A maximum diversity-based path sparsification for geometric graph matching

Abd Errahmane Kiouche, Hamida Seba, Karima Amrouche

A Maximum Diversity-based Path Sparsification for Geometric Graph Matching

This is a preprint: the final paper is published in PRL https://doi.org/10.1016/j.patrec.2021.09.019

Abd Errahmane Kiouche[1,2], Hamida Seba[1]*, Karima Amrouche[2]

[1]University lyon 1, LIRIS, CNRS UMR 5205, Villeurbanne, 69100, France

[2]Laboratoire de Communication dans les Systèmes Informatiques (LCSI)

Ecole nationale Supérieure d'Informatique

BP 68M, 16309, Alger, Algérie.

Abstract

This paper presents an effective dissimilarity measure for geometric graphs representing shapes. The dissimilarity measure is a distance that combines a sparsification of the geometric graph based on the maximum diversity problem and a new node embedding that captures the topological neighborhood of nodes. The sparsification step aims to correct the misdistribution of nodes on the geometric graph induced by the noise of image handling. Computational experiments on two popular datasets indicate that our approach retains the form of the shapes while decreasing the number of processed nodes which yields interesting results both on accuracy and time processing[1].

Keywords: Geometric graphs, Shape matching, Graph matching, Graph sparsification, Maximum diversity problem

1 Introduction

Graphs are a highly appreciated modelling tool which has proven its expressiveness in several domains and applications especially in pattern recognition [31] where graphs introduce an abstraction from data to a model and help categorising objects. A graph $G = (V, E)$ is a set of vertices V, also called nodes, that are connected to each other by a set of edges E. Vertices and edges are used to represent objects and their interactions. When the considered objects have specific shapes, the corresponding graphs are called geometric. So, a geometric graph is a graph that somehow retains the geometric form or silhouette of the object that it represents. This is achieved by augmenting the structure of the graph with attributes that render the geometric form of the object such as coordinates for the vertices or angles between edges or by directly modelling the parts of the object that define its shape as with Blum's skeleton [2].

*Corresponding author: hamida.seba@univ-lyon1.fr

[1]This is a preprint: the final, more elaborated version, is published in PRL https://www.sciencedirect.com/science/article/abs/pii/S0167865521003457

Object comparison is a basic task encountered in almost all human activities. Graph matching states for the methods that are able to measure how much similar (or dissimilar) are two graphs and hence infer a comparison between the corresponding objects. These methods can be classified into two main categories : exact graph matching, also called isomorphism, that looks for an exact mapping between the vertices and edges of the compared graphs and fault-tolerant graph matching that aims to quantify the degree of similarity between two graphs. Fault-tolerant graph matching is more suitable for classification and search/rank based applications. Several graph similarity/dissimilarity measures have been proposed in the literature and several approaches have been used including genetic algorithms [29], neural networks [17], probability theory [18], clustering techniques [26], spectral methods [24], decision trees [16], etc. We refer the reader to [4, 5, 31] for more exhaustive surveys.

One of the most flexible graph matching method is graph edit distance which defines the similarity of graphs by the minimum costing sequence of edit operations that convert one graph into the other [3, 25]. An edit operation is either an insertion, a suppression or a re-labelling of a vertex or an edge in the graph. A cost function associates a cost to each edit operation. However, computing graph edit distance is NP-hard in general [33] which motivated several approximating solutions. A comprehensive survey on graph edit distance and the approaches proposed to compute it can be found in [8].

Geometric graph matching is a special case of graph matching that aims to quantify how much two geometric graphs are alike. Several geometric graph matching methods are extensions of general graph matching methods.

In [28], the authors captures the shape of a 2D objects with shock graphs, i.e., a 2D-object is decomposed into a set of qualitative parts, captured in a directed acyclic graph. To match two shock graphs, the authors rely on a algorithm that reduces shock graphs into rooted trees and use eigen-decomposition of the obtained tress to found similarities among them. In [11], the authors propose a redefinition of edit distance to match shock graphs by introducing new edit operations corresponding to shock transitions. In [1], the authors formalise the geometric graph matching problem in a maximum likelihood estimation framework and use the expectation maximisation technique to estimate the match between two graphs. Paths have been used to match geometric graphs in [20]. In this approach, each path of at most k edges is modelled by a superedge and described by a path descriptor that summarises the curve between the endpoints of the path. To find matching between descriptors, they formalise the problem as an integer quadratic program and use weighted random walks to find an approximate solution. In [14], the authors extended, to geometric graphs, an approximation of edit distance proposed in [23] and [22]. It is a polynomial-time framework based on a fast bipartite assignment procedure mapping nodes and their local structures of one graph to nodes and their local structures of another graph. The local structure in [14] is a node embedding that includes the values of angles between edges to capture the shape of the geometric graph. In [19] authors propose a geometric graph matching algorithm based on Monte Carlo tree search. More recently, [6] describe a distance measure between geometric graphs where every node has a coordinate position in a two-dimensional plane. This distance combines the Euclidean distance between vertices and the orientation and length of edges.

Apart from the work of [20] that reduces the size of the geometric graph before matching, existing solutions use the whole nodes of the geometric graph. However, this can bias the matching process as the geometric graphs can have regions with a high concentration of nodes and more sparse regions where nodes are dotted here and there. This is related to the image segmentation process and to the noise induced by how the object is transformed into a graph. Moreover, the time complexity of existing methods is function of the number of nodes in the graph and reducing this number will benefit scalability. In fact, the fastest graph matching methods are approximations of graph edit distance using bipartite matching and the Hungarian algorithm are $O(n^3)$ where n is the number of nodes in the graph [23].

Recently, graph compression and reductionmethods that aim to reduce the size of the graph while retaining part or the whole properties of the graph are proposed to deal with the complexity of graph data and its increasing size in several domains and especially in pattern recognition [30]. Graph sparsification is a manner to simplify a graph by reducing its number of edges and/or nodes [7]. The simplest way to sparsify a graph is to sample some of its edges or nodes. However, edges and nodes have not the same importance depending on the graph and the application and consequently how to choose the retained edges and nodes is very important.

In this paper, we provide the following contributions:

- We introduce a new geometric graphmatchingmethod that deterministically select a subset of the nodes of a geometric graph by formulating this selection as a Maximum Diversity Problem [13]. The maximum diversity problem consists of selecting a subset of m elements from a set of n elements in such a way that the sum of the distances between the chosen elements is maximised. The maximumdiversity problemis NP-hard but due to its theoretical significance and applications, various heuristics and meta-heuristics are provided [15].
- We provide a new node embedding that retains the topological information of each node and consequently is adapted for geometric graph matching.

Our experiments show that the obtained sparsified geometric graphs retain perfectly the original shapes and yield good performance in both accuracy and execution time.

In the remainder of the paper, we describe the new proposed approach in Section 2 and focus in its experimental validation in Section 3. Then, Section 4 concludes the paper with some future research hints.

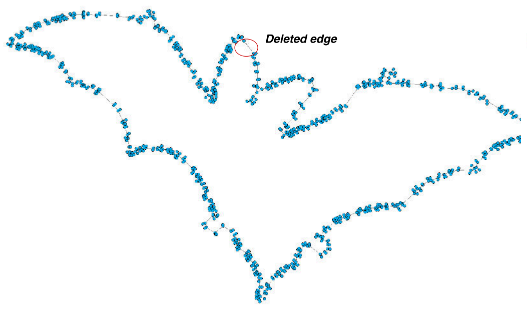
2 The Proposed Approach

In this section, we present a graph distance especially devised for geometric graphs representing shapes. We use this distance as a dissimilarity measure between two geometric graphs G_1 and G_2. Algorithm 1 highlights the main steps of the computation of the proposed dissimilarity measure.

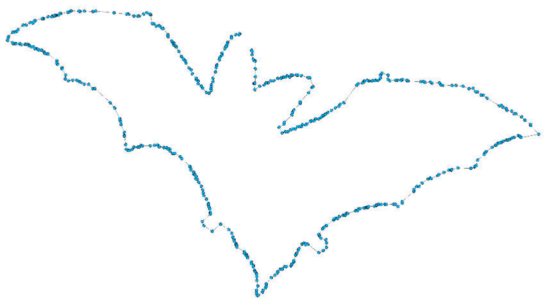
```
  Data: Two geometric graphs $G_1$ and $G_2$
  Result: A dissimilarity measure between $G_1$ and $G_2$
1 Approximate the longest paths $P_1$ and $P_2$ of $G_1$ and $G_2$ respectively using Algorithm 2 ;
2 Sparsify paths $P_1$ and $P_2$ using Algorithm 3 ;
3 Embed each node in the sparsified paths $P'_1$ and $P'_2$ into a vector using Algorithm 4 ;
4 Construct a complete weighted bipartite graph by linking vectors of $P'_1$ to vectors of $P'_2$ ;
5 Solve the minimum linear assignment problem;
6 return Minimum weighted matching cost ;
```

Algorithm 1: The proposed geometric graph distance computation

As highlighted by Algorithm 1, the first step of our approach is extracting the longest path, in term of the number of nodes in the path, from each shape graph. The extraction of the longest path



(a)



(b)

Figure 1: (a) A geometric graph representing a bat's shape. (b) The approximated longest path found after breaking the cycle by removing one edge.

allows to retrieve the main structure of the shape (see Figure 1b). However, finding the longest path in an undirected cyclic graph (as shape graphs) is a NP-hard problem [10]. Therefore, we use a new simple method to approximate the longest path which takes advantage from the characteristics of geometric graphs.

The approximation of the longest path is detailed in Algorithm 2. The main idea is to first delete the edge that may cause a cycle in the geometric graph (see Figure 1a, for an example). Note that such an edge can always be found in shape graphs since they contain some regions (subgraphs) which can be represented by simple plain paths. Then, the algorithm computes the shortest path connecting the two endpoints of the deleted edge using Breadth First Search (BFS). This shortest past is the requested approximated longest path of the geometric graph. Algorithm 2 finds the approximated longest path of a geometric graph $G = (V, E)$ in $O(|E|^2 + |E||V|)$ time steps in the worst case. An example of the longest path approximation is illustrated in Figure 1.

```
   Data: An undirected graph $G = (V, E)$
   Result: Approximated longest path in $G$
1  $max\_length \leftarrow 0$;
2  $longest\_path \leftarrow \{\}$;
3  for $(u, v) \in E$ do
4      $E \leftarrow E \setminus \{(u, v)\}$ ;
5      path = compute shortest path from $u$ to $v$ ;
6      if $max\_length < |path|$ then
7          $longest\_path \leftarrow path$;
8          $max\_length \leftarrow |path|$;
9      end
10     $E \leftarrow E \cup \{(u, v)\}$ ;
11 end
12 return $longest\_path$;
```

Algorithm 2: Approximating the longest path in a cyclic geometric graph

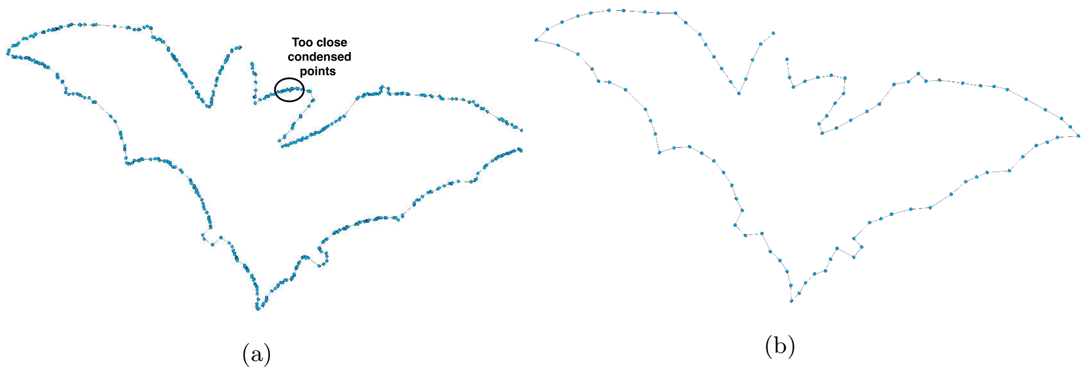


Figure 2: (a) A path before sparsification. (b) the corresponding sparsified path with $N = 100$.

Algorithm 2 returns a path that matches the geometric form of the shape but this path does not correct the misdistribution of nodes within the path. In fact, even with a path representation of a shape, denser regions may subsist as depicted in Figure 2a and may bias the comparison of the two paths. To deal with this issue, we propose to sparsify the obtained path to keep only a small number of nodes distributed as uniformly as possible across the path. This will also allow us to normalise the number of nodes within compared paths which makes their comparison much more easier. Moreover, this will reduce the number of nodes in the compared paths and consequently speed up the dissimilarity computation time.

Our path sparsification process consists in choosing a number $N \lll |P|$ of nodes from the path $P = \{v1, ..., v_{|P|}\}$ in such a way that the diversity of the chosen nodes is maximised and all chosen nodes are distributed as uniformly as possible across the path. However, the maximum diversity problem is an NP-hard problem [9]. Therefore, we use a greedy algorithm to maximise the diversity of the chosen nodes. For this, we extend the constructive heuristic $C2$ proposed in [9] for the maximum diversity problem so that the chosen nodes are distributed uniformly across the path. The greedy sparsification procedure, we propose, is given by Algorithm 3.

Algorithm 3 starts, as in $C2$ [9], with one selected node (line 3). Then, at each iteration, node v_{i*} that has the greatest minimal distance to the already selected nodes (S) is chosen. This process is repeated until N nodes are selected. However, unlike the original procedure $C2$ presented in [9], the composite distance in our case $d[v_i]$ of vertex v_i represents its euclidean distance to the closest node belonging to the set S (line 7) and not the sum of distances between v_i and all the nodes of S. This slight modification of the composite distance allows us to get a set of nodes distributed as uniformly as possible across the original path. Figure 2 illustrates an example of the output of the sparsification algorithm. For a path $P = \{v1, ..., v_{|P|}\}$, the sparsification algorithm takes $O(|V|N)$ time steps in the worst case since $|P| \leq |V|$ always holds ($|V|$ is the number of nodes in the initial geometric graph).

After path sparsification, we embed each node into a vector. To achieve this, we represent each node v_i of the sparsified path by a vector $A[i]$ of K elements embedding its neighborhood geometric topology, such that the j-th element of the vector (i.e., $A[i][j]$) is related to the j-th closest neighbor of v_i. For instance, in Figure 3, the 5 first closest neighbors of node B are D, A, S, O *and* F, ordered in the increasing order of their euclidean distance to B. The value $A[i][j]$

```
Data: A path graph $P = \{v1, ...., v_{|P|}\}$
A positive number $N$
Result: A sparsified path $P'$ with $N$ nodes
$P' \leftarrow \{\}$ ;
$P_{copy} \leftarrow P$;
$S \leftarrow \{v_1\}$;
/* Select $N$ diversified nodes                                        */
while $|S| < N - 1$ do
    for $v_i \in P_{copy} \setminus S$ do
        $d[v_i] \leftarrow min\{euclidean\_distance(v_i, v_s), v_s \in S\}$
    end
    Find $v_{i^*}$ such that $d_{i^*} = max\{d[v_i], v_i \in P_{copy} \setminus Sel\}$;
    $SS_{el} \leftarrow Sel \cup \{v_{i^*}\}$;
    $P_{copy} \leftarrow P_{copy} \setminus \{v_{i^*}\}$ ;
end
/* Construct the sparsified path                                       */
for $i = 1$ to $|P|$ do
    if $v_i \in S$ then
        Add $v_i$ to $P'$ ;
    end
end
return $P'$;
```

Algorithm 3: The Path sparsification procedure

represents the ratio between the real euclidean distance between v_i and its j-th closest neighbor (with respect to the euclidean space) and the length of the path connecting the two nodes multiplied by the sum of all angles in this path. More formally, let v_i be a node in the sparsified path (i.e., $v_i \in P'$) and $v_m \in P'$ is its j-th closest neighbor in the euclidean space, the two nodes are connected in P' by a subpath $P_{im} = \{v_i, v_{i+1}, ..., v_m\}$ (or $P_{im} = \{v_m, v_{m+1}, ..., v_i\}$ if $i > m$). We suppose without loss of generality that $i < m$, the value of $A[i][j]$ is given by:

$$A[i][j] = \frac{d(v_i, v_m)}{\sum_{l=i}^{l=m-1} d(v_l, v_{l+1})} \sum_{l=i}^{l=m-2} \widehat{v_l v_{l+1} v_{l+2}} \tag{1}$$

where $d(u, v)$ is the euclidean distance between nodes u and v using their Cartesian coordinates, and the angle $\widehat{ABC}$ between three points A, B and C is computed using the Al-kashis theorem given by:

$$\widehat{ABC} = \arccos\left(\frac{d(B,C)^2 + d(A,B)^2 - d(A,C)^2}{d(B,C)d(A,B)}\right) \tag{2}$$

Note that the total time complexity of embedding all the nodes of a sparsified path is $O(KN^2)$ the worst case.

Our node embedding scheme is detailed in Algorithm 4, it's adapted to geometric graphs since it takes into account the distance distortion and path orientation between nodes. In addition, it allows to embed the neighborhood geometric topology of each node into a vector, which will be very useful in the computation of the dissimilarity between geometric graphs.

```
  Data: A sparsified path $P' = \{v1, ...., v_N\}$
  A positive integer $K$: size of the resulting vector
  Result: A matrix $A$ of size $N \times K$
1 for $v_i \in P'$ do
2     Find the top $K$ nearest neighbors of $v_i$ ;
3     for $j = 1$ to $K$ do
4         compute $A[i][j]$ using Equation 1;
5     end
6 end
7 return $A$ ;
```

Algorithm 4: Geometric node embedding

Let G_1 and G_2 be two shape graphs with their corresponding sparsified paths P'_1 and P'_2 respectively, and A_1, A_2 are two matrices representing the embedding of the nodes of P'_1 and P'_2 respectively. To compare the two graphs G_1 and G_2, we construct a complete bipartite graph with two sets of nodes S_1 and S_2 where S_1 (resp. $S2$) represent the set of the vectors corresponding to the nodes of the path P'_1 (resp. P'_2). Note that each edge $e = (A_1[i], A_2[j])$ is weighted with w_{ij} which is the Euclidean distance between node vectors $A_1[i]$ and $A_2[j]$ (i.e., $w_{ij} = \sqrt{\sum_{m=1}^{K} (A_1[i][m] - A_2[j][m])^2}$). The construction of the bipartite graph takes $O(KN^2)$ time steps. Figure 4 illustrates an example of such bipartite graph.

Finally, we use the minimum weighted matching $cost_{min}$ in the resulting bipartite graph to compute the final dissimilarity between the two graphs. To find the minimum weighted matching

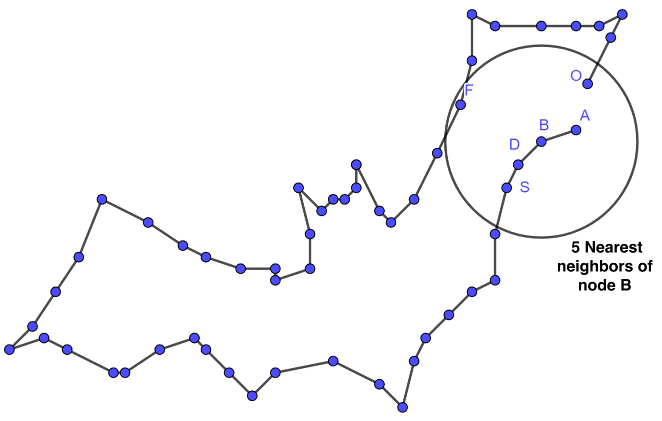


Figure 3: Example of a node's neighborhood

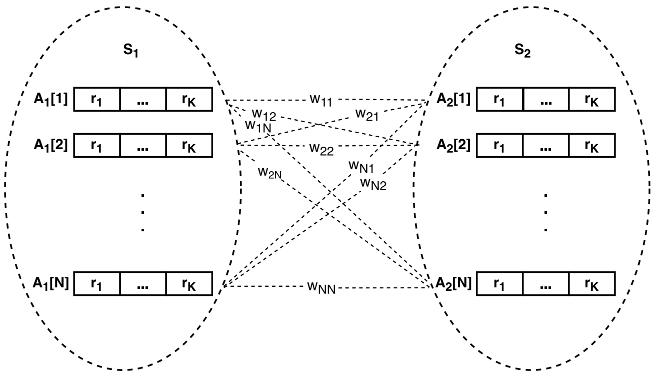
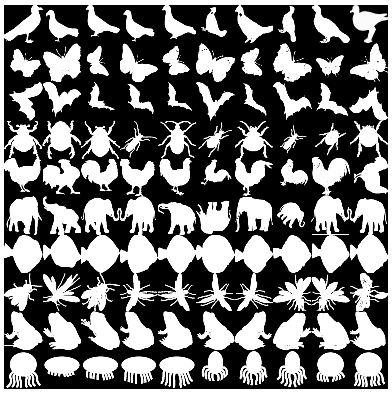


Figure 4: Bipartite Graph Construction.

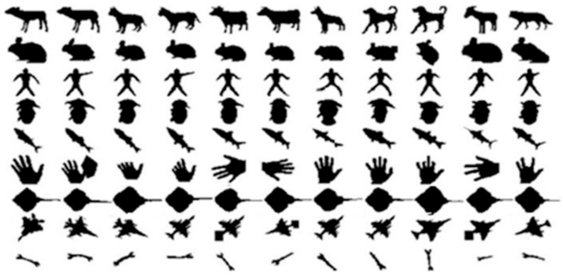
we solve a linear assignment problem using the well-known Hungarian Algorithm [12]. The final geometric graph distance between the two graphs is equal to the minimum weighted matching cost $cost_{min}$

3 Experimental Evaluation

To assess the effectiveness and the performance of our method, we use two shape datasets transformed into geometric graphs. The construction of graphs from shapes and their characteristics are described in the following subsection. The recognition performance is evaluated in terms of runtime and precision.



(a)



(b)

Figure 5: (a) MPEG-7 dataset [21] . (b) Kimia99 dataset [27] .

Table 1: Characteristics of the datasets

	#Graphs	Avg #nodes	Avg #edges
MPEG7	100	1342	1351
Kimia99	99	748	751

3.1 Datasets

We use two known benchmarks for shape recognition. The first one, is a dataset including 100 shapes of animals and insects with 10 shapes in each class chosen from the dataset MPEG-7 [21] (see Figure 5a). The second dataset is Kimia99 [27] (see Figure 5b) which consists of 9 different classes of objects with 11 shapes in each class.

To transform shapes into geometric graphs, we use image segmentation techniques to recognise shapes from images and then represent them with graphs. We used the framework of graph construction from images proposed in [14], which is based on the Line Segment Detection (LSD) algorithm [32]. An example of this transformation is illustrated in Figure 6. The characteristics of the obtained graphs are summarised in Table 1.

3.2 Recognition performance

All experiments have been carried on a machine having an i7-9700 (3GHZ) CPU with 64 gigabytes of RAM, all algorithms described in the previous section are implemented using Python 3.7 under windows 10 (64 bits).

The first part of experiments consists in setting the parameter N, which represents the number of nodes in each sparsified path. For this, we've tested tree values of N for the two datasets. These values are $N = \{50, 100, 200\}$ for MPEG-7 and $N = \{20, 50, 80\}$ for KIMIA99. In order to evaluate the recognition performance, each shape graph has been used as a query and compared to all other graphs in the same dataset. The measured metric is the so-called bull's eye retrieval rate [21]. The bull's eye score for each query is the number of shapes from the same class among the $2C$ most similar shapes, where C is the number of shapes in each class ($C = 10$ for MPEG-7 and $C = 11$ for

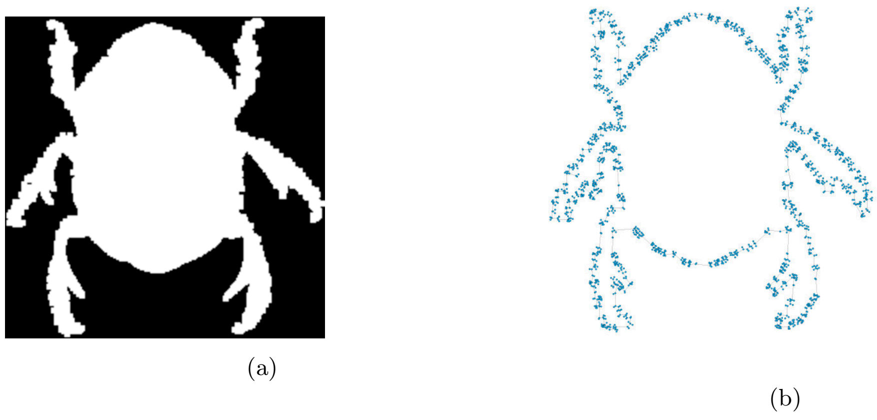

(a) (b)

Figure 6: (a) A beetle image from MPEG-7 dataset. (b) image of the constructed geometric graph from the shape.

Table 2: Recognition performances on MPEG7 dataset using different values of parameter N

N	bulls eye rate (%)
50	78.4
100	**80.5**
200	71.04

Kimia99). The total bull's eye retrieval rate is the ratio of the total sum of bull's eye scores to the highest possible number (which is 10×100 for MPEG-7 and 11×99 for Kimia99). Thus, the best possible rate is 100% [21]. The results are given in Table 2 and Table 3 where we can see that the best value of N in term of shape recognition is $N = 100$ for the MPEG-7 and $N = 80$ for Kimia99. Therefore, we set the number of nodes in the sparsified paths to $N = 100$ for the MPEG-7 dataset and $N = 80$ for Kimia99, in all the rest of experiments.

Table 4 gives the average run-time per graph of the sparsification step. Note that the extraction of the approximated sparsified longest paths from all shapes is done only once and offline. In fact, one of the advantages of our approach is that shape graphs can be stored as sparsified paths. No more need to keep the original geometric graphs. We notice that this operation is slow especially for the MPEG7 dataset. Indeed, the average number of nodes exceeds 1000 which makes this operation time consuming. However, this step is done offline and only once and its benefits are important as the number of nodes in the resulting paths is divided by more than 100 in this case. In fact, the

Table 3: Recognition performances on KIMIA99 dataset using different values of parameter N

N	bulls eye rate (%)
20	62.8
50	69.7
80	**72.81**

Table 4: Average runtime of the sparsification step

	Average runtime per graph (sec)
MPEG7	22.3
Kimia99	3.5

Table 5: Recognition performances on the MPEG7 dataset

K	Embedding runtime per graph (ms)	Distance computation runtime per comparison (ms)	bull's eye rate (%)	mAP (%)
5	**68**	**231**	74.5	59.82
10	142	244	80.5	66.19
20	329	273	**81.9**	69.22
30	535	287	**81.9**	70.76
40	780	296	81.4	**72.2**
50	1050	304	80.9	71.67

resultant paths contain 100 (resp. 80) nodes for MPEG-7 (resp. Kimia99) graphs. These values are much smaller than the number of nodes in the original graphs.

The second part of experiments aims to assess the recognition performance of our dissimilarity measure. The measured metrics are: the running time, the mean average precision (mAP) and the bull's eye retrieval rate [21]. Tables 5 and 6 show the recognition performances on the two datasets using different values of K where K is the size of the embedded vectors, which represents the size of the embedded geometric neighborhood.

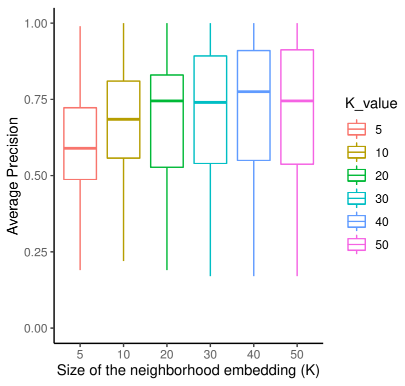
From the results, we notice that the running times are very promising. For $K = 5$, the embedding step takes less than 70 ms per graph in both datasets and the distance computation takes less than 0.25 sec. For the mAP and bull's eye score, the results are also satisfactory for the MPEG-7 dataset. Indeed, we achieve a bull's eye rate of $> 81\%$ and mAP $> 72\%$ within less than $300ms$ per comparison. For Kimia99, we obtain a maximum rate of bull's eye of 72% and a maximum mAP

Table 6: Recognition performances on the KIMIA99 dataset

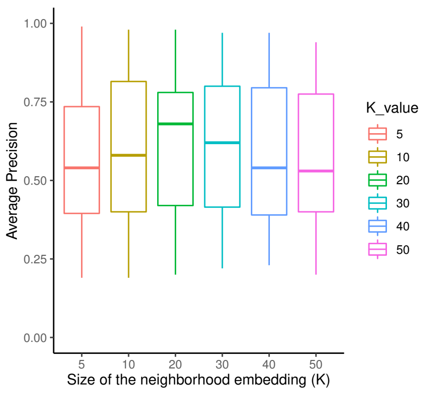
K	Embedding runtime per graph (ms)	Distance computation runtime per comparison (ms)	bull's eye rate (%)	mAP (%)
5	**46**	**134**	64.83	56.36
10	101	140	66.06	59.57
20	232	157	**72.81**	**61.46**
30	383	158	71.074	59.98
40	557	173	67.77	57.91
50	753	180	68.87	57.36

Table 7: Recognition performances without sparsification

Dataset	Embedding runtime per graph (ms)	Distance computation runtime per comparison (ms)	bulls eye rate (%)	mAP (%)
KIMIA99	2500	2744	19.65	22.8
Customized MPEG7	14299	21322	28.5	29.26



(a)



(b)

Figure 7: (a) The effect of parameter K on the AP (MPEG-7). (b) The effect of parameter K on the AP (Kimia99).

rate equal to 61.46%. The influence of the size of the embedded geometric neighborhood in each node vector (parameter K) is illustrated through the boxplots in Figures 7a and 7b on the two datasets. From the boxplot, we notice that the average precision is improved by increasing the value of K. However, the performance starts to drop when the neighborhood size becomes too large. The best values of K are 20 for the Kimia99 dataset and and 40 MPEG-7.

In order to assert the effectiveness of our sparsification procedure, we tested our geometric similarity algorithm (algorithm 1) on the two datasets without the sparsification step (without line 2 of Algorithm 1) using the best combination of parameters, the results are given in Table 7. We notice that the distance computation and the graph embedding steps are very slow, this is obviously due to the large number of nodes in the original paths. The distance computation is 70 (resp. 17) times slower without sparsification than with sparsification on the MPEG7 dataset (resp. KIMIA99). In addition, we can also notice that the sparsification allows also to improve recognition performance as illustrated by the values reported in Table 7. This is justified, by the fact that our sparsification tries to distribute as uniformly as possible across the shape by sparsifying denser and condensed regions which may bias similairty score.

4 Conclusion

We've presented, in this paper, a new dissimilarity measure for geometric graphs representing shapes. Our dissimilarity measure uses a sparsification method that aims to reduce drastically the number of nodes in graphs while preserving the geometric form of the shape. Furthermore, it aims to normalise the number of nodes in all graphs without adding any dummy node. The computational experiments conducted on known shape datasets show the effectiveness of the approach in term of the speed of the comparison as well as its accuracy. We've achieved a satisfactory bull's eye and mAP rates with relatively very short comparison running time. As a perspective, we plan to design a new node embedding scheme that describes in a more accurate way the geometric neighborhood topology in order to improve the precision of the recognition. It is also interesting to generalise the sparsification method to deal with the comparison of massive general graphs.

Acknowledgments

This work received grants from the IXXI research center (http://www.ixxi.fr), INS2I PEPS GRAM and Agence Nationale de Recherche under grant ANR-20-CE23-0002.

References

[1] Ayser Armiti and Michael Gertz. Geometric graph matching and similarity: A probabilistic approach. In *Proceedings of the 26th International Conference on Scientific and Statistical Database Management*, SSDBM 14, New York, NY, USA, 2014. Association for Computing Machinery.

[2] Harry Blum. Biological shape and visual science (part i). *Journal of Theoretical Biology*, 38(2):205 – 287, 1973.

[3] H. Bunke and G. Allerman. Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters-PRL*, 1(4):245–253, 1983.

[4] Horst Bunke and Kaspar Riesen. Recent advances in graph-based pattern recognition with applications in document analysis. *Pattern Recognition*, 44:1057–1067, 2011.

[5] Donatello Conte, Pasquale Foggia, Carlo Sansone, and Mario Vento. Thirty Years of Graph Matching in Pattern Recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 18:265–298, 2004.

[6] Shri Prakash Dwivedi and Ravi Shankar Singh. Error-tolerant geometric graph similarity and matching. *Pattern Recognition Letters*, 125:625 – 631, 2019.

[7] Wai Shing Fung, Ramesh Hariharan, Nicholas J.A. Harvey, and Debmalya Panigrahi. A general framework for graph sparsification. In *Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing*, STOC 11, page 7180, New York, NY, USA, 2011. Association for Computing Machinery.

[8] X. Gao, B. Xiao, D. Tao, and X. Li. A survey of graph edit distance. *Pattern Analysis Applications*, (13):113–129, 2010.

[9] Fred Glover, Ching-Chung Kuo, and Krishna S Dhir. Heuristic algorithms for the maximum diversity problem. *Journal of information and Optimization Sciences*, 19(1):109–132, 1998.

[10] David Karger, Rajeev Motwani, and Gurumurthy DS Ramkumar. On approximating the longest path in a graph. *Algorithmica*, 18(1):82–98, 1997.

[11] Philip N. Klein, Thomas B Sebastian, and Benjamin B Kimia. Shape matching using edit-distance: an implementation. In *SODA '01: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, page 781790, January 2001.

[12] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[13] C.C. Kuo, F. Glover, and K.S. Dhir. Analyzing and modeling the maximum diversity problem by zero-one programming. *Decision Sciences*, 24(6):1171–1185, 1996.

[14] Kamel Madi, Hamida Seba, Hamamache Kheddouci, and Olivier Barge. A graph-based approach for kite recognition. *Pattern Recognition Letters*, 87:186 – 194, 2017. Advances in Graph-based Pattern Recognition.

[15] Rafael Martí, Micael Gallego, Abraham Duarte, and Eduardo G. Pardo. Heuristics and metaheuristics for the maximum diversity problem. *Journal of Heuristics*, 19:591–615, 2013.

[16] Bruno T. Messmer and Horst Bunke. A decision tree approach to graph and subgraph isomorphism detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence - TPAMI*, 32(12):1979–1998, 1999.

[17] Alessio Micheli. Neural network for graphs : A contextual constructive approach. *IEEE Transactions on Neural Networks*, 20(3):498–511, 2009.

[18] Richard Myers, Richard C. Wilson, and Edwin R. Hancock. Bayesian graph edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence - TPAMI*, 22(6):628–635, 2000.

[19] M. A. Pinheiro, J. Kybic, and P. Fua. Geometric graph matching using monte carlo tree search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(11):2171–2185, Nov 2017.

[20] Miguel Amável Pinheiro and Jan Kybic. Path descriptors for geometric graph matching and registration. In Aurélio Campilho and Mohamed Kamel, editors, *Image Analysis and Recognition*, pages 3–11, Cham, 2014. Springer International Publishing.

[21] Richard Ralph. Mpeg-7 core experiment ce-shape-1. 1999.

[22] Romain Raveaux, Jean-Christophe Burie, and Jean-Marc Ogier. A graph matching method and a graph matching distance based on subgraph assignments. *Pattern Recognition Letters*, 31:394–406, 2010.

[23] K. Riesen and H. Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image and Vision Computing*, 27:950–959, 2009.

[24] Antonio Robles-kelly and Edwin R. Hancock. A Riemannian approach to graph embedding. *Pattern Recognition -PR*, 40(3):1042–1056, 2007.

[25] A. Sanfeliu and K.S. Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics (Part B)*, 13(3):353–363, 1983.

[26] Alberto Sanfeliu, René Alquézar, and Francesc Serratosa. Clustering of attributed graphs and unsupervised synthesis of function-described graphs. volume 2, pages 6022–6025, 2000.

[27] T Sebastin, P Klein, and B Kimia. Recognition of shapes by editting shock graphs, 2001. In *IEEE International Conference in Computer Vision*, 2001.

[28] Kaleem Siddiqi, Ali Shokoufandeh, Sven J. Dickinson, and Steven W. Zucker. Shock graphs and shape matching. *International Journal of Computer Vision*, 35(1):13–32, Nov 1999.

[29] Ponnuthurai N. Suganthan. Structural pattern recognition using genetic algorithms. *Pattern Recognition - PR*, 35(9):1883–1893, 2002.

[30] **S. Lagraa**, **H. Seba**, **R. Khennoufa**, **A. M'Baya**, and **H. Kheddouci**. **A distance measure for large graphs based on prime graphs**. *Pattern Recognition*, 47(9):2993 – 3005, 2014.

[31] Mario Vento. A long trip in the charming world of graphs for pattern recognition. *Pattern Recognition*, 48(2):291–301, 2015.

[32] Rafael Grompone Von Gioi, Jeremie Jakubowicz, Jean-Michel Morel, and Gregory Randall. Lsd: A fast line segment detector with a false detection control. *IEEE transactions on pattern analysis and machine intelligence*, 32(4):722–732, 2008.

[33] Zhiping Zeng, Anthony K. H. Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou. Comparing stars: On approximating graph edit distance. *Proceedings of The Vldb Endowment - PVLDB*, 2(1):25–36, 2009.