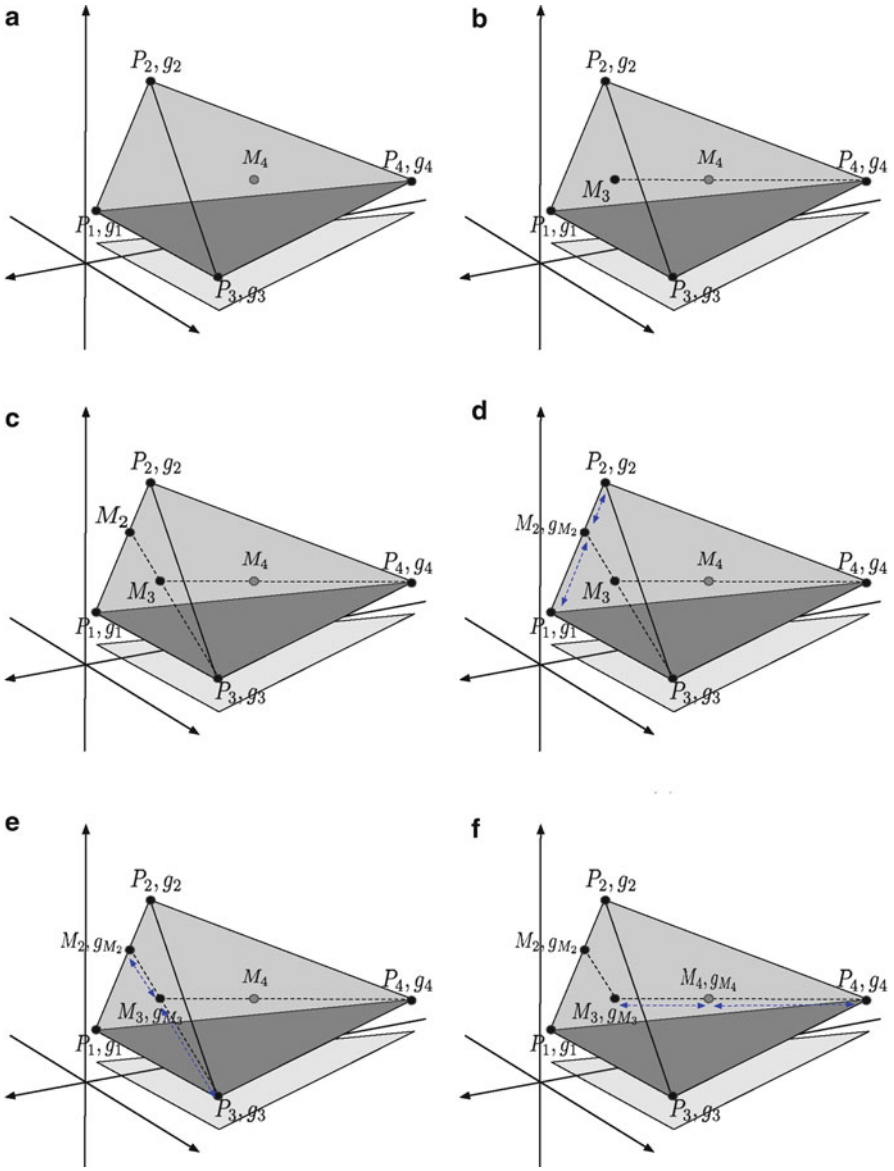


**Fig. 4** Complete example of the median recovering with four points $\{P_1, P_2, P_3, P_4\}$

As mentioned before, in order to use the weighted mean of a pair of graphs to calculate the graph corresponding to $M_n$, we need to first find a point (whose corresponding graph is known) that lies on the line defined by the vector $(\mathbf{P_n} - \mathbf{M_n})$ and specifically on the ray extending $M_n$ (so that $M_n$ lies between $P_n$ and the new point). Now we have two points ($P_n$ and $M_{n-1}$) and the original point $M_n$

falling on the line defined by them. However, although we already know the graph corresponding to the point $P_n$ ($P_n$ comes from the graph $g_n$), we do not know yet the graph corresponding to the point $M_{n-1}$. Therefore, we cannot apply the weighted mean to find the graph corresponding to $M_n$. However, we can follow exactly the same procedure as before and consider a new line defined by the vector $(\mathbf{P_{n-1}} - \mathbf{M_{n-1}})$ ($(\mathbf{P_3} - \mathbf{M_3})$ in Fig. 4c). Again, as we did for $M_{n-1}$, we can define the point of intersection of the above line with the $n-3$ dimensional hyperplane $H_{n-3}$ which is defined by the $n-2$ remaining points $\{P_1, P_2, \ldots, P_{n-2}\}$. Then, we will get a new point $M_{n-2}$ ($M_2$ in Fig. 4c which lies on the line defined by points $P_1$ and $P_2$).

This process is recursively repeated until $M_2$ is obtained. The case of $M_2$ is solvable using the weighted mean of a pair of graphs, as $M_2$ will lie on the line segment defined by $P_1$ and $P_2$ which correspond to the known graphs $g_1$ and $g_2$ (we obtain $g_{M_2}$ corresponding to $M_2$ in Fig. 4d).

Having calculated the graph $g_{M_2}$ corresponding to the point $M_2$, the inverse process can be followed all the way up to $M_n$. Once $g_{M_2}$ is found, in the next step, the graph $g_{M_3}$ corresponding to $M_3$ can be calculated as the weighted mean of the graphs corresponding to $M_2$ and $P_3$ (Fig. 4e). Generally the graph $g_{M_k}$ corresponding to the point $M_k$ will be given as the weighted mean of the graphs corresponding to $M_{k-1}$ and $P_k$. The weighted mean algorithm can be applied repeatedly until the graph $g_{M_n}$ corresponding to $M_n$ is obtained ($g_{M_4}$ in Fig. 4f).

In this procedure we claim that the method to recover the graph from a vector should permit to obtain the exact graph in case that:

- The embedding preserves the distance structure.
- We were able to perform exact computations of the graph edit distance.

In general, these two conditions are not easy to satisfy. Concerning the first condition, the procedure simply requires that the edit path between two graphs follows a path along the straight line joining the two corresponding vectors in the vector space. Although there are some cases where using the selected embedding procedure this can be shown to be true, in general, it is not always satisfied. Regarding the second condition, the exact computation of the edit distance is a well-known NP-problem. So, we are forced to use some approximation. For these reasons, we are only able to get approximations of the graph corresponding to the point.

## 4 Median Graph Computation

In this section we will make use of the procedure to recover a graph from an embedded vector to propose a generic procedure to compute the median graph via embedding. In our case, the embedding of graphs into points in a suitable vector space will permit us to carry the median computation in the vector domain instead

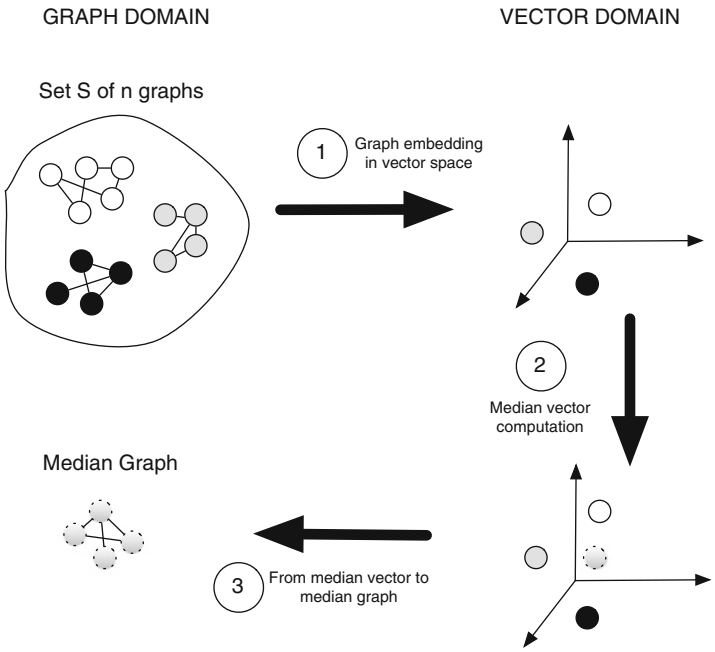


**Fig. 5** Overview of the approximate procedure for median graph computation

of performing this operation in the graph domain, which is considerably more complex. This generic procedure we present is composed by three main steps (see Fig. 5).

- Given a set $S = \{g_1, g_2, \ldots, g_n\}$ of $n$ graphs, the first step is to embed every graph in $S$ into the real $n$-dimensional space. That is, each graph in $S$ becomes a point in $\mathbb{R}^n$. Theoretically, any embedding which fulfils this condition, i.e., each graph becomes an $n$-dimensional point, could be used in this step. However, it is expected to obtain better results if the distance relationships resemble as much as possible both in the original graph space and the vector space.
- The second step consists of computing a representative of the set in the vector space. As in the case of the first step, several solutions could be applied here. However, the median vector $\mathbf{M}$ arises as a natural solution [13], since it is the vectorial counterpart of the median graph but in the vector domain: given a set $\wp = \{P_1, P_2, \ldots, P_m\}$ of $m$ points with $P_i \in \mathbb{R}^n$ for $i = 1, \ldots, m$, the median vector is a point $M_n \in \mathbb{R}^n$ that minimizes the sum of the distances to all the points in $\wp$. Thus, if the embedding preserves the distance structure of the graph domain, the median vector should be a good representation of the median graph in the vector space.
- Finally, the resulting median vector has to be mapped back to a corresponding graph. For this, we will use the procedure described in the previous section.

It is important to notice that the three above mentioned steps are generic and independent of each other. That means that different approaches and solutions can be used in each of them and combined. In the following section we will propose a particular implementation of each step in order to compute the median graph. We explain the choices we have made for the first two steps, which are basically the same as in [13], and we discuss some practical considerations about the application of the recursive procedure to recovering the median graph from the median vector in the last step.

### *4.1 Step I: Graph Embedding*

In our proposal, we will use a class of graph embedding procedures based on the selection of some prototypes and graph edit distance computation. This approach was first presented in [37], and it is based on the work proposed in [31]. The basic intuition of this work is that the description of the regularities in observations of classes and objects is the basis to perform pattern classification. Thus, based on the selection of concrete prototypes, each point is embedded into a vector space by taking its distance to all these prototypes. Assuming these prototypes have been chosen appropriately, each class will form a compact zone in the vector space. For the sake of completeness, we briefly describe this approach in the following.

Assume we have a set of training graphs $T = \{g_1, g_2, \ldots, g_n\}$ and a graph dissimilarity measure $d(g_i, g_j)$ $(i, j = 1, \ldots, n; g_i, g_j \in T)$. 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 dissimilarity between a given graph $g \in T$ 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 $P$.

For our purposes, given a set of graphs $S = \{g_1, g_2, \ldots, g_n\}$, we use the graph embedding method described above to obtain the corresponding $n$-dimensional points $\{P_1, P_2, \ldots, P_n\}$ in $\mathbb{R}^n$. However, in our case, we set $P = S$, i.e., we avoid the problem of selecting a proper subset $P \subseteq S$ of prototypes and use the whole set of graphs.

It is important to mention that, as long as there are no identical graphs in the set $S$, the vectors $\mathbf{v_i} = (\mathbf{P_i} - \mathbf{O})$, where $O$ is the origin of the $n$-dimensional space defined, can be assumed to be linearly independent. This arises from the way the coordinates of the points were defined during graph embedding (Fig. 6).
An important relation that has been shown in [37] is

$$\| \phi(g) - \phi(g') \| \leq \sqrt{n} \cdot d(g, g') \tag{3}$$

where $\phi(g)$ and $\phi(g')$ denote the mappings in the vector space of graphs $g$ and $g'$, respectively, after embedding. That is, the upper bound of the Euclidean distance of

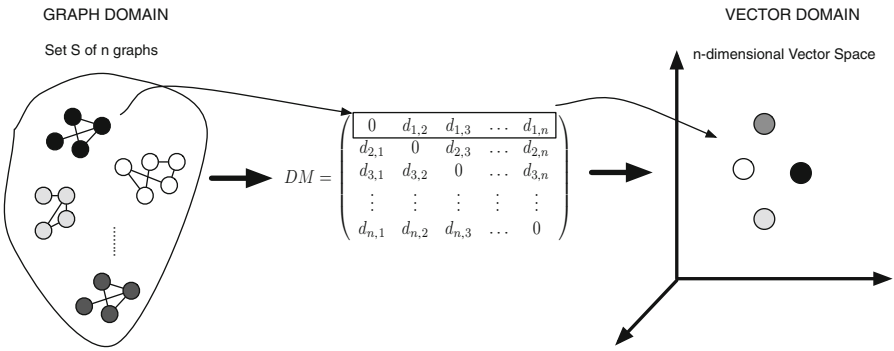


**Fig. 6** Step 1. Graph embedding

a pair of graph maps $\phi(g)$ and $\phi(g')$ is given by $\sqrt{n} \cdot d(g, g')$. In other words, if $g$ and $g'$ have a small distance in the graph domain, they will have a small distance after embedding in the Euclidean space as well.

Therefore, at the end of this first step we will have a collection of points in an $n$-dimensional space, each of them corresponding to one of the original graphs.

## *4.2 Step II: Median Vector Computation*

To obtain the representative of the set in the vector domain, we will use the concept of median vector as we already commented at the beginning of Sect. 3.

The median vector cannot be calculated in a straightforward way. The exact location of the median vector can not be found when the number of elements in $\wp$ is greater than 5 [1]. No algorithm in polynomial time is known, nor has the problem been shown to be NP-hard [17]. In this work we will use the most common approximate algorithm for the computation of the median vector, i.e., Weiszfeld's algorithm [44]. It is a form of iteratively re-weighted least squares that converge to the median vector. To this end, the algorithm first selects an initial estimate solution $M'_{n_0}$ (this initial solution is often chosen randomly). Then, the algorithm defines a set of weights that are inversely proportional to the distances from the current estimate $M'_{n_i}$ to the samples $x$ and creates a new estimate $M'_{n_i+1}$ that is the weighted average of the samples according to these weights.

$$M'_{n_{i+1}} = \frac{\sum_{j=1}^{m} \frac{x_j}{\|x_j - M'_{n_i}\|}}{\sum_{j=1}^{m} \frac{1}{\|x_j - M'_{n_i}\|}} \tag{4}$$

The algorithm may finish when a predefined number of iterations are reached, or under some other criteria, such as that the difference between the current estimate and the previous one is less than a predefined threshold.

Note that the median vector will always fall within the volume of the $n-1$ dimensional simplex whose vertices are the set of points used to compute the median. Thus, it fulfills the required constraint to use the recursive procedure to compute the graph associated to that point. Figure 3 shows an example for $n = 4$ and $n = 3$.

### *4.3 Step III: Median Graph Recovering*

We propose to obtain the graph corresponding to the median vector by means of the recursive application of the weighted mean of a pair of graphs. As it has already been remarked in the previous section, this recursive procedure relies on a set of approximations concerning the metric space and the computation of the graph edit distance, which will result in the calculation of an approximation of the median graph.

In order to analyze the effect of all these approximations in the final result, we can examine the order in which points $P_i$ in the vector space are considered in the recursive procedure. This is an issue not defined in the original procedure as, if computations were exact, the order would not matter. However, in case of approximate computations, the order can be important for the final solution. For instance, if we start the process of recovering the median graph using the points that are further from the optimal solution to define the connecting line in the vector space, we will probably start introducing some approximation errors in the first steps as the quality of the weighed mean is better the shortest the edit path is. However, in the final steps we will consider the points that are closer to the optimal solution and thus, we will probably balance this effect as we will give more weight to these points in the final solution. If we take the reverse order the expected effect would be the contrary. The final result of these opposite effects is not clear.

Therefore, we have defined different sorting schemes to consider the points in the recursive procedure according to the SOD of every point, calculated either in the graph or the vector domain, to the rest of points. Points with a low SOD will correspond to points close to the optimal solution.Thus, we present four variants of the basic recursive scheme presented in Sect. 3 (BRS in short), which include a preprocessing to sort the graphs. Note that to be consistent with the notation and the explanations performed in Sect. 3, the words *ascending* or *descending* used in the following refer to graphs from $g_n$ to $g_1$. These sorted schemes will be referred as SRS (sorted recursive schemes).

- *Graph-domain-based Sorted Recursive Scheme in descending order* (GSRSD): In this approach, the graphs are ordered in descending order, taking into account

the SOD to the rest of the graphs in $S$ of each of them. Consequently, $g_n$ is the graph with maximum SOD and $g_1$ is the set median graph. Under this sorting, the graph corresponding to the point $M_2$ is calculated as the weighted mean of $g_1$ and $g_2$, the two graphs with lowest SOD, i.e., the set median ($g_1$) and the next one in terms of the minimum SOD to $S$ ($g_2$).

- *Graph-domain-based Sorted Recursive Scheme in ascending order* (GSRSA): This sorting is the inverse to the previous one. The graphs are ordered upwards, based on the SOD. This way, the graph corresponding to $M_2$ is obtained from the two graphs with maximum SOD, and the graph corresponding to $M_n$ is obtained from the weighted mean between the graphs corresponding to $M_{n-1}$ and $g_n$ (the set median).
- *Vector-domain-based Sorted Recursive Scheme in descending order* (VSRSD): Here the criterion for the ordering is still the SOD, but it is evaluated in the Euclidean space. That is, the SOD of each of the points $\{P_n, \ldots, P_1\}$ to the other points of the set. In this case, $g_n$ is the graph, the corresponding point of which has the maximum SOD,

$$P_{\max} = \arg \max_{P \in \{P_n, \ldots, P_1\}} \sum_{i=1}^{n} \|P_i - P\|.$$

- *Vector-domain-based Sorted Recursive Scheme in ascending order* (VSRSA): As before, in this last sorting, the SOD in the Euclidean space is used to sort the points. The points are ordered upwards with respect to the SOD, such that the first two points used to compute the weighted mean are those with maximum SOD.

In addition note that, given $n$ graphs, in the procedure to recover the median graph we obtain $n-1$ intermediate graphs (from $M_2$ to $M_n$). As we go through the process we get closer to the graph corresponding to the median vector. But, at the same time, at every step we are also introducing more approximation in the final solution. As a result, it could happen that some of the intermediate graphs has an SOD better than the final median graph. Given this situation, we have also analyzed the SOD of these intermediate graphs.

In order to see the differences along these five recursive schemes (BRS and the four variations) we computed several medians using the letter dataset [34]. In this dataset, we consider the 15 capital letters of the Roman alphabet that consist of straight lines only (A, E, F, H, I, K, L, M, N, T, V, W, X, Y, Z). For each class, a prototype line drawing is manually constructed. These prototype drawings are then converted into prototype graphs by representing lines by undirected edges and ending points of lines by nodes. Each node is labeled with a two-dimensional attribute giving its position relative to a reference coordinate system. Edges are unlabeled (see [34] for more characteristics of this dataset). More concretely, we took sets of 50 and 100 elements randomly from the dataset and we computed the median with each of the methods. Figure 7 shows the evolution of the SOD of the intermediate median graphs for each recursive method. The $x$-axis represents the

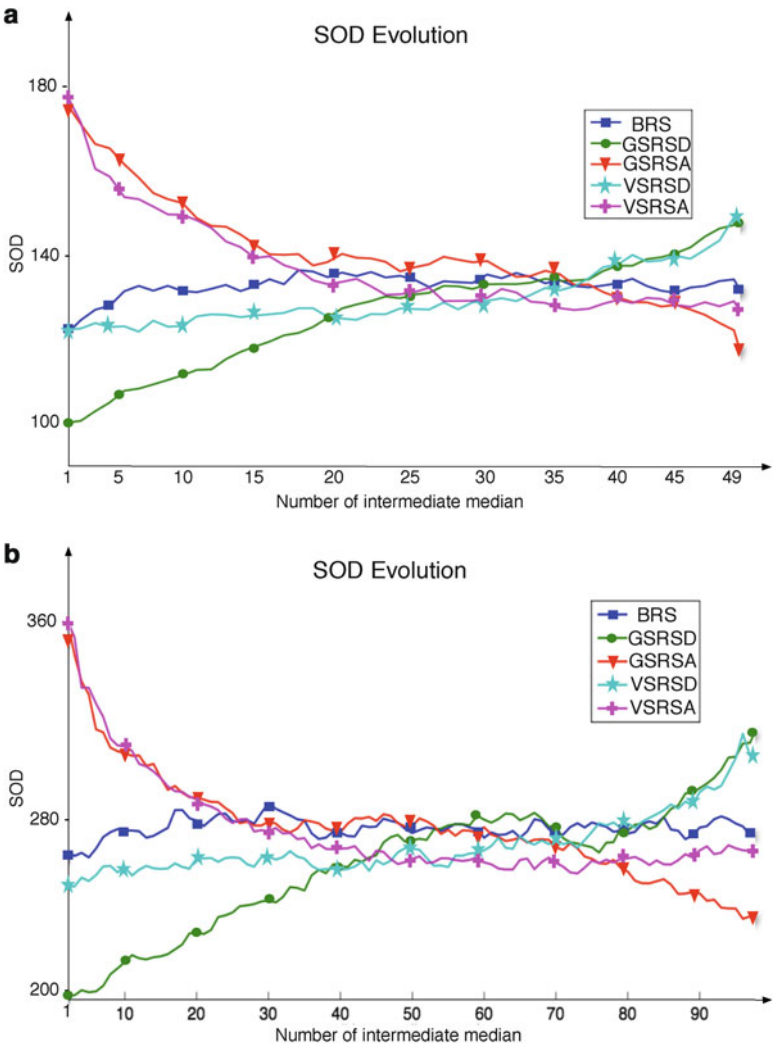


**Fig. 7** SOD evolution for the letter dataset using (**a**) sets of 50 elements and (**b**) sets of 100 elements

recursive level, being 1 for the first graph we obtain (i.e., $M_2$), and the last point representing the last final median (i.e., $M_n$). The $y$-axis represents the SOD of each corresponding intermediate graph. Results are the mean over ten repetitions for each size of the set.

First of all, as expected, it is important to note that the results are different for each of the five recursive schemes. As it can be seen in Fig. 7, the evolution of the SOD shows different behavior depending on the initial sorting. However, while the BRS approach shows a random-like behavior (there is no clear tendency in the

evolution of the SOD), the sorted schemes show a general tendency in the SOD evolution. Note also that this tendency is independent of the size of the set used to compute the median. One of the most striking facts is that the domain on which the sorting is based is unimportant. That is, in the descending methods (GSRSD and VSRSD), there is a clear tendency in starting with graphs or vectors with lower SODs and terminate with higher SODs. This fact can be explained because in the descending methods, the first intermediate graph (i.e., $M_2$) is computed using graphs having lower SOD (in the case of GSRSD method, $M_2$ is computed with the set median and the next graph in terms of the lower SOD). Consequently, $M_2$ has a low SOD. Then, as we compute more intermediate graphs, they are computed using graphs with higher SODs. This translates into a degradation in terms of the SOD in the intermediate graph. On the contrary, in the ascending schemes (GSRSA and VSRSA) the tendency in the evolution is exactly complementary. Here, we start with graphs having high SODs (and consequently $M_2$ has a high SOD) and then we use better graphs in terms of their SOD. This translates to a decreasing curve. As a conclusion, we can state that we get better solutions as we consider points that are closer to the optimal solution. However, the behavior of the two sorting schemes is not completely complementary in the sense that the loss in terms of SOD in the descending methods is not the same as the gain obtained in the ascending methods. For this reason, the minimum (or maximum) values of SOD in these evolutions differ. However, the fact that the tendency is kept regardless of the domain of the sorting supports the idea that relative distances are well conserved after mapping graphs into points in the particular embedding considered here.

Another important observation is that if we analyze the SOD of the intermediate graphs we can find intermediate solutions along the recursive path with a lower SOD than the final solution. This fact validates our previous hypothesis that there is a compromise between the amount of approximation and how close we are to the final solution. For this reason, when we compare these methods to other existing approaches for the median graph computation in the next section, we will take into account not only the final solution but also the best solution along the recursive path.

Recursive methods sorted in descending order (specially GSRSD) obtain, in general, the best intermediate graphs. This fact seems to lead to the conclusion that it is better to start the approximation with a graph as closer as possible to the optimal solution. In addition, in these methods, the best median is usually obtained in a very interior call, when few intermediate graphs have been computed. Table 1 shows for each dataset and for each of the five recursive schemes the mean position of the best intermediate median (for 50/100 elements) along all the repetitions. Note that the values obtained by the BRS method are very close to the mid position (i.e., 25 in the case of 50 elements and 50 in the case of 100 elements), while the descending methods have in general lower values than the mid value and the ascending methods have in general higher values than the mid value. This could be used in a future work to improve the method in order to obtain good approximations of the median without need of computing all the intermediate graphs.

**Table 1** Average position of the median with minimum SOD

| Method | Letter | Molecule | Mutagenicity |
|---|---|---|---|
| BRS | 21/50 | 18/48 | 23/48 |
| GSRSD | 11/18 | 21/39 | 15/25 |
| GSRSA | 36/56 | 25/58 | 31/63 |
| VSRSD | 20/50 | 15/33 | 14/20 |
| VSRSA | 28/48 | 33/60 | 33/76 |

## 5 Experimental Evaluation

In this section we provide the results of an experimental evaluation of the three proposed methods. To this end, and since the objective of the methods is to find a representative of a set of graphs, we will perform cluster analysis. The median is typically used as a representative of a class, hence it is natural to think that if clustering results are better, then the corresponding theoretical representative is a better prototype of the cluster and the corresponding method to obtain the representatives is better. To this end, different clustering indices will be employed as evaluation measures. The main goal is not only to compare the different methods we propose to compute the representative but also to give an idea of which of them could be a better choice to compute the representative of a given set. Therefore, our reference system in all the experiments will be the set median.

With this objective in mind, we first proceed to provide the basic notions about clustering. In Sect. 5.1 we briefly explain graph clustering with focus on explaining the *k*-means graph-based clustering method that will be used in the subsequent experiments. In Sect. 5.1.1 we present the four standard clustering quality measures we will use to perform the evaluation of the methods. Finally, we present the results in Sect. 5.2.

### *5.1 Graph Clustering*

Cluster analysis or clustering is the task of assigning a set of objects into groups (called clusters) so that the objects in the same cluster are more similar (in some sense or another) to each other than to those in other clusters. In our case, we use a clustering strategy, the well known *k*-means algorithm, in which at each iteration of the algorithm a representative for each cluster is needed. For the sake of completeness, the k-means clustering algorithm applied to graphs is presented in Algorithm 3.

The k-means clustering algorithm is one of the most simple and straightforward methods for clustering data [26]. Given k, the desired number of clusters, the k centers of the clusters are randomly initialized picking up $k$ graphs from the original set of $n$ graphs. Then, the remaining graphs are assigned to the cluster corresponding to the closest center. The centers of the clusters are recomputed and the graphs are

**Algorithm 3:** Graph k-means algorithm

**input** : A set $S = \{g_1, g_2, \ldots, g_n\}$ of $n$ data items (represented as graphs) and the number of clusters $k$ to create

**output**: The centers of the clusters and for each data item an integer [1, k] indicating the cluster the item belongs to

**begin**

1. Assign randomly each graph $g_i$ to a cluster
2. Using this initial assignment, compute the median graph of each cluster.
3. Assign each data item to be in the cluster of its closest center using the graph edit distance.
4. Recompute the centers as in Step 2.
5. Repeat Steps 3 and 4 until the centers do not change.

**end**

assigned again to the cluster with the closest center until the clusters remain stable or a maximum number of iterations are reached. Note that clustering of data items represented by graphs is then possible by letting the median graph or the barycenter be the center and using graph edit distance whenever a distance is needed.

### 5.1.1 Clustering Performance Measures

In this work, graph clustering is used as a tool for the evaluation of the different median graph approaches. It is natural to think that if clustering results are better, then the corresponding theoretical representative is a better prototype of the cluster and the corresponding method is better.

Thus let $X = \{g_1, \ldots, g_n\}$ be a set of $n$ graphs belonging to classes $\{C_1, \ldots, C_k\}$, which represents he ground truth, and let $D = \{D_1, \ldots, D_l\}$ be a clustering of $X$. Let us denote $n_i^j = D_i \cap C_j$ the number of elements of class $C_j$ clustered in $D_i$.

Before presenting the results obtained for the different datasets, we introduce the clustering performance measures in which we base our evaluation of the clusterings, which have already been used in previous graph-based clustering experiments [10, 35, 41].

We use four standard performance measures. Three of them, the Rand index, the mutual information, and the bipartite index, base their scoring in the comparison between the ground truth and the clustering. In the case of this study, we are aware of the real classification of the data we work with. But it is important to remark that these measures of quality cannot be used when unclassified data are clustered. We compute one more quality measure, which is independent of the ground truth. The Dunn index is based on the assumption that items clustered together must be near each other while being far from items belonging to other clusters. Let us define all of them.

**Rand Index:** To compute this index, a pairwise comparison between all pairs of items in the dataset is computed. If two elements fall in the same class and belong

to the same cluster, it counts as an agreement. Similarly, if the two elements belong to different classes and fall into different clusters it counts as an agreement too. Otherwise, it counts as a disagreement. Let $A$ be the number of agreements and $D$ the number of disagreements. The Rand index [32]

$$R = \frac{A}{A+D} \tag{5}$$

measures the matching of the obtained clusters to the ground truth classes. The Rand index produces measures in the interval [0, 1], with 1 meaning a perfect match between the result of the algorithm and the ground truth.

**Mutual Information:** The mutual information[6, 43]

$$M = \frac{1}{n}\sum_{j=1}^{l}\sum_{h=1}^{k} n_j^h \log_{lk}\left(\frac{n_j^h n}{\sum_{i=1}^{l} n_i^h \sum_{i=1}^{k} n_j^i}\right) \tag{6}$$

represents the overall degree of agreement between the clustering and the ground truth with a preference for clusters that have high purity. Higher values indicate better performance.

**Bipartite Index:** Let $\wp_k$ denote the symmetric group, i.e., the set of all the permutations, of the set $1,\ldots,k$. We use permutations to evaluate all the possible assignments of clusters to classes and then compute the bipartite index over the optimal such assignment, as follows [35]:

$$B_I = \max_{\sigma\in\wp} \frac{1}{n}\sum_{i=1}^{k} n_i^{\sigma(i)} \tag{7}$$

This index is also normalized in the $[0,1]$ range, with higher values denoting better performance .

**Dunn Index:** Let $d_{\min}$ be the minimum distance between any two objects in different clusters and $d_{\max}$ the maximum distance between any two objects in the same cluster. The *Dunn index* [9]

$$D_I = \frac{d_{\min}}{d_{\max}} \tag{8}$$

is a measure of the compactness and separation of the clusters. Higher values of the Dunn index indicate better clustering.

## 5.2 Experimental Results

In this section, the application of the median graph and the barycenter graph for data clustering purposes will be presented.

**Table 2** Clustering quality measures for the letter database

| | RI | DI | MI | BI |
|---|---|---|---|---|
| SM | 0.805627 | 0.031315 | 0.116952 | 0.222400 |
| E2P | 0.885333 ● | 0.026163 | 0.197808 ● | 0.395333 ● |
| E3P | 0.895852 ● | 0.027030 | 0.214651 ● | 0.424000 ● |
| BRS | 0.854425 ● | 0.026506 ⋆ | 0.157418 ● | 0.288750 ● |
| GSRSD | 0.836309 ● | 0.021861 | 0.148605 ● | 0.287500 ● |
| GSRSA | 0.901697 ●, ○ | 0.025869 | 0.234569 ●, ○ | 0.454000 ●, ⋆ |
| VSRSD | 0.840083 ● | 0.024761 | 0.148055 ● | 0.289412 ● |
| VSRSA | 0.891671 ●, ⋆ | 0.030308 ○ | 0.203391 ●, ⋆ | 0.406889 ● |
| BRS/Best | 0.899922 ●, ○ | 0.028445 ○ | 0.292317 ●, ○ | 0.448000 ●, ○ |
| GSRSD/Best | 0.903065 ●, ○ | 0.030218 ○ | **0.245440** ●, ○ | 0.465333 ●, ○ |
| GSRSA/Best | **0.905512** ●, ○ | 0.030634 ○ | 0.238033 ●, ○ | **0.476833** ●, ○ |
| VSRSD/Best | 0.901601 ●, ○ | **0.037744** ●, ○ | 0.236525 ●, ○ | 0.457810 ●, ○ |
| VSRSA/Best | 0.897984 ●, ○ | 0.031610 ●, ○ | 0.230356 ●, ○ | 0.444667 ●, ○ |

### 5.2.1 Experimental Setup

The clustering experiments have been applied to the letter, the molecule, and the mutagenicity datasets [34]. Notice that the total number of methods being compared grows up to 11 (set median, E2P, E3P and the five recursive methods), which implies a large number of total instances of the clustering process for each database. For each of these methods, and ten times for each database, 50 elements of each class are randomly picked up and the clustering is carried out. In each instance of the experiments, i.e., for each clustering performed, we compute the value of the four quality measures that are explained in Sect. 5.1.1, and the mean value over the ten repetitions is reported.

### 5.2.2 Results

The results of these experiments are displayed in three tables, one for each database. Each table contains, for each of the methods, the mean value over the ten repetitions, for each of the four quality indices. Table 2 shows the values of the indices for the experiments made with graphs from the letter database, Table 3 corresponds to Molecules and Table 4 refers to the experiments with mutagen datasets. Results marked with a (●) are those medians performing better as class representatives than the set median. Results marked with a (○) are the medians performing better than the E2P and E3P methods. Medians marked with the (⋆) are those performing better than E2P or E3P methods. The best median for each index is marked with bold face. Recall that the Rand index (RI), the mutual information (MI), and the bipartite index (BI) are groundtruth-based indices, while the Dunn index (DI) is not. For the sake of simplicity, we will sometimes refer to groundtruth-based indices as GT indices. Recall also that all four indices give higher values to better clusterings.

**Table 3** Clustering quality measures for the molecule database

| | RI | DI | MI | BI |
|---|---|---|---|---|
| SM | 0.533500 | 0.367934 | 0.063922 | 0.617000 |
| E2P | 0.562738 ● | 0.130584 | 0.102330 ● | 0.676923 ● |
| E3P | 0.697138 ● | 0.168914 | 0.210778 ● | 0.813846 ● |
| BRS | 0.548760 ● | 0.263230 | 0.088487 ● | 0.656000 ● |
| GSRSD | 0.534226 ● | 0.455628 ● | 0.052444 | 0.588710 |
| GSRSA | 0.676083 ●,⋆ | 0.144330 | 0.188662 ●,⋆ | 0.785833 ●,⋆ |
| VSRSD | 0.517486 | **0.459920** ● | 0.034428 | 0.560952 |
| VSRSA | 0.617291 ●,⋆ | 0.158076 | 0.147863 ●,⋆ | 0.737273 ●,⋆ |
| BRS/Best | 0.705227 ●,○ | 0.126002 | 0.214477 ●,○ | 0.809333 ●,⋆ |
| GSRSD/Best | **0.781244** ●,○ | 0.126477 | **0.277252** ●,○ | **0.870000** ●,○ |
| GSRSA/Best | 0.651940 ● | 0.115464 | 0.176478 ●,○ | 0.775000 ● |
| VSRSD/Best | 0.701178 ●,○ | 0.135930 | 0.214217 ●,○ | 0.813333 ●,⋆ |
| VSRSA/Best | 0.770160 ●,○ | 0.093471 | 0.266716 ●,○ | 0.864000 ●,○ |

**Table 4** Clustering quality measures for the mutagenicity database

| | RI | DI | MI | BI |
|---|---|---|---|---|
| SM | 0.512620 | **0.230830** | 0.012476 | 0.571000 |
| E2P | 0.531980 ● | 0.055990 | 0.027048 ● | 0.621000 ● |
| E3P | 0.532200 ● | 0.051731 | 0.026672 ● | 0.624000 ● |
| BRS | 0.519187 ● | 0.070817 | 0.015229 ● | 0.586875 ● |
| GSRSD | 0.500000 ● | 0.314786 | 0.022740 ● | 0.500000 |
| GSRSA | 0.527580 ● | 0.047640 | 0.022250 ● | 0.613000 ● |
| VSRSD | 0.500000 ● | 0.314786 | 0.021190 ● | 0.500000 |
| VSRSA | 0.512053 ● | 0.051021 | 0.009267 ● | 0.573333 ● |
| BRS/Best | 0.534200 ●,○ | 0.048426 | 0.027776 ●,○ | 0.630000 ● |
| GSRSD/Best | **0.540818** ●,○ | 0.050847 | **0.034978** ●,○ | **0.642727** ● |
| GSRSA/Best | 0.517640 ● | 0.077794 | 0.014892 ● | 0.570000 |
| GSRSD/Best | 0.527674 ● | 0.081238 | 0.023582 ● | 0.607895 ● |
| VSRSA/Best | 0.527778 ● | 0.057365 | 0.023378 ● | 0.611111 ● |

**Letter:** In general, except for the Dunn index, the recursive give better results than the set median graph. Which means that the graph embedding approach turns out to be a useful tool for median graph computation. In addition, the recursive methods give almost always better results than at least one of the non-recursive embedding methods, which means that using the whole set of graphs to obtain the median gives a significant improvement. In all the indices, the best results are given by one of the recursive methods (taking the best intermediate median). In addition to that, regarding the type of ordering used in the recursive methods, the ascending order gives the better results in the general case. However, if we take the best intermediate median along the recursive path, such a difference is less evident and the best results are spread among both ascending and descending schemes.

**Molecule:** Similar results can be drawn for the molecule dataset. In general, the embedding methods are able to obtain better representatives than the set median. However, only the best intermediate medians give better results than the two of the non-recursive embedding methods, although most of the recursive embedding methods give better results than at least one of the non-recursive embedding methods. Here, the best intermediate medians of the GSRSD methods give the best results in general. In relation to the sorting type, the ascending schemes are those giving the best results in general in the recursive methods when the full recursive path is taken into account. This tendency is not followed when the best intermediate median is taken along the recursive path. In this case, better results are given in three out of the four indices by the GSRSD method.

**Mutagenicity:** In this case, the differences between all the embedding methods seem to be less than before. Although in general, the embedding methods give better results than the set median, the recursive embedding methods perform only slightly better than the non-recursive embedding methods, although three out of the four best results are given by the SRSGD method using the best intermediate median. Again, the ascending order gives the best results in general when the final graph along the recursive path is taken as the median. But, descending order and especially the GSRSD method give the best results (three out of the four indices) when the best intermediate median is taken as the final median.

These clustering experiments confirm that in general the embedding methods give better results than the set median in terms of their quality as representative points of a class. In addition, the recursive embedding methods give in most cases better results than the non-recursive embedding methods. This may suggest that using the whole set of original graphs to compute the median is important to the final result. It is important to remark that some differences can be seen between the ascending and descending sorting schemes. This suggests that a deep study of the implications of each sorting scheme should be performed in order to try to improve the median computation, for instance stopping the computation before the whole recursive path is computed.

## 6 Summary

Graph embedding methods have become very popular in the last few years since they permit to use the whole repository of machine learning algorithms with graphs. However an unsolved problem yet is the reverse step, i.e., how to recover the graph that corresponds to a point in the vector space. In this chapter we have described a generic recursive procedure that permits to recover such a graph given that the point lies inside the convex hull of $n$ previously embedded graphs. This procedure is based on recursively projecting the point into hyperplanes of decreasing dimensionality and recovering the graph from these projections using the concept of the weighted mean of a pair of graphs.

One problem where this procedure can be successfully applied is the computation of the median graph. The median graph has been shown to be a good choice to obtain a representative of a set of graphs. However, its computation is extremely complex. As a consequence, in real applications we are forced to use suboptimal methods in order to obtain approximate solutions for the generalized median graph in reasonable time. The procedure proposed in this chapter comprises a new algorithm for the computation of the median graph based on graph embedding. First, the graphs are mapped to points in an $n$-dimensional vector space using an embedding based on the graph edit distance. Then, the crucial point of obtaining the median of the set is carried out in the vector space, not in the graph domain, which dramatically simplifies this operation. Finally, the new procedure to recover a graph from the vector space permits to obtain the graph corresponding to the median vector. This last step is the main difference with previous methods that also compute the median graph using graph embedding. We analyze four variations of the base algorithm taking into account the order in which the graphs are considered in the recursive path.

In order to evaluate the proposed method (and all its variations), we have made experiments on three different graph databases, one semi-artificial and two containing real-world data. The underlying graphs have no constraints regarding the number of nodes and edges. We compared this approach with state-of-the-art embedding-based methods for median graph computation and also with the set median approach. Results show that with the proposed recursive approach we can obtain, in general, better medians, in terms of their SOD and their clustering performance, than existing embedding-based methods or the set median.

The proposed novel method for median graph computation is approximate in a double sense, namely through the graph embedding and graph recovery step. Nevertheless, as experiments on a number of databases with quite different characteristics have shown, it is able to discover median graphs of better quality than previous approximate methods that use the set median or the closest two or three points as the basis for approximation.

A number of important questions remain open regarding the nature of the proposed procedure. For instance, a deep analysis of the influence of the different approximations in the final graph that is obtained, should be carefully investigated. It would also be interesting to establish some relation between the degree of the exactness of the recovered graph and the characteristics of several embeddings that exist in the literature and, particularly, regarding how well these embeddings preserve the graph distances in the vector space. Finally, applications of this procedure to problems other than the median graph computation should also be investigated.

## References


1. Bajaj C (1988) The algebraic degree of geometric optimization problems. Discrete Comput Geom 3(2):177–191
2. Bunke H, Allerman G (1983) Inexact graph matching for structural pattern recognition. Pattern Recogn Lett 1(4):245–253
3. Bunke H, Günter S (2001) Weighted mean of a pair of graphs. Computing 67(3):209–224
4. Caelli T, Kosinov S (2004) Inexact graph matching using eigen-subspace projection clustering. IJPRAI 18(3):329–354. DOI http://dx.doi.org/10.1142/S0218001404003186
5. Conte D, Foggia P, Sansone C, Vento M (2004) Thirty years of graph matching in pattern recognition. Int J Pattern Recogn Artif Intell 18(3):265–298
6. Cover TM, Thomas JA (1991) Elements of information theory. Wiley-Interscience, New York
7. Demirci MF, Shokoufandeh A, Keselman Y, Bretzner L, Dickinson SJ (2006) Object recognition as many-to-many feature matching. Int J Comput Vision 69(2):203–222
8. Duda R, Hart P, Stork D (2000) Pattern classification, 2nd edn. Wiley Interscience, New York
9. Dunn J (1974) Well separated clusters and optimal fuzzy partitions. J Cibernet 4:95–104
10. Ferrer M (2008) Theory and algorithms on the median graph. Application to graph-based classification and clustering. PhD Thesis, Universitat Autònoma de Barcelona
11. Ferrer M, Serratosa F, Sanfeliu A (2005) Synthesis of median spectral graph. 2nd Iberian Conference on Pattern Recognition and Image Analysis. Estoril, Portugal. Lecture Notes in Computer Science, Springer-Verlag, vol 3523, pp 139–146
12. Ferrer M, Valveny E, Serratosa F, Bardají I, Bunke H (2009a) Graph-based -means clustering: A comparison of the set median versus the generalized median graph. In: Jiang X, Petkov N (eds) CAIP. Lecture notes in computer science, vol 5702. Springer, Berlin, pp 342–350
13. Ferrer M, Valveny E, Serratosa F, Riesen K, Bunke H (2010) Generalized median graph computation by means of graph embedding in vector spaces. Pattern Recognition 43(4): pp. 1642–1655.
14. Friedman M, Kandel A (1999) Introduction to pattern recognition. World Scientific, New York
15. Gibert J, Valveny E, Bunke H (2012) Graph embedding in vector spaces by node attribute statistics. Pattern Recogn 45(9):3072–3083. DOI 10.1016/j.patcog.2012.01.009. URL http://dx.doi.org/10.1016/j.patcog.2012.01.009
16. Grauman K, Darrell T (2004) Fast contour matching using approximate earth mover's distance. In: Proceedings of the IEEE conference on Computer Vision and Pattern Recognition Washington, DC, pp 220–227
17. Hakimi SL (2000) Location theory. CRC, West Palm Beach
18. de la Higuera C, Casacuberta F (2000) Topology of strings: Median string is NP-complete. Theor Comput Sci 230(1–2):39–48
19. Hlaoui A, Wang S (2006) Median graph computation for graph clustering. Soft Comput 10(1):47–53
20. Indyk P (2001) Algorithmic applications of low-distortion geometric embeddings. FOCS '01 Proceedings of the 42nd IEEE symposium on Foundations of Computer Science. IEEE Computer Society Washington, DC, USA, pp 10–33
21. Jiang X, Münger A, Bunke H (2001) On median graphs: Properties, algorithms, and applications. IEEE Trans Pattern Anal Mach Intell 23(10):1144–1151
22. Jouili S, Tabbone S (2010) Graph embedding using constant shift embedding. In: Proceedings of the 20th international conference on recognizing patterns in signals, speech, images, and videos, ICPR'10. Springer, Berlin, pp 83–92. URL http://dl.acm.org/citation.cfm?id=1939170.1939183
23. Justice D, Hero AO (2006) A binary linear programming formulation of the graph edit distance. IEEE Trans Pattern Anal Mach Intell 28(8):1200–1214
24. Kramer S, Raedt LD (2001) Feature construction with version spaces for biochemical applications. In: Proceedings of the eighteenth international conference on machine learning, ICML '01. Morgan Kaufmann Publishers Inc., San Francisco, pp 258–265. URL http://dl.acm.org/citation.cfm?id=645530.655667

25. Luo B, Wilson RC, Hancock ER (2003) Spectral embedding of graphs. Pattern Recogn 36(10):2213–2230
26. Mitchell TM (1997) Machine learning. McGraw-Hill, New York
27. Münger A (1998) Synthesis of prototype graphs from sample graphs. Diploma thesis, University of Bern (in German)
28. Neuhaus M, Bunke H (2004) An error-tolerant approximate matching algorithm for attributed planar graphs and its application to fingerprint classification. In: Fred ALN, Caelli T, Duin RPW, Campilho AC, de Ridder D (eds) SSPR/SPR. Lecture notes in computer science, vol 3138. Springer, Berlin, pp 180–189
29. Neuhaus M, Riesen K, Bunke H (2006) Fast suboptimal algorithms for the computation of graph edit distance. In Proc. Joint IAPR Workshops on Structural and Syntactic Pattern Recognition and Statistical Techniques in Pattern Recognition. Lecture Notes on Computer Science, Springer-Verlag, Vol 4109, pp 163–172
30. Pekalska E, Duin R (2005) The dissimilarity representation for pattern recognition – foundations and applications. World Scientific, Singapore
31. Pekalska E, Duin RPW, Paclík P (2006) Prototype selection for dissimilarity-based classifiers. Pattern Recogn 39(2):189–208
32. Rand WM (1971) Objective criteria for the evaluation of clustering methods. J Am Stat Assoc 66:846–850
33. Ren P, Wilson RC, Hancock ER (2011) Graph characterization via ihara coefficients. IEEE Trans Neural Networks 22(2):233–245
34. Riesen K, Bunke H (2008a) IAM graph database repository for graph based pattern recognition and machine learning. In N. da Vitoria Lobo et al., (eds) SSPR&SPR. Lecture Notes in Computer Science, Springer-Verlag, vol 5342, pp 287–297
35. Riesen K, Bunke H (2008b) Kernel k-means clustering applied to vector space embeddings of graphs. In: Prevost L, Marinai S, Schwenker F (eds) ANNPR. Lecture notes in computer science, vol 5064. Springer, Berlin, pp 24–35
36. Riesen K, Bunke H (2009) Approximate graph edit distance computation by means of bipartite graph matching. Image Vision Comput 27(7):950–959
37. Riesen K, Bunke H (2010) Graph classification and clustering based on vector space embedding. World Scientific, Singapore
38. Riesen K, Neuhaus M, Bunke H (2007) Graph embedding in vector spaces by means of prototype selection. In: 6th IAPR-TC-15 international workshop, GbRPR 2007. Lecture notes in computer science, vol 4538. Springer, Berlin, pp 383–393
39. Robles-Kelly A, Hancock ER (2007) A Riemannian approach to graph embedding. Pattern Recogn 40(3):1042–1056
40. Sanfeliu A, Fu K (1983) A distance measure between attributed relational graphs for pattern recognition. IEEE Trans Syst Man Cybernet 13(3):353–362
41. Schenker A, Bunke H, Last M, Kandel A (2005) Graph-theoretic techniques for web content mining. World Scientific, Singapore
42. Shokoufandeh A, Macrini D, Dickinson S, Siddiqi K, Zucker SW (2005) Indexing hierarchical structures using graph spectra. IEEE Trans Pattern Anal Mach Intell 27(7):1125–1140. DOI 10.1109/TPAMI.2005.142. URL http://dx.doi.org/10.1109/TPAMI.2005.142
43. Strehl A, Ghosh J, Mooney R (2000) Impact of similarity measures on web-page clustering. In: Proceedings of the 17th national conference on artificial intelligence: workshop of artificial intelligence for web search, (AAAI 2000), Austin, Texas, 30–31 July 2000, pp 58–64
44. Weiszfeld E (1937) Sur le point pour lequel la somme des distances de $n$ points donnés est minimum. Tohoku Math J (43):355–386
45. White D, Wilson RC (2006) Mixing spectral representations of graphs. In: 18th international conference on pattern recognition (ICPR 2006). IEEE Computer Society, Hong Kong, China, 20–24 August 2006, pp 140–144
46. Wilson RC, Hancock ER, Luo B (2005) Pattern vectors from algebraic graph theory. IEEE Trans Pattern Anal Mach Intell 27(7):1112–1124