Erdos-Renyi Random Graphs

Pseudocode

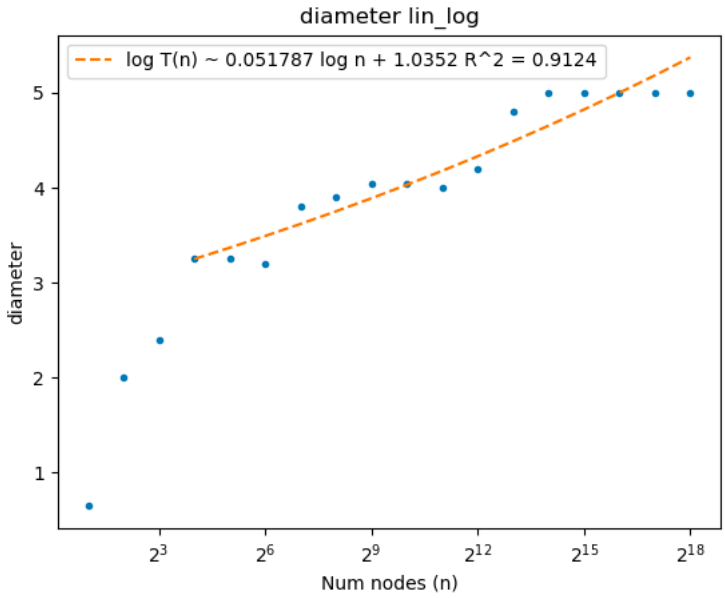
```
# If your student ID is an odd number, you should test the above algorithms on Erdos-
Renyi random graphs, G(n,p),
# with p = 2(ln n)/n, where "ln n" denotes the natural logarithm of n.
def erdos_renyi(n: int) -> Iterable[tuple[int, int]]:
    p = 2 * (math.log(n)) / n
    edges = set()
    v = 1
    w = -1
    while v < n:
        r = random.random()
        # Almost all module functions depend on the basic function random(), which
generates a random float uniformly in the half-open range 0.0 <= X < 1.0
        # https://docs.python.org/3/library/random.html

        w += 1 + int(math.log(1-r) / math.log(1-p))
        while w >= v and v < n:
            w -= v
            v += 1
        if v < n:
            edges.add(tuple([v, w]))

    return edges
```

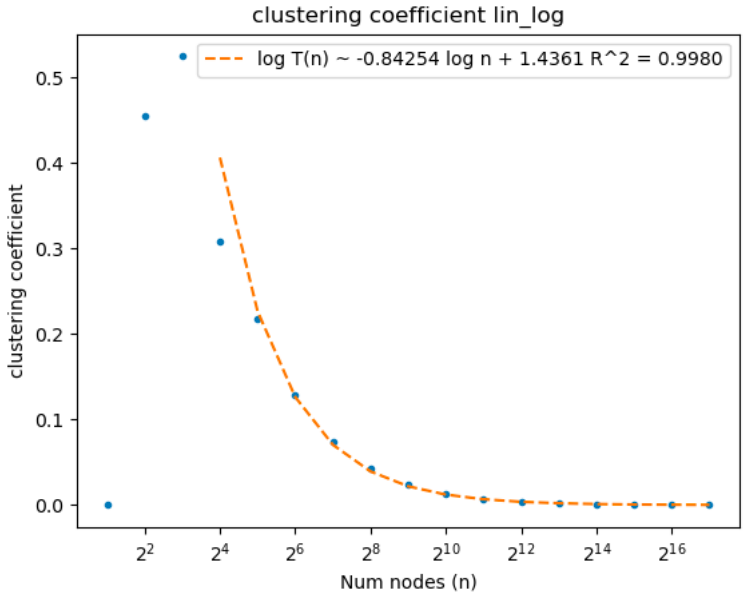
This algorithm iteratively selecting a random vertex and then offset based on the calculated probabilities to generates a random graph. It continues visiting until all vertices have been visited, and then it adds the corresponding vertex edge's pairs as to the edge set.

Diameter



From the graph, it seems like the diameter does not relate to the num of nodes much due to the ratio of the diameter with the number of nodes, but in fact they have relationship, when the number of nodes is much bigger, then the size diameter may increase a little bit.
Thus, from the experiment, we can see the diameter values grow as a function of n. It grows proportional to the function, log n.

Clustering Coefficients



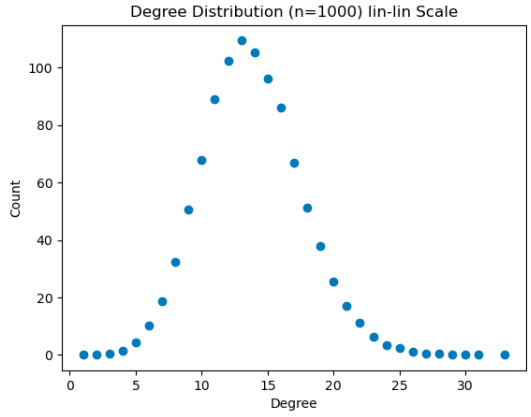
The clustering coefficient is proportional to the number of nodes since it decreases as the number of nodes increases. As the number of n grows, the clustering coefficient approach p.

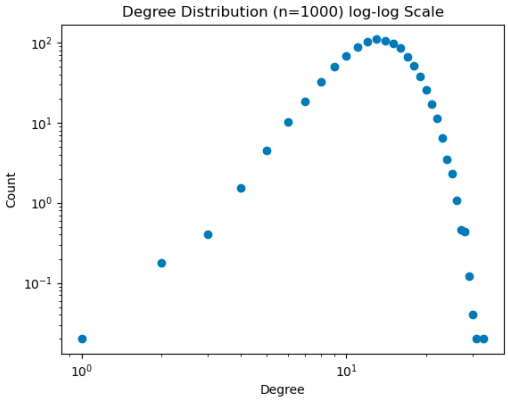
$$p = E / (N(N-1)/2)$$

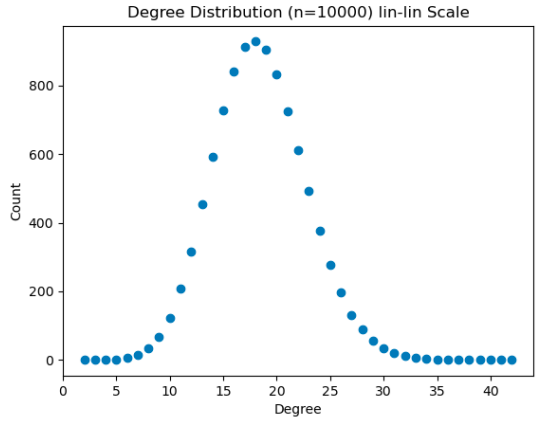
$p = 2(\ln n)/n$

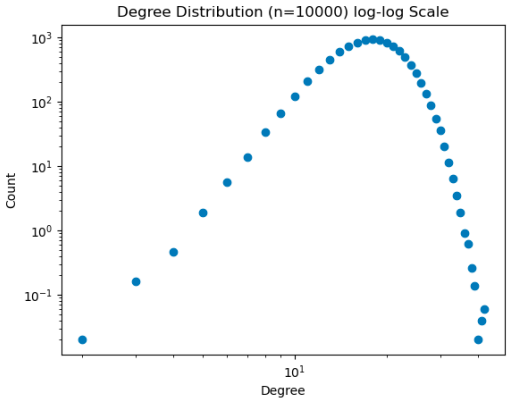
From TA lecture, it looks like log(n)/x and Desmos shows that it seems like an exact fit. Thus, I have a confidence in this result.

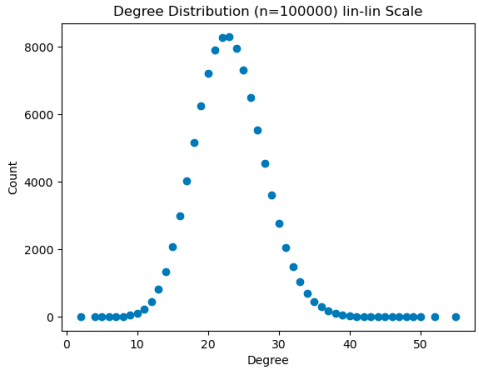
Degree Distribution

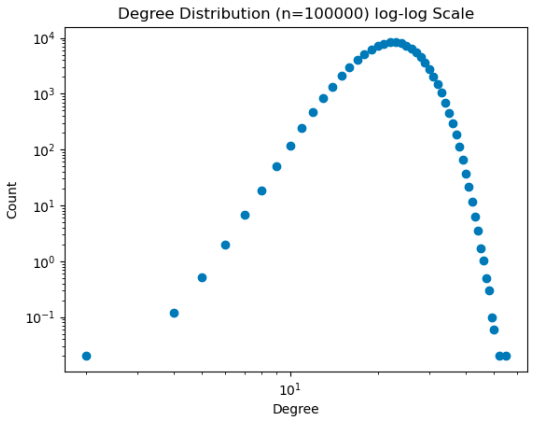












From the graph, we can see that when the size of node is large, it roughly follows Poisson distribution, so the Erdos-Renyi graphs does not follow a power-law.

Extra Credit

Barabasi-Albert Random Graph

Pseudocode

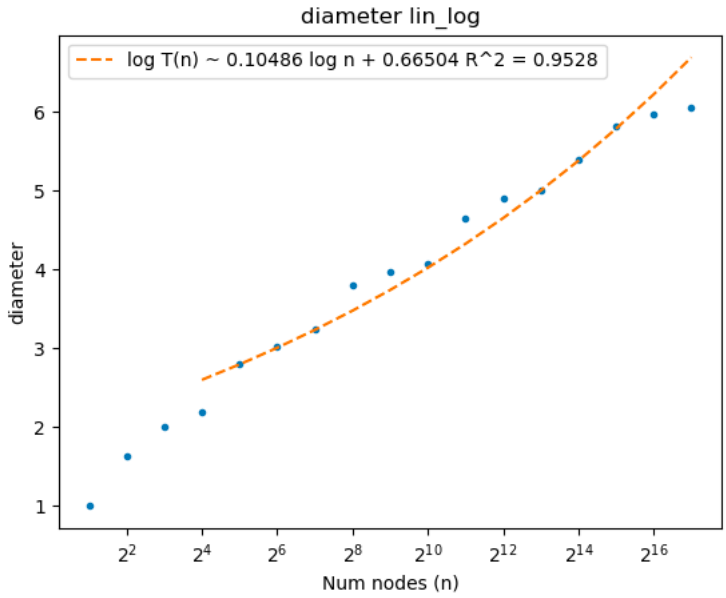
```
# If your student ID is an even number, you should test the above algorithms on
Barabasi-Albert random graphs,
# generated with the parameter d = 5 as the number of neighbors each new vertex
chooses.
def barabasi_albert(n: int) -> Iterable[tuple[int, int]]:
    d = 5
    m = [0] * (2 * n * d)
    for v in range(n):
        for i in range(d):
            m[2 * (v*d + i)] = v
            r = random.randint(0, 2 * (v*d + i))
            m[2 * (v*d + i) + 1] = m[r]

    edges = set()
    for i in range(n*d):
        edges.add(tuple([m[2*i], m[2*i + 1]]))

    return edges
```

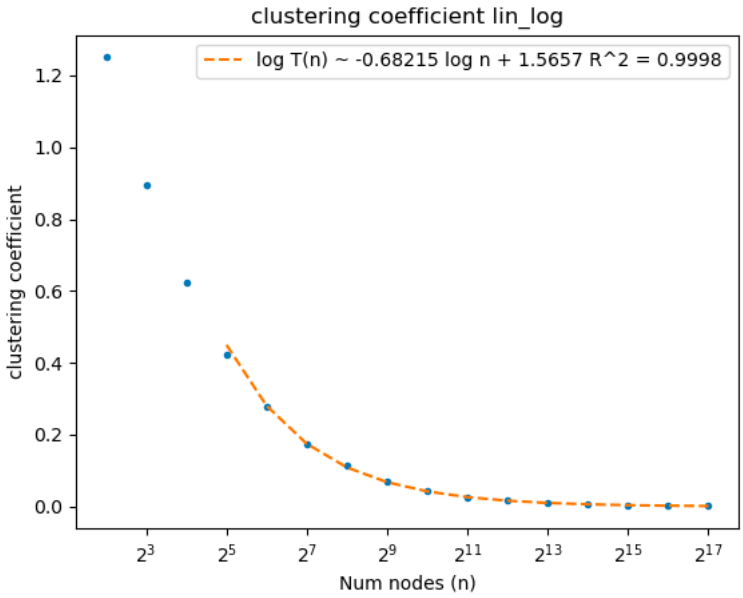
The Barabasi-Albert algorithm use the preferential attachment principle to generates a random graph. It iteratively adds vertices and connects each of them to existing vertices. The algorithm makes sure that new vertices are more likely to connect to vertices with a higher degree.

Diameter



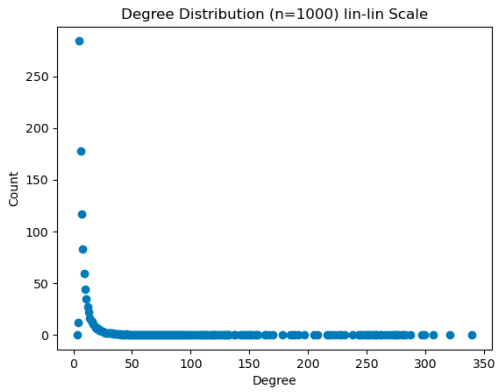
This diameter grows as the number of nodes increase, which is proportional to log n.

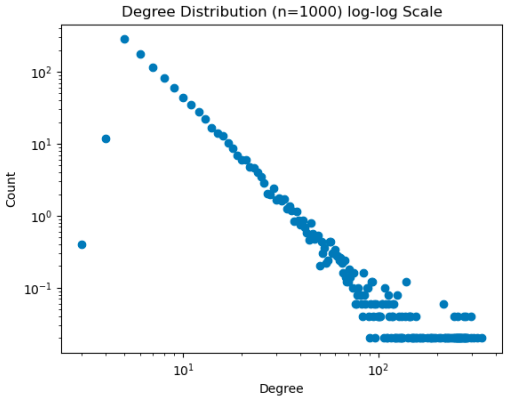
Clustering coefficient

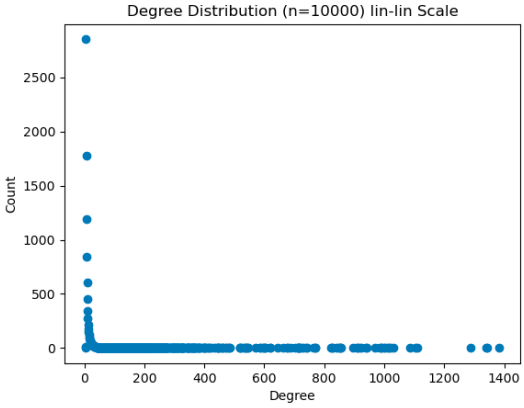


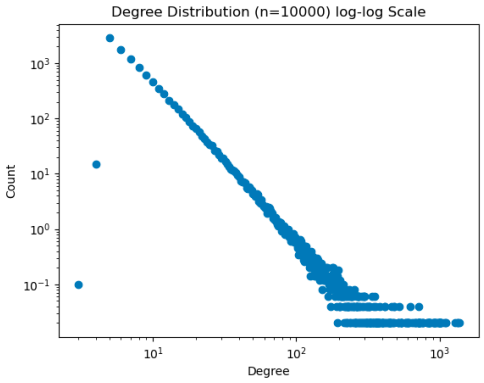
This cluster coefficient decrease as the number of nodes increases, which is proportional to n.

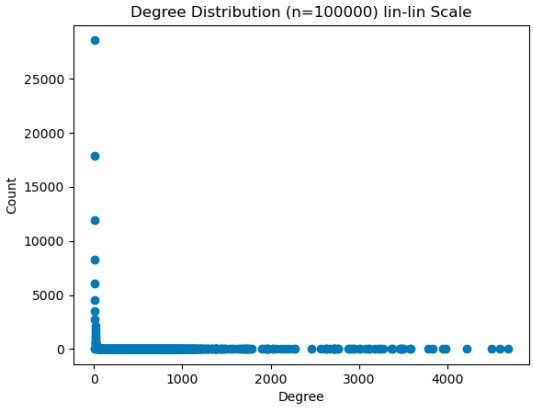
Degree Distribution

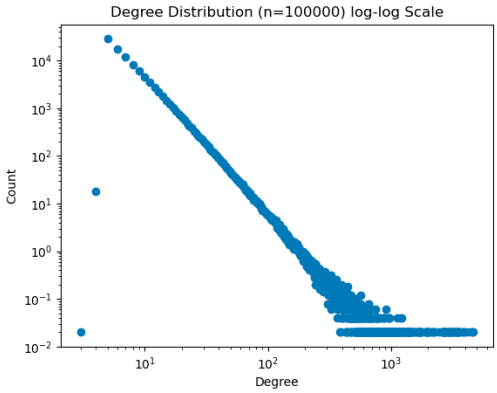












It seems like have the power like with the exponent of -3 according to lecture slide.