Class Representative Computation Using Graph Embedding

Fahri Aydos, Ahmet Soran, and M. Fatih Demirci

TOBB University of Economics and Technology,
Computer Engineering Department,
Sogutozu Cad. No:43, 06560 Ankara, Turkey
{aydos,asoran,mfdemirci}@etu.edu.tr

Abstract. Due to representative power of graphs, graph-based object recognition has received a great deal of research attention in literature. Given an object represented as a graph, performing graph matching with each member of the database in order to locate the graph which most resembles the query is inefficient especially when the size of the database is large. In this paper we propose an algorithm which represents the graphs belonging to a particular set as points through graph embedding and operates in the vector space to compute the representative of the set. We use the k-means clustering algorithm to learn centroids forming the representatives. Once the representative of each set is obtained, we embed the query into the vector space and compute the matching in this space. The query is classified into the most similar representative of a set. This way, we are able to overcome the complexity of graph matching and still perform the classification for the query effectively. Experimental evaluation of the proposed work demonstrates the efficiency, effectiveness, and stability of the overall approach.

Keywords: object recognition, graph embedding, clustering.

1 Introduction

Graph-based object recognition has received a great deal of research attention in literature. Once the objects are represented as graphs such that vertices correspond to features and edges encode some relations between the features, the problem of object recognition is reformulated as that of exact or inexact graph matching. Given a query represented as a graph, one can perform graph matching with each member of the database to locate the graph which most resembles the query. Considering the fact that graph matching is computationally expensive, this experimental setup is not practical especially when the size of the database is large.

To deal with this problem, some techniques concentrates on computing the representative graphs for a given set. Jiang et al. defines the median graph as the one whose sum distance to the other graphs of the set is minimized [11]. The median graph is drawn from the set, however, a more general concept of

A. Petrosino (Ed.): ICIAP 2013, Part I, LNCS 8156, pp. 452–461, 2013.

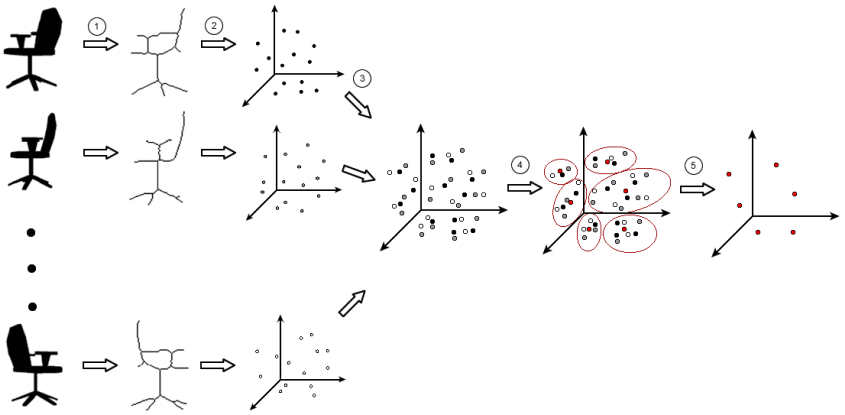

Fig. 1. Overview of the proposed algorithm. Each view of an object class is represented as a tree (transition 1). After embedding trees into the vector space (transition 2), we bring the embedded points to the same dimension (transition 3). The k-means clustering algorithm is then used to learn the centroids, which forms the class representatives (transitions 4 and 5).

the set median graph, which is not constrained to come from the set, is also introduced in this work. Both the median and set median graphs are used as the set representatives. This method has a number of applications where the graph representatives are needed. However, the cost of computing the such graphs are exponential in the number of database graphs [3].

The concept of graph embedding has been used by a number of approaches to map the graphs into the vector space, where the final matching is performed. The main advantage of moving from graph space to vector space is two-fold: using the representational power of graphs and approximating some computationally expensive algorithms in polynomial time. For instance, Demirci et al. [6,5] propose an algorithm using a graph embedding such that each vertex in the graph correspond to a point and the distance between a vertex pair is realized by its corresponding points. Once this transformation is computed, the matching and distance between two graphs can be easily obtained using their point sets. The reader is referred to [10,1] for an extensive review of graph embedding.

When graphs are embedded, the median and mean of these point sets can be easily computed in the vector space. Ferrer et al. [7] propose such an algorithm, which first embeds the graphs into the vector space and computes the median for the corresponding point sets using the Weiszfeld algorithm [17]. Obtaining the median in the vector domain requires simpler operations than those in the graph domain. Given a median in the vector space, the algorithm then uses a triangulation procedure that enables to go back to the graph domain in order to obtain an approximation of the median graph.

Lozano and Escolano [12] study the related problem of learning a probabilistic graph from a set of input graphs, in which node and edge probabilities show relative frequencies of node and edges in the input graphs. Torsello and Hancock [15,14] use a union tree model to capture the within-class variation. This model is learned from the class members by optimizing a minimum description length criterion. The authors' more recent approaches on this subject concentrates on tree edit-distance and tree clustering [16,18].

Although powerful, after computing the representative graph for a given set, one still faces the computational complexity of graph matching in order to obtain the most similar representative graph to the query. In this paper we propose an algorithm which represents the graphs belonging to a particular set (or, an object class) as points through graph embedding and operates in the vector space to compute the representative of the set. We use the k-means clustering algorithm to learn centroids forming the representatives. Once the representative of each set is obtained, we do not go back to the graph domain as done by the previous work. Instead, we embed the query into the vector space and compute the matching in the vector domain. The query is classified into the most similar representative of a set. This way, we are able to overcome the complexity of graph matching and still perform the classification for the query. Experimental evaluation of the proposed work shows that our approach is more efficient and effective than exhaustive search of the query in the original database without the representatives. Figure 1 shows an overview of the proposed approach.

The rest of the paper is organized as follows. In Section 2, we describe the graph embedding procedure, which takes a set of input graphs (in particular, trees) into the vector space and presents how the representative points are computed. Since the embedding procedure we used in this paper works with trees, we also introduce the process of representing input objects as trees in this section. In Section 3 we evaluate the proposed framework by conducting experiments in the domain of object recognition. Finally, we finish the paper with the conclusion and future work in Section 4.

2 Embedding into Vector Spaces and Computing Representative Points

Graph embedding represents an input graph as a set of points in a vector space such that the distance between a pair of vertices in the graph space realized by the distance between their corresponding points in the vector space. In this paper we follow the embedding procedure used in [5]. Since this embedding process is designed for trees, our algorithm starts by representing input images as trees through skeleton points. After defining this process in Section 2.1, we present the embedding procedure and for embedded points, we describe how we compute their representatives in Section 2.2 and Section 2.3, respectively.

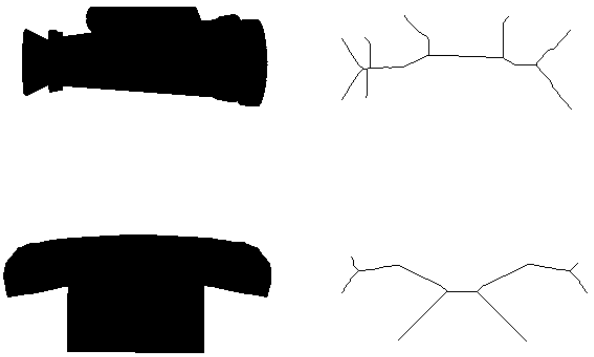

Fig. 2. Skeleton points of two silhouette images in left are shown in right

2.1 Tree Representations for Input Images

A skeleton point (or, shock point) is defined in [8] as the dynamic view of the medial axis where the propagation of waves from the shape boundary results in the formation of singularities. In [2] medial axis is described as the locus of centers of circles inside the region which are bi-tangent to the boundary in at least two places. Each skeleton point p is associated with a 3-dimensional vector $v(p) = (x, y, r)$, where (x, y) are the Euclidean coordinates of the point and r is the radius of the maximal bi-tangent circle centered at the point. Each shock point represented as a vertex in the skeleton graph, which takes over significant role especially on structural and statistical pattern recognition. Each pair of skeleton points in the graph is connected by an edge whose weight reflects the Euclidean distance between them. We convert the graph to a tree by computing its minimum spanning tree. As a result, nodes correspond to skeleton points, and edges connect nearby skeleton points. The root of the tree is the node that minimizes the sum of the shortest path distances to all other nodes in the tree. Figure 2 shows two silhouette images and their skeleton points.

2.2 Embedding through Caterpillar Decomposition

The embedding procedure used in this paper is based on a graph-theoretical concept, caterpillar decomposition, which can be defined as a way to represent a tree by its edge-disjoint root-leaf paths. This concept has been used before by Demirci [4] for efficient retrieval of database graphs, which are similar to a given query. The concept of the caterpillar decomposition is described in a sample tree shown in Figure 3. The two edge-disjoint root-leaf paths between a and h and a and i are called level 1 paths and represent first two paths in caterpillar decomposition. If we remove these two level 1 paths from the tree, we are left with the subtree rooted at c in which two edge-disjoint root-leaf paths exist.

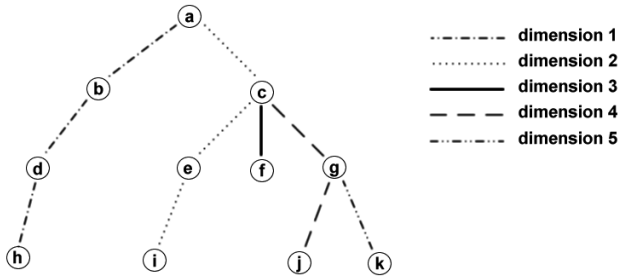

Fig. 3. The union of each edge-disjoint (sub)root-leaf paths forms the caterpillar decomposition of the tree. Each path in the caterpillar decomposition corresponds to a dimension in the vector space.Thus, the tree in this example is represented in a 5-dimensional vector space.

These are the paths between c and f and c and j, called level 2 paths, which represent the other two paths in caterpillar decomposition. Upon removal of these paths, only one path, between g and k, remains and this path is the level 3 path. If removing the level 3 path had left additional connected components, the process would be repeated until all the edges in the tree had been removed. The union of the paths is called the caterpillar decomposition of the tree. The total number of paths in the caterpillar decomposition specifies the dimensionality of the vector space into which the tree is embedded, i.e., each path corresponds to a different dimension. Thus, the tree shown in Figure 3 is embedded into 5 dimensional vector space. The reader is referred to [13] for a detailed description of the caterpillar decomposition and its properties.

To compute the coordinate of a vertex in the vector space, we follow the path from this vertex up to the root (the root is always embedded in the origin). The weight of the potion of the path in each level specifies the value of the coordinate for the corresponding dimension. As an example, the path between k and a consists of three edges; one between k and g corresponding to dimension 5, the other one between g and c corresponding to dimension 4, and the last one between c and a corresponding to dimension 2. Assuming that each edge has a weight of 1.0, the coordinate of vertex k is (0,1,0,1,1). It is easy to see that this embedding is distortion-free under the ℓ_1 norm. That is, the weight of the unique path between any pair of vertices in the tree equals the ℓ_1 distance of their corresponding points in the vector space.

One may notice that this embedding procedure is not unique and depends on the selection of the root and the root-leaf paths. To be consistent in our procedure, the vertex with total minimum distance to all other vertices is chosen as the root and the paths in the caterpillar decomposition is selected based on their weights. This way we ensure that the vertices of one tree is always represented by the same points in the vector space.

Note that since the dimension of the vector space into which an input tree is embedded is defined by its caterpillar decomposition, two trees with different structures are likely to be embedded into different dimensions. Thus before we proceed with computing the representative points of the set, we should perform a post-processing step whose objective is to bring the embedded points into the same dimension. There are two possible ways to do this. One is to bring the higher dimensional embeddings into the lower dimensions by some dimensionality reduction techniques, such as PCA. In general this is not possible without introducing some distortion. The other one is to bring the lower dimensional embeddings up to the higher dimensions by padding them with zeros. As used in [5], we employed the latter method since it does not result in distortion.

2.3 Computing Representative Points

Having embedded the input trees of one class into the same vector space as a set of points, we compute the representative points based on the k-means clustering algorithm. Given a set of n points in $d-$dimensional space $\Re^d$, the objective here is to find k representative points in $\Re^d$ such that the distance between each point to its closest representative point is minimized. More specifically, the k-means algorithm starts by initializing the cluster representatives. After assigning each point to its nearest cluster representative, the representatives are recomputed based on the current cluster memberships. This last step is repeated until no further change in the assignment of the data to new cluster representatives occurs. Although k-means is an effective clustering approach, it is well-known that it may converge to a local optimum. In order to avoid this, we start the algorithm from several initial cluster representatives, increasing the likelihood of obtaining a global optimum.

In the original formulation of k-means, each point has an equal importance in computing the representative of the cluster. However, in our case each point has a different importance reflected by its weight. Therefore, to adapt the original formulation of k-means to our application, we follow the following process. For each point $v(p) = (x, y, r)$ in the set, we insert as many points to location (x, y) as the value of r. This process moves cluster representatives to points with higher weights.

Given a set of images belonging to the same object class, we apply the above process to obtain their representative points. In order to classify the query, it is first embedded into the vector space and the distance with the representative points of each object class in the vector space is computed. As mentioned before, the proposed framework does not require an extra step to go back to the graph domain avoiding the computational complexity of graph matching.

3 Experiments

In this section we evaluate the proposed approach in the context of an object recognition experiment. We use a silhouette dataset, consisting of 9 different

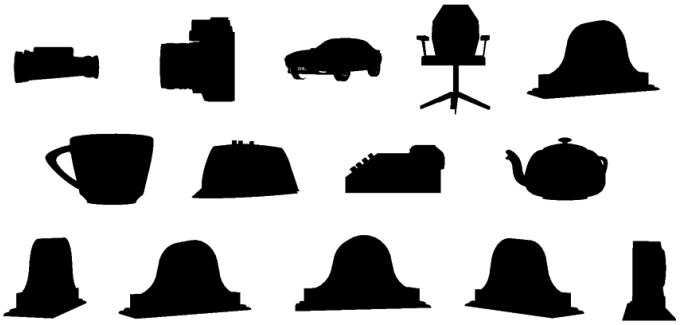

Fig. 4. The top two row presents sample silhouettes from the dataset, while the bottom row shows some sample views for the same object class

objects with 180 views for each, i.e., objects are rotated 2 degrees in depth and for each rotation its 2D view is captured. The top two rows of Figure 4 presents sample silhouettes, while the bottom row shows some sample views for the same object. To test the proposed approach, we selected 19 equidistant views of each object. Each silhouette in the dataset is represented as a rooted tree using the method described in Section 2.1.

To evaluate the proposed approach on this database, we first compute the representative points for each class in the database using the proposed framework. Each view in the database is then used as a query. We say that the proposed approach is correct if the closest representative set to the query has been formed using the views of the query class. In our experiments we use the Hausdorff distance [9] to compute the dissimilarity between two point sets. Given two point sets, $P = \{p_1, \ldots, p_m\}$ and $Q = \{q_1, \ldots, q_n\}$, the Hausdorff distance is defined as follows:

$$d_H(P, Q) = \max\{\sup_{p \in P} \inf_{q \in Q} d(p, q), \sup_{q \in Q} \inf_{p \in P} d(p, q)\}, \tag{1}$$

where $d(p, q)$ is the distance between points p and q. Since our embedding is isometric under ℓ_1, we used the ℓ_1 norm (or, Manhattan distance) in the vector domain. It is well-known that the Hausdorff distance is sensitive to outliers. Thus, instead of using the original formulation, for each point in the first set, we compute its closest distance to the second set and took the average of all distances,i.e., the modified Hausdorff distance we used in this paper is computed as follows:

$$d'_H(P, Q) = \max\{\frac{1}{m} \sum_{p \in P} \inf_{q \in Q} d(p, q), \frac{1}{n} \sum_{q \in Q} \inf_{p \in P} d(p, q)\}, \tag{2}$$

where m is the size of set P, and n is the size of set Q.

The dataset used in the experiments has been employed before in [6], where the distance between each view and each of the remaining database entries is computed. This approach reports the average distance between a pair of objects.

Table 1. Recognition results for proposed, median, and exhaustive search algorithms for different sampling resolutions. The proposed algorithm outperforms the two algorithms in both experiments. The experiment in the bottom row is performed with higher sampling resolution than the top row.

Experiment	Proposed	Median	Exhaustive
RECOGNITION	97.4%	40.1%	67.3%
RECOGNITION	97.8%	42.0%	88.2%

That is given two objects A and B, for each 19 view of object A , its closest view of object B is found and the average over the resulting distances is computed. Since we aim at computing the class representatives, the result of this experiment is not appropriate for comparison purposes. However, the performance of our algorithm in the above experiment is still compared with two alternative approaches. In the first approach, we replace representative points of each class with the median view drawn from this class, i.e., from the set of all views used to construct the representative points, we choose the view whose sum distance to all other views is minimum. In the second approach, we eliminate the representative points and compare the query directly to each of the database views except itself.

The results are shown in the top row of Table 1 and reveal that the proposed framework outperforms both median, and exhaustive search algorithms. Specifically, the proposed framework, median, and exhaustive search algorithms achieve %97.4, %40.1, and %67.3 recognition rates, respectively. Since the proposed approach takes into consideration all views of a class for generating its representative points, the recognition results of our approach are better than the median algorithm. Upon taking a closer look at the results, the exhaustive search algorithm could have been expected to achieve a higher recognition rate. However, we should note that the skeletons are very sensitive to noise, thus even skeletons of neighboring views can be quite different in the database. When the sampling resolution increases, we expect the performance of exhaustive search to improve.

To measure how the sampling resolution of the database effects these results, we take all views in the database and repeat the above experiment. The retrieval performance of the proposed, median, and the exhaustive search algorithms are recorded as %97.8, %42.0, and %88.2, respectively (bottom row of Table 1). This demonstrates that the proposed algorithm is stable under the sampling resolution compared to the other algorithms. In fact, the results suggest that even only 19 equidistant views of each object are sufficient to generate a good representative for the class. The main advantage of our algorithm, on the other hand, lies in the number of comparisons it performs. For a single query, while the exhaustive search performs $n \times m$ comparisons, where n is the number of classes and m is the number of views per class, our approach needs only n comparisons.

To test the sensitivity of the proposed algorithm to perturbation, we perturbed the representative points of each class by first deleting a randomly selected 5%, 10%, and 20% of points for all its views in the vector space and applying the k-means technique to the resulting points. The recognition rates were recorded

as 97.4%, 97.2%, and 96.9%, respectively for the database of 19 views per object. Note that a number of embedded points are placed nearby in the vector space. Thus, removing some of these points do not change the position of the cluster representatives much. Although not a true occlusion experiment, these results reflect the algorithm's stability to missing data, which is required for occlusion experiments.

4 Conclusions

In this paper we have proposed a new framework for graph-based object recognition based on graph embedding. Given a set of objects grouped in different classes, our algorithm first represents the objects as points through graph embedding and generates one representative for each class in the vector space. After embedding the query into the vector space, its classification is performed by comparing to each class representative only. This process precludes the need for comparing the query against each database view. Experimental evaluation of the framework, including a comparison with the two approaches demonstrates the efficacy of the overall algorithm. In addition, a set of perturbation experiments show the stability of the algorithm against missing data. We plan to extend the proposed approach in a number of ways. Performing a more comprehensive experimental test using a larger dataset with different image formats, measuring the recognition performance as a function of increasing database size using different alternative approaches, employing different clustering algorithms in the vector space, and applying our framework to different domains are our future plans.

References

1. Babilon, R., Matousek, J., Maxová, J., Valtr, P.: Low-distortion embeddings of trees. Journal of Graph Algorithms and Applications 7(4), 399–409 (2003)
2. Blum, H.: Biological shape and visual science (part i). Journal of Theoretical Biology 38(2), 205–287 (1973), http://www.sciencedirect.com/science/article/B6WMD-4F1Y9M7-D5/2/1b17959a78e759a89f524d9f3eae0938
3. Bunke, H., Münger, A., Jiang, X.: Combinatorial search versus genetic algorithms: a case study based on the generalized median graph problem. Pattern Recognition Letters 20(11-13), 1271–1277 (1999), http://dx.doi.org/10.1016/S0167-86559900094-X
4. Demirci, M.: Retrieving 2D shapes using caterpillar decomposition. Machine Vision and Applications 24(2), 435–445 (2013)
5. Demirci, M., Osmanlioglu, Y., Shokoufandeh, A., Dickinson, S.: Efficient many-to-many feature matching under the ℓ_1 norm. Computer Vision and Image Understanding 115(7), 976–983 (2011), http://dx.doi.org/10.1016/j.cviu.2010.12.012
6. Demirci, M., Shokoufandeh, A., Keselman, Y., Bretzner, L., Dickinson, S.: Object recognition as many-to-many feature matching. International Journal of Computer Vision 69(2), 203–222 (2006)

7. Ferrer, M., Valveny, E., Serratosa, F., Riesen, K., Bunke, H.: An approximate algorithm for median graph computation using graph embedding. In: 19th International Conference on Pattern Recognition, pp. 1–4. IEEE (2008)
8. Giblin, P., Kimia, B.: On the local form and transitions of symmetry sets, medial axes, and shocks. International Journal of Computer Vision 54(1-3), 143–156 (2003)
9. Huttenlocher, D., Klanderman, D., Rucklige, A.: Comparing images using the Hausdorff distance. IEEE Transactions on Pattern Analysis and Machine Intelligence 15(9), 850–863 (1993), `citeseer.ist.psu.edu/huttenlocher93comparing.html`
10. Indyk, P.: Algorithmic aspects of geometric embeddings. In: Proceedings of the 42nd Annual Symposium on Foundations of Computer Science (2001)
11. Jiang, X., Münger, A., Bunke, H.: On median graphs: Properties, algorithms, and applications 23(10), 1144–1151 (2001)
12. Lozano, M., Escolano, F.: Protein classification by matching and clustering surface graphs. Pattern Recognition 39(4), 539–551 (2006)
13. Matousek, J.: On embedding trees into uniformly convex banach spaces. Israel Journal of Mathematics 237, 221–237 (1999)
14. Torsello, A., Hancock, E.: Graph embedding using tree edit-union. Pattern recognition 40(5), 1393–1405 (2007)
15. Torsello, A., Hancock, E.R.: Learning shape-classes using a mixture of tree-unions. IEEE Transactions on Pattern Analysis and Machine Intelligence 28(6), 954–967 (2006)
16. Torsello, A., Robles-Kelly, A., Hancock, E.: Discovering shape classes using tree edit-distance and pairwise clustering. International Journal of Computer Vision 72(3), 259–285 (2007)
17. Weiszfeld, E.: Sur le point pour lequel la somme des distances de n points donns est minimum. Thoku Mathematical Journal 43, 355–386 (1937)
18. Xiao, B., Torsello, A., Hancock, E.: Isotree: Tree clustering via metric embedding. Neurocomputing 71(10), 2029–2036 (2008)