Chapter 4

Graph Matching utilizing Node Centrality Information

4.1 Introduction

The applications of exact graph matching to real-world applications is rather limited due to the presence of noise or error during the processing of the graphs. Error-tolerant graph matching offers an alternative to performing approximate graph matching. Due to exponential complexity associated with graph edit distance, other methods have been introduced to perform efficient graph matching at the cost of a slight decrease in accuracy. The node contraction technique described in chapter 3, is based on removing the nodes based on their degree centrality to decrease the size of the matching graphs. However, the degree centrality may not always be the best criteria to ignore the nodes. Depending on the structure and properties of the different dataset, we can select the appropriate centrality measure to delete the nodes for reducing the size of the graphs. In this chapter, we use eigenvector, betweenness and PageRank centrality in addition to degree centrality to reduce the size of the graphs for estimating an early approximate graph matching between two graphs.

Now we briefly explain the above centrality measures [111]. The centrality of a node in the graph signifies its relative importance in the graph. The centrality measures aim to find the most important or central nodes of a graph or network. Simplest centrality measure

is *degree centrality*, which simply refers to the degree of the given node. A node with more adjacent nodes will have higher degree centrality as compared to nodes with a fewer connection. *Betweenness* centrality of a node is based on the extent by which this node lies on the paths between other nodes. *Eigenvector* centrality is a generalization of degree centrality, which assigns each node a value proportionate to the sum of the values of its neighbors. For a node u_i its eigenvector centrality is given by $x_i = \kappa_1^{-1} \sum_j A_{ij} x_j$, where κ_1 is the largest eigenvalue of adjacency matrix A and A_{ij} is an element of A. In PageRank centrality, the centrality of a node is proportionate to the centrality of its neighbors divided by their outgoing degree. The *PageRank* centrality is defined by $x_i = \alpha \sum_j A_{ij} \frac{x_j}{k_j} + \gamma$, where α is a free parameter, k_j is the outgoing degree and γ is a constant.

This chapter is organized as follows. Section 4.2, presents error-tolerant graph matching using centrality measures and introduces r-centrality graph edit distance. Section 4.3, describes experimental evaluation. Lastly, section 4.4 provides a summary.

4.2 Error-Tolerant Graph Matching using Centrality Measures

To reduce the computation time of error-tolerant graph matching, we ignore the nodes from the graphs with less centrality value before computing a similarity score using GED between two graphs.

Definition 4.2.1. r-centrality node contraction is the process of contracting r fractions of nodes from a graph G with least centrality values of a given centrality measure.

In the above definition, r stands for ratio, which is equal to the decimal equivalent of the percentage of nodes from a graph G. If $r = 0.1$, then 10% of nodes will be contracted from G.

The above definition implies that starting from the node with the lowest centrality value in a graph G, up to $r.|G|$ nodes are deleted provided they are not a cut vertex. Where $|G|$ is the number of nodes in the graph G. Depending on the centrality measure used, r-centrality node contraction (r-NC) can be r-degree centrality node contraction (rDC-NC),

r-betweenness centrality node contraction (r-BC-NC), r-eigenvector node contraction (r-EV-NC) and r-PageRank node contraction (r-PR-NC).

Definition 4.2.2. r-degree centrality node contraction is the operation of contracting $r.|G|$ nodes of the smallest degree from graph G.

When $r.|G|$ is equal to the number of nodes of degree k in a graph, then r-degree centrality node contraction corresponds to k-degree node contraction.

Definition 4.2.3. r-betweenness centrality node contraction is the operation of contracting $r.|G|$ nodes with the lowest betweenness score from graph G.

Definition 4.2.4. r-eigenvector centrality node contraction is the process of contracting $r.|G|$ nodes having the lowest eigenvector centrality from graph G.

Definition 4.2.5. r-PageRank centrality node contraction is the process of contracting $r.|G|$ nodes with the lowest PageRank score from graph G.

Definition 4.2.6. r-centrality GED computation between two graphs G_1 and G_2 is defined as GED between these graphs, when $r.|G_1|$ nodes of G_1 and $r.|G_2|$ of G_2 of least centrality value have been contracted.

In the above definition depending on the actual centrality criteria used r-centrality GED computation (r-GED) corresponds to r-degree centrality GED computation (r-DC-GED), r-betweenness centrality GED computation (r-BC-GED), r-eigenvector GED computation (r-EV-GED) and r- PageRank GED computation (r-PR-GED). When $r = 0$, r-GED corresponds to standard GED computation.

4.2.1 Edit Cost

We can define the edit cost of r-GED by using an additional operation $c(u \to \varepsilon) = 0$, for $r.|G|$ vertices of the graph G having the lowest score of the given centrality measure.

r-GED utilizes the Euclidean distance and allocates the constant cost to insertion, deletion and substitution of vertices and links. For two graphs G_1 and G_2, having vertices $u \in V_1$, $v \in V_2$ and links $e \in E_1$, $f \in E_2$, we specify the extended edit cost function as given below.

$c(u \to \varepsilon) = x_{node}$
$c(\varepsilon \to v) = x_{node}$
$c(u \to v) = y_{node}.||\mu_1(u) - \mu_2(v)||$
$c(e \to \varepsilon) = x_{edge}$
$c(\varepsilon \to f) = x_{edge}$
$c(e \to f) = y_{edge}.||\nu_1(e) - \nu_2(f)||$
$c(u \to \varepsilon) = 0$, if u is one of the $r.|G|$ nodes of the lowest centrality value and is not a cut vertex.
Here x_{node}, y_{node}, x_{edge}, y_{edge} are positive constants.

4.2.2 Algorithm

The computation of error-tolerant graph matching using r-centrality node contraction is outlined in Algorithm 4. The input to the r-Centrality-Graph-Edit-Distance algorithm is two graphs $G_1 = (V_1, E_1, \mu_1, \nu_1)$, $G_2 = (V_2, E_2, \mu_2, \nu_2)$ and a parameter r. The output to the algorithm is the minimum cost r-GED between G_1 and G_2. It calls Algorithm 5 in line 1 to remove $\lceil r.n \rceil$ nodes in G_1 having the lowest centrality value provided they are not cut vertex. Here, $\lceil \rceil$ is a *ceiling* function, which returns the least integer greater than equal to rn. Similarly line 2 to remove $\lceil r.m \rceil$ in G_2 having the lowest centrality value. G_1' and G_2' are the resultant graphs obtained after performing r-Centrality-Node-Contraction on G_1 and G_2 respectively, such that $V_1' = \{u_1', ..., u_{n'}'\}$ and $V_2' = \{v_1', ..., v_{m'}'\}$. Line 3 initializes an empty set A. The vertex u_1' of G_1' is substituted by each vertex v_j' of G_2' in the *for* loop of lines 3–6, and deletion of u_1' is performed in line 7. The computation of the minimum cost edit path is performed in the *while* loop of lines 8–27. *If* loop in line 10 checks, whether C_{min} is a complete edit path so that it completely transform G_1' to G_2'. If all nodes V_1' are processed (line 13), then remaining nodes of V_2' are simply inserted in C_{min} in *for* loop of lines 14–16. Similarly, all unprocessed vertices of V_1' are substituted by all vertices of V_2' along with the deletion of vertices of V_1' in the *for* loop of lines 19–23, and A is updated in line 24.

Algorithm 5 describes the steps to perform r-centrality node contraction. The *for* loop in lines 1–7 iterates up to $\lceil r.|G| \rceil$ times to check for nodes in G to be a cut vertex. In case the node is a cut vertex it deletes the node and its connected edges and updates the graph G.

Algorithm 4 : r**-Centrality-Graph-Edit-Distance** (G_1, G_2)

Input: Two Graphs G_1, G_2, where $V_1 = \{u_1, ..., u_n\}$ and $V_2 = \{v_1, ..., v_m\}$ and a parameter r

Output: A minimum cost r-GED between G_1 and G_2

```
1:  G'_1 ← r-Centrality-Node-Contraction (G_1, ⌈r.n⌉)
2:  G'_2 ← r-Centrality-Node-Contraction (G_2, ⌈r.m⌉)
3:  A ← ∅
4:  for each (v'_j ∈ V'_2) do
5:      A ← A ∪ {u'_1 → v'_j}
6:  end for
7:  A ← A ∪ {u'_1 → ε}
8:  while (True) do
9:      Compute minimum cost edit path C_min from A
10:     if (C_min is a complete edit path) then
11:         return C_min
12:     else
13:         if (all vertices (u'_i ∈ V'_1) are visited) then
14:             for all unvisited (v'_j ∈ V'_2) do
15:                 C_min ← C_min ∪ {ε → v'_j}
16:             end for
17:             A ← A ∪ {C_min}
18:         else
19:             for (all unvisited vertices (u'_i ∈ V'_1)) do
20:                 for (each (v'_j ∈ V'_2)) do
21:                     C_min ← C_min ∪ {u'_i → v'_j} ∪ {u'_i → ε}
22:                 end for
23:             end for
24:             A ← A ∪ {C_min}
25:         end if
26:     end if
27: end while
```

Proposition 4.2.1. *r-Centrality-Graph-Edit-Distance algorithm performs error-tolerant graph matching of G'_1 and G'_2.*

Using the properties of the edit costs of r-GED, the Algorithm 4 returns the minimum cost of complete edit path which transform input graph G'_1 to output graph G'_2, so that every vertex of G'_1 uniquely corresponds to a vertex of G'_2. Also the algorithm r-Centrality-Node-Contraction ensures that $V'_1 \subset V_1$ and $V'_2 \subset V_2$.

Proposition 4.2.2. *r-Centrality-Node-Contraction algorithm executes in $O(n)$ time.*

We can check whether a node u is a cut vertex in $O(n)$ time. Therefore the *for* loop of the algorithm takes $O(r.|G|.n)$ time, that is $O(n)$.

The worst case computational complexity of the r-Centrality-Graph-Edit-Distance algorithm is exponential in the number of vertices in input graphs. We can use an appropriate variable r to minimize the overall computation time.

Algorithm 5 : r-Centrality-Node-Contraction (G, r)

Input: A Graph G and a parameter r
Output: Transformed graph after applying r-Centrality-Node-Contraction on G

1: **for** $(i \leftarrow 1$ to $\lceil r.|G| \rceil)$ **do**
2: Select node u with minimum centrality
3: **if** (u is not cut vertex) **then**
4: $V \leftarrow V \setminus \{u\}$
5: $E \leftarrow E \setminus \{(u,v)|(u,v) \in E, \forall v \in G\}$
6: **end if**
7: **end for**
8: **return** G

4.3 Experimental Evaluation

In this section, we apply r-Centrality-Graph-Edit-Distance algorithm for error-tolerant graph matching using the degree, betweenness, eigenvector and PageRank centrality. We use IAM graph database [109] for the comparison of execution time and accuracy obtained by these centrality techniques. We use letter and AIDS dataset for the evaluation of the proposed error-tolerant graph matching scheme.

4.3.1 Execution Time Comparison

For the comparison purpose, we have used three distinct values of r in r-GED, which are $r = 0.1, r = 0.3$ and $r = 0.5$. We have used these three values of r to compute 0.1-GED,0.2-GED and 0.3-GED. Comparison of the average execution time of graph matching in milliseconds using r-Centrality-Graph-Edit-Distance algorithm as applied to letter A and E of high distortion letter dataset using different centrality measures is shown in Figure 4.1 and Figure 4.2 respectively.

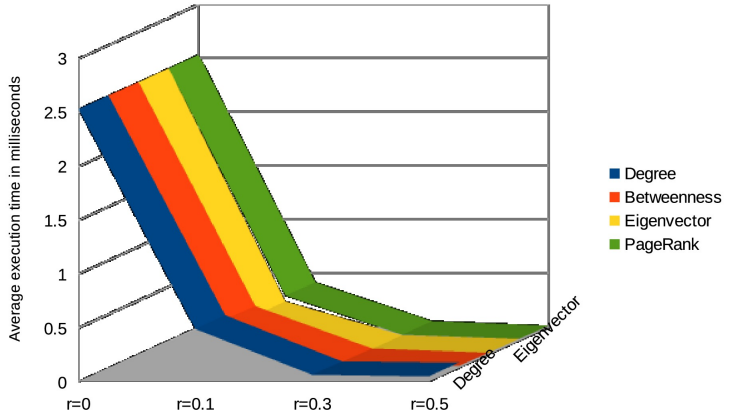

FIGURE 4.1: Comparison of execution time for letter A dataset

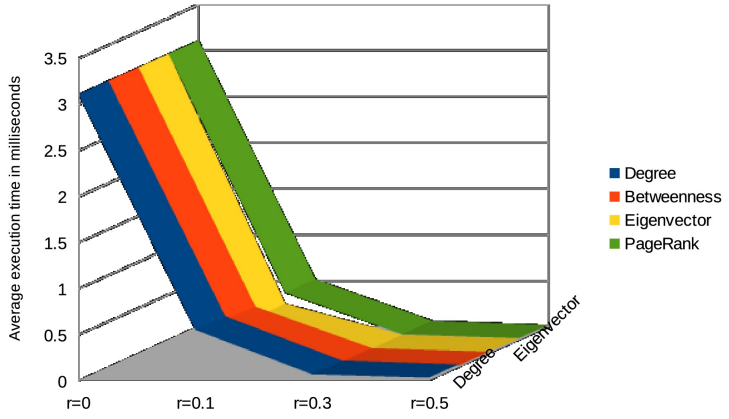

FIGURE 4.2: Comparison of execution time for letter E dataset

We can observe that graph matching time using eigenvector criteria is least, whereas time using degree centrality is large. Computation time for letter E is higher as it contains more nodes than letter A.

Comparison of the average running time of graph matching in milliseconds using beam search heuristic (beam width $w = 10$) for the four different centrality measures for the active class of AIDS dataset are shown in Figure 4.3. From this figure, we observe that Algorithm 4 usually takes less time using eigenvector and betweenness centrality as compared to the degree and PageRank centrality.

Figure 4.4 shows the corresponding average execution time of graphs for inactive AIDS dataset using the four centrality measures. Here again, the computation time using eigenvector takes less time than the degree, betweenness, and PageRank centrality

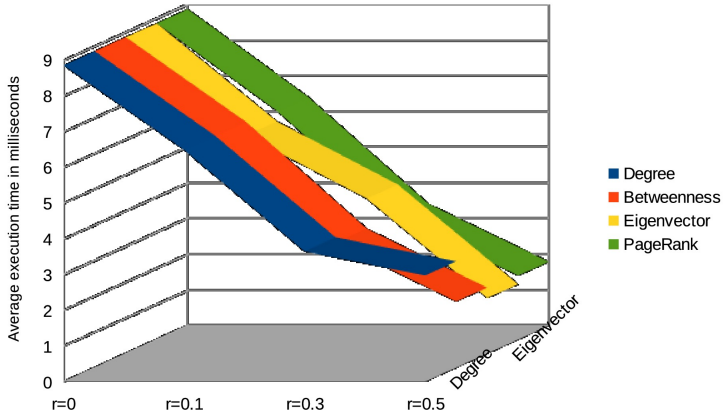

FIGURE 4.3: Comparison of execution time for active class of AIDS dataset

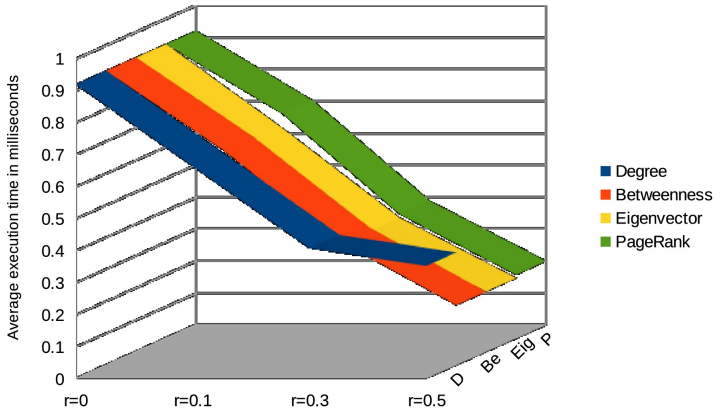

FIGURE 4.4: Comparison of execution time for inactive class of AIDS dataset

measures. Running time using betweenness centrality is usually less than that of degree and PageRank centrality measures.

4.3.2 Accuracy Comparison

For accuracy assessment, we consider the problem of classification of graphs by the nearest neighbor classifier. Letter dataset of high distortion level consists of 750 graphs for each of training as well as test sets. Each of these training, as well as test dataset, contains 50 graphs for every 15 letters. Classification accuracy of the proposed graph matching for letter A of high distortion using the four centrality indicators is given in Figure 4.5, while the accuracy of graph matching for letter E for the same measures is shown in Figure 4.6.

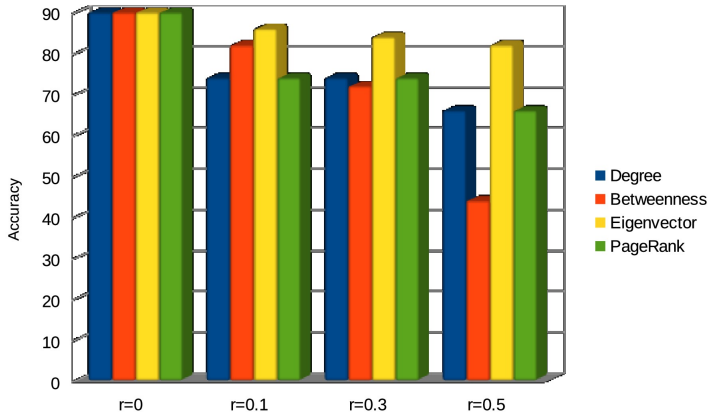

FIGURE 4.5: Comparison of accuracy ratio of letter A dataset

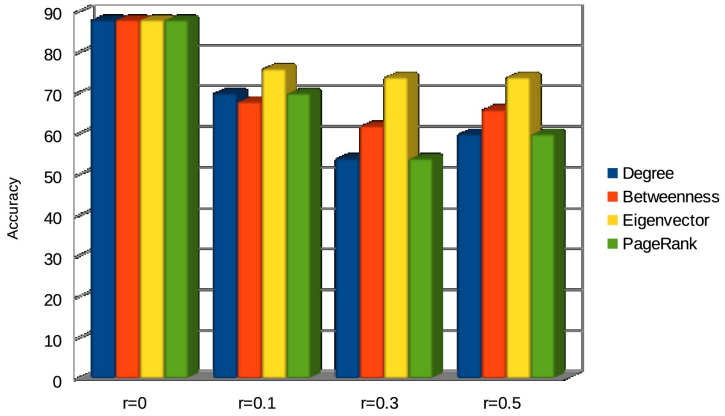

FIGURE 4.6: Comparison of accuracy ratio of letter E dataset

Here we note that the accuracy of both letters A and E using eigenvector centrality is higher than that of the other three measures. It shows that eigenvector centrality better captures the relative importance of nodes for letter dataset. Table 4.1 provides the accuracy results of all fifteen letters using degree centrality for $r = 0.1, 0.3$ and 0.5. Similarly, Tables 4.2, 4.3 and 4.4 show accuracy results for all letters using betweenness, eigenvector and PageRank centrality respectively.

To find the accuracy on AIDS dataset, we utilize test dataset consisting of 300 graphs from active class and 1200 graphs from inactive class, whereas training dataset consists of 50 graphs from active class and 200 graphs from the inactive class of AIDS dataset. We can observe the accuracy ratio of the proposed error-tolerant scheme using the four different centrality measure for the active class of AIDS dataset in Figure 4.7. In this figure, we

TABLE 4.1: Accuracy on letter dataset using degree centrality

Class	$r = 0.1$	$r = 0.2$	$r = 0.3$
A	74	74	66
E	70	54	60
F	72	44	42
H	58	42	26
I	90	86	84
K	68	58	44
L	78	64	54
M	82	50	66
N	58	50	50
T	70	46	52
V	88	54	60
W	88	74	60
X	76	52	40
Y	80	62	52
Z	78	62	64

TABLE 4.2: Accuracy on letter dataset using betweenness centrality

Class	$r = 0.1$	$r = 0.2$	$r = 0.3$
A	82	72	44
E	68	62	66
F	62	54	40
H	54	38	40
I	94	92	74
K	76	36	38
L	82	68	52
M	86	70	32
N	62	50	30
T	68	56	62
V	78	52	80
W	84	60	46
X	58	48	34
Y	82	76	74
Z	62	50	48

TABLE 4.3: Accuracy on letter dataset using eigenvector centrality

Class	$r = 0.1$	$r = 0.2$	$r = 0.3$
A	86	84	82
E	99	96	92
F	58	42	22
H	40	32	42
I	96	92	58
K	68	38	42
L	76	60	56
M	84	54	30
N	64	38	30
T	54	34	36
V	72	36	66
W	90	62	34
X	62	34	34
Y	74	80	72
Z	62	56	36

TABLE 4.4: Accuracy on letter dataset using PageRank centrality

Class	$r = 0.1$	$r = 0.2$	$r = 0.3$
A	74	74	66
E	70	54	60
F	72	44	42
H	58	42	26
I	90	86	84
K	68	58	44
L	78	64	54
M	82	50	66
N	58	50	50
T	70	46	52
V	88	54	60
W	88	74	60
X	76	52	40
Y	80	62	52
Z	78	62	64

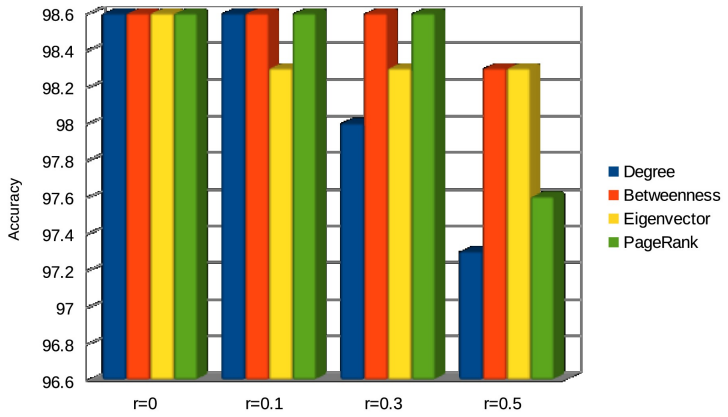

FIGURE 4.7: Comparison of accuracy for active class of AIDS dataset

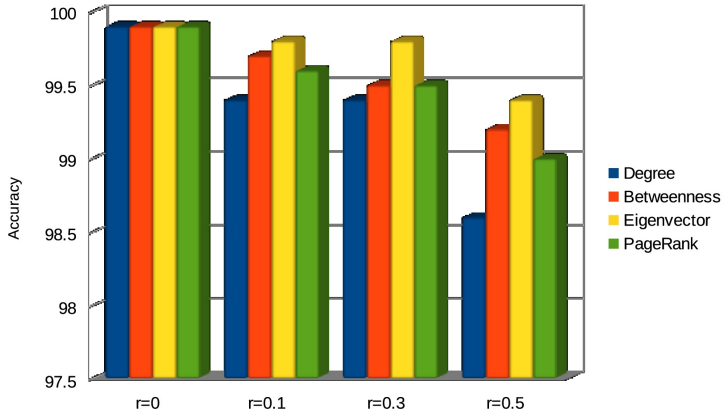

FIGURE 4.8: Comparison of accuracy for inactive class of AIDS dataset

observe that the accuracy ratio for the active class of AIDS dataset using betweenness centrality is usually higher than other centrality measures.

Figure 4.8 shows the comparison of accuracy for the inactive class of AIDS dataset using the four centrality measures. In this figure, we observe that the accuracy ratio of inactive class AIDS dataset using eigenvector centrality is highest followed by betweenness centrality.

Tables 4.5–4.8 show the accuracy ratio of AIDS dataset using degree, betweenness, eigenvector and PageRank centrality for three different values of $r = 0.1, 0.3$ and 0.5.

From the above experiments, we observe that node contraction using various centrality measure can have a different effect on the execution time and accuracy ratio for a given graph dataset. Therefore we can select the best suitable centrality measure for a given dataset depending on the requirement of execution time and accuracy.

TABLE 4.5: Accuracy on AIDS dataset using degree centrality

Class	$r = 0.1$	$r = 0.2$	$r = 0.3$
Active	98.6	98	97.3
Inactive	99.4	99.4	98.6

TABLE 4.6: Accuracy on AIDS dataset using betweenness centrality

Class	$r = 0.1$	$r = 0.2$	$r = 0.3$
Active	98.6	98.6	98.3
Inactive	99.7	99.5	99.2

TABLE 4.7: Accuracy on AIDS dataset using eigenvector centrality

Class	$r = 0.1$	$r = 0.2$	$r = 0.3$
Active	98.3	98.3	98.3
Inactive	99.8	99.8	99.4

TABLE 4.8: Accuracy on AIDS dataset using PageRank centrality

Class	$r = 0.1$	$r = 0.2$	$r = 0.3$
Active	98.6	98.6	97.6
Inactive	99.6	99.5	99

4.4 Summary

In this chapter, we presented a technique to approximate graph matching utilizing the concept of centrality measure to reduce the size of the graphs by ignoring the nodes with a lower value of given centrality criteria. In particular, we have used eigenvector, betweenness and PageRank centrality apart from degree centrality to perform the node contraction for the computation for error-tolerant graph matching. Experimental results show that these

centrality criteria can be used as computation time versus accuracy trade-off for different graph dataset.