Pattern Recognition Using Graph Edit Distance

Shri Prakash Dwivedi and Ravi Shankar Singh

1 Introduction

The graph is a fundamental and ubiquitous mathematical structure in mathematics, engineering, and computer science. Its strength lies in the flexibility to represent itself as a structural model in the various domains of science and engineering. Graph edit distance (GED) can be stated as the least number of modifications needed to convert one graph into another. The edit operations can add, delete, and substitute nodes and edges. The edit operations were initially used as string edit operations for converting one string into another using the lowest count of edit operations, such as addition, deletion, or substitution of alphabets. Later on, the idea of string edit distance was applied to tree edit distance and further generalized to GED. Due to its flexibility, the edit distance approach is powerful, and it can be applied to different problems.

One of the significant applications of GED is inexact graph matching. In graph matching (GM), we measure the similarity between two objects represented in the form of graphs. It is mainly categorized into two classes, exact and inexact GM. In exact GM, a strict correspondence must be there between the vertices and edges of the two graphs. In the error-tolerant GM, also known as inexact GM, some flexibility or tolerance is allowed during the matching between the two graphs. The limitation of exact GM is that it can be only used to find the strict matching between the two graphs, and therefore, it cannot take into account distortion incurred to the graph

S. P. Dwivedi (✉)
Department of Information Technology, G.B. Pant University of Agriculture & Technology, Pantnagar, India
e-mail: shriprakashdwivedi@gbpuat-tech.ac.in

R. S. Singh
Department of Computer Science & Engineering, Indian Institute of Technology (BHU), Varanasi, India

S. Awasthi et al. (eds.), *Sustainable Computing*,
https://doi.org/10.1007/978-3-031-13577-4_21

due to the existence of noise in the process of matching. Error-tolerant GM offers flexibility during the process of GM [2]. There are many approaches to error-tolerant GM but GED being very adaptable is one of the most crucial techniques for the GM problem. A comprehensive review of diverse GM methods is described in [4] and [14].

In [15], the authors introduced an approach to present patterns by trees instead of strings. As a tree is a more general high-dimensional structure than string, its use will lead to an efficient description of a higher-dimensional pattern. Then tree system representation of patterns is used for syntactic pattern recognition. A distance measure for attributed relational graph [29] using the computation of the least number of modifications required to change an input graph to the output one is described in [27]. This chapter also considers the costs of recognizing the vertices in the distance computation. In [3], the authors proposed using heuristic information inexact GM of attributed graphs derived from a state-space search. The matching process is generalized to arbitrary graphs, and the edit cost functions are designed so that the GED satisfies the properties of metrics in some situations.

GED is applicable in a broad range of applications as it allows specific edit cost functions to be defined for various applications. A significant limitation of GED is that its computation becomes too costly. It uses exponentially ample execution time to compute GED concerning the number of vertices in the input graph. Graph edit distance problem is shown to be NP-hard in [30]. Since a fast deterministic algorithm is unavailable for this problem, many approximate and suboptimal algorithms have been proposed recently.

The paper [17] proposed an efficient algorithm for GED computation of attributed planar graphs by iteratively matching small subgraphs to optimize structural correspondence. Then it applied the above technique to the fingerprint classification problem. In [19], the authors describe the fast suboptimal algorithm for the GED by reducing the space essential for computing the GED using A^* search technique[16]. They describe the different variants of A^*, such as A^*-beamsearch and A^*-pathlength, to reduce the search space that may not be pertinent to particular classification tasks. The computation of approximate GED by considering only local instead of global edge structure through the optimizing process is given in [23]. In [28], the authors represent the GED as a basis in the label space and use it to define a class of GED cost. They also describe the various characteristics and use of this GED cost. An improvement over the above method by manipulating the initial assignment of the approximation algorithm so that the assignment is ordered based on the individual confidence is provided in [13]. Estimating exact GED considering lower and upper bounds of bipartite approximation utilizing regression analysis is described in [25]. Different search strategies for improving the approximation of bipartite GED computation using the beam search, iterative search, and greedy search, etc., are provided by Riesen and Bunke [24]. The book [21] provides a detailed description of structural pattern recognition and describes various algorithms for structural pattern recognition by using GED.

A novel category of structural pattern recognition using GED is recently proposed [5] that decreases the graph size, reducing search space by ignoring the less

relevant vertices using some measure of importance. In [8], the authors presented homeomorphic GED for topologically equivalent graphs and used to perform GM by measuring the structural similarity between two graphs. The proposed technique utilizes the path contraction to remove the vertices having degree two to construct simple paths of input graphs in which every node except first and last has degree two. An extension to GED utilizing the notion of node contraction in which a graph is changed into another by contracting the lesser degree vertices is given in [9]. In [12], the authors proposed centrality GED to perform inexact GM using the various centrality measures for removing the vertices having the least centrality value in the graph. Some other recent works are given in [6, 7, 10, 11, 26].

This chapter is outlined as follows. Section 2 introduces basic concepts and definitions related to GED. Section 3 presents essential algorithms and techniques to compute GED. Section 4 shows some experimental results, and at last, Sect. 5 includes the conclusion.

2 Basic Concepts and Definitions

A description of fundamental concepts and definitions associated with GED is provided in this section. To get an in-depth description, the reader can refer to the texts such as [1, 5, 18].

In computer science, a graph is commonly defined as $G = (V, E)$, where V is the vertex set and E is the edge set in which every edge connects the two vertices. To define a graph in pattern recognition field, we use two additional parameters, node label mapping and edge label mapping, to identify the vertices and links in a graph.

We define a *graph* G as a tuple $G = (V, E, \mu, \nu)$; here, V and E are defined as above, $\mu : V \rightarrow L_V$, and $\nu : E \rightarrow L_E$. Here, μ is a function that assigns each vertex $v \in V$ a unique label $l_v \in L_V$. Similarly, ν is a function that assigns each vertex $e \in E$ a unique label $l_e \in L_E$.

A graph may be directed or undirected based on its edges; if $\nu(u, v) = \nu(v, u)$, then the graph is undirected since from both directions, edges have the same value, whereas for undirected graphs, $\nu(u, v) \neq \nu(v, u)$. When $L_V = L_E = \epsilon$, i.e., vertex label and edge label sets are empty, the graph G is called an unlabeled graph.

A graph can be converted into another graph using a set of edit operations. A set of edit operations are inserting, deleting, and substituting nodes and edges. A chain of edit operations that convert an input graph into the output one is defined as an *edit path* from the input graph to the output one. To insert a node u, we denote $\epsilon \rightarrow u$; to delete a node u, we represent $u \rightarrow \epsilon$; to substitute the vertex u by vertex v, we represent $u \rightarrow v$. Likewise, to insert an edge e, we denote by $\epsilon \rightarrow e$, $e \rightarrow \epsilon$ represents deletion of the edge e, and $e \rightarrow f$ defines substitution of the edge e by edge f.

In the following definitions, for simplicity, we denote a graph G_i by $G_i = (V_i, E_i, \mu_i, \nu_i)$.

Definition 1 The *GED* from G_1 to G_2 is stated by

$$GED(G_1, G_2) = min_{(e_1,\ldots,e_k)\in\varphi(G_1,G_2)} \sum_{i=1}^{k} c(e_i);$$

here $c(e_i)$ represents the costs of corresponding edit operations of e_i, and $\varphi(G_1, G_2)$ denotes the sequence of edit path to convert G_1 into G_2.

Two graphs are homeomorphic when both graphs are a subdivision of another graph. Subdivision of an edge is the process of inserting an additional vertex along an edge. The subdivision on graph G creates another graph after performing the subdivision on the edges of this graph. We can observe that the subdivision operation on a graph only changes the number of nodes of degree two.

Definition 2 The homeomorphic GED *HGED* from graph G_1 to G_2 is defined as

$$HGED(G_1, G_2) = GED(H_1, H_2) = min_{(e_1,\ldots,e_k)\in\varphi(H_1,H_2)} \sum_{i=1}^{k} c(e_i),$$

where G_1' is the graph obtained from G_1 by doing path contraction from every vertex. Similarly, G_2' is the graph got from G_2 by performing path contraction from each vertex.

Path contraction is the technique of removing every intermediate vertex having degree 2, except first and last nodes. *Node contraction* is the method of removing vertices along with the incident edges, given that it is not an articulation point or cut vertex [9].

A brief explanation of the essential centrality measures [20] follows. Centrality indicates the importance of a node in a network or graph. A node with high centrality will be more significant concerning other nodes in the graph. The *degree centrality* denotes the degree of a node in the graph. When a graph is a directed graph, it will have both the indegree and outdegree centrality. The *betweenness centrality* denotes the number of times a node occurs between other two nodes on their shortest path. The *eigenvector centrality* measures the node's influence using the count of its links to other vertices in the graph. The *PageRank centrality* considers the importance of a vertex proportional to the influence of its neighboring vertices divided by their outdegree.

Definition 3 r-CentralityNodeContraction is the operation to contract the r ratio of vertices from G having minimum centrality scores of a specified centrality identifier.

Definition 4 r-CentralityGraphEditDistance computation between G_1 and G_2 is defined as the GED between these two graphs, where $r.G_1$ nodes of G_1 and $r.G_2$ nodes of G_2 having minimum centrality value are contracted.

Definition 5 In the t-CentralityNodeContraction, t nodes having the minimum centrality score of a specified centrality measure are removed, given that these nodes are not the cut vertex.

Definition 6 t-CentralityGraphEditDistance is the method of computing GED from G_1 to G_2 after deleting the t nodes from both the graphs having the minimum centrality score of a given centering measure.

3 Algorithms

The computation of GED is generally accomplished utilizing tree-search-based algorithms. Tree-search-based techniques will be able to traverse the entire search space for the assignments of vertices and edges from the one graph to another for finding the optimal edit transformation. A commonly used technique is based on A^* search method using an ordered tree to explore the complete search space. The tree's root node will start with the initial solution. Intermediate nodes denote the subsequent partial solution, whereas the leaf nodes represent the complete edit path. A heuristic function is used for selecting the next successor node to be explored at the given level.

Algorithm 1 describes to compute the GED. The inputs to the GraphEditDistance algorithm are G_1 and G_2, where graph G_1 has n vertices and graph G_2 has m vertices. The outcome of this algorithm is the least-cost GED from graph G_1 to graph G_2. The algorithm starts with the empty set A in line 1. Set A includes all the partial edit paths created so far. During the *for* loop of lines 3–4, every vertex of the second graph is substituted by the u_1 vertex of the first graph. Each of these substitutions is then inserted into the set A. Deletion of node u_1 is also appended to the set A. The *while loop* of lines 6–25 computes the minimum cost edit path C_{min} from A. The algorithm uses a heuristic function $g(C) + h(C)$ to select the minimum cost edit path. $g(C)$ represents the cost of optimal edit path from the top root vertex to the current vertex C. $h(C)$ is used to denote the estimated cost from the current node C to the leaf node. The sum $g(C) + h(C)$ provides the total cost assigned to a vertex in the search tree. Here the optimal edit path from root vertex to a leaf vertex will be computed. The *if* loop of line 8 tests to know if the constructed C_{min} is one of the complete edit paths. If this is the case, the algorithm returns the corresponding complete edit path in line 9. If all nodes of first graph G_1 are visited (line 11), then the remaining unvisited nodes of graph G_2 (line 12) are inserted in to the graph (line 13) and A is updated in line 16. Similarly, in the *for* loop of lines 18–22, all the unvisited nodes of G_1 are substituted by each node of G_2 along with the deletion of each node of G_1 in line 19, and finally A is updated in line 22.

GraphEditDistance algorithm is an exact algorithm that explores the complete search space to find the optimal edit path to transform G_1 into G_2. This algorithm is exact, but it is computationally very expensive; therefore, this algorithm may

Algorithm 1 GraphEditDistance (G_1, G_2)

Require: Two Graphs G_1, G_2, where $V_1 = \{u_1, \ldots, u_n\}$ and $V_2 = \{v_1, \ldots, v_m\}$
Ensure: A minimum cost GED between G_1 and G_2

```
1:  A ← ∅
2:  for each (v_j ∈ V_2) do
3:      A ← A ∪ {u_1 → v_j}
4:  end for
5:  A ← A ∪ {u_1 → ϵ}
6:  while (True) do
7:      Compute minimum cost edit path C_min = min_{C∈A}{g(C) + h(C)} from A
8:      if (C_min is a complete edit path) then
9:          return C_min
10:     else
11:         if (all vertices (u_i ∈ V_1) are visited) then
12:             for all unvisited (v_j ∈ V_2) do
13:                 C_min ← C_min ∪ {ϵ → v_j}
14:             end for
15:             A ← A ∪ {C_min}
16:         else
17:             for (all unvisited vertices (u_i ∈ V_1)) do
18:                 for (each (v_j ∈ V_2)) do
19:                     C_min ← C_min ∪ {u_i → v_j} ∪ {u_i → ϵ}
20:                 end for
21:             end for
22:             A ← A ∪ {C_min}
23:         end if
24:     end if
25: end while
```

not be feasible for the graphs having large sizes. To overcome this disadvantage, several approximate and suboptimal algorithms have been proposed. One of the approximate algorithms for GED is outlined in Algorithm 2. The basic idea of the suboptimal algorithm is to prune the search space using some optimizing and heuristic techniques so that the resulting search space is reduced, thereby reducing the computation time. The steps of ApproximateGraphEditDistance algorithm are the following. The inputs to the ApproximateGraphEditDistance algorithm are G_1 and G_2. The output of this algorithm is the minimum cost approximate GED from graphs G_1 to G_2. The steps of this algorithm are similar to the GraphEditDistance algorithm except for the *while* loop of lines 6–26. Before selecting the next node for consideration, set A is pruned using some optimizing techniques such as beam search in which a fixed number of nodes known as beam width are only explored for selecting the next candidate. Since the heuristic methods view only the partial set of edit operations in the complete search space, this results in an approximate but comparatively efficient solution.

Algorithm 2 ApproximateGraphEditDistance (G_1, G_2)

Require: Two Graphs G_1, G_2, where $V_1 = \{u_1, \ldots, u_n\}$ and $V_2 = \{v_1, \ldots, v_m\}$
Ensure: A minimum cost GED between G_1 and G_2

$A \leftarrow \emptyset$
for each ($v_j \in V_2$) **do**
    $A \leftarrow A \cup \{u_1 \rightarrow v_j\}$
end for
$A \leftarrow A \cup \{u_1 \rightarrow \epsilon\}$
while (True) **do**
    Prune A using an optimizing/heuristic technique
    Find min. cost edit path C_{min} from A
    if (C_{min} is a complete edit path) **then**
        return C_{min}
    else
        if (all vertices ($u_i \in V_1$) are visited) **then**
            for all unvisited ($v_j \in V_2$) **do**
                $C_{min} \leftarrow C_{min} \cup \{\epsilon \rightarrow v_j\}$
            end for
            $A \leftarrow A \cup \{C_{min}\}$
        else
            for (all unvisited vertices ($u_i \in V_1$)) **do**
                for (each ($v_j \in V_2$)) **do**
                    $C_{min} \leftarrow C_{min} \cup \{u_i \rightarrow v_j\} \cup \{u_i \rightarrow \epsilon\}$
                end for
            end for
            $A \leftarrow A \cup \{C_{min}\}$
        end if
    end if
end while

3.1 Homeomorphic Graph Edit Distance

As discussed before, we say two graphs to be homeomorphic when both graphs are a subdivision of another graph. During the homeomorphic GED, first, all the nodes of degree 2 are deleted except the first and last nodes along all the simple paths of the graphs, and after that, GED is computed.

Given two graphs G_1 and G_2, homeomorphic edit cost function $\forall$ vertices $u \in V_1$, $v \in V_2$ and $\forall$ edges $e \in E_1$, $e' \in E_2$ is defined as:

$$c(u \rightarrow \epsilon) = x_{node}$$
$$c(\epsilon \rightarrow v) = x_{node}$$
$$c(u \rightarrow v) = y_{node}.||\mu_1(u) - \mu_2(v)||$$
$$c(e \rightarrow \epsilon) = x_{edge}$$
$$c(\epsilon \rightarrow e') = x_{edge}$$
$$c(e \rightarrow e') = y_{edge}.||\nu_1(e) - \nu_2(e')||$$
$$c((u_1, \ldots, u_n) \rightarrow (u_1, u_n)) = z_{path}.||\mu_1(u_1) - \mu_1(u_n)||$$

Here x_{node}, x_{edge}, y_{node}, y_{edge}, and z_{path} are positive values, and $c(u \rightarrow \epsilon)$ and $c(\epsilon \rightarrow v)$ are the costs of deleting vertex u and inserting vertex v, respectively, $c(u \rightarrow v)$ is the charge of substituting vertex u by vertex v, $c(e \rightarrow \epsilon)$ and $c(\epsilon \rightarrow e')$ are the costs of deleting edge e and inserting edge e', respectively, and $c((u_1, \ldots, u_n) \rightarrow (u_1, u_n))$ is the charge of performing path contraction from $(u_1, \ldots, u_n)$ to (u_1, u_n).

The computation of homeomorphic GED is outlined in Algorithm 3. Input to this algorithm is the two input graphs, G_1 and G_2, and the outcome of this algorithm is the least-cost homeomorphic GED from G_1 to G_2. The steps to perform HomeomorphicGraphEditDistance algorithm are follows. The first *for* loop performs the path contraction operations over all the simple paths of graph G_1 to delete all the intermediate nodes of degree 2 except the first and last nodes along the path. The *if* loop checks for all the candidate paths that are eligible for the process of path contraction. Similarly, the second *for* loop performs the path contraction operations over all the simple paths of graph G_2 to delete all the intermediate nodes of degree 2 except the first and last nodes along the path. After performing the path contraction operation, both input graphs G_1 and G_2 are updated along with their modified number of vertices and edges. Finally, the ApproximateGraphEditDistance algorithm is called on the updated G_1 and G_2 to compute the GED.

Algorithm 3 HomeomorphicGraphEditDistance (G_1, G_2)

INPUT: Two Graphs G_1, G_2, where $G_i = (V_i, E_i, \mu_i, \nu_i)$ for $i = 1, 2$
where $V_1 = \{u_1, \ldots, u_n\}$ and $V_2 = \{v_1, \ldots, v_m\}$
OUTPUT: A min. cost homeomorphic GED between G_1 and G_2
for each $(u_i \in V_1)$ **do**
  if (there is a path $(u_i, u_{i+1}, \ldots, u_{i+k})$ such that
$deg(u_{i+1}) = deg(u_{i+2}) = \ldots = deg(u_{i+k-1}) = 2)$ **then**
    $(u_i, u_{i+1}, \ldots, u_{i+k}) \rightarrow (u_i, u_{i+k})$

    $V_1 \leftarrow V_1 \setminus \{u_{i+1}, \ldots, u_{i+k-1}\}$

  end if
end for
for each $(v_j \in V_2)$ **do**
  if (there is a path $(v_j, v_{j+1}, \ldots, v_{j+k})$ such that
$deg(v_{j+1}) = deg(v_{j+2}) = \ldots = deg(v_{j+k-1}) = 2)$ **then**
    $(v_j, v_{j+1}, \ldots, v_{j+k}) \rightarrow (v_j, v_{j+k})$

    $V_2 \leftarrow V_2 \setminus \{v_{j+1}, \ldots, v_{j+k-1}\}$

  end if
end for
Update $G_1, G_2, n \leftarrow n', m \leftarrow m'$
ApproximateGraphEditDistance (G_1, G_2)

3.2 GED Utilizing Centrality Information

To decrease the computing time of GED, first we remove the vertices from the graphs having lower centrality scores prior to computing the GED between the input graphs. r-CentralityGraphEditDistance is an extension to GED to compute the approximate value of GED by ignoring the r fraction of nodes from the input graphs using the specific centrality measure.

Definition 7 r-DegreeCentralityNodeContraction is a process of deleting $r.|G|$ vertices with lowest degree from G, provided these nodes are not cut vertices.

Definition 8 r-BetweennessCentralityNodeContraction is a task of deleting $r.|G|$ vertices having minimum betweenness value from G, provided these nodes are not cut vertices.

Definition 9 r-EigenvectorCentralityNodeContraction is a task of deleting $r.|G|$ vertices having least eigenvector value from G, provided these nodes are not cut vertices.

Definition 10 r-PageRankCentralityNodeContraction is a task of deleting $r.|G|$ vertices having minimum PageRank value from G, provided these nodes are not cut vertices.

The edit cost of r-centrality GED can be defined utilizing an extra cost $c(a \rightarrow \epsilon) = 0$, for $r.|G|$ nodes of the G with the least value of the specified centrality indicator.

r-GED uses the Euclidean's distances to assign a fixed cost for inserting, deleting, and substituting the nodes and edges. Suppose graphs G_1 and G_2 have nodes $a \in V_1$, $b \in V_2$ and links $e \in E_1$, $f \in E_2$, the modified edit cost function can be specified as given below:

$c(a \rightarrow \epsilon) = p_{node}.$
$c(\epsilon \rightarrow b) = p_{node}.$
$c(a \rightarrow b) = q_{node}.||\mu_1(a) - \mu_2(b)||.$
$c(e \rightarrow \epsilon) = p_{edge}.$
$c(\epsilon \rightarrow f) = p_{edge}.$
$c(e \rightarrow f) = q_{edge}.||\nu_1(e) - \nu_2(f)||.$
$c(a \rightarrow \epsilon) = 0$, when a is one of the $r.|G|$ vertices with minimum centrality score, and it is not an articulation point.

$p_{node}, q_{node}, p_{edge}, q_{edge}$ are positive constants.

The steps to perform r-CentralityGraphEditDistance (G_1, G_2) are outlined in Algorithm 4. The r-CentralityGraphEditDistance algorithm's inputs are G_1 and G_2 and the parameter r. The outcome of this algorithm is minimum cost r-CentralityGraphEditDistance between G_1 and G_2. Line 1 of the algorithm calls the procedure r-CentralityNodeContraction that removes $r.|G|$ vertices of minimum

Algorithm 4 r-CentralityGraphEditDistance (G_1, G_2)

Require: Two Graphs G_1, G_2, where $|V_1| = n$ and $|V_2| = m$, a constant r
Ensure: A min. cost r-GED between G_1 and G_2
1: $G_1' \leftarrow r$-CentralityNodeContraction $(G_1, \lceil r.n \rceil)$
2: $G_2' \leftarrow r$-CentralityNodeContraction $(G_2, \lceil r.m \rceil)$
3: **ApproximateGraphEditDistance** (G_1', G_2')
4: **procedure** r-**CentralityNodeContraction**$(G, \lceil r.|G| \rceil)$
5: **for** $(i \leftarrow 1$ to $\lceil r.|G| \rceil)$ **do**
6: Choose the node v having least centrality value
7: **if** (v is not an articulation point) **then**
8: $V \leftarrow V \setminus \{v\}$
9: $E \leftarrow E \setminus \{(u, v) | (u, v) \in E$ for every $u \in G\}$
10: **end if**
11: **end for**
12: **return** G
13: **end procedure**

centrality value from G_1. Similarly, line 2 of the algorithm calls the procedure r-Centrality node contraction that removes $r.|G|$ nodes of least centrality value from G_2. The procedure r-CentralityNodeContraction is described in lines 4–13. This procedure uses *for* loop in lines 5–11 to delete $\lceil r.|G| \rceil)$ vertices from a G with minimum centrality value of specified centrality criteria. It selects the node u with minimum centrality value (line 6), verifies that it is not a cut vertex (line 7), after that the corresponding node is deleted in line 8, and the associated edges are removed in line 9. The preprocessed graph is returned in line 12. Finally, line 3 executes ApproximateGraphEditDistance and computes minimum cost edit path that also satisfies for complete edit path.

4 Results and Discussion

Pattern recognition is among the significant applications of GED. Especially in structural pattern recognition, where the underlying pattern has structures that fixed dimensional vectors cannot represent, graphs can be utilized to represent such structures. When a pattern represents a graph, pattern recognition is usually known as GM. GED is a crucial technique for GM. In Sect. 3, we discussed a few important algorithms for computing the GED between two graphs. Since Algorithm 1 explores the complete search space to find the optimum edit path, it takes an exponential amount of time to output this GED. Due to its computationally exponential complexity, this algorithm cannot be used for the graph exceeding 10–15 nodes. Algorithm 2 computes the inexact approximate GED between two graphs by pruning the search space using some heuristic function for finding the successive node at every levels of the search tree. Algorithm 3 uses the inexact GED after performing the path contraction of every simple path of both input graphs. Algorithm 4 also uses the idea of Algorithm 2 to prune the search space

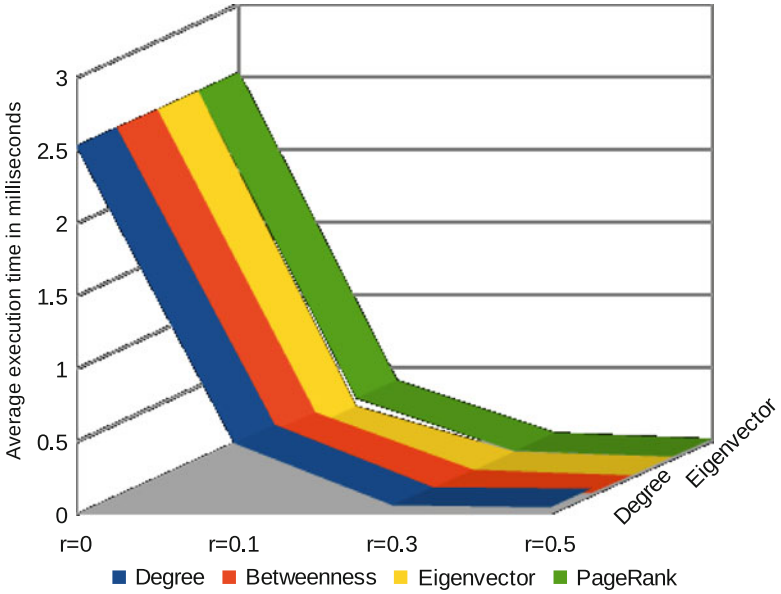

Fig. 1 Computation time for letter A datasets

after performing the node contraction on both input graphs using a given centrality measure.

In this section, r-CentralityGraphEditDistance algorithm's computation time and accuracy to perform the GM are observed. This section utilizes letters and AIDS datasets from IAM graph database [22] to perform the GM. Computation time is the execution time of the algorithm to perform the GM using r-CentralityGraphEditDistance for a given centrality measure. Accuracy of the algorithm is described in terms of classification accuracy on the test set after performing training of the algorithms using a training set of the given datasets.

The computation time taken by Algorithm 4 in milliseconds to perform the GM on the letter A datasets using the four centrality measures, degree, betweenness, eigenvector, and PageRank, is shown in Fig. 1. This figure shows the difference in time taken by this algorithm using the various centrality measures.

Figure 2 shows the computation time used by Algorithm 4 in milliseconds to perform GM for the active AIDS datasets utilizing the various centrality criteria. To prune the search space and to select the successive node in the search tree, we have used beam search heuristic technique.

Accuracy of Algorithm 4 on letter A datasets of high distortions utilizing the given centrality measures for four different values of $r = 0, 0.1, 0.3$, and 0.5 is provided in Fig. 3. This figure demonstrates that the accuracy ratio using the eigenvector centrality is usually more than other centrality indicators. From this, we can infer that eigenvector centrality can be more suitable to perform GM on the letter datasets.

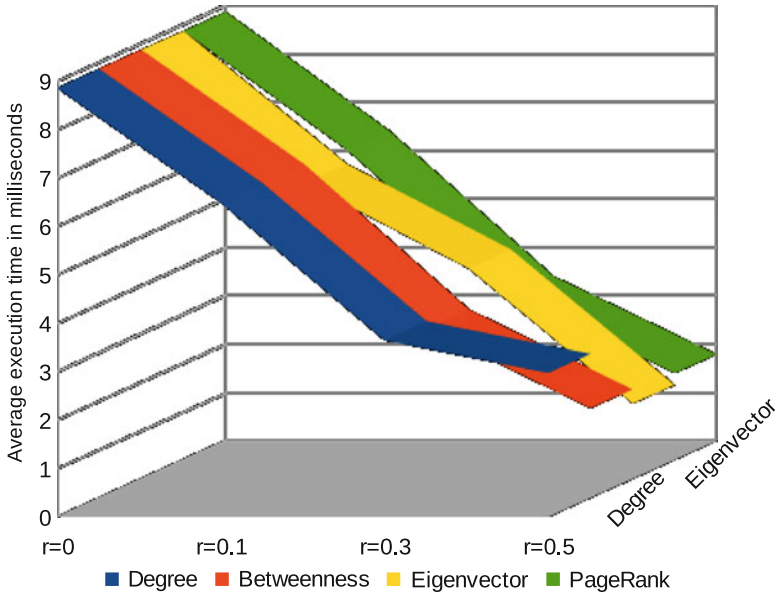

Fig. 2 Computation time for active AIDS datasets

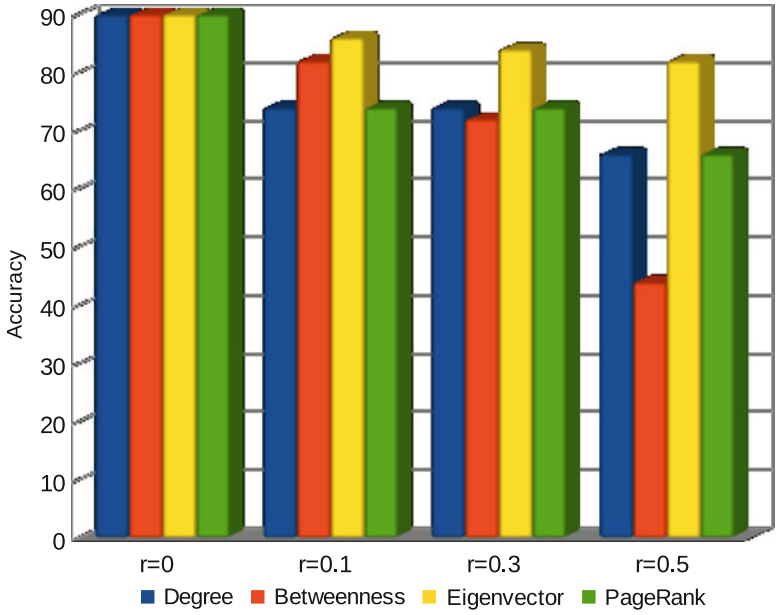

Fig. 3 Accuracy ratio of letter A datasets

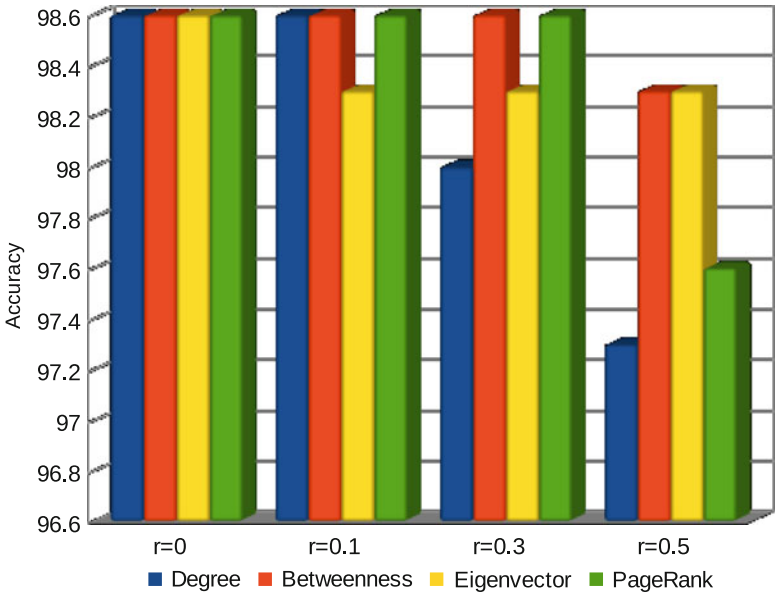

Fig. 4 Accuracy ratio for active AIDS datasets

Classification accuracy of r-CentralityGraphEditDistance to perform GM for the active AIDS datasets using the four centrality measures for four different values of $r = 0, 0.1, 0.3$, and 0.5 is provided in Fig. 4. This figure indicates that the classification accuracy utilizing betweenness criteria is generally more than other centrality indicators.

5 Conclusion

This chapter discussed the GED-based techniques for structural pattern recognition. GED is a crucial technique to measure the similarity between two graphs. We presented the important techniques and algorithms to compute the GED. We also described the various extensions and advancements to compute the GED that can be used for the trade-off between efficiency and accuracy consideration.

References

1. C.C. Aggarwal, H. Wang, Managing and mining graph data, in *Advances in Database Systems* (Springer, Berlin, 2010)
2. H. Bunke, Error-tolerant graph matching: A formal framework and algorithms, in *Advances in Pattern Recognition, International Workshop on Structural, Syntactic and Statistical Pattern Recognition (S+SSPR)*. Lecture Notes in Computer Science (Springer, Berlin, 1998)

3. H. Bunke, G. Allerman, Inexact graph matching for structural pattern recognition. Pattern Recog. Lett. **1**, 245–253 (1983)
4. D. Conte, P. Foggia,C. Sansone, M. Vento, Thirty years of graph matching in pattern recognition. Int. J. Pattern Recog. Artif. Intell. **18**(3), 265–298 (2004)
5. S.P. Dwivedi, Some algorithms on exact, approximate and error-tolerant graph matching. PhD Thesis, Indian Institute of Technology (BHU), Varanasi, 2019. arXiv:2012.15279
6. S.P. Dwivedi, Inexact graph matching using centrality measures (2021). *arXiv:2201.04563*
7. S.P. Dwivedi, Approximate bipartite graph matching by modifying cost matrix. Lecture Notes Electr. Eng. **837**, 415–422 (2022)
8. S.P. Dwivedi, R.S. Singh, Error-tolerant graph matching using homeomorphism, in *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)* (2017), pp. 1762–1766
9. S.P. Dwivedi, R.S. Singh, Error-tolerant graph matching using node contraction. Pattern Recog. Lett. **116**, 58–64 (2018)
10. S.P. Dwivedi, R.S. Singh, Error-tolerant geometric graph similarity, in *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*. Lecture Notes in Computer Science, vol. 11004 (Springer, Berlin, 2018), pp. 337–344
11. S.P. Dwivedi, R.S. Singh, Error-tolerant geometric graph similarity and matching. Pattern Recog. Lett. **125**, 625–631 (2019)
12. S.P. Dwivedi, R.S. Singh, Error-tolerant approximate graph matching utilizing node centrality information. Pattern Recog. Lett. **133**, 313–319 (2020)
13. M. Ferrer, F. Serratosa, K. Riesen, Improving bipartite graph matching by assessing the assignment confidence. Pattern Recog. Lett. **65**, 29–36 (2015)
14. P. Foggia, G. Percannella, M. Vento, Graph matching and learning in pattern recognition in the last 10 years. Int. J. Pattern Recog. Artif. Intell. **28**, 1450001.1–1450001.40 (2014)
15. K.S. Fu, B.K. Bhargava, Tree systems for syntactic pattern recognition. IEEE Trans. Comput. **22**, 1087–1099 (1973)
16. P.E. Hart, N.J. Nilson, B. Raphael, A formal basis for heuristic determination of minimum cost paths. IEEE Trans. Syst. Sci. Cyber. **4**, 100–107 (1968)
17. M. Neuhaus, H. Bunke, An error-tolerant approximate matching algorithm for attributed planar graphs and its application to fingerprint classification, in *International Workshop on Structural, Syntactic and Statistical Pattern Recognition (SSPR and SPR)*. Lecture Notes in Computer Science, vol. 3138 (Springer, Berlin, 2004), pp. 180–189
18. M. Neuhaus, H. Bunke, *Bridging the Gap Between Graph Edit Distance and Kernel Machines* (World Scientific, Singapore, 2007)
19. M. Neuhaus, K. Riesen, H. Bunke, Fast suboptimal algorithms for the computation of graph edit distance, in *Proceedings of the 11th International Workshop on Structural and Syntactic Pattern Recognition*. Lecture Notes in Computer Science, vol. 4109 (Springer, Berlin, 2006), pp. 163–172
20. M.E.J. Newman, *Networks–An Introduction* (Oxford University Press, Oxford, 2010)
21. K. Riesen, *Structural Pattern Recognition with Graph Edit Distance, Approximation Algorithms and Applications* (Springer, Berlin, 2015)
22. K. Riesen, H. Bunke, IAM graph database repository for graph based pattern recognition and machine learning, in *International Workshop on Structural, Syntactic and Statistical Pattern Recognition (S+SSPR)*. Lecture Notes in Computer Science, vol. 5342 (Springer, Berlin, 2008), pp. 287–297
23. K. Riesen, H. Bunke, Approximate graph edit distance computation by means of bipartite graph matching. Image Vision Comput. **27**(4), 950–959 (2009)
24. K. Riesen, H. Bunke, Improving bipartite graph edit distance approximation using various search strategies. Pattern Recog. **48**(4), 1349–1363 (2015)
25. K. Riesen, A. Fischer, H. Bunke, Estimating graph edit distance using lower and upper bounds of bipartite approximations. Int. J. Pattern Recog. Artif. Intell. **29**(2), 1550011 (2015)

26. A. Robles-Kelly, E. Hancock, Graph edit distance from spectral seriation. IEEE Trans. Pattern Analy. Mach. Intell. **27**(3), 365–378 (2005)
27. A. Sanfeliu, K.S. Fu, A distance measure between attributed relational graphs for pattern recognition. IEEE Trans. Syst. Man Cyber. **13**(3), 353–363 (1983)
28. A. Sole-Ribalta, F. Serratosa, A. Sanfeliu, On the graph edit distance cost: properties and applications. Int. J. Pattern Recog. Artif. Intell. **26**(5), 1260004.1–1260004.21 (2012)
29. W.H. Tsai, K.S. Fu, Error-correcting isomorphisms of attributed relational graphs for pattern analysis. IEEE Trans. Syst. Man Cyber. **9**, 757–768 (1979)
30. Z. Zeng, A.K.H. Tung, J. Wang, J. Feng, L. Zhou, Comparing stars: on approximating graph edit distance. PVLDB **2**, 25–36 (2009)