Computer Vision and Image Understanding 115 (2011) 919–928

Contents lists available at ScienceDirect

Computer Vision and Image Understanding

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

A generic framework for median graph computation based on a recursive embedding approach

M. Ferrer [a,*], D. Karatzas [b], E. Valveny [b], I. Bardaji [a], H. Bunke [c]

[a] Institut de Robòtica i Informàtica Industrial, CSIC-UPC, C.Llorens Artigas 4-6, 2a planta, 08028 Barcelona, Spain
[b] Centre de Visió per Computador, Departament de Ciències de la Computació, Universitat Autònoma de Barcelona, 08193 Bellaterra, Spain
[c] Institute of Computer Science and Applied Mathematics, University of Bern, Neubrückstrasse 10, CH-3012 Bern, Switzerland

ARTICLE INFO

Article history:
Available online 17 March 2011

Keywords:
Median graph
Graph embedding
Graph matching
Structural pattern recognition

ABSTRACT

The median graph has been shown to be a good choice to obtain a representative of a set of graphs. However, its computation is a complex problem. Recently, graph embedding into vector spaces has been proposed to obtain approximations of the median graph. The problem with such an approach is how to go from a point in the vector space back to a graph in the graph space. The main contribution of this paper is the generalization of this previous method, proposing a generic recursive procedure that permits to recover the graph corresponding to a point in the vector space, introducing only the amount of approximation inherent to the use of graph matching algorithms. In order to evaluate the proposed method, we compare it with the set median and with the other state-of-the-art embedding-based methods for the median graph computation. The experiments are carried out using four different databases (one semi-artificial and three containing real-world data). Results show that with the proposed approach we can obtain better medians, in terms of the sum of distances to the training graphs, than with the previous existing methods.

1. Introduction

In structural pattern recognition, the use of graphs to represent complex and structured objects has gained popularity in recent years. Thus, a number of different graph matching approaches have been proposed in the literature. For an extensive review of graph matching methods and applications, we refer the reader to Conte et al. [1]. However, one drawback of graphs arises from the fact that there is little mathematical structure in the domain of graphs. For example, computing the weighted sum, or the product of a pair of entities (which are elementary operations, required in many algorithms) is not possible in the graph domain.

Another important operation is the computation of the median graph [2]. Given a set of graphs, the median graph is defined as the graph that has the minimum sum of distances (SOD) to all the graphs in the set. It can be seen as the representative of the set. In fact, it has been successfully applied in classical learning algorithms such as k-means clustering [3] and kNN-based classification [4]. Moreover, it can be potentially applied to any graph-based algorithm where a representative of a set of graphs in needed. However, the cost of the computation of the median graph is exponential both in the number of input graphs and their size [5]. Up to now, two exact algorithms have been presented [6,7]. 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 [2,6], greedy algorithms [8] and spectral graph theory [9,10]. However, all these algorithms can only be applied to restricted sets of graphs, regarding either the type or the size of the graphs.

An alternative to simplify the computation of the median graph is graph embedding. In general, embeddings try to convert points of an original space into another space with better properties that, potentially, permit to simplify some of the operations over the original space. Some work has been devoted to establish a theoretical framework and to propose generic embeddings with good properties concerning the distortion introduced by the embedding (see [11] for a good review). Some of these ideas have been applied to trees [12] and graphs [13] in the context of image categorization. The application of the generic framework of embedding to graphs permits to convert graphs into points in any vector space. Thus, graph embedding techniques emerge as a powerful way to provide access to the rich repository of algorithmic tools available in statistical pattern analysis. For that reason, a number of specific graph embeddings have been recently proposed. For instance, features derived from the eigen-decomposition of graphs are studied in [14]. An approach dealing with string edit distance applied to the eigensystem of graphs is presented in [15]. This procedure results in distances between graphs which are used to embed the graphs into a vector space by means of multidimensional scaling. In

* Corresponding author. Fax: +34 93 401 57 50.
E-mail address: mferrer@iri.upc.edu (M. Ferrer).

1077-3142/$ - see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.cviu.2010.12.010

[16], the authors show how the elements of the spectral decomposition of the Laplacian matrix can be used to construct symmetric polynomials. The coefficients of these polynomials are used as graph features in order to encode graphs as vectors. Another approach for graph embedding has been proposed in [17]. The authors use the relationship between the Laplace–Beltrami operator and the graph Laplacian to embed a graph onto a Riemannian manifold. Recently, graph embedding by means of the graph edit distance [18] has been used to perform classification tasks [19].

Graph embedding has been already used for the median graph computation [4]. In that approach, a three-step procedure is followed to approximate the median graph. In the first step, graphs are embedded into a vector space by means of the graph edit distance computation. As a result, every graph in the original set becomes a point in a real-vector space. In the second step, the median of this set of points (which is supposed to be the vector representation of the median graph) is computed using the Weiszfeld algorithm [20]. Performing this operation in the vector domain is simpler than computing it in the graph domain. Finally, in the third step, an approximation of the median graph is recovered from the vector domain using the weighted mean of a pair of graphs [21]. Although this work has been shown to be able to obtain good approximations of the median graph, it is also true that one is not able to recover the graph corresponding to the median vector but only a graph corresponding to the median of a subset of the original points (supposed to be close to the median vector). This may cause a deterioration in the obtained median graph.

The present paper presents a new generic framework to compute the median graph using graph embedding. The proposed methodology also relies on 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 this representative point. However, the main contribution of the paper is made in the third step, i.e. the way of recovering the median graph from the median vector. We propose a recursive approach that, ideally, would permit to recover the graph that corresponds exactly to the median vector. The basic idea of this approach is as follows. Once all graphs have been mapped to their corresponding points in the n-dimensional real space and the median of these points has been computed, we sequentially project the median 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 median point is obtained. However, due to the complexity of graph matching problems, we are forced to use approximate algorithms and, therefore we will only be able to obtain partial approximations of the real median graph. Nevertheless, the proposed approach takes into account all graphs of the given set in the recovery of the median graph from the median vector. This is in contrast with previous approaches [3,4], where only a small subset is used for the reconstruction. Due to the larger set of graphs, we may expect to obtain a better approximation of the median graph by this procedure. In this sense, we analyze four additional variations of this method which take into account different sorting schemes of the original set of graphs. These variations can help to understand the influence of these approximations in the final result. 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. A preliminary version of this paper appeared in [22]. The current paper has been significantly extended with respect to the underlying methodology and the experimental evaluation.

In order to test the quality of the proposed methods, we have made experiments on four different graph databases, one semi-artificial and three containing real-world data. The underlying graphs have no constraints regarding the number of nodes and edges. The results are evaluated, according to the definition of the median graph, in terms of the sum of distances of the median graph to all other elements in the training set. We will show that, in most cases, the new method obtains better results than the set median and other embedding-based methods. With these results at hand, we can apply this new approach to any real world graph-based application in pattern recognition and machine learning that requires to compute a median, for instance, classification and clustering.

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 the proposed generic method for the median computation is described. After that, Section 4 presents a practical implementation of the proposed generic framework. Section 5 reports a number of experiments and presents the results achieved with our method. Also a comparison with a reference system is provided. Finally, in Section 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 paper.

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, $\mu: V \to L$ is the node labeling function and ν: $V \times V \to L$ 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, that is, 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.

2.2. Graph edit distance

The basic idea behind the graph edit distance [18,23] 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 an 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

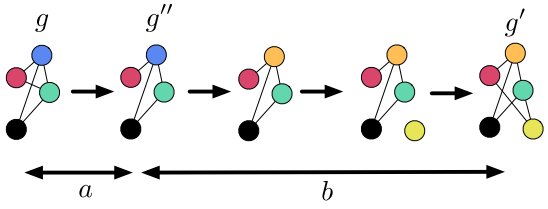

Fig. 1. Example of a possible edit path between two graphs, g and g'. It consists on an edge deletion, a node substitution, a node insertion and two edge insertions. Furthermore g' is the Weighted Mean of the Graphs g and g'.

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.

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

A number of optimal and approximate algorithms for the computation of the graph edit distance have been proposed up to now. 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 [23]. The major drawback of such an approach is its computational complexity, which is exponential in the number of nodes of the involved graphs. Consequently, its application 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 [24]. A linear programming method to compute the graph edit distance with unlabeled edges is presented in [25]. Such a method can be used to obtain lower and upper edit distance bounds in polynomial time. In [26] 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 [27]. In this paper we will use these two last approximate methods for the graph edit distance computation.

In [18] 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 [26,27] used in this paper, however, the metric property is no longer guaranteed. But this does not have any negative impact on the approach proposed in this paper 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 [28] 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 [21] is a crucial tool. For this reason we include its definition in the following.

Let g and g' be graphs and let U be the set of graphs that can be constructed using the labels of their nodes and edges. Let

$$I = \{h \in U |\ d(g,g') = d(g,h) + d(h,g')\}$$

be the set of *intermediate* graphs between g and g, i.e. any of the graphs along the edit path between g and g'. Given a weight $a \in \mathbb{R}$ where $0 \leqslant a \leqslant d(g,g')$, the weighed mean of g and g' is a graph g'' such that,

$$g'' = \arg\min_{h\in I} |d(g,h) - a| \quad (2)$$

That is, given two graphs, g and g' and a parameter a, the weighted mean is an intermediate graph g'' between them, whose distance to g is as close as possible to a. Consequently, its distance to g' is also the closest to $d(g,g') - a$. From this point on, we will refer the weighted mean as a-mean.

Please note that, due to the discrete nature of the domain of graphs and the graph edit distance, not for any value of a, a graph g'' may exist such that $d(g,g'') = a$ and $d(g'',g') = d(g,g') - a$.

Fig. 1 shows an example where a coincides with the cost of deletion of the edge between the red and the green nodes. Therefore, since this deletion is an edit operation of an optimal edit path, the graph g'' is an a-mean for which $|d(g,g'') - a| = 0$.

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.

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,

$$\bar{g} = arg\min_{g\in U} \sum_{g_i\in S} d(g,g_i) \quad (3)$$

That is, the generalized median graph $\bar{g}$ of S 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. 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 in needed.

Despite its simple mathematical definition (Eq. 3), the computation of the median graph is extremely complex. As implied by Eq. (3), a 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 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 [29]). 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 [2,6,8–10] apply some heuristics in order to reduce the complexity of the graph distance computation and the size of the search space. Recent works [3,4] rely on graph embedding into vector spaces. Since they are the foundation of this work, we introduce them in detail in the next section.

An alternative to the generalized median graph, which is computationally less demanding, is the set median graph. The difference between the two concepts consists in the search space where the median is looked for. As it is shown in Eq. (3), the search space for the generalized median graph is U, that is, the whole universe of graphs. In contrast, the search space for the set median graph is simply S, that is, the set of given graphs. 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. 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 reference system in the experiments presented in Section 5.

3. A generic approach to compute the median graph via embedding

Generally speaking, graph embedding [11] aims to convert graphs into another structure, such as real vectors, and then operate in the associated space to facilitate some typical graph-based

tasks, such as matching and clustering [30,31]. To this end, as we already explained in the introduction, different graph embedding procedures have been proposed in the literature so far.

Graph embedding has been used recently for the approximate median graph computation [4]. In that work, a three-step based process is used to perform the median computation. Here, we take this three-step procedure as the basis to propose a generic framework to compute the median graph. In the following we will describe the three steps of this procedure, although our main contribution is made in the third step where we will introduce a new generic method to recover the graph corresponding to the median vector. For the rest of the section we will suppose that a set S of n graphs $S = \{g_1, g_2, \ldots, g_n\}$ is given.

Step I: In a first step every graph in S is embedded into the real n-dimensional space, i.e. each graph becomes a point in $\mathbb{R}^n$. In principle, any embedding which fulfils this condition could be used in this step. However, it is expected that the best results will be obtained when the distance relationships in the vector space resemble as much as possible the distance relationships in the original graph space.

Step II: The second step consists of computing a representative of the set in the vector space. Here, the median vector $\vec{M}$ arises as a natural choice [4]. 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 $\wp$. As we can see, its definition is exactly the same as the median graph but in the vector space. Thus, if the embedding preserves the distance structure of the graph domain, the median vector should be the a good representation of the median graph in the vector space.

Step III: Finally, the resulting median vector has to be mapped back to a corresponding graph. This last step of mapping back from the vector space to the graph domain is a difficult problem for a number of reasons. To mention just two, depending on the embedding technique not every point in the (continuous) vector space corresponds to a graph. Secondly it might be that for a particular vector a one-to-many relationship to the graph domain exists. These difficulties have been shown in recent works. For instance, in [4] the three closest points to the computed median vector $\vec{M}$ are used to compute the median $\vec{M}'$ of the three points (which always falls on the plane defined by them). Using these three points (corresponding to known graphs) and the new median $\vec{M}'$, the weighted mean approach [21] is used to recover a graph $\bar{g}'$ (corresponding to $\vec{M}'$), which is taken as an approximation of the median graph $\bar{g}$ of S. Although the results of this approach are better than other approximations of the median graph, the method to recover the median graph from the median vector is not exact. It only permits to recover the graph that corresponds to the median vector of the three selected points and not the graph that corresponds to the median vector of the whole set.

In the remaining of this section we present a new recursive approach to get the mapping between the median vector and the median graph. The novelty of the approach lies in the fact that the complete set of graphs, or more precisely, their representation in the vector space, will be used for establishing this mapping. 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 between two graphs, we would be able to obtain the graph that corresponds to the median vector of the whole set. However, as we will discuss later in Section 4, we will be forced to use several approximations in the practical implementation of the procedure. As a result we will only be able to obtain approximations of the median graph but still better than those obtained using existing methods as we will show in Section 5.

3.1. Median graph recovering

The graph corresponding to the median vector will be recovered by means of the recursive application of the weighted mean of a pair of graphs. 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).
2. The Euclidean median M_n[1] of these n points will always fall on the hyperplane H_{n-1}. More concretely, it will fall within the volume of the $n-1$ dimensional simplex with vertices P_i with $i = 1, 2, \ldots, n$. Fig. 2 shows an example for $n = 4$ and $n = 3$.
3. 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 calculated median M_2 lies on this line segment. We can then calculate the graph g_{M_2} corresponding to the median M_2 as the weighted mean of g_1 and g_2.

From the third point we can observe that, given n embedded points $\{P_1, P_2, \ldots, P_n\}$ and their corresponding median 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 on these two points in order to find the graph corresponding to M_n. In the following we will describe how we can obtain these two points and, thus, such a graph. We will illustrate this procedure with the example shown in Fig. 3 with four points. Fig. 3a shows the four points $\{P_1, P_2, P_3, P_4\}$ and their median 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 $(M_n - P_n)$ (vector $(M_4 - P_4)$ in Fig. 3b). 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 $\{P_1, P_2, \ldots, P_{n-1}\}$, that is all the original points except P_n, then the intersection of the line defined by the vector $(M_n - P_n)$ 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. 3b which lies on the hyperplane H_2 (plane) defined by P_1, P_2 and P_3). 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 $(M_n - P_n)$, 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 median 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

[1] For clarity, in the remainder, we will refer to the median of n points as M_n.

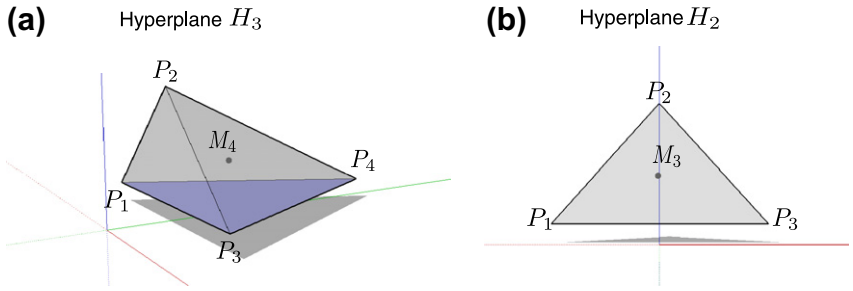

Fig. 2. (a) The 3D-Hyperplane H_3 is defined by the four points $P_i = \{P_1, P_2, P_3, P_4\}$. The Euclidean median M_4 falls in the 3D space defined by the four points and specifically within the pyramid (3D simplex) with vertices P_i ($i = 1, \ldots, 4$). (b) The 2D-Hyperplane H_2 is defined by the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

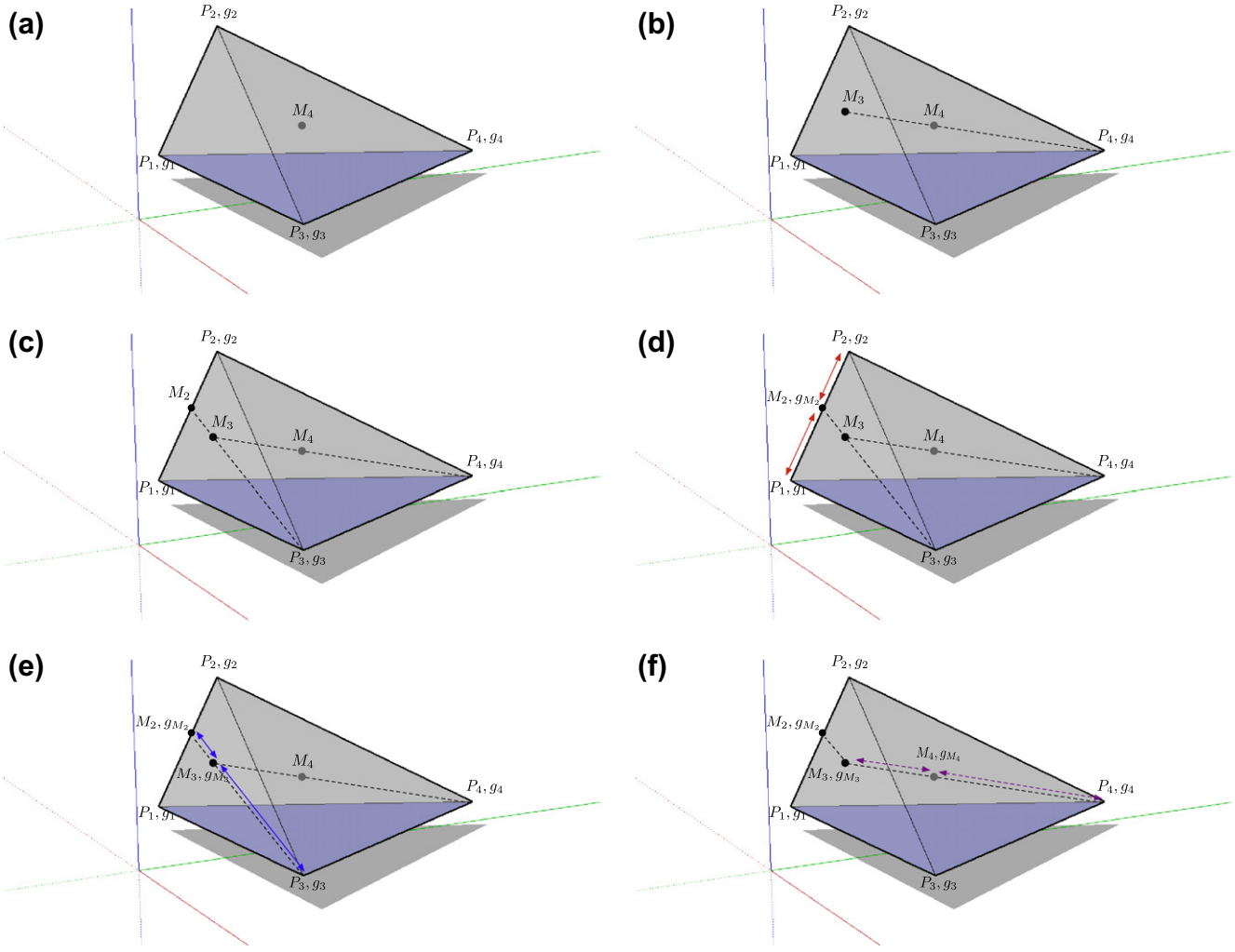

Fig. 3. Complete example of the median recovering with four points $\{P_1, P_2, P_3, P_4\}$.

the vector $(M_{n-1} \vec{-} P_{n-1})$ ($(M_3 \vec{-} P_3)$ in Fig. 3c). 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. 3(c) 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. 3d).

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. 3e). 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, which is the median graph of the set (g_{M_4} in Fig. 3f).

4. A practical implementation of the generic framework

In the previous section we introduced the generic framework to compute the median graph as a three-step process and, particulary, our contribution to recover the median graph from the median vector in the last step. In this section we describe the practical issues concerning our specific implementation of this generic framework. We explain the choices we have made for the first two steps, that are basically the same as in [4], and we discuss some practical considerations about the approximation introduced in the last step.

Step I (*graph embedding*): In this work 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 [32], and it is based on the work proposed in [33]. 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 $m \leqslant n$). 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 $v_i = (P_i - O)$, 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. Note that this point was an important observation in Section 3.1.

An important relation that has been shown in [32] is,

$$\|\phi(g) - \phi(g')\| \leqslant \sqrt{n} \cdot d(g, g') \tag{4}$$

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

Step II (*median vector computation*): As we already commented at the beginning of Section 3, the median vector is used as the representative of the set in the vector domain.

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 [34]. No algorithm in polynomial time is known, nor has the problem been shown to be NP-hard [35]. In this work we will use the most common approximate algorithm for the computation of the median vector, that is, the Weiszfeld's algorithm [20]. It is s a form of iteratively re-weighted least squares that converges 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.

$$M'_{n_{i+1}} = \frac{\sum_{j=1}^{m} \frac{x_j}{\left\| x_j - M'_{n_i} \right\|}}{\sum_{j=1}^{m} \frac{1}{\left\| x_j - M'_{n_i} \right\|}} \tag{5}$$

The algorithm may finish when a predefined number of iterations is 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.

Step III (*median graph recovering*): To recover the median graph corresponding to the median vector M_n, we will use the recursive procedure presented in Section 3.1. In that section, it was claimed that the method to recover the median graph from the median vector should permit to obtain the exact median graph in case that the embedding preserves the distance structure and that 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. Finally, the computation of the median vector is also based on an approximate algorithm. For all these reasons, we are only able to get approximations 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 (and consequently the order in which the graphs are taken). 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 in 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 sum of distances of every point in the graph or the vector domain to the rest of points. Points with a low sum of distances will correspond to points close to the optimal solution.Thus, we present four variants of the basic recursive scheme presented in Section 3.1 (BRS in short), which include a pre-processing to sort the graphs. Note that, to be consistent with the notation and the explanations performed in Section 3.1, the words *ascending* or *descending* used in the following, refers to graphs from g_n to g_1. These sorted schemes will be referred as SRS (sorted recursive schemes).

- *Graph-domain-based Recursive Scheme sorted in descending order* (SRS-GD): 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 Recursive Scheme sorted in ascending order* (SRS-GA): 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 graph corresponding to M_{n-1} and g_n (the set median).
- *Vector-domain-based Recursive Scheme sorted in descending order* (SRS-VD): 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 such that the corresponding point has the maximum sum of distances,

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

- *Vector-domain-based Recursive Scheme sorted in ascending order* (SRS-VA): 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 computed 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 a 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. In this dataset, we consider the 15

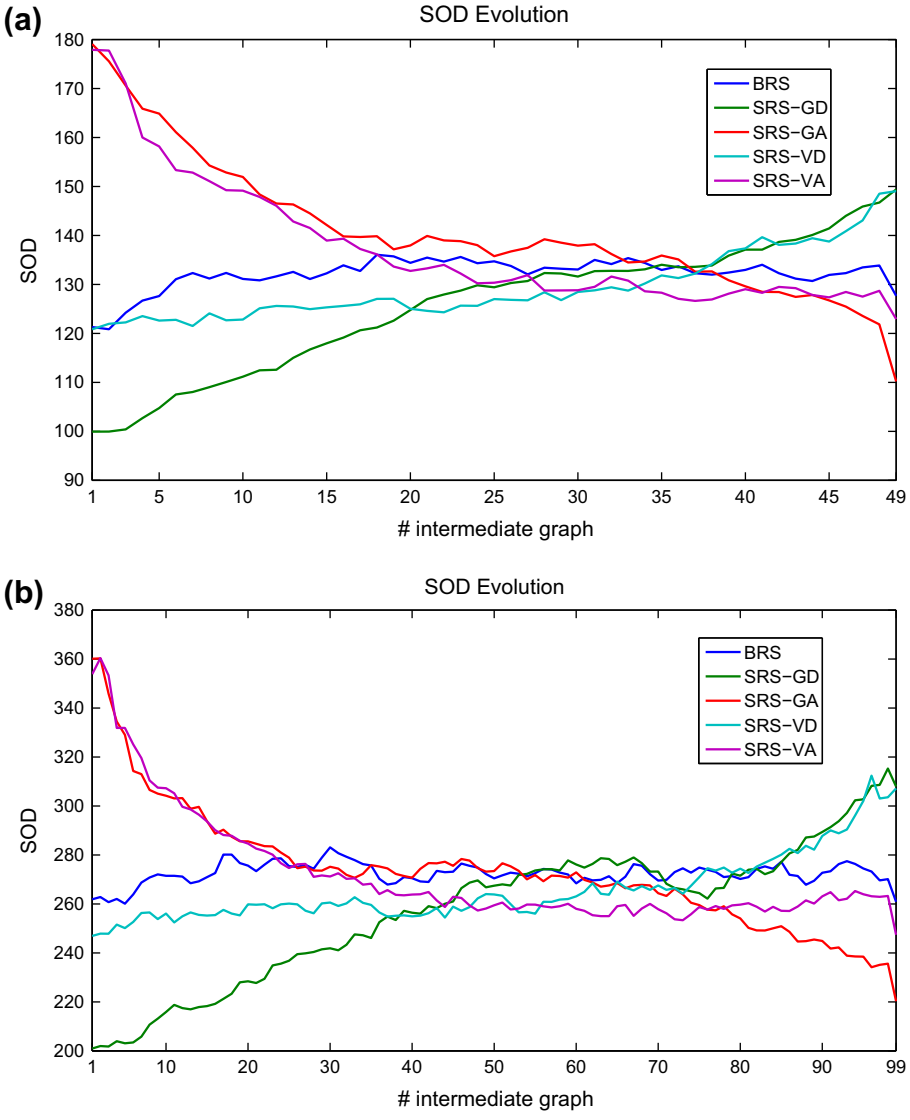

Fig. 4. SOD Evolution for the Letter Dataset using (a) sets of 50 elements and (b) sets of 100 elements.

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 Table 2 for characteristics of this dataset, or [36]). More concretely, we took sets of 50 and 100 elements randomly from the dataset and we computed the median with each of the methods. Fig. 4 shows the evolution of the SOD of the intermediate medians graphs for each recursive method. The x-axis represents the 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 10 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. 4, 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 (SRS-GD and SRS-VD), there is a clear tendency in starting with graphs 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 SRS-GD method, M_2 is computed with the set median and the next graph in terms of the lower SOD). Consequently, M_2 has a SOD close to that of the set median. 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 (SRS-GA and SRS-VA) 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 the SOD. This translates in a decreasing curve. As a conclusion, we can state that we get better solutions as we take 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 differs. 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.

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 SRS-GD) 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
Mean value of the position of the median with minimum SOD.

	Letter	Molecule	Mutagenicity	Webpages
BRS	21/50	18/48	23/48	24/42
SRS-GD	11/18	21/39	15/25	23/44
SRS-GA	36/56	25/58	31/63	26/55
SRS-VD	20/50	15/33	14/20	23/45
SRS-VA	28/48	33/60	33/76	28/56

5. Experimental evaluation

In this section we provide the results of an experimental evaluation of the proposed algorithm against the two previous embedding methods for the median graph computation. To this end we have used four different graph databases representing Letter shapes, Molecular compounds (two databases), and Webpages. Table 2 show some characteristics of each dataset. For more information of these databases see [36].[2]

5.1. Experimental setup

In the experiments presented in this section we have proceeded as follows. For each class in each dataset 50 and 100 elements were randomly chosen. Then, we calculate the approximate median of each set using 8 different methods. Namely, the set median (SM), the previous embedding method using the two closest points to the median vector to recover the approximate median graph (E2P) [3], the embedding method using the three closest points to the median vector (E3P) [4], and the five recursive schemes explained before. For these five recursive schemes we also include results taking the best median along the recursive path. This procedures were repeated 10 times each.

The set median graph (SM) is used in the experiments as a reference line. As the set median graph is the graph belonging to the training set with minimum SOD, it is a good reference to evaluate the generalized median graph quality. For a given dataset, the same edit distance algorithm is used to compute the set median and the approximate median. In this sense, since they are computed using the same method, the same amount of error or distortion (due to the approximation) is introduced to both of them. Thus, we can say that they are fairly comparable. Clearly, with the size of the sets and the graphs we are managing, it is not possible to obtain the true median graph. This is the main reason for which we use the set median as the reference line for our comparisons.

Tables 3 and 4 show, for each dataset, the mean value of the SOD of the median obtained for each method. Results marked with the •are those better than the set median. Results marked with the ◦ are those better than both of the previous embedding methods. Results marked with the ☆ are those better than one of the previous embedding methods. The best results for each dataset and size of the training set (50 or 100) is marked with **bold** face.

[2] http://www.iam.unibe.ch/fki/databases/iam-graph-database.

Table 2
Summary of dataset characteristics, viz. the size, the number of classes (# classes), the average size of the graphs (∅ nodes) and the maximum size of the graph.

Database	Size	# classes	∅ nodes	max nodes
Letter	2250	15	4.7	8
Molecules	2000	2	15.7	95
Mutagenicity	4337	2	30.3	417
Webpages	2340	6	186.1	834

Table 3
SOD comparison for the Letter and the Molecule datasets (•: Methods better than the set median. ○: Methods better than both of the previous embedding methods. ☆: Methods better than one of the previous embedding methods. Global best results are marked with **bold** face).

	Letter		Molecule	
	50	100	50	100
SM	97.02	194.08	306.9	587.1
E2P	117.03	241.95	341.5	705.8
E3P	120.80	245.42	277.7 •	566.5 •
BRS	127.83	260.86	374.1	590.3
SRS-GD	149.39	308.47	504.9	1038.9
SRS-GA	109.71 ○	220.31 ○	290.9 •	599.3
SRS-VD	149.39	307.20	508.6	1069.8
SRS-VA	122.92	247.42	314.9	649.7
BRS/Best	98.93 ○	196.75 ○	259.5 •,○	**493.2** •,○
SRS -GD/Best	**92.60** •,○	**183.79** •,○	266.3 •,○	518.2 •,○
SRS-GA/Best	103.10 ○	193.68 ○	**255.23** •,○	508.9 •,○
SRS-VD/Best	100.54 ○	195.83 ○	269.9 •,○	523.7 •,○
SRS-VA/Best	102.26 ○	196.35 ○	265.35 •,○	515.28 •,○

Table 4
SOD comparison for the Mutagenicity and the Web datasets (•: Methods better than the set median. ○: Methods better than both of the previous embedding methods. ☆: Methods better than one of the previous embedding methods. Global best results are marked with **bold** face).

	Mutagenicity		Web	
	50	100	50	100
SM	367.1	723.9	1910.8	3886.4
E2P	336.9 •	666.9 •	1344.1 •	2693.6 •
E3P	317.6 •	643.3 •	1467.4 •	2998.2 •
BRS	399.6	945.2	2106.1	4141.1
SRS-GD	909.5	1732.1	3019.1	6195.3
SRS-GA	373.1	713.3 •	1865.1 •	3387.9 •
SRS-VD	877.6	1895.9	2754.5	5602.5
SRS-VA	388.3	771.1	1997.4	4246.1
BRS/Best	**314.7** •,○	**610.1** •,○	1298.1 •,○	**2418.1** •,○
SRS -GD/Best	321.5 •,☆	622.5 •,○	1298.5 •,○	2453.5 •,○
SRS-GA/Best	329.5 •,☆	613.8 •,○	1309.8 •,○	2446.6 •,○
SRS-VD/Best	316.4 •,○	611.41 •,○	**1293.7** •,○	2441.7 •,○
SRS-VA/Best	322.1 •,☆	615.3 •,○	1306.5 •,○	2482.7 •,○

5.2. Results and discussion

First of all, it is important to note that in all cases, the best results in terms of the SOD are achieved by one of the recursive methods, and more specifically, when we take the best intermediate graph along the recursive path. In addition to that, in 80% of the cases (32 out of 40) the best recursive methods outperform the set median. In all cases they outperform at least one of the previous embedding methods and in the 92.5% of cases they outperform both previous embedding methods. Keeping in mind that the previous embedding methods are already better than the set median in 10 out of the 16 cases, this last result shows that the best recursive schemes are able to outperform two already good previous methods for the median graph computation.

For the non-best recursive methods, the SRS-GA method shows to be superior over the others. In particular, it achieves better SOD than the set median in four out of the eight cases, and it is also better than the embedding methods in two cases. These results show a high correlation with the SOD evolution experiment, where the SRS-GA method achieved the lower SOD among all the recursive methods.

As a final conclusion we can say that in practice, in the recursive methods, selecting the best intermediate graph is better than taking the graph at the end of the recursive path and, as we said, better that the set median and the existing embedding methods. Moreover, we can also observe that all the best intermediate graphs are always better than all the best final graphs obtained with any of the recursive methods. In addition, recursive methods perform similarly among them. There is no recursive method clearly better than the others. Thus, it is worth to highlight and recall the fact that by means of descendent methods the best intermediate graph is found earlier. This could give us the option to avoid some of the computation, by stopping the recursion without calculating all the intermediate graphs. The proposed recursive embedding method shows to be an excellent guide through the search space, so that we have been able to find intermediate graphs which, with few exceptions, have lower SOD than those returned by the rest of the methods.

6. Conclusions

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. Recently, two different procedures using graph embedding into vector spaces have been presented for its approximate computation. Both procedures are based on three key steps. However, they introduce a source of error due to some approximation in the third step (i.e. mapping back from the vector domain to the graph domain), which may cause a deterioration in the obtained medians.

In this paper we proposed a generic recursive embedding procedure based on the weighted mean of a pair of graphs which tries to minimize such an approximation in the third step. First, the graphs are mapped to points in an n-dimensional vector space using 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, we proposed a recursive application of the weighted mean of a pair of graphs to obtain the graph corresponding to the median vector. This last step is the main difference with the previous existing embedding-based methods. We also proposed four variations of the base algorithm taking into account the order the graphs (and therefore the points) are considered in the recursive path.

In order to evaluate the proposed method (and all its variations), we have made experiments on four different graph databases, one semi-artificial and three containing real-world data. The underlying graphs have no constraints regarding the number of nodes and edges. We compared our approaches with these state-of-the-art embedding-based methods for the 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 the SOD, than the previous embedding methods and also 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 find median graphs of better quality than previous approximate methods that use the set median or the closest two or three points.

A number of important questions remain open regarding the nature of the median graph and the selection of the best representative graph for of a given set. For example it would be interesting to establish under what circumstances such a graph could be unique and in what ways it depends on the particular costs associated with the edit distance. These are interesting and relevant questions that should guide future research on the topic.

Finally, it should be noted that although we have concentrated on the concept of median graph, our generic approach can be easily adapted to obtain other graph representatives. For instance, if the barycenter vector is computed instead of the median vector, we can think that the corresponding graph should be more similar to the sample mean of graphs than the median graph. With these results at hand, we can think of applying this new approach to real world graph-based applications in pattern recognition and machine learning, such as classification and clustering, or more generally to any machine learning algorithm where a representative is needed.

Acknowledgments

This work has been supported by the Spanish research programmes Consolider Ingenio 2010 CSD2007-00018, TIN2006-15694-C02-02 and TIN2008-04998 and the fellowship RYC-2009-05031.

References

[1] D. Conte, P. Foggia, C. Sansone, M. Vento, Thirty years of graph matching in pattern recognition, Int. J. Pattern Recognit. Artif. Intell. 18 (3) (2004) 265–298.
[2] X. Jiang, A. Münger, H. Bunke, On median graphs: properties, algorithms, and applications, IEEE Trans. Pattern Anal. Mach. Intell. 23 (10) (2001) 1144–1151.
[3] M. Ferrer, E. Valveny, F. Serratosa, I. Bardají, H. Bunke, Graph-based -means clustering: a comparison of the set median versus the generalized median graph, in: X. Jiang, N. Petkov (Eds.), CAIP, Lecture Notes in Computer Science, vol. 5702, Springer, 2009, pp. 342–350.
[4] M. Ferrer, E. Valveny, F. Serratosa, K. Riesen, H. Bunke, Generalized median graph computation by means of graph embedding in vector spaces, Pattern Recognit., in press. doi:10.1016/j.patcog.2009.10.013.
[5] H. Bunke, A. Münger, X. Jiang, Combinatorial search versus genetic algorithms: a case study based on the generalized median graph problem, Pattern Recognit. Lett. 20 (11-13) (1999) 1271–1277.
[6] A. Münger, Synthesis of Prototype Graphs from Sample Graphs, in: Diploma Thesis, University of Bern (in German), 1998.
[7] M. Ferrer, E. Valveny, F. Serratosa, Median graph: a new exact algorithm using a distance based on the maximum common subgraph, Pattern Recognit. Lett. 30 (5) (2009) 579–588.
[8] A. Hlaoui, S. Wang, Median graph computation for graph clustering, Soft Comput. 10 (1) (2006) 47–53.
[9] M. Ferrer, F. Serratosa, A. Sanfeliu, Synthesis of median spectral graph, in: Second Iberian Conference of Pattern Recognition and Image Analysis, vol. 3523, LNCS, 2005, pp. 139–146.
[10] D. White, R.C. Wilson, Mixing spectral representations of graphs, in: 18th International Conference on Pattern Recognition (ICPR 2006), 20–24 August 2006, Hong Kong, China, IEEE Computer Society, 2006, pp. 140–144.
[11] P. Indyk, Algorithmic applications of low-distortion geometric embeddings, in: IEEE Symposium on Foundations of Computer Science, 2001, pp. 10–33.
[12] R. Babilon, J. Matousek, J. Maxová, P. Valtr, Low-distortion embeddings of trees, J. Graph Algorithms Appl. 7 (4) (2003) 399–409.
[13] M.F. Demirci, A. Shokoufandeh, S.J. Dickinson, Skeletal shape abstraction from examples, IEEE Trans. Pattern Anal. Mach. Intell. 31 (5) (2009) 944–952.
[14] B. Luo, R.C. Wilson, E.R. Hancock, Spectral embedding of graphs, Pattern Recognit. 36 (10) (2003) 2213–2230.
[15] R.C. Wilson, E.R. Hancock, Levenshtein distance for graph spectral features, in: 17th International Conference on Pattern Recognition, 2004, pp. 489–492.
[16] R.C. Wilson, E.R. Hancock, B. Luo, Pattern vectors from algebraic graph theory, IEEE Trans. Pattern Anal. Mach. Intell. 27 (7) (2005) 1112–1124.
[17] A. Robles-Kelly, E.R. Hancock, A Riemannian approach to graph embedding, Pattern Recognit. 40 (3) (2007) 1042–1056.
[18] H. Bunke, G. Allerman, Inexact graph matching for structural pattern recognition, Pattern Recognit. Lett. 1 (4) (1983) 245–253.
[19] K. Riesen, H. Bunke, Graph classification based on vector space embedding, IJPRAI 23 (6) (2009) 1053–1081.
[20] E. Weiszfeld, Sur le point pour lequel la somme des distances de n points donnés est minimum, Tohoku Math. Journal (43) (1937) 355–386.
[21] H. Bunke, S. Günter, Weighted mean of a pair of graphs, Computing 67 (3) (2001) 209–224.
[22] M. Ferrer, D. Karatzas, E. Valveny, H. Bunke, A recursive embedding approach to median graph computation, in: A. Torsello, F. Escolano, L. Brun (Eds.), GbRPR, Lecture Notes in Computer Science, vol. 5534, Springer, 2009, pp. 113–123.
[23] A. Sanfeliu, K. Fu, A distance measure between attributed relational graphs for pattern recognition, IEEE Trans. Syst. Man Cybernet. 13 (3) (1983) 353–362.
[24] M. Neuhaus, H. Bunke, An error-tolerant approximate matching algorithm for attributed planar graphs and its application to fingerprint classification, in: A.L.N. Fred, T. Caelli, R.P.W. Duin, A.C. Campilho, D. de Ridder (Eds.), SSPR/SPR, Lecture Notes in Computer Science, vol. 3138, Springer, 2004, pp. 180–189.
[25] D. Justice, A.O. Hero, A binary linear programming formulation of the graph edit distance, IEEE Trans. Pattern Anal. Mach. Intell. 28 (8) (2006) 1200–1214.
[26] M. Neuhaus, K. Riesen, H. Bunke, Fast suboptimal algorithms for the computation of graph edit distance, in: Joint IAPR International Workshops, SSPR and SPR 2006, Lecture Notes in Computer Science 4109, 2006, pp. 163–172.
[27] K. Riesen, H. Bunke, Approximate graph edit distance computation by means of bipartite graph matching, Image Vision Comput. 27 (7) (2009) 950–959.
[28] E. Pekalska, R. Duin, The Dissimilarity Representation for Pattern Recognition – Foundations and Applications, World Scientific, 2005.
[29] C. de la Higuera, F. Casacuberta, Topology of strings: median string is NP-complete, Theor. Comput. Sci. 230 (1–2) (2000) 39–48.
[30] K. Grauman, T. Darrell, Fast contour matching using approximate earth mover's distance, in: Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004, pp. 220–227.
[31] M.F. Demirci, A. Shokoufandeh, Y. Keselman, L. Bretzner, S.J. Dickinson, Object recognition as many-to-many feature matching, Int. J. Comput. Vision 69 (2) (2006) 203–222.
[32] K. Riesen, H. Bunke, Graph Classification and Clustering Based on Vector Space Embedding, World Scientific, 2010.
[33] E. Pekalska, R.P.W. Duin, P. Paclík, Prototype selection for dissimilarity-based classifiers, Pattern Recognition 39 (2) (2006) 189–208.
[34] C. Bajaj, The algebraic degree of geometric optimization problems, Discrete Comput. Geom. 3 (2) (1988) 177–191.
[35] S.L. Hakimi, Location Theory, CRC Press, 2000.
[36] K. Riesen, H. Bunke, IAM graph database repository for graph based pattern recognition and machine learning, in: SSPR/SPR, 2008, pp. 287–297.