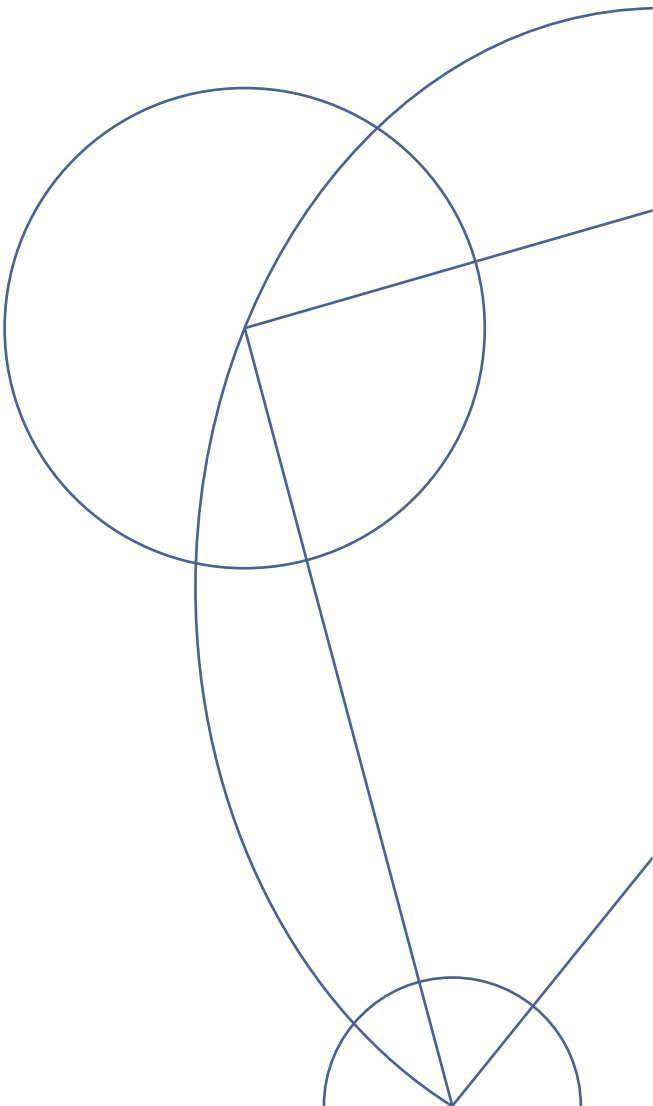
National Technical University of Athens
School of Electrical and Computer Engineering
Inter-Faculty Postgraduate Studies Programme
Data Science and Machine Learning

Second Set of Exercises

The second assignment written in partial fulfillment of the requirements for the completion of the Algorithmic Data Science core course of the Data Science & Machine Learning post-graduate studies programme.

Instructors
Prof. A. Pagourtzis
Dr. D. Souliou

Written by
[redacted]



June 23, 2022

Table of Contents

1 Hash Functions & Universality

In what follows, the congruence modulo n, i.e. $a = b \pmod{n}$, will be denoted as $a \equiv_n b$ for brevity. In addition, we follow the notation of [1], where $\mathbb{Z}_m := \{0, \dots, m-1\}$, i.e. $\mathbb{Z}_m = [m]$ using the instructors' notation.

(a) Consider the class $\mathscr{H} = \{h : U \to \mathbb{Z}_m\}$ of hash functions $h_{a,b}(x) = (ax+b) \bmod m$, where $U = \mathbb{Z}_{m^k}$ for $k \geq 2$ and $a \in \mathbb{Z}_m^*$, $b \in \mathbb{Z}_m$. The fact that this class of hash functions is not universal can be proven via a simple counter-example. Consider two numbers $x_1 = \kappa m$ and $x_2 = \lambda m$, where $\kappa, \lambda \in \mathbb{N}$ and $0 \leq \kappa < \lambda < m^{k-1}$ (for example, $\kappa = 1$ and $\lambda = 2$ are two such choices). Then, for any a, b, the following holds:

$$h_{a,b}(x_1) = (a\kappa m + b) \bmod m = 0 + b \bmod m = (a\lambda m + b) \bmod m = h_{a,b}(x_2) \tag{1.1}$$

Since $h_{a,b}(x_1) = h_{a,b}(x_2)$ holds for any choice of a and b, while $x_1 \neq x_2$, the fact that

$$\Pr_{h \in \mathscr{H}}\left[h_{a,b}(x_1) = h_{a,b}(x_2)\right] = 1 > \frac{1}{m} \tag{1.2}$$

proves that $\mathscr{H}$ is not universal.

(b) Suppose now that $h_{a,b}(x) = ((ax+b) \bmod p) \bmod m$, where p is a prime number with $p > m^k$ and $a \in \mathbb{Z}_p^*$, $b \in \mathbb{Z}_p$. In this case, $\mathscr{H}$ is indeed universal for $U = \mathbb{Z}_{m^k}$. To show this, first consider two numbers $x_1, x_2 \in U$ with $x_1 \neq x_2$ and then define

$$y_1 = (ax_1 + b) \bmod p \qquad \text{and} \qquad y_2 = (ax_2 + b) \bmod p. \tag{1.3}$$

Obviously, $y_1 - y_2 \equiv_p a(x_1 - x_2)$ holds. Nonetheless, $a \bmod p \neq 0$ and $(x_1 - x_2) \bmod p \neq 0$ also hold, since $p > a \neq 0$, $x_1 \neq x_2$ and $p > x_1, x_2$. Therefore, since p is a prime number, Theorem 31.6 of [1] indicates that $a(x_1 - x_2) \bmod p \neq 0$ holds as well. This in turn implies that $y_1 \neq y_2$.

Up to this point, we have proven that distinct choices for x_1 and x_2 with $x_1 \neq x_2$ map to distinct values of y_1 and y_2. In addition, for given values of y_1 and y_2, one may write

$$y_1 - y_2 \equiv_p a(x_1 - x_2) \quad \Rightarrow \quad a \equiv_p (y_1 - y_2) \cdot \left(\frac{1}{x_1 - x_2} \bmod p\right) \tag{1.4}$$

and

$$b \equiv_p y_2 - ax_2, \tag{1.5}$$

where the second factor of Eq. (1.4) denotes the unique multiplicative inverse of $x_1 - x_2$, modulo p. This indicates that there is a one-to-one correspondence between pairs of (a, b) and pairs of (y_1, y_2), since each of the $p(p-1)$ choices for the pair (a, b) yields a unique pair (y_1, y_2), with $y_1 \neq y_2$. The consequence of this is that for any given pair of (x_1, x_2), if a and b are chosen at random from $\mathbb{Z}_p^*$ and $\mathbb{Z}_p$, respectively, then the resulting pair of (y_1, y_2) is equally likely to be any pair of distinct values modulo p. This reduces the probability of a collision between the distinct keys x_1 and x_2 to the probability that $y_1 \equiv_m y_2$, when y_1 and y_2 are randomly chosen (since a and b are randomly chosen).

Taking everything into consideration, for a given value of y_1, the number of values y_2 such that $y_2 \neq y_1$ and $y_1 \equiv_m y_2$ is at most $\lceil p/m \rceil - 1$, where $\lceil \alpha \rceil$ denotes the ceiling of α. This is because by fixing the value of y_1, the value of y_2 can be $y_1 \pm m$, $y_1 \pm 2m$, ..., as long as $y_2 < p$. Now, since there are $(p-1)$ such choices for y_2, the probability of a collision satisfies

$$\Pr_{h\in\mathscr{H}}\left[h_{a,b}(x_1)=h_{a,b}(x_2)\right]=\Pr\left[y_1\neq y_2, y_1\equiv_m y_2\right]\leq\frac{\lceil p/m\rceil-1}{p-1}. \tag{1.6}$$

Taking into account the fact that

$$\left\lceil\frac{a}{b}\right\rceil\leq\frac{a+b-1}{b}, \tag{1.7}$$

as shown in Eq. (3.6) of [1], the probability of Eq. (1.6) is further bounded by

$$\Pr_{h\in\mathscr{H}}\left[h_{a,b}(x_1)=h_{a,b}(x_2)\right]\leq\frac{\frac{p+m-1}{m}-1}{p-1}=\frac{p-1}{m(p-1)}=\frac{1}{m}, \tag{1.8}$$

thus concluding the proof.

(c) The proof holds even in the case $U=\mathbb{Z}_p$, since this does not alter any of the steps shown in **(b)** in any way: $x_1<p$ and $x_2<p$ still hold and therefore $x_1-x_2<p$ holds as well, thus ensuring the fact that $(x_1-x_2)\mod p\neq 0$, as well as the existence of $(x_1-x_2)^{-1}\mod p$.

2 Open Addressing Hashing

(a) Consider a hash table T with a total of m slots and a key k, in the context of the open addressing hashing. The snippets shown below contain the pseudocode corresponding to the algorithms for the insertion (Pseudocode 1) and the search (Pseudocode 2) of the key.

```
1 i = 0
2 repeat
3     j = h(k,i)
4     if T[j] == NULL
5         T[j] = k
6         return j
7     else i = i + 1
8 until i == m
9 error "hash table overflow"
```

Pseudocode 1: HASH-INSERT(T,k)

```
1 i = 0
2 repeat
3     j = h(k,i)
4     if T[j] == k
5         return j
6     i = i + 1
7 until T[j] == NULL or i == m
8
9 return NIL
```

Pseudocode 2: HASH-SEARCH(T,k)

It becomes evident that the algorithm for the search of key k probes the same sequence of slots in T that the insertion algorithm examines when trying to insert the same key. Consequently, both procedures require the same number of steps and by extension the same average time.

(b) In order to prove the claim, we will first show that given an open-address hash table with load factor $\alpha = n/m < 1$ and assuming uniform hashing, the expected number of probes in an unsuccessful search is at most $(1-\alpha)^{-1}$. Let us denote the number of probes performed in an unsuccessful search by S and define an event E_i, for $i = 1, 2, \ldots$, as the event that an i-th probe occurs and corresponds to an occupied slot. Then, the probability that $S \geq v$ is equivalent to the probability of the intersection of events $E_1, E_2, \ldots, E_{v-1}$, i.e.

$$\begin{aligned}\Pr[S \geq v] &= \Pr[\mathrm{E}_1 \cap E_2 \cap \cdots \cap E_{v-1}] \\ &= \Pr[E_1] \cdot \Pr[E_2|E_1] \cdot \Pr[E_3|E_1 \cap E_2] \cdot \ldots \cdot \Pr[E_{v-1}|E_1 \cap E_2 \cap \cdots \cap E_{v-2}]\,. \end{aligned} \tag{2.1}$$

The hash table contains n elements and a total of m slots, therefore $\Pr[E_1] = n/m$. For $i > 1$, the probability that there is an i-th probe to an occupied slot, given that the first $i-1$ probes were performed on occupied slots, is equal to

$$\Pr[E_i|E_1 \cap \cdots \cap E_{i-1}] = \frac{n-i+1}{m-i+1}, \tag{2.2}$$

due to the uniformity assumption[1]. Combining these two equations yields

$$\Pr[S \geq v] = \frac{n}{m} \cdot \frac{n-1}{m-1} \cdot \ldots \cdot \frac{n-v+2}{m-v+2} \leq \underbrace{\frac{n}{m} \cdot \frac{n}{m} \cdot \ldots \cdot \frac{n}{m}}_{v-1 \text{ times}} = \left(\frac{n}{m}\right)^{v-1} = \alpha^{v-1}, \tag{2.3}$$

thus providing us with an upper bound for the studied probability. The expected number of probes is therefore the sum of probabilities for all values of v, which can also be upper bound by using Eq. (2.3) as follows:

[1] During the i-th probe, one of the remaining $n-(i-1)$ elements is found in one of the $m-(i-1)$ unexamined slots. Since we have assumed uniform hashing, the corresponding probability is the ratio of these quantities.

$$\mathbb{E}[S] = \sum_{v=1}^{\infty} \Pr[S \geq v] \leq \sum_{v=1}^{\infty} \alpha^{v-1} = \sum_{v=0}^{\infty} \alpha^{v} = \frac{1}{1-\alpha}, \tag{2.4}$$

since $\alpha < 1$, thus ensuring the convergence of the geometric series.

An important remark is that, since the insertion of a key into the hash table requires an unsuccessful search followed by placing the new key into the first available empty slot, this upper bound also corresponds to the maximum number of average probes required to insert an element into the table. Also, as explained in **(a)**, searching for a key k reproduces the same probe sequence which is produced during the insertion of the key. Combining these two observations, it becomes evident that if k was the $(v+1)$-st key inserted into the hash table, the expected number of probes made in a search for k is upper bound by the quantity

$$\frac{1}{1-\frac{v}{m}} = \frac{m}{m-v}, \tag{2.5}$$

as the load factor during the insertion of k is v/m. An upper bound for the expected number of probes, P, in a successful search is then given by averaging over all n keys of the hash table:

$$P = \frac{1}{n}\sum_{v=0}^{n-1}\frac{m}{m-v} = \frac{m}{n}\sum_{v=0}^{n-1}\frac{1}{m-v} = \frac{1}{\alpha}\sum_{x=m-n+1}^{m}\frac{1}{x} \leq \frac{1}{\alpha}\int_{m-n}^{m}\frac{1}{x}\,dx = \frac{1}{\alpha}\ln\frac{m}{m-n} = \frac{1}{\alpha}\ln\frac{1}{1-\alpha}, \tag{2.6}$$

thus concluding the proof.

3 The Girvan-Newman Algorithm

(a) First of all, it must be stressed that MMDS [2] gives a presentation of the Girvan-Newman algorithm that does not actually correspond to the algorithm presented by Girvan and Newman in their 2002 paper [3]. The original Girvan-Newman algorithm for a graph with $|V|$ vertices and $|E|$ edges is summarized in the following pseudocode snippet (Pseudocode 3).

```
num_edges = E
repeat
    between_scores = BETWEENNESS(graph(E,V))
    sorted = SORT(between_scores)
    graph(E,V) = REMOVE(graph(E,V), sorted[0].key)

    num_edges = num_edges - 1
until num_edges == 0
```

Pseudocode 3: The actual Girvan-Newman algorithm

In the above, `BETWEENNESS(G)` is a function that calculates the betweenness for the edges of a graph, $G(E, V)$, and stores the results in a key-value object, such as a dictionary. This object's values are then sorted, so that the `REMOVE(G,e)` function can remove from the graph the key e (i.e. the edge) that corresponds to the highest betweenness value. This process is repeated until all edges have been removed.

The Girvan-Newman algorithm presented in [2] is, instead, a version of Newman's algorithm for the calculation of edge betweenness centrality, as presented in his paper [4], i.e. an implementation of the `BETWEENNESS()` function. The corresponding algorithm is summarized in the following pseudocode snippet (Pseudocode 4).

```
num_nodes = V
node = 0
edge_scores[:] = 0
repeat
    BFS(node)

    SORT_LAYERS_TOP_DOWN()
    v = 0
    repeat
        if v != node
            paths_from_node[v] = PATHS(node,v)
        v = v + 1
    until v == num_nodes

    SORT_LAYERS_BOTTOM_UP()
    credits = DISTRIBUTE_CREDITS()
    edge_scores = edge_scores + FIND_SCORE(paths_from_node,credits)

    node = node + 1
until node == num_nodes

edge_scores = edge_scores/2
```

Pseudocode 4: The Girvan-Newman algorithm presented in MMDS

Starting from a given node, a BFS algorithm is performed using the `BFS()` function, in order to layer the graph into a DAG (Directed Acyclic Graph) using the selected node as the root. Then, following a top-down approach, the number of all shortest paths leading from the selected node towards all other nodes, v, is calculated using the `PATHS(node,v)` function and stored into an array. Switching to a

bottom-up approach, credits are distributed to each node using the `DISTRIBUTE_CREDITS()` function, which works as follows: every leaf of the graph gets 1 credit and then every other node gets 1 credit plus its children's credits. Finally, the edge betweenness scores are calculated using the `FIND_SCORE()` function, which distributes each node's credits along the edges connected to it, by taking into account the total number of paths from the source as weights. This process is repeated in the same manner for all other starting nodes, and the individual edge betweenness scores are summmed and divided by two, to avoid double counting.

As far as the time complexity of this algorithm is concerned, the execution of the BFS algorithm requires time $\mathcal{O}(|E| + |V|)$, and so do the steps where the paths are numbered and the scores are distributed. Since the smallest possible value of $|E|$ is $|V|-1$, while typically $|E|$ assumes considerably larger values, it suffices to say that the overall complexity is $\mathcal{O}(|E|)$. Finally, due to the fact that the procedure is repeated for all nodes, the total complexity for the Girvan-Newman algorithm as it is presented in [2] is $\mathcal{O}(|V| \cdot |E|)$.

When it comes to the actual Girvan-Newman algorithm, its time complexity will be $\mathcal{O}\left(|V| \cdot |E|^2\right)$, assuming that the `BETWEENNESS()` function is utilized by an algorithm with complexity $\mathcal{O}(|V| \cdot |E|)$. This is because the function is called $|E| - 1 = \mathcal{O}(|E|)$ times, one for each node removed.

(b) In the case where the input graph is a tree, the aforementioned procedures are greatly simplified. First of all, performing the first BFS algorithm is not necessary, since trees are by default DAGs. In addition, the calculation of the number of shortest paths per vertex from the root node is unnecessary, as the number is always equal to 1. This indicates that the calculation of the edge betweenness score for any given root node is reduced to the calculation of the number of nodes below each edge. Of course, the time complexity of this procedure remains $\mathcal{O}(|E| + |V|) = \mathcal{O}(|E|)$ and since it has to be performed $|E|$ times, the overall complexity of the Girvan-Newman algorithm as presented in [2] is found to be $\mathcal{O}\left(|V|^2\right)$, where we have used the fact that $\mathcal{O}(|V|) = \mathcal{O}(|E|)$, since $|E| = |V| - 1$ in the case of trees. By extension, the complexity of the actual Girvan-Newman algorithm, which corresponds to the sequential removal of the graph's edges based on their edge betweenness scores, will be $\mathcal{O}\left(|V|^3\right)$.

While it seems that the order of magnitude of the time complexity remains the same in the case of trees, the truth is that in practical applications there will be significant differences in the computation times between random graphs and trees. The concept of the time complexity takes into account the worst case scenario by default. In the case of random graphs this worst case scenario is not far from the average, however in trees the same does not hold. Most importantly, as previously mentioned, the number of edges in graphs is typically much larger than the corresponding number of vertices, so a complexity of $\mathcal{O}\left(|V|^3\right)$ is not in general equivalent to a complexity of $\mathcal{O}\left(|V| \cdot |E|^2\right)$, even though this holds in the case of trees.

4 Social-Network Graphs Exercises

The graph considered for the following two exercises is shown in Fig. 4.1.

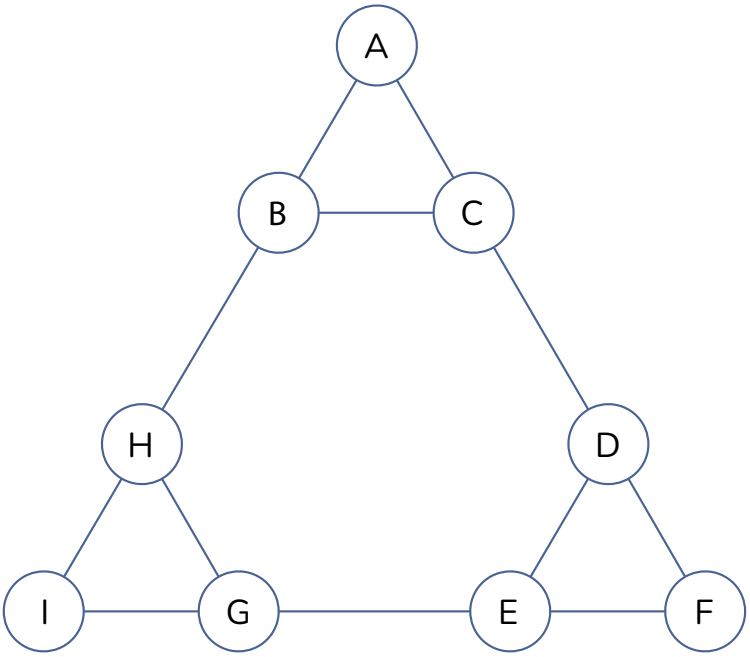


Figure 4.1: The studied graph.

4.1 Exercise 10.4.1, MMDS

(a) The adjacency matrix element A_{ij} is equal to 1 if there is a link between nodes i and j, otherwise it is equal to zero. That being said, the adjacency matrix corresponding to the graph of Fig. 4.1 is

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}, \tag{4.1}$$

where the order of the rows/columns corresponds to the order A,B,C,D,E,F,G,H,I.

(b) Following the same convention, the degree matrix is the diagonal matrix where the element D_{ii} corresponds to the degree of node i. In this case, we have

$$\mathbf{D} = \mathrm{diag}\,(2, 3, 3, 3, 3, 2, 3, 3, 2)\,. \tag{4.2}$$

(c) The Laplacian matrix is defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$, therefore in this case it is found equal to

$$\mathbf{L} = \begin{pmatrix} 2 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & 0 & 0 & 0 & 0 & -1 & 0 \\ -1 & -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 3 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 3 & -1 & -1 \\ 0 & -1 & 0 & 0 & 0 & 0 & -1 & 3 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 2 \end{pmatrix}, \tag{4.3}$$

4.2 Exercise 10.4.2, MMDS

In order to solve the eigenvectors-eigenvalues problem for the Laplacian matrix of Eq. (4.3), we develop the Python code shown in Snippet 1.

```
import numpy as np

L = np.array([[2,-1,-1,0,0,0,0,0,0],
              [-1,3,-1,0,0,0,0,-1,0],
              [-1,-1,3,-1,0,0,0,0,0],
              [0,0,-1,3,-1,-1,0,0,0],
              [0,0,0,-1,3,-1,-1,0,0],
              [0,0,0,-1,-1,2,0,0,0],
              [0,0,0,0,-1,0,3,-1,-1],
              [0,-1,0,0,0,0,-1,3,-1],
              [0,0,0,0,0,0,-1,-1,2]])

vals, vecs = np.linalg.eig(L)
```

Python Code Snippet 1: Eigenvectors-eigenvalues calculation for the Laplacian matrix.

The Laplacian matrix's eigenvalues are shown sorted in Table 4.1, along with their corresponding multiplicities.

Eigenvalue	0	$\frac{1}{2}\left(5-\sqrt{13}\right)$	3	$\frac{1}{2}\left(5+\sqrt{13}\right)$	5
Multiplicity	1	2	3	2	1

Table 4.1: The eigenvalues of the Laplacian matrix and their multiplicities.

The smallest nonzero eigenvalue is equal to $\lambda = \frac{1}{2}\left(5-\sqrt{13}\right)$ and the corresponding eigenvectors are (to second decimal accuracy):

$$\begin{aligned} v_1 &= (-0.59, -0.38, -0.38, 0.09, 0.29, 0.29, 0.29, 0.09, 0.29)^\top, \\ v_2 &= (0.14, -0.02, 0.21, 0.36, 0.19, 0.42, -0.34, -0.40, -0.56)^\top. \end{aligned} \tag{4.4}$$

Finally, since $\operatorname{sgn}(v_1) = (-,-,-,+,+,+,+,+,+)$, the partition proposed by v_1 is $\mathscr{C}_1 = \{\text{A,B,C}\}$ and $\mathscr{C}_2 = \{\text{D,E,F,G,H,I}\}$. Similarly, since $\operatorname{sgn}(v_2) = (+,-,+,+,+,+,-,-,-)$, the partition proposed by v_2 is $\mathscr{C}_1 = \{\text{A,C,D,E,F}\}$ and $\mathscr{C}_2 = \{\text{B,G,H,I}\}$.

5 Partitioning by Spectral Analysis Graph

The graph studied for the purposes of the present exercise is shown in Fig. 5.1.

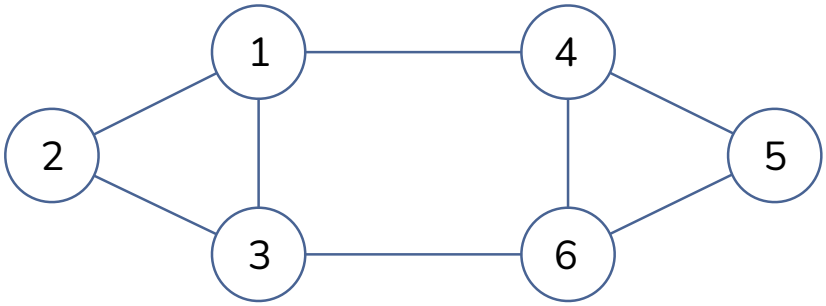


Figure 5.1: The studied graph.

(a) Assuming the cut $S = \{1, 2, 3\}$, $T = \{4, 5, 6\}$, then the normalized cut, NC, is simply

$$NC = \frac{\text{Cut}\,(S,T)}{\text{Vol}\,(S)} + \frac{\text{Cut}\,(S,T)}{\text{Vol}\,(T)} = \frac{2}{5} + \frac{2}{5} = \frac{4}{5}. \tag{5.1}$$

(b) Moving on with the same cut, i.e. $S = \{1, 2, 3\}$, $T = \{4, 5, 6\}$, we proceed with the calculation of the modularity, Q, defined as

$$Q = \frac{1}{2m} \sum_{i,j} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta\left(c_i, c_j\right). \tag{5.2}$$

In this expression, m is the graph's total number of edges, A_{ij} is an element of the graph's adjacency matrix, k_i is the degree of node i and $\delta\left(c_i, c_j\right) = 1$ if i and j belong in the same community, otherwise it is equal to 0. For our purposes, $m = 8$ and $A_{ij} = 1$ for all intra-community calculations, except for on-site interactions. This means that Eq. (5.2) may assume the form

$$Q = 2 \cdot \frac{1}{16} \left[2 \sum_{i<j} \left(1 - \frac{k_i k_j}{16} \right) - \frac{1}{16} \sum_i {k_i}^2 \right], \quad i, j = 1, 2, 3, \tag{5.3}$$

where the first factor of 2 is due to the symmetry between the $\{1, 2, 3\}$ and $\{4, 5, 6\}$ communities. Since $k_1 = k_3 = 3$ and $k_2 = 2$, the relevant calculations are

$$\begin{aligned} Q &= \frac{1}{8} \left[2 \cdot \left(1 - \frac{6}{16} + 1 - \frac{9}{16} + 1 - \frac{6}{16} \right) - \frac{1}{16} (9 + 4 + 9) \right] \\ &= \frac{1}{8} \left(2 \cdot \frac{27}{16} - \frac{22}{16} \right) = \frac{1}{8} \cdot \frac{32}{16} = \frac{1}{4}. \end{aligned} \tag{5.4}$$

Let us now consider the cut $\{1\}$, $\{2, 3\}$, $\{4, 5\}$, $\{6\}$. Again, there is a symmetry (computations-wise) between $\{1\}$, $\{2, 3\}$ and $\{6\}$, $\{4, 5\}$. As a result, Eq. (5.2) can now be reduced to

$$Q' = 2 \cdot \frac{1}{16} \left[2 \cdot \left(1 - \frac{k_2 k_3}{16} \right) - \frac{1}{16} \sum_i {k_i}^2 \right], \quad i, j = 1, 2, 3, \tag{5.5}$$

The relevant calculations yield

$$Q' = \frac{1}{8} \left[2 \cdot \frac{10}{16} - \frac{1}{16} (9 + 4 + 9) \right] = \frac{1}{8} \left[\frac{10}{8} - \frac{11}{8} \right] = -\frac{1}{64}. \tag{5.6}$$

6 Web Advertising Exercises

6.1 Exercise 8.2.1, MMDS

Consider the ski-buying problem, where renting the skis costs 10$ per time, while buying the relevant equipment costs 100$. The optimal off-line policy for the problem is buying the equipment from day 1 if we intend to use it for $100/10 = 10$ times or more, otherwise renting the skis every time, if we intend to stop skiing after $n \leq 10$ times. In other words, the optimal cost is

$$C_{\text{opt}} = \min\{10n, 100\}, \quad n \geq 1. \tag{6.1}$$

In our attempt to construct an on-line algorithm with the best possible competitive ratio, we first consider the case of never buying the equipment. In this case, the worst case scenario is taking up skiing and therefore spending 10$ every time we go skiing for the rest of our lives. This algorithm's worst case scenario indicates that the corresponding competitive ratio is unbounded. Therefore, it becomes evident that our on-line policy must include buying the equipment at some point.

Assume that we buy the equipment after renting it for the first $n-1$ times we go skiing. Then, the total cost will be

$$C = 10\,(n-1) + 100. \tag{6.2}$$

We will now split our investigation in two cases: buying the equipment after having rented it for 10 times or more and buying the equipment at some point before renting it for a 10th time. In both cases, the worst case scenario will be quitting ski the day after buying the equipment. In the first case, the domain of n is $n \geq 11$, therefore the competitive ratio, $R_1(n)$, of the algorithm is

$$R_1(n) = \frac{C}{C_{\text{opt}}} = \frac{10\,(n-1)+100}{100} = 0.9 + 0.1n, \quad n \geq 11. \tag{6.3}$$

Since $n \geq 11$, the competitive ratio satisfies $R_1(n) \geq 2$ with $R_{1(\min)} = 2$, corresponding to $n = 11$.

In the second case, the domain of n is $n \leq 10$. As a consequence, the competitive ratio, $R_2(n)$, of the algorithm is

$$R_2(n) = \frac{C}{C_{\text{opt}}} = \frac{10\,(n-1)+100}{10n} = 1 + \frac{9}{n}, \quad n \leq 10. \tag{6.4}$$

Since $n \leq 10$, the competitive ratio satisfies

$$1 \leq n \leq 10 \;\Leftrightarrow\; \frac{1}{10} \leq \frac{1}{n} \leq 1 \;\Leftrightarrow\; 0.9 \leq \frac{9}{n} \leq 9 \;\Leftrightarrow\; 1.9 \leq R_2(n) \leq 10. \tag{6.5}$$

This indicates that $R_{2(\min)} = 1.9$ and it is achieved for $n = 10$. Since this minimum is smaller than the minimum corresponding to the first case, we have proven that the on-line algorithm with the best possible competitive ratio is the following:

Rent the equipment for the first 9 times and then buy it.

The competitive cost for this algorithm is $R = 1.9$ and it is the smallest possible, as proven above.

6.2 Exercise 8.4.1, MMDS

(a) For a sequence of queries $xxyyzz$, the best case scenario is the one where an algorithm allocates the two x's to A, the two y's to B and the two z's to C, thus yielding a revenue of 6. We will now show that the worst case scenario is the one where A or B receive no queries and the remaining two advertisers (B and C or A and C, respectively) share the remaining four queries.

- Assume that A receives no queries. Then, the two x's must be allocated either both to B, or both to C, or one to each (otherwise, A would receive one or two x's). If both x's are allocated to B, then C can receive two y's, or two z's or one y and one z. If both x's are allocated to C, then B receives the two y's. If B and C each receive one x, then they each also receive one y, or B receives one y and C receives one z.

- Assume that B receives no queries. This implies that the two x's have been allocated to A and the two y's have been allocated to C, since any other configuration would imply that B can receive one or more x's or y's.

- We cannot assume that C receives no queries, since it is the only advertiser that can receive z's.

We have therefore proven that the worst case scenario corresponds to 4 out of the 6 queries being allocated to advertisers, implying that the greedy algorithm's competitive ratio is $2/3$ in this problem.

(b) Constructing such examples of sequences is easy, if we consider sequences of length 4. Obviously, the best case scenario is the one where all queries are allocated to advertisers. The worst case scenario must be such, that only the 2 queries are allocated to advertisers, since we want half the queries to be unassigned. The sequences that meet these criteria are sequences of the form $\bullet\bullet zz$. This is because advertiser C can receive any type of query, however it is the only advertiser that can receive z queries. As a result, the worst case scenario in such sequences will be the $\bullet\bullet$ queries being allocated to C, thus leaving the two z's unallocated. In fact, all such possible queries are $xxzz$, $yyzz$, $xyzz$, as well as all permutations thereof.

7 Recommendation Systems Exercises

7.1 Exercise 9.2.3, MMDS

Table 7.1 contains the computers' features discussed in the present exercise.

Feature	A	B	C
Processor Speed	3.06	2.68	2.92
Disk Size	500	320	640
Main-Memory Size	6	4	6

Table 7.1: Feature vectors for three computers A, B and C.

(a) The three given values are the only available ratings for the user, therefore the normalization is going to be performed by extracting the mean of these ratings:

$$\mu = \frac{4+2+5}{3} = \frac{11}{3} \tag{7.1.1}$$

The corresponding normalized rating for computer i is given by $\text{Norm}(i) = \text{Rating}(i) - \mu$. The results for all computers are shown in Table 7.2.

i	A	B	C
Norm(i)	$1/3$	$-5/3$	$4/3$

Table 7.2: Feature vectors for three computers A, B and C.

(b) The user profile $P(f)$ for each feature f is given by

$$P(f) = \sum_{i=A,B,C} f(i) \cdot \text{Norm}(i), \tag{7.1.2}$$

therefore we have:

$$\begin{aligned} P(\text{Processor Speed}) &= 3.06 \cdot \frac{1}{3} - 2.68 \cdot \frac{5}{3} + 2.92 \cdot \frac{4}{3} = \frac{67}{150} \\ P(\text{Disk Size}) &= 500 \cdot \frac{1}{3} - 320 \cdot \frac{5}{3} + 640 \cdot \frac{4}{3} = \frac{1460}{3} \\ P\left(\text{Main-Memory Size}\right) &= 6 \cdot \frac{1}{3} - 4 \cdot \frac{5}{3} + 6 \cdot \frac{4}{3} = \frac{10}{3} \end{aligned}$$

7.2 Exercise 9.3.2, MMDS

The original utility matrix used in this exercise is shown in Table 7.3.

	a	b	c	d	e	f	g	h
A	4	5		5	1		3	2
B		3	4	3	1	2	1	
C	2		1	3		4	5	3

Table 7.3: The original utility matrix.

(a) Following the suggested method, the utility matrix assumes the form shown in Table 7.4.

	a	b	c	d	e	f	g	h
A	1	1	0	1	0	0	1	0
B	0	1	1	1	0	0	0	0
C	0	0	0	1	0	1	1	1

Table 7.4: The reformed utility matrix.

Then, all pairwise Jaccard distances are evaluated as

$$d_J(i,j) = 1 - J(i,j) = 1 - \frac{|i \cup j|}{|i \cap j|}. \tag{7.2.1}$$

The results are shown in Table 7.5. Note that due to the $i \leftrightarrow j$ symmetry of the Jaccard distance, it suffices to show only the lower/upper triangular part of the table.

	a	b	c	d	e	f	g	h
a	0							
b	1/2	0						
c	1	1/2	0					
d	2/3	1/3	2/3	0				
e	1	1	1	1	0			
f	1	1	1	2/3	1	0		
g	1/2	2/3	1	1/3	1	1/2	0	
h	1	1	1	2/3	1	0	1/2	0

Table 7.5: Jaccard pairwise distances.

We are interested in a hierarchical clustering scheme, so the first step consists of assigning each element to its own cluster:

$$\mathscr{C}_1 = \{a\}, \quad \mathscr{C}_2 = \{b\}, \quad \mathscr{C}_3 = \{c\}, \quad \mathscr{C}_4 = \{d\}, \quad \mathscr{C}_5 = \{e\}, \quad \mathscr{C}_6 = \{f\}, \quad \mathscr{C}_7 = \{g\}, \quad \mathscr{C}_8 = \{h\}$$

The next step will consist of merging $\mathscr{C}_6$ and $\mathscr{C}_8$ into a single cluster, since the distance between f and h is zero. We therefore have:

$$\mathscr{C}_1 = \{a\}, \quad \mathscr{C}_2 = \{b\}, \quad \mathscr{C}_3 = \{c\}, \quad \mathscr{C}_4 = \{d\}, \quad \mathscr{C}_5 = \{e\}, \quad \mathscr{C}_6 = \{f, h\}, \quad \mathscr{C}_7 = \{g\}$$

For the next step, the minimum distance is 1/3 and it corresponds to the distance between b and d, as well as the distance between d and g. Since the distance between an element and a cluster is defined as the minimum distance between the element and any of the cluster's elements, all b, d and g will end up clustered together, so the order does not matter. We can therefore say that by the end of this step the clusters are:

$$\mathscr{C}_1 = \{a\}, \quad \mathscr{C}_2 = \{b, d, g\}, \quad \mathscr{C}_3 = \{c\}, \quad \mathscr{C}_5 = \{e\}, \quad \mathscr{C}_6 = \{f, h\}$$

For the final step, we consider clustering together clusters with a distance of 1/2, which can be done in many ways. For example, we can merge $\mathscr{C}_1$ and $\mathscr{C}_2$, since the distance between a and b is 1/2, or we can merge $\mathscr{C}_2$ with $\mathscr{C}_6$, since the distance between g and f is also 1/2. We decide to break this tie randomly and cluster together $\mathscr{C}_1$ and $\mathscr{C}_2$ thus ending up with the following four clusters:

$$\mathscr{C}_1 = \{a, b, d, g\}, \quad \mathscr{C}_3 = \{c\}, \quad \mathscr{C}_5 = \{e\}, \quad \mathscr{C}_6 = \{f, h\}$$

(b) In order to compute the average cluster score per user, we only average over the nonblank entries for each user. Therefore, for example, the average score of $\mathscr{C}_1$ for user A will be $(4 + 5 + 5 + 3)/4 = 17/4$, while the average score of $\mathscr{C}_1$ for user B will be $(3 + 3 + 1)/3 = 7/3$, since the entry of a is blank for user B (it is not assumed zero). The results for all clusters per user are concentrated in Table 7.6.

	$\mathscr{C}_1$	$\mathscr{C}_3$	$\mathscr{C}_5$	$\mathscr{C}_6$
A	17/4	0	1	2
B	7/3	4	1	2
C	10/3	1	0	7/2

Table 7.6: User scores per cluster.

(c) Given the results of Table 7.6, the vector norms for each user are

$$\|A\| = \frac{\sqrt{369}}{4}, \quad \|B\| = \frac{\sqrt{238}}{3}, \quad \|C\| = \frac{\sqrt{877}}{6}.$$

As a result, the cosine similarity scores are

$$\cos(A, B) = \frac{\frac{17}{4} \cdot \frac{7}{3} + 0 \cdot 4 + 1 \cdot 1 + 2 \cdot 2}{\frac{\sqrt{369}}{4} \cdot \frac{\sqrt{238}}{3}} = \frac{179}{\sqrt{87822}} \simeq 0.604$$

$$\cos(A, C) = \frac{\frac{17}{4} \cdot \frac{10}{3} + 0 \cdot 1 + 1 \cdot 0 + 2 \cdot \frac{7}{2}}{\frac{\sqrt{369}}{4} \cdot \frac{\sqrt{877}}{6}} = \frac{508}{\sqrt{323613}} \simeq 0.893$$

$$\cos(B, C) = \frac{\frac{7}{3} \cdot \frac{10}{3} + 4 \cdot 1 + 1 \cdot 0 + 2 \cdot \frac{7}{2}}{\frac{\sqrt{238}}{3} \cdot \frac{\sqrt{877}}{6}} = \frac{338}{\sqrt{208726}} \simeq 0.740$$

and the corresponding cosine distances are

$$d(A,B) \simeq \cos^{-1}(0.604) \simeq 52.84^\circ$$
$$d(A,C) \simeq \cos^{-1}(0.893) \simeq 26.75^\circ$$
$$d(B,C) \simeq \cos^{-1}(0.740) \simeq 42.27^\circ$$

References

[1] T. H. Cormen, C. E. Leiserson, E. L. Rivest, and C. Stein, *Introduction to Algorithms*. The MIT Press, 2009.

[2] J. Leskovec, A. Rajaraman, and J. Ullman, *Mining of Massive Datasets*. Cambridge University Press, 2019.

[3] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks," *Proceedings of the National Academy of Sciences*, vol. 99, no. 12, pp. 7821–7826, 2002. DOI: 10.1073/pnas.122653799.

[4] M. E. J. Newman, "Scientific collaboration networks. ii. shortest paths, weighted networks, and centrality," *Phys. Rev. E*, vol. 64, p. 016 132, 1 2001. DOI: 10.1103/PhysRevE.64.016132.