Lab 4 of
Information Retrieval (IR)

Qiaorui Xiang

1 Implementation

We use **Python** as our programming language for this lab. We also use the package **network** to generate random graph and to calculate clustering coefficient and shortest path. In order to show plot as required, we used the package **matplotlib.pyplot**. The detail of implementation is in the file **er.py** and **ws.py**.

We implement the **Erdos-Renyi** model by introduce two parameters: n nodes and m edges. Since ER model is chosen uniformly at random, the m can be calculated from $m = \binom{n}{2}p$ where p is the probability to assign the edge. However, the value of p should not be the same with different size of graph. So in order to keep the graph in same ratio and connected at same time, we use the value ϵ to determine the value p. If $p > \frac{(1+\epsilon)\ln n}{n}$ then a graph in $G_{ER}(n, m)$ will almost surely be connected. Thus we build the script that runs max iterations, for each iteration i, $n = 10 * 2^i$ and with fixed ϵ to obtain p dynamically.

The **Watts-Strogatz model** is implemented using three parameters: n nodes, k nearest neighbors to be connected for each node initially and p probability of rewiring the edges in the initial network. The k and n is fixed for all WS graph we want to compare. However, the p value has to change by each iteration, so we build the script that runs max iterations, for each iteration i, $p = \frac{1}{1.5^i}$.

2 Results

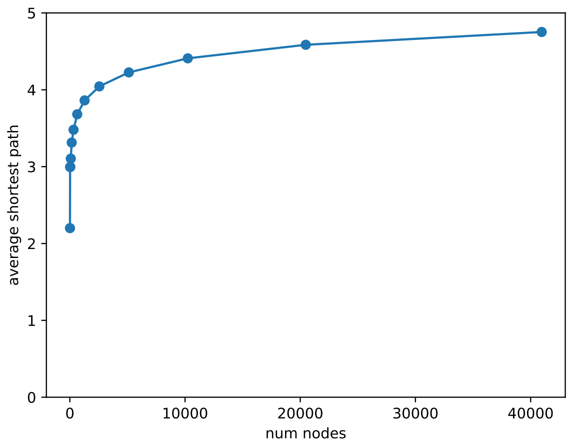

Figure 1: Erdos-Renyi model

The result of following command is shown in Figure 1. We ran the script with 13 iterations and $\epsilon = 0.05$. After 13 iterations we can obtain a graph of 40960 nodes, this is the maximum number that our machine can bear.

```
python er.py -max 13 -e 0.05
```

As shown in Figure 1, the G_{ER} shows **small diameter** even when we increase exponentially the number of nodes. Which is exactly what we want to mimic the *small world* phenomenon.

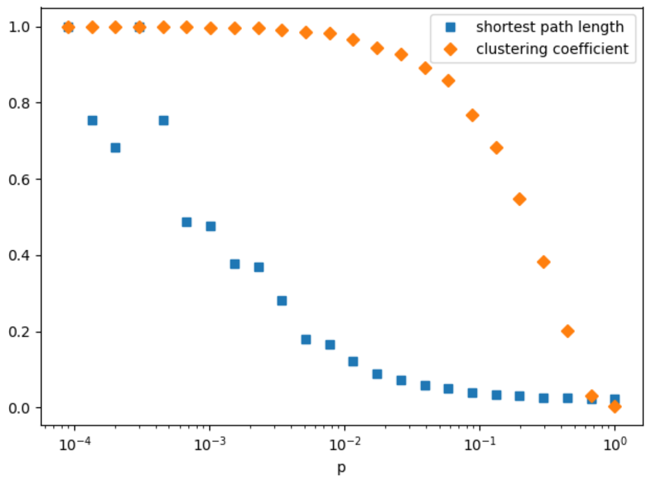

Figure 2: Watts-Strogatz model

The result of following command is shown in Figure 2. We ran the script with 23 iterations and $n = 2048$ $k = 4$. 23 iterations can give us different value of p with range 0.0001 to 1.

```
python ws.py -max 23 -n 2048 -k 4
```

As shown in Figure 2, the G_{WS} shows high clustering and big diameter when p is very small, low clustering and small diameter when p is nearly 1. On the other hand we can observe **small diameter** and **high clustering** around $p \approx 0.01$. If we want to mimic the real world network, then set $p = 0.01$ is a reasonable choice.