# Experimentally Investigating three Properties of Large-Scale Real World Graphs

Johnny Ceja
COMPSCI 165
University of California, Irvine

## Prologue

### Introduction

Here we experimentally determine various properties of large-scale random graphs. Three particular properties are under investigation here. The first property is the diameter of a graph, which is the longest shortest path between any two pairs of vertices in a given graph. Secondly, we investigate the clustering-coefficient of a graph, which is meant to measure how closely coupled the nodes in the graph are with each other. The final property under investigation is the degree-distribution of a graph, which measures to what extent are nodes connected to other nodes. For the mentioned properties, we aim to find a (if any) correlation between the graph size and the given property; we calculate the property for graphs of monotonically increasing sizes. Each reported size is the result of five graphs being generated for that size and the results of all five being averaged to minimize variance.

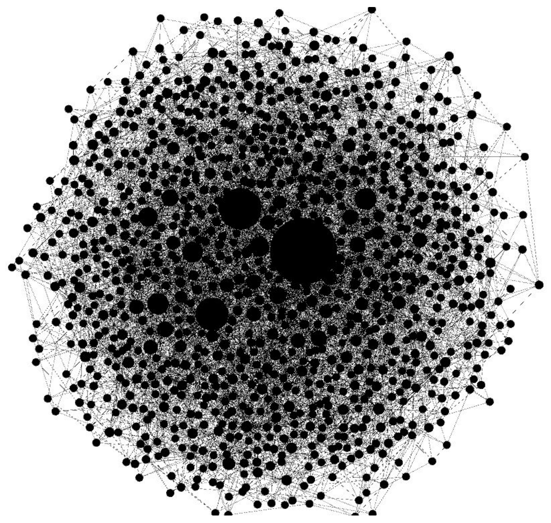
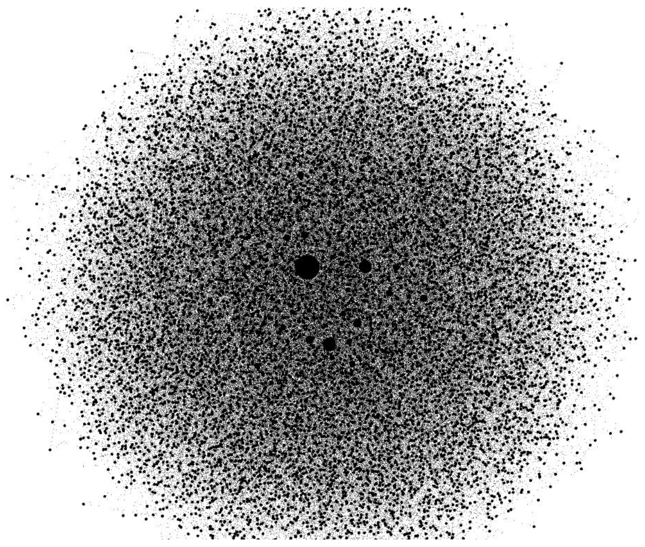
The graphs being used along with the properties being tested are done so under a philosophy that keeps the “real-world” in mind. Graph creation is done via the Barabasi-Albert model, which creates a pseudo-random graph of arbitrary size. We use this model to simulate real-world networks like the global network of Facebook friends. Additionally, the certain properties being investigated can also be related to “real-world” networks. The diameter of a graph can be seen as the link of mutual friends one has to traverse to go from one person to another person. The clustering coefficient quantifies the likelihood that two of your friends are also friends with each other. The last property, the degree-distribution, demonstrates how easy it is for already popular people to become even more popular relative to people who aren't already popular (also known as "the richer get richer and the poor get poorer").

## Graph Generation

```
func Barabasi (n, d = 5):
    G = undirected graph with 2 nodes connected to each other
    for i = 0 to n:
        add node v to G
            for j = 0 to d:
                for every previous node in G:
                    p = degree of current node / sum of all degrees in graph
                    add an edge from v to previous node with probability p
    return G
```

The Barabasi-Albert model for creating a random undirected graph works by starting with just two vertices connected via an edge. It then adds one vertex to the graph. This vertex is connected to one of the previous vertices via one edge with probability proportional to its degree. The number of edges a new vertex has with the existing graph is equal to *d*, which in our case is set to five, that is, a new vertex has five edges connected to the existing graph. This logic is repeated *n* times. In the end, there will be a small number of vertices that have a relatively high degree but most of the vertices will have a relatively low degree. By using the Barabasi-Albert model, we simulate a real-world network like the Facebook network of friends example previously mentioned. Note that there are alternative implementations with better running times than the aforementioned model. The implementation used in our working code uses an alternative, more efficient implementation, albeit the core logic of the algorithm remains the same.

The Barabasi-Albert model is able to generate graphs similar to this graph of 1000 vertices and 10000 vertices respectively (nodes are sized by their degree, using the Force Atlas graph visualization algorithm):

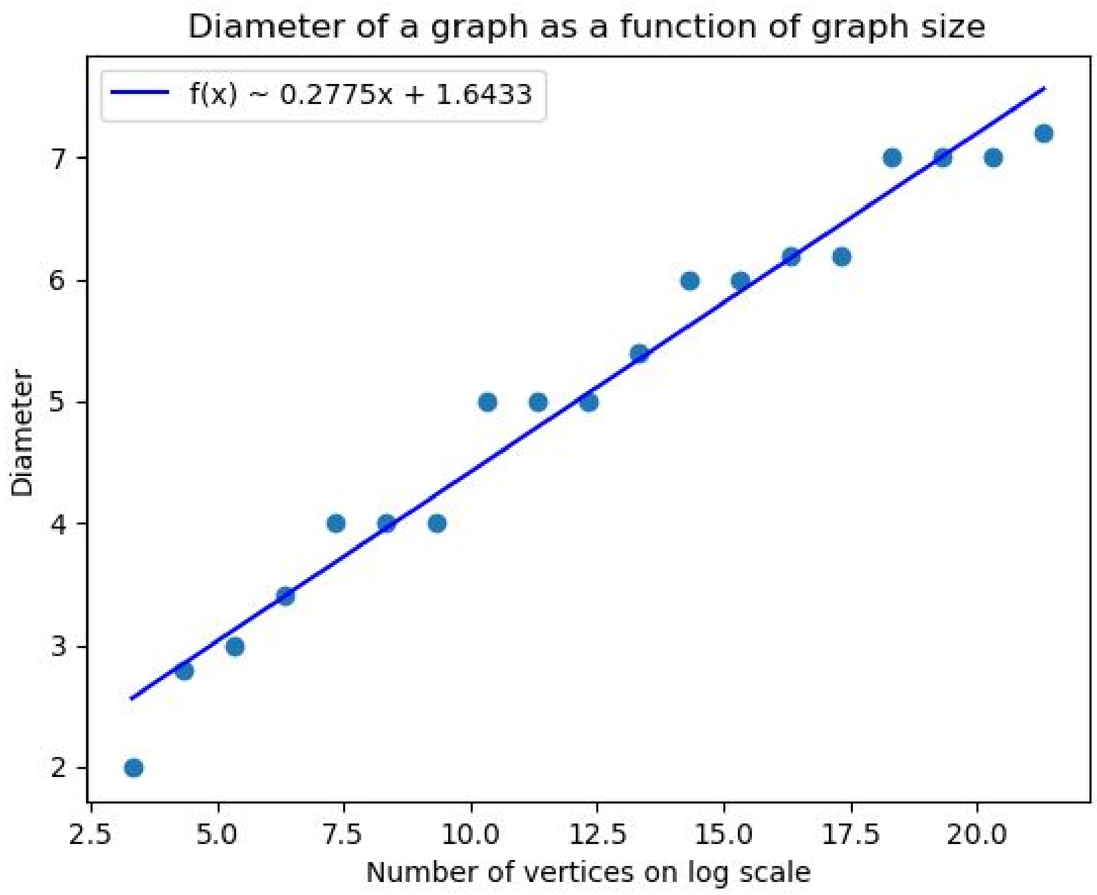



## Results

### *Diameter*

```
func Diameter(G):
    Dmax = 0
    result_struct = get_eccentricity(G.random_vertex)
    while(result_struct.result  > Dmax):
        Dmax = result
        # struct has from and to vertex, get the to vertex
        result = get_eccentricity(result_struct.to)
    return Dmax
```

The diameter of a graph is the longest shortest path of two vertices in the graph. One way to acquire this value is to run an algorithm like the Floyd Warshall algorithm. A downfall to this approach has to do with the size of the graph that we are working with; running fully correct algorithms would be too computationally expensive, as such, we turn to using heuristics in calculating the diameter of a graph. The above pseudocode gives the details of the heuristic we used. It calculates the eccentricity of a random vertex in the graph. It follows this logic of continuously calculating the eccentricity of the *to* vertex of the previous calculation, until the current calculation returns an eccentricity that is not bigger than the last.

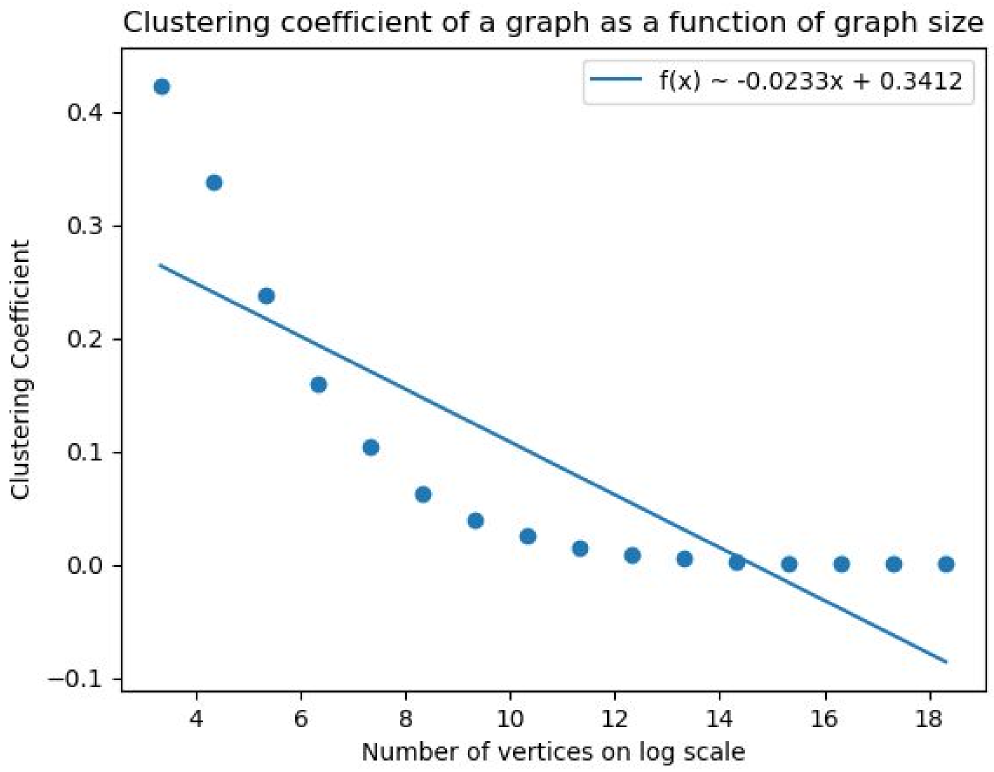


The results of our testing is shown above. The x-axis is on a $log_2$ scale and the y-axis is on a linear scale. Do notice that the linear regression line fits nicely with the data shown. Since the x-axis is on a *log* scale, we can say that the diameter of a graph grows proportionally to the *logarithm* of the number of vertices in the graph.

**Clustering Coefficient**

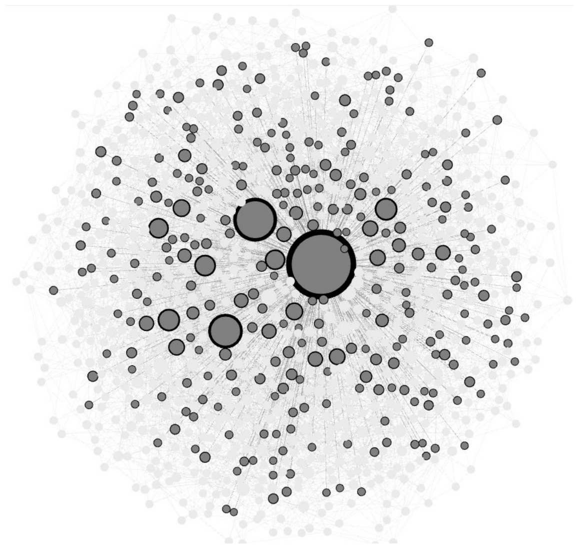
```
func clustering_coefficient(G):
    triangles = 0
    # get number of triangles
    for every vertex v in G:
        for every distinct pair of edges u, w of v:
            if u, w have higher degree than v or higher id if equal degree:
                if u, w are connected via an edge:
                    ++triangles;
    # get total number of two-path edges
    two_paths = 0
    for every vertex v in G:
        two_paths += degree(v) * (degree(v) - 1) / 2
    return 3 * triangles / two_paths
```

The clustering coefficient of a graph returns a real number that quantifies how tightly coupled vertices in the graph are. The way to calculate this value is essentially to calculate the ratio of three times the number of triangles in the graph over the sum of all the two-path edges.



The results of our testing is shown above. The x-axis is on a $log_2$ scale and the y-axis is on a linear scale. It is debatable as to whether the linear regression line fits the data. We have made the judgement call that it does **not** fit the data. Given the scale used in this particular graph, it appears that the data forms a line inversely proportional to the x-axis. Since the x-axis is on a log scale, we can say that the clustering coefficient appears to shrink proportionally to $log_2(\frac{1}{n})$ of the graph, where *n* is the number of vertices in the graph. Note that $log_2(\frac{1}{n})$ shrinks much more slowly than $log_2(n)$. The horizontal asymptote in this case will clearly be the x-axis, and the vertical asymptote will be the y-axis; meaning that the clustering coefficient will never be less than 0, and more than 1, which makes logical sense.

**Degree-Distribution**

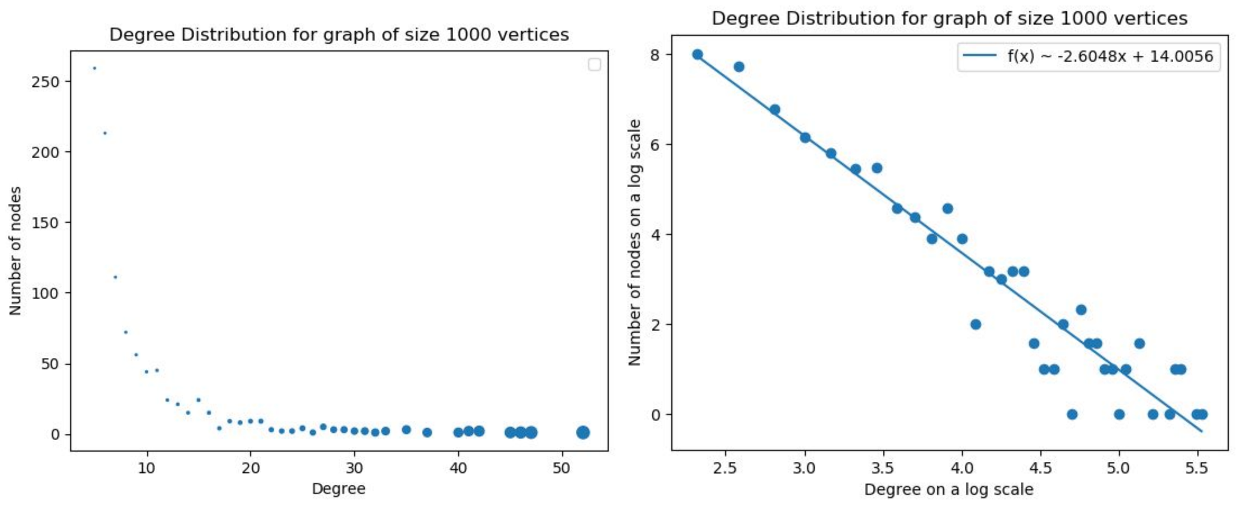


```
func deg_dist(G):
    # map degree to count of vertices having that degree
    map = {}
    for every vertex v in G:
        ++map[degree_of_vertex(v)]
    return map
```

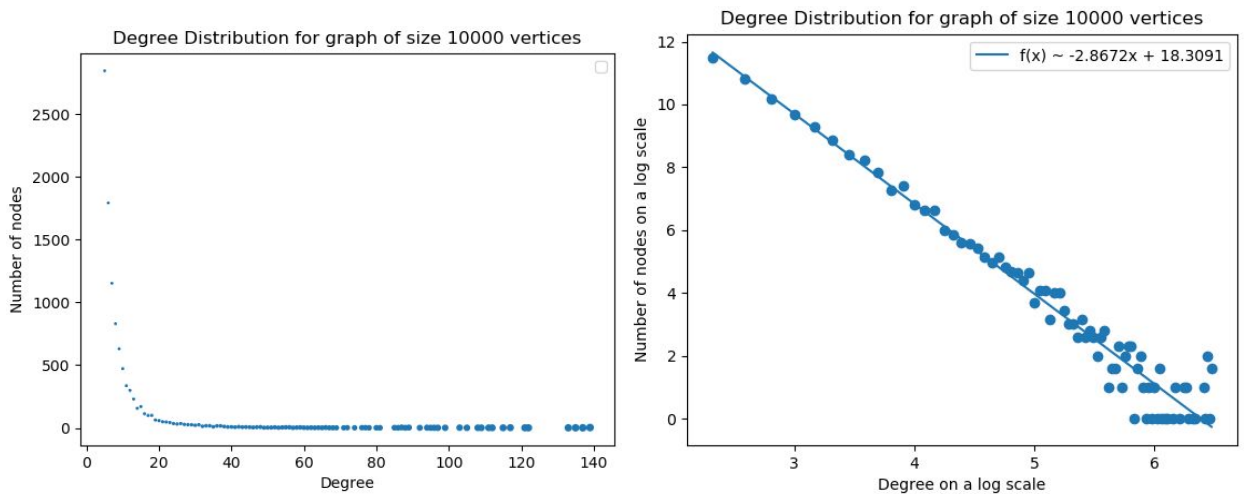
The degree-distribution is a simple concept; it maps each valid degree in G with the number of vertices in G that have that degree. In the above graph, we can see the vertex with the highest degree as the biggest node; notice it is connected with many other vertices. The "rich get richer" phenomenon is exhibited in this graph.

The above pseudocode is nearly trivial; it iterates through all vertices in the graph and maps the degree of that vertex with its corresponding degree. Using a map allows for all vertices with the same degree to be clumped together so as to keep a count of all vertices with that degree.

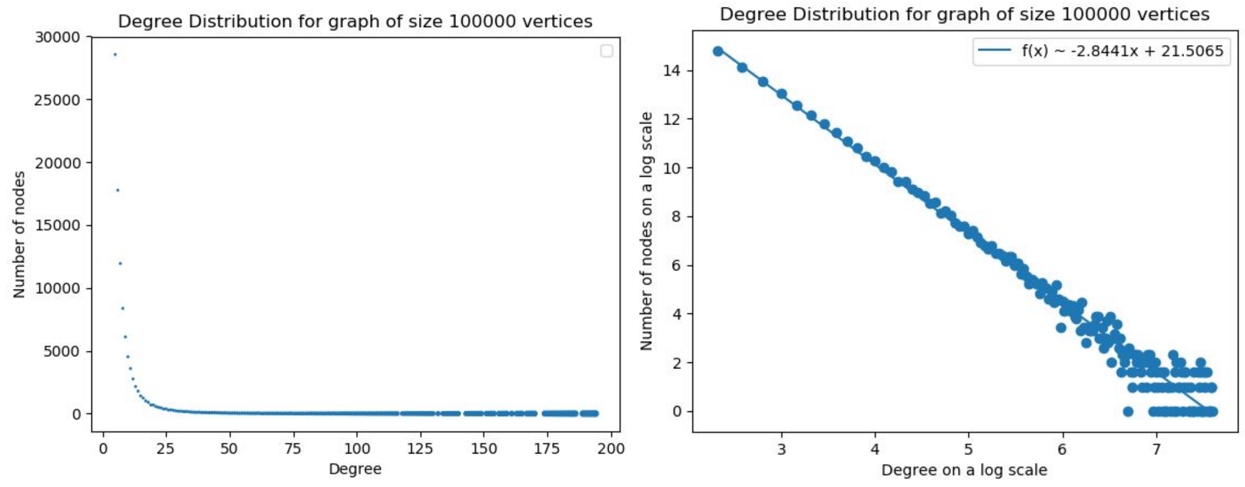
Only three discrete graph sizes were tested with this graph property, namely graphs with 1000, 10000, and 100000 vertices. As expected, there are many vertices associated with relatively low numbered degrees. There is only a few vertices associated with a relatively high numbered degree. Shown below are the graphs of the number of vertices as a function of degree (left) and degree on a log scale (right).



*Pertaining to a graph of 1,000 vertices*



*Pertaining to a graph of 10,000 vertices*



*Pertaining to a graph of 100,000 vertices*

For the above graphs on the left, the x-axis is the order of the degree and the y-axis is the number of vertices associated with that degree. The above graphs on the right are slightly different, the x-axis is the order of the degree on a log scale and the y-axis is the number of vertices associated with that degree on a log scale. It is clear that the linear regression line fits the data; furthermore, the graphs obey a power law inasmuch as the asymptotes allow them to. The slopes of the power laws are reported on the graphs above. Since the linear regression lines fit the data points, we can say that the number of vertices that are associated with certain degree shrinks linearly proportional to the order of the degree.

## <u>Conclusion</u>

| ***Property*** | ***Proportional to*** |
|---|---|
| **Diameter** | $log_2(n)$ |
| **Clustering Coefficient** | $log_2(\frac{1}{n})$ |
| **Degree Distribution** | $n$ |

We have discovered that all three properties investigated are proportional in some way to the number of vertices in the graph when using the Barabasi-Albert model. If we consider the graphs generated by such a model to be sufficiently similar to real-world networks, we can extend these findings to those networks. Our findings relating to the proportionality of the tested properties can be summarized in the tables above;  all of which consider *n* to be the number of vertices in a given graph.