As we need to construct a continuous mapping $f : H(\in \mathbb{R}^{|V|\times|E|}) \longrightarrow A(\in \mathbb{R}^{|V|\times|V|})$ we start by considering HH^T. By definition,

$$h_{ij} = \begin{cases} 0 & \text{if } v_i \notin e_j \\ 1 & \text{if } v_i \in e_j \end{cases}$$

Let $C = HH^T$. Then, $c_{ij} = \sum_{k=1}^{n} h_{ik}h_{jk}$. Note that c_{ij} can be rewritten as

$$c_{ij} = \sum_{k=1}^{n} \mathbb{1}_{v_i \in e_k} \times \mathbb{1}_{v_j \in e_k} \tag{1}$$

Since G given in the question is an undirected graph,

$$c_{ij} = \begin{cases} \text{No. of edges shared between } v_i \text{ and } v_j & \text{if } i \neq j \\ d(v_i) & \text{if } i = j \end{cases}$$

Therefore, required mapping is

$$A = HH^T - D \tag{2}$$

where D is the degree matrix of the graph G.

Theorem 0.1 (Hoeffding's Inequality)**.** *Let $X_1, X_2, ..., X_n$ be a series of independent random variable with $a_i \leq X_i \leq b_i$. Let $S_n = \sum_{i=1}^{n} X_i$ then,*

$$\mathbb{P}\big[S_n - \mathbb{E}[S_n] \geq t\big] \leq \exp\left(-\frac{2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right)$$

Proof. Let X be any random variable and $s > 0$. Then,

$$P\big(X \geq t\big) = P\big(e^{sX} \geq e^{st}\big) \leq e^{-st}E(e^{sX})$$

, by using Markov's inequality and the fact that e^x is a monotonically increasing function. Let's now look at $\sum_{i=1}^{n} X_i - E[X_i]$. Then,

$$\begin{aligned} P\big[\sum_{i=1}^{n} X_i - E[X_i] \geq t\big] &\leq e^{-st}E\left[e^{s\left(\sum_{i=1}^{n} X_i - E[X_i]\right)}\right] \\ &= e^{-st}E\left[\prod_{i=1}^{n} e^{s\left(X_i - E[X_i]\right)}\right] \\ &= e^{-st}\prod_{i=1}^{n} E\left[e^{s\left(X_i - E[X_i]\right)}\right] \end{aligned}$$

The last step follows from independence of the X_i's.

Lemma 0.2. *Let X be a random variable with $E[X] = 0$ and $a \leq X \leq B$ with probability one. Then,*

$$E[e^{sX}] \leq e^{\frac{s^2(b-a)^2}{8}}$$

.

Proof. By the convexity of exponential function we get

$$e^{sx} \leq \frac{x-a}{b-a}e^{sa} + \frac{b-x}{b-a}e^{sb}, \text{ for } a \leq x \leq b$$

. Thus,

$$\begin{aligned} E[e^{sx}] &\leq E\left[\frac{x-a}{b-a}\right]e^{sa} + E\left[\frac{b-x}{b-a}\right]e^{sb} \\ &= \frac{b}{b-a}e^{sb} - \frac{a}{b-a}e^{sa} \text{ as } E[X] = 0 \\ &= \big(1 - \lambda + \lambda e^{s(b-a)}\big)e^{-\lambda s(b-a)} \text{ for } \lambda = -\frac{a}{b-a} \end{aligned}$$

Let $u = s(b-a)$ and define,

$$\phi(u) \equiv -\lambda u + \log(1 - \lambda + \lambda e^u)$$

. From Taylor's theorem we have

$$\phi(u) = \phi(0) + u\phi'(0) + \frac{u^2}{2}\phi''(v), \text{ for some } v \in [0, u].$$

$$\begin{aligned}
\phi(0) &= 0 \\
\phi'(u) &= -\lambda + \frac{\lambda e^u}{1 - \lambda + \lambda e^u} \implies \phi'(0) = 0 \\
\phi''(u) &= \frac{\lambda e^u}{1 - \lambda + \lambda e^u} - \frac{\lambda e^u}{(1 - \lambda + \lambda e^u)^2} \\
&= \frac{\lambda e^u}{1 - \lambda + \lambda e^u}\left(1 - \frac{\lambda e^u}{1 - \lambda + \lambda e^u}\right) \\
&= \theta(1 - \theta), \text{where } \theta = \frac{\lambda e^u}{1 - \lambda + \lambda e^u}
\end{aligned}$$

$\theta(1-\theta)$ has a max value of $\frac{1}{4}$ for $\theta = \frac{1}{2}$. Therefore, $\phi(u) \leq \frac{u^2}{8}$ and $E[e^{sX}] \leq \frac{s^2(b-a)^2}{8}$. □

Let us apply this result on Hoeffding's inequality.

$$\begin{aligned}
P\big[S_n - E[S_n] \geq t\big] &\leq e^{-st} \prod_{i=1}^{n} E\left[e^{s\left(X_i - E[X_i]\right)}\right] \\
&\leq e^{-st} \prod_{i=1}^{n} e^{\frac{s^2(b_i-a_i)^2}{8}} \\
&= e^{-st} e^{s^2 \sum_{i=1}^{n} \frac{(b_i-a_i)^2}{8}} \\
&= e^{ks^2 - st}, \text{where } k = \sum_{i=1}^{n} \frac{(b_i - a_i)^2}{8} \\
&\leq e^{\frac{-t^2}{4k}}, \text{ for } s = \frac{t}{2k} \\
&= \exp\left(-\frac{2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right)
\end{aligned}$$

□

This proof is based on the lecture notes of Prof. Robert D Nowak: https://nowak.ece.wisc.edu/SLT09/lecture7.pdf.

Let $u \in V = \{1, 2, ..., n\}$ be a node in an ER graph denoted by $G_{n,p}$. Let us define a random variable X_i s.t.,

$$X_i = \begin{cases} 1 & \text{if } e_{ui} \in E(G_{n,p}) \\ 0 & \text{if } e_{ui} \notin E(G_{n,p}) \end{cases}$$

Denote $S_n = \sum_{i \neq u} X_i$. Note that $S_n = d_u$. We also have $E[S_n] = (n-1)p$. Choosing $t = \epsilon np$, $\epsilon > 0$ and applying Hoeffding's inequality we get,

$$P[S_n - (n-1)p \geq \epsilon np] \leq \exp\left(- \frac{2n^2\epsilon^2p^2}{n-1} \right)$$

. For large n, $n - 1 \approx n$ and we have

$$P[d_u \geq (1+\epsilon)np] \leq \exp\big(- 2n\epsilon^2p^2 \big)$$

.

This question is code based. The following code generates Erdös-Rényi graph 100 times and check if it is connected.

```
connected_count = 0
for expt in range(100):
    G = gen_er_graph(n, p_f)
    if is_connected(G):
        connected_count += 1
```

The following is the code for the abstract method `gen_er_graph(n, p)`

```
def gen_er_graph(num_nodes, p_act):
    g = Graph()
    nodelist = [i for i in range(num_nodes)]
    for e in combinations(nodelist, 2):
        if flip(p_act):
            g.add_edge(e[0], e[1])
return g
```

The `flip(p)` method simulates tossing a biased coin with probability of showing head as p. For reproducability the `np.random.seed()` is set to 33

	n				
p	100	200	400	800	1600
0.95	95	100	100	100	100
0.98	96	99	97	100	100
1.02	98	100	100	100	100
1.05	98	100	100	100	100

Table 1: Number of times (out of 100) graph is connected for different n, p combinations.

The number of edges in a complete undirected graph is $\binom{n}{2} = \frac{n(n-1)}{2}$, where $|V(G)| = n$. For an ER graph, expected number of edges,i.e. $E\big[|V(G)|\big] = \frac{n(n-1)p}{2}$. Given in the question $n = 2708$.Then,

$$\frac{n(n-1)p}{2} = 2708$$
$$p = 0.00148119$$
$$p = 0.5074\frac{\log n}{n}$$

Again we take help of `gen_er_graph(n, p)` to generate an ER graph.

```
G_2 = gen_er_graph(num_nodes, p)
```

The next part is to compute clustering coefficient of a node and clustering coefficient of a graph.

```
def cluster_coefficient(graph: Graph, u: int):
    nbrs_u = Neighbors of node u
    num_links = 0
    for each node v in nbrs_u:
        if there is an edge between v and node of nbrs_u which is not u:
        num_links = num_links + 1
    k = number of neighbors of node u
    if k < 2:
        return 0
    cc = num_links / k * (k - 1)
    return cc
```

Note that in the above pseudocode we do not divide by $\binom{k}{2}$. Instead we divide by $k \times (k-1)$. The reason is we are counting every link twice by this approach. For example, let us consider $nbrs_u = \{v, w, x\}$. If there is an edge $\{v, w\}$, it will appear both times when considering v and w.

```
def graph_cluster_coefficient(graph: Graph):
    cc_avg = 0.0
    for node in graph.nodelist:
        cc_i = cluster_coefficient(graph, node)
        cc_avg = cc_avg + cc_i
    cc_avg = cc_avg / graph.num_nodes()
    return cc_avg
```

Results

```
Number of edges in G_1(cora-network) is 5278
G_1 Clustering coeff:  0.2406732985019372
Number of edges in G_2 is 5495
G_2 Clustering coeff:  0.0010952706441428241
```

Let us denote the number of edges in the Watts-Strogatz model as E and the degree of a node u as d_u. Then,

$$\sum_{u \in V} d_u = 2E.$$

As each node is connected to K nodes ($K/2$ on the right-side and $K/2$ on left-side) then,

$$E = \frac{NK}{2}, \text{ where } N \text{ is the number of nodes.}$$

Given in the question $N = 2708, E = 5429$. Thus

$$\begin{aligned} K &= \left\lceil \frac{2E}{N} \right\rceil \\ &= \left\lceil \frac{2 \times 5429}{2708} \right\rceil \\ &= 4 \end{aligned}$$

Then we just iterate over edges and based on the probability we rewire the edges. This is done as follows:

```
G_tmp = G_2.__copy__()
    edges_to_traverse = list(G_tmp.edge_list)
    for e in edges_to_traverse:
        if flip(q):
            G_tmp.remove_edge(e[0], e[1])
            new_e = np.random.randint(0, N, (2, ))
            G_tmp.add_edge(new_e[0], new_e[1])

    print('q:', q, 'CC: ', cc_avg)
```

To estimate the rewiring parameter q we can do a linear search in the interval $(0, 1)$ with a step of 0.05.

```
q_range = [0.05, 0.1, ..., 1]
for q in q_range:
    # Rewire Watts-Strogatz model as above
    cc_avg = cluster_coefficient of rewired Watts-Strogatz graph
    report argMin(|clustering_coefficient_of_cora - cc_avg|)
# Report nearest clustering coefficient and rewiring probability q.
```

Result By the above method it turns out q to be 0.25.

```
q: 0.25, Nearest cluster coefficient  0.22497479543269888
Clustering coefficient of G_1 is 0.2406732985019372
```

Note that we can refine q by refining the interval $(0.20, 0.25)$. Also instead of linear sweep we can do a random search.

The maximum connected component can be found out using bfs repeatedly and maintaining the list of nodes each bfs pays visit to.

```
# G_1 is the original cora network
    for i, n in enumerate(G_1.nodelist):
        if n not in visited:
            comp = bfs_visit(G_1, n, visited)
            connected_component_dict[n] = [k for k in comp]
            comp_len = len(connected_component_dict[n])
            if comp_len > max_comp:
                max_comp = comp_len
                max_start = n
    print('G_4 size:', max_comp)
    connected_component = connected_component_dict[max_start]
```

Following is the code for the abstract method `bfs_visit(graph, entry_node, visited)`. Note that this method returns a list of node during bfs.

```
def bfs_visit(graph: Graph, s: int, visited: dict) -> set:
    connected = set()
    q: List[int] = list()
    q.append(s)
    while q:
        u = q.pop()
        visited[u] = True
        connected.add(u)
        for v in graph.nodes[u].neighbors:
            if v not in visited:
                q.append(v)
    return connected
```

Given D is the degree matrix and A is the adjacency matrix, we compute the Graph Laplacian and do the necessary steps.

```
    # Compute D^-1/2
    DH = np.nan_to_num(np.divide(D, np.sqrt(D)))

    # Compute Laplacian and Normalized Laplacian
    L = D - A
    L_sym = np.identity(num_subgraph_nodes, dtype=float) - np.matmul(np.matmul(DH, A), DH)

    # Compute Eigen values and Eigen vectors
    w, v = np.linalg.eig(L)
    wsym, vsym = np.linalg.eig(L_sym)

    # Sort the Eigen values
    xind = np.argsort(w)
    x_sym_ind = np.argsort(wsym)

    # Compute the Eigen vector corresponding to the second smallest Eigen value
    x = v[:, xind[1]]
    xsym = vsym[:, x_sym_ind[1]]

    # Sort the components of x (or D^-1/2 x_sym)
    x_ind = np.argsort(x)
    x_sym_ind = np.argsort(np.matmul(DH, xsym))

    # Perform Sweep cut on x
    k, val = sweep_cut(x_ind.flatten(), 'ncut')
    print('Cut found at k =', k)
    print('N-cut(s) = ', val)

    # Perform Sweep cut on x_sym
    k, val = sweep_cut(x_sym_ind.flatten(), 'ncut')
    print('Cut found at k =', k)
    print('N-cut(s‘) = ', val)
```

Details are needed how the sweep cut is performed.

```
# Reports min cut

def sweep_cut(nodes, metric):
    nodes: Sorted list of nodes
    for c in range(1,...,n-1):
        min_cut =  min(min_cut, calculate_cut_at_c)
    return min_cut

# Calculates cut based on chosen k
def calculate_cut(s: set, v_s: set, metric):
```

```
    for ed in G_4.edge_list:
        if {u, v} is across the cut:
            cut = cut + 1
    if metric == 'ncut':
        return n_cut(cut, s, v_s)
    elif metric == 'rcut':
        return r_cut(cut, s, v_s)
```

N-cut and Ratio-Cut is computed based on the formula given in the class.

Results

```
N-cut(s) =  0.014592068351562624
N-cut(s‘) =  0.18868274639250632
```

```
    k, val = sweep_cut(x_ind.flatten(), 'rcut')
    print('Cut found at k =', k)
    print('R-cut(s) = ', val)

    k, val = sweep_cut(x_sym_ind.flatten(), 'rcut')
    print('Cut found at k =', k)
    print('R-cut(s‘) = ', val)
```

The R-cut values are computed as follows:

```
def r_cut(cut, s, v_s):
    return float(cut / len(s)) + float(cut / len(v_s))
```

The `cut` variable is defined in the method `calculate_cut(s: set, v_s: set, metric)`.

Results

```
    R-cut(s) =  0.04207300555329811
    R-cut(s‘) =  0.7842617261247545
```

Information sheet
CS59000-MLG: Computation and Machine Learning on Graphs

Assignment Submission Fill in and include this information sheet with each of your assignments. This page should be the last page of your submission. Assignments are due at 11:59pm and are always due on a Thursday. All students please submit their homework via Brightspace. Students can typeset or scan their homework. Make sure that you answer each (sub-)question on a separate page. That is, one answer per page regardless of the answer length. Regarding the programing question, please answer them with your idea/pseudo codes. Students also need to upload their code together. Put all the code for a single question (not for each sub-question) into a single file, compress all the files and this PDF into one file (.zip) and upload the compressed file.

Late Homework Policy *Homework are due on Thursdays at 11:59pm PT and one late period expires on the following Monday at 11:59pm PT.* Only one late period may be used for an assignment. Any homework received after 11:59pm PT on the Monday following the homework due date will receive no credit. Once these late periods are exhausted, any assignments turned in late will receive no credit.

Honor Code I strongly encourage students to form study groups. Students may discuss and work on homework problems in groups. However, each student must write down their solutions independently, i.e., each student must understand the solution well enough in order to reconstruct it by him/herself. Students should clearly mention the names of all the other students who were part of their discussion group. Using code or solutions obtained from the web (GitHub/Google etc.) is considered an honor code violation. I will randomly check the submissions for plagiarism.

Your name: Soham Mukherjee

Email: mukher26@purdue.edu **SUID:** 29920150

Discussion Group: _____

I acknowledge and accept the Honor Code.

(Signed) SM