Hoan Tran

7 June 2019

CS 165 Project 3: Network Algorithms

Overview:

Examples of networks can be found throughout the world so modelling and analysing them becomes important. Networks are typically represented as graphs with nodes being different entities and the edges are the connections between them. Graphs can have different properties that represent different phenomena of the network it represents. For example, the Barabasi-Albert modelling of graphs highlights networks having "Preferential Attachment" which can be shown by how nodes connect to each other. The different properties that this report analyzes are the Diameter of a graph, the Clustering Coefficient of a graph, and the Degree Distribution for each node of a graph.

1. Barabasi-Albert Model

a. Summary: The goal of a Barabasi-Albert graph model is to produce a graph that highlights "Preferential Attachment" which means that a nodes with more connections are more likely to make even more connections to other nodes. In other words, while generating edges between nodes, nodes with a higher degree (or connections) will be more likely to generate a connection with a new node that is having its edges being processed. In this implementation, I also filtered out self-loops and multi-edges to simplify the graph.

b. Pseudocode:

```
Input: integer n, integer d
Allocate array M of size 2*n*d
For v in range(0, n-1)
      For i in range(0, d-1)
            M[2*(n*d+i)] = Node v //(v+1) for 1-indexing
            Random Value r between ( 0, 2*(v*d+i) )
            M[2*(n*d+i)+ 1] = M[r]
Edge Hash Table E
```

```
For j in range(0, (n*d)-1)
      Filter out Multi-edges and Self-loops
      Insert {M[2*j],M[2*j+1]} as an edge pair into E
```

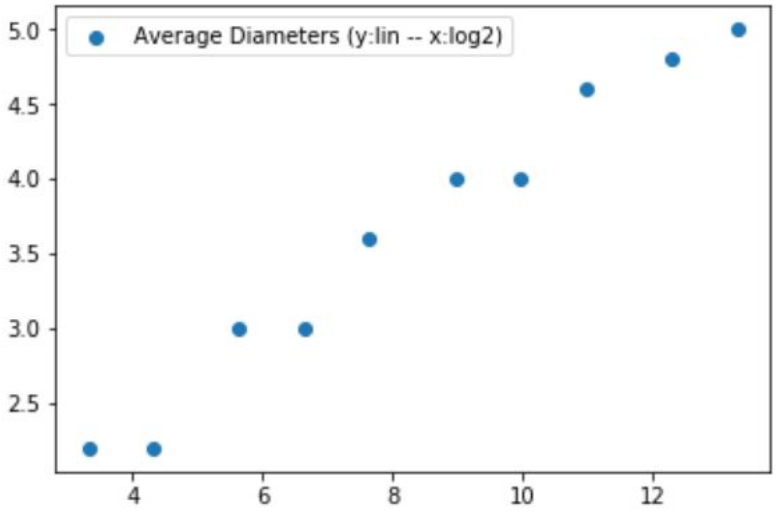
2. Diameter

a. Summary: Diameter is defined as the longest, shortest-path length over all node pairs. Unfortunately, calculating the diameter has a time complexity of O(nm) where n = number of nodes and m = number of edges which can be very time consuming. However, there are heuristic algorithms that can get the correct value or in some cases a value that is close to the correct diameter.

b. Pseudocode:

```
Vertex r = random vertex from the graph
Max Distance (so far) = 0
Perform a Breadth-First Search starting from vertex r
      and record the farthest distance
While farthest distance > Max Distance
      Max Distance = farthest distance
      Vertex r = farthest Node
      Repeat BFS
```

c. Results:



The diameter is plotted on a lin-log2 scale where the x-axis is log2(graph sizes) while the y-axis is the normal values from the diameter results. Each data point is an average diameter for each graph size {10, 20, 50, 100, 200, 500, 1000, 2000, 5000, 10000}. Based on the shape of the plot, diameter seems to grow as a function proportional to log(n) because it appears to be somewhat linear and a function of log(n) appears linear on a lin-log scale.

3. Cluster Coefficient

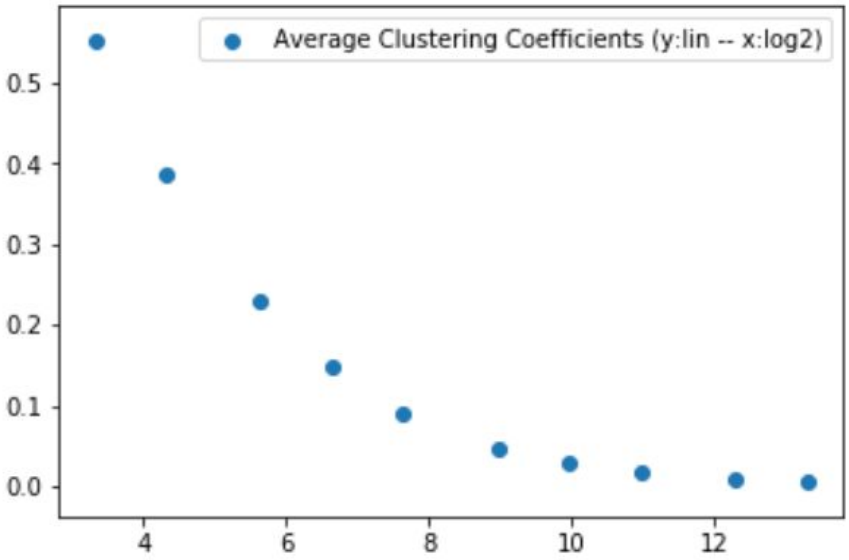
a. Summary: Clustering coefficient describes how nodes in a graph tend to group together and form cliques or groups. This can be representative of human connections and networks as people will sometimes become part of groups to socialize. Clustering coefficient is calculated by counting the number of triangles and dividing that by the number of 2-edge paths. Calculating 2-edge paths is easy because it is simply the (degree of a vertex)(degree of a vertex - 1) / 2. However, calculating the number of triangles can be very time consuming. A naive implementation may have a time complexity of $O(n^4)$. The degeneracy ordering of the graph can help calculate triangles faster. Degeneracy is the smallest value d for which every subgraph has a vertex with a degree of at most d. That means the nodes can be ordered in an array, list, etc. such that each node in the ordering has at most d neighbors that appear earlier within the ordering. By utilizing this special ordering, nearby nodes (from the degeneracy ordering) can be checked to see if they form triangles rather than iterating over every possible node combination.

b. Pseudocode:

```
Degeneracy Algorithm:
      Output List L
      Hash table H // checking if in L
      Map each value d_v to a corresponding node v
            // initially d_v is just degree of the node v
      Map lists of nodes that have the same d_v value in map D
      Map N_v : list of nodes that come before node v in L
      Pick smallest i in D[i] that is not empty
```

```
            Pick a node n from D[i] list
            Push n to front of L, Remove n from D[i] list, Insert n to H
            For each neighbor w of n
                  If w not in L // check via hash table H
                        Decrement value of d_v[w]
                        Remove w from old D[] list
                        Move w to new D[] list
                        Push w into N_v[n] list
            Return L and N_v
      Counting Triangles:
            For each node v in Degeneracy Ordering L:
                  For each pair of nodes (u,w) in N_v
                        If there is an edge between u and w
                              Increment Triangle Count
      Cluster Coefficient
            Calculate the number of Triangles
            Denom = 0
            For each node n:
                  Denom += (degree of n)*(degree of n - 1)/2
            Return (# of Triangle / Denom)
```

c. Results:



The clustering coefficient seems to decrease as the size of a graph grows. This might occur because the Barabasi-Albert Model makes more nodes make connections the more it already has so more nodes will make it easier for a given node to make a lot of connections meaning that there will be less grouping. From the lin-log scale, it is hard to pinpoint what the actual function is, but it is probably a function that "grows" faster than log(n) based on the curve if it increasing rather than decreasing.

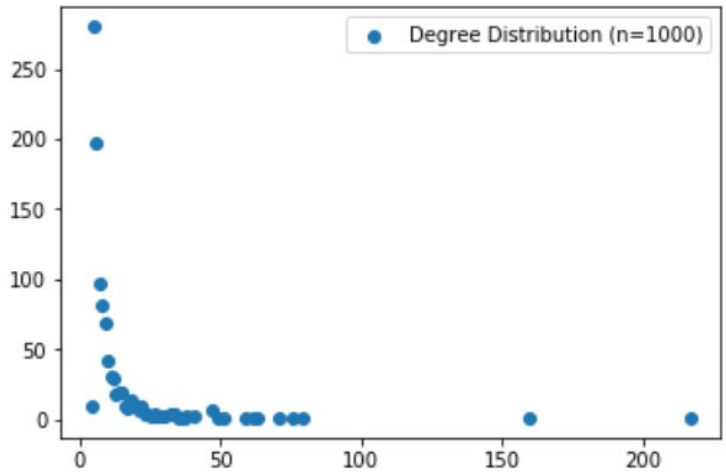
4. **Degree-Distribution**
    a. Summary: The degree distribution counts how many nodes have a certain degree (or number of neighbors). The Barabasi-Albert model ensures that the degree distribution will look like a power law where the graph has a high value at the beginning and draws out smaller values towards the end as a long tail. Calculating the degree distribution is simple since it just needs to map nodes to its degree value (or number of neighbors).
    b. Pseudocode:

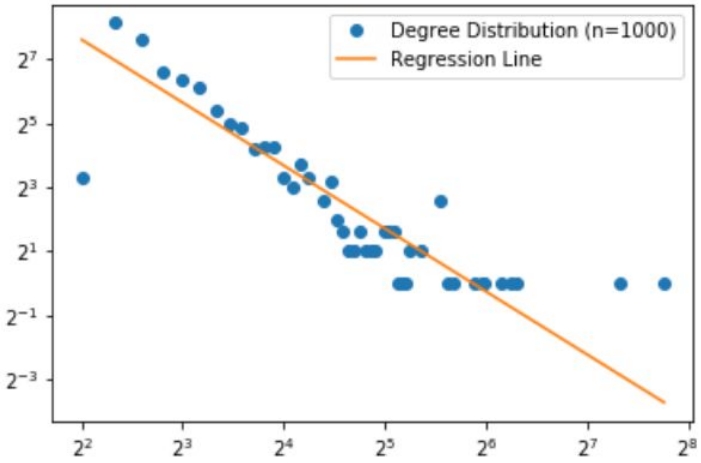
```
Map Hist
For each node n:
      Hist[degree of n] += 1
Return Hist
```

    c. Results:

The degree-distribution was recorded for graph sizes of n = {1000, 10000, 100000}. The data is plotted on a lin-lin scale to show the power law distribution and on a log-log scale to find a linear regression line to find the exponent of the power law. The regression line was made while excluding some values such as the first data point because they appeared to be outliers and would affect how the regression line's slope represents the power law for the original distribution.

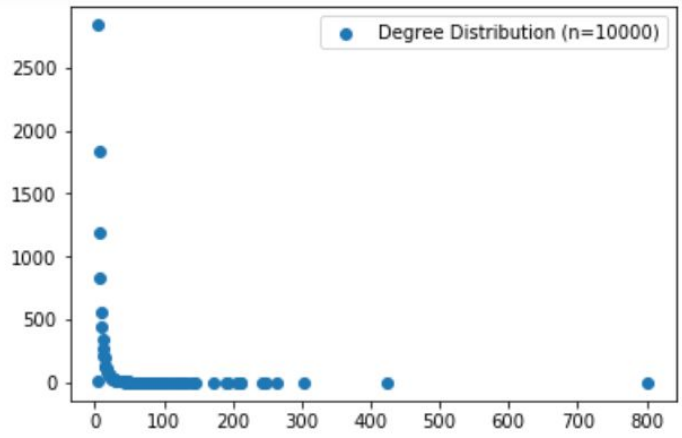
N = 1,000

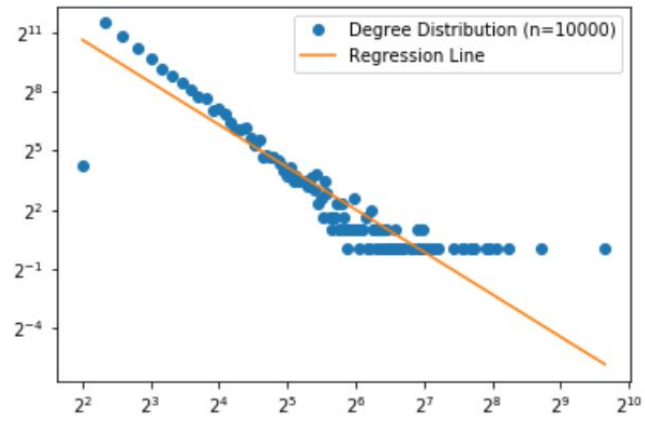





```
Regression Line Function (log2log2 scale): -1.9685913980469893x + 11.54478445246081
Degree(n) = n^-1.9685913980469893
```

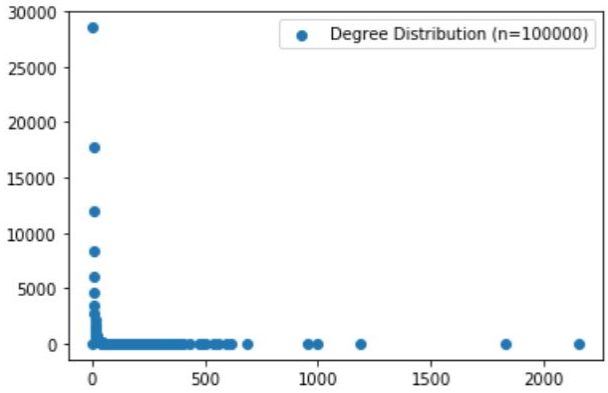
N = 10,000

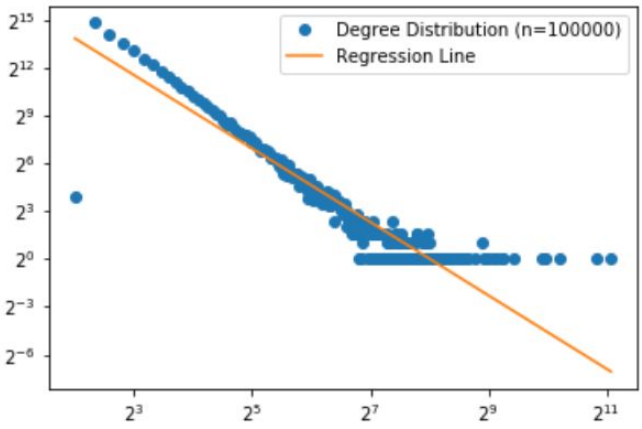





```
Regression Line Function: -2.153412625629466x + 14.925656519010616
Degree(n) = n^-2.153412625629466
```

N = 100,000






```
Regression Line Function: -2.3025461332026027x + 18.436553909222127
Degree(n) = n^-2.3025461332026027
```