Sarah Kirby

Module 5 Assignment

Results:

Average clustering coefficient: 0.5706384782076823

Global clustering coefficient (transitivity): 0.2556818181818182

Random graph clustering coefficient: 0.13903743315508021

Program:

package assignments;  
  
import graph.GraphUtils;  
import org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.GraphTraversalSource;  
import org.apache.tinkerpop.gremlin.structure.Direction;  
import org.apache.tinkerpop.gremlin.structure.Vertex;  
import org.apache.tinkerpop.gremlin.tinkergraph.structure.TinkerGraph;  
  
import java.util.HashMap;  
import java.util.Iterator;  
import java.util.Map;  
import java.util.Vector;  
  
public class Module5 {  
  
 private String GRAPH\_FILE;  
  
 public Module5(){  
 GRAPH\_FILE = "GraphDatabases\\karate.graphml";  
 TinkerGraph graph = GraphUtils.*readGraphML*(GRAPH\_FILE);  
  
 System.*out*.println("Average clustering coefficient: " +  
 clusteringCoefficientAvg(graph));  
 System.*out*.println("Global clustering coefficient (transitivity): " +  
 clusteringCoefficientGlobal(graph));  
 System.*out*.println("Random graph clustering coefficient: " +  
 clusteringCoefficientRandom(graph));  
 }  
  
 */\*\*  
 \* Calculates the percentage of a vertex's neighbors that are linked  
 \* out of all possible links between neighbors  
 \** ***@param*** *v Vertex to calculate on  
 \** ***@return*** *local clustering coefficient for Vertex v  
 \*/* public double clusterCoefficientLocal(Vertex v){  
 double result = 0;  
 int triangles = 0;  
 int degree = 0;  
 Map<Integer, Vertex> neighborMap = new HashMap<>();  
  
 //add neighbors to map  
 Iterator<Vertex> neighbors = v.vertices(Direction.*BOTH*);  
 while(neighbors.hasNext()){  
 Vertex neighbor = neighbors.next();  
 neighborMap.put(neighbor.hashCode(), neighbor);  
 degree++;  
 }  
  
 //see how many neighbors are linked  
 neighbors = v.vertices(Direction.*BOTH*);  
 while(neighbors.hasNext()){  
 Vertex neighbor = neighbors.next();  
 Iterator otherNeighbors = neighbor.vertices(Direction.*BOTH*);  
 while(otherNeighbors.hasNext()){  
 if(neighborMap.containsKey(otherNeighbors.next().hashCode())){  
 triangles++;  
 }  
 }  
 }  
  
 double denominator = (degree \* (degree - 1));  
 if(denominator > 0) {  
 result = triangles / denominator;  
 }  
 return result;  
 }  
  
 */\*\*  
 \* Calculates the average of the local clustering coefficients  
 \* of each vertex in the graph  
 \** ***@param*** *graph The graph to operate on  
 \** ***@return*** *The average local clustering coefficient  
 \*/* public double clusteringCoefficientAvg(TinkerGraph graph){  
 double result = 0;  
 Vector<Double> coefficientVec = new Vector<>();  
 double sum;  
  
 //calculate coefficient for each node.  
 Iterator<Vertex> it = graph.vertices();  
 while(it.hasNext()){  
 coefficientVec.add(clusterCoefficientLocal(it.next()));  
 }  
  
 sum = MyUtils.*sumVector*(coefficientVec);  
 if(!coefficientVec.isEmpty()) {  
 result = sum / coefficientVec.size();  
 }  
 return result;  
 }  
  
 */\*\*  
 \* Calculates the global clustering coefficient of a graph by counting  
 \* all triples, and all closed triples.  
 \** ***@param*** *graph The graph to operate on  
 \*/* public double clusteringCoefficientGlobal(TinkerGraph graph){  
 int totalTriples = 0;  
 int closedTriples = 0;  
 double result = 0;  
  
 //for each node  
 Iterator<Vertex> vertices = graph.vertices();  
 while(vertices.hasNext()){  
 Vertex startVertex = vertices.next();  
 Iterator<Vertex> neighbors = startVertex.vertices(Direction.*BOTH*);  
  
 while(neighbors.hasNext()){ //for each neighbor  
 Vertex neighbor = neighbors.next();  
 Iterator<Vertex> neighbors2 = neighbor.vertices(Direction.*BOTH*);  
  
 while(neighbors2.hasNext()){ //for each neighbor's neighbor  
 Vertex neighbor2 = neighbors2.next();  
 if(!neighbor2.equals(startVertex)){  
 totalTriples++; //add to total triple count  
  
 //If triple is closed add to triangle count  
 if(*hasLink*(neighbor2, startVertex)){  
 closedTriples++;  
 }  
 }  
 }  
 }  
 }  
  
 if(totalTriples > 0){  
 result = (double)closedTriples/totalTriples;  
 }  
 return result;  
 }  
  
 public double clusteringCoefficientRandom(TinkerGraph graph){  
 double result = 0;  
 GraphTraversalSource g = graph.traversal();  
 int edgeCount = g.E().count().next().intValue();  
 int vertexCount = g.V().count().next().intValue();  
  
 //Get mean degree for a random graph  
 double avgDegree = (edgeCount \* 2.0) / vertexCount;  
 double denominator = vertexCount - 1;  
  
 if(denominator > 0){  
 // C = avgDegree / number of vertices - 1  
 result = avgDegree / denominator;  
 }  
 return result;  
 }  
  
 */\*\*  
 \* Checks if one vertex is linked to another  
 \** ***@param*** *from Vertex linked from  
 \** ***@param*** *to Vertex linked to  
 \** ***@return*** *True if the vertices are linked  
 \*/* public static boolean hasLink(Vertex from, Vertex to){  
 boolean result = false;  
 Iterator it = from.vertices(Direction.*BOTH*);  
 while(it.hasNext()){  
 if(it.next().equals(to)){  
 result = true;  
 break;  
 }  
 }  
 return result;  
 }  
  
 public static void main(String[] args){  
 new Module5();  
 }  
}