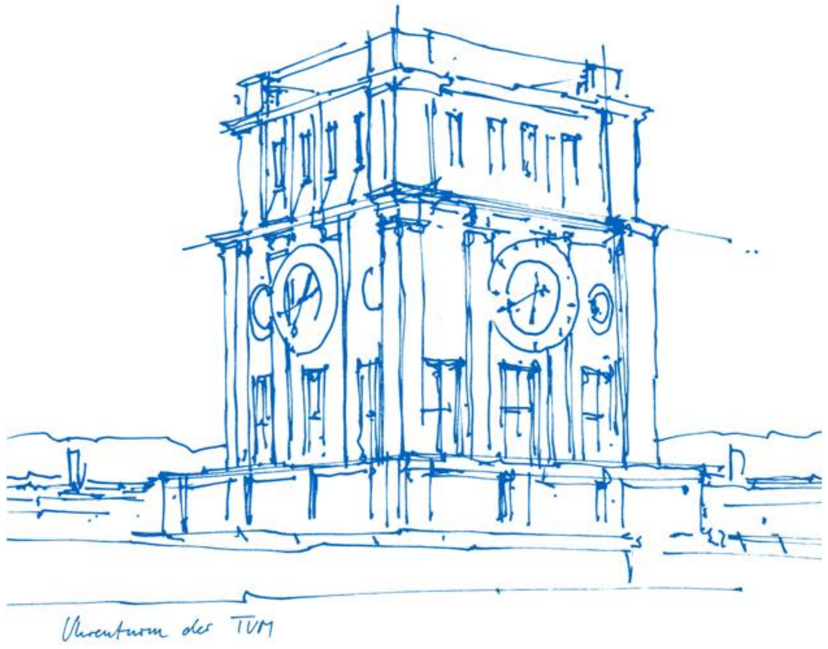
Exercises for Social Gaming and Social Computing (IN2241 + IN0040) – Introduction to Exercise 5



Exercise Content

Sheet Number	Exercise	Working Time
1	• Introduction to Python: basic Python programming language exercises • Graph Drawing using igraph	Monday, May 27 - Monday, June 3, 24:00
2	• Centrality measures	Monday, June 3 - Monday, June 17, 24:00
3	• Recommender Systems as an example for systems using simple forms of social context: Collaborative Filtering	Monday, June 17 - Monday, June 24, 24:00
4	• Inferring social tie strength from activity data in social networking platforms with linear regression	Monday, June 24- Monday, July 01, 24:00
5	• finding social groups with clustering methods on profile date (K-Means) and graph clustering (Girvan Newman method)	Monday, July 01 - Monday, July 08, 24:00
6	• analyzing short-term social context using mobile interaction data (Reality Mining)	Monday, July 08 - Monday, July 15, 24:00

Problem 5.1: Finding Groups in Social Networks with K-Means

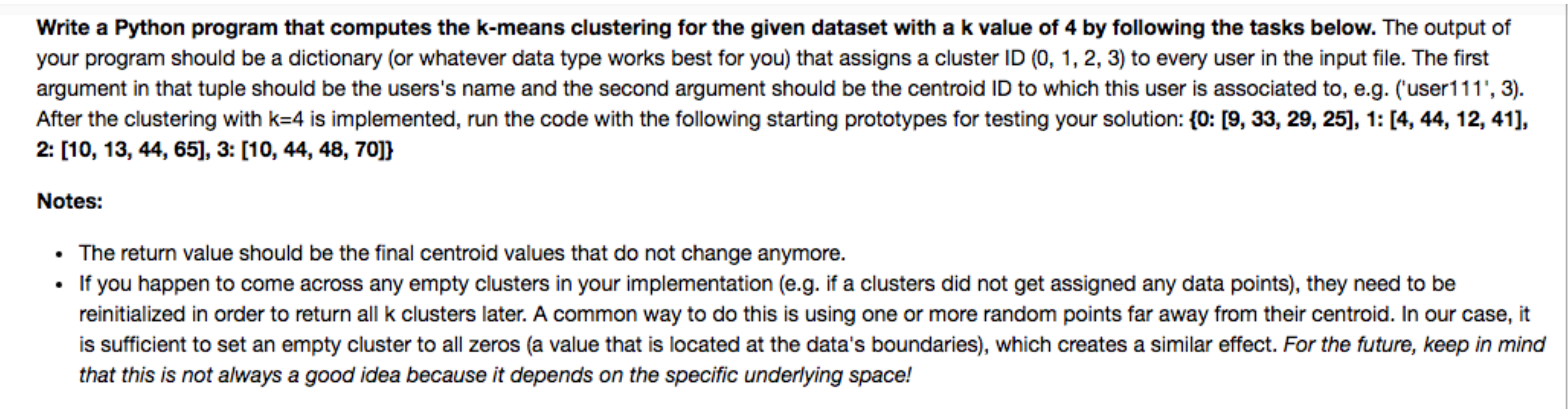
- data: *NetworkActivityData.csv* : for each user:
  - Number of posts
  - Number of comments
  - Number of likes (on posts and comments)
  - Number of friends

- K-Means: objective function (see lecture):

$$J(\mu) = \sum_{k=1}^{K} \sum_{\{n|x_n \in C_k\}} ||x_n - \mu_k||^2$$

where $\mu_k \in \mathbb{R}^d$ and $x_n \in \mathbb{R}^d$

Problem 5.1: Finding Groups in Social Networks with K-Means



Write a Python program that computes the k-means clustering for the given dataset with a k value of 4 by following the tasks below. The output of your program should be a dictionary (or whatever data type works best for you) that assigns a cluster ID (0, 1, 2, 3) to every user in the input file. The first argument in that tuple should be the users's name and the second argument should be the centroid ID to which this user is associated to, e.g. ('user111', 3). After the clustering with k=4 is implemented, run the code with the following starting prototypes for testing your solution: **{0: [9, 33, 29, 25], 1: [4, 44, 12, 41], 2: [10, 13, 44, 65], 3: [10, 44, 48, 70]}**

Notes:

- The return value should be the final centroid values that do not change anymore.
- If you happen to come across any empty clusters in your implementation (e.g. if a clusters did not get assigned any data points), they need to be reinitialized in order to return all k clusters later. A common way to do this is using one or more random points far away from their centroid. In our case, it is sufficient to set an empty cluster to all zeros (a value that is located at the data's boundaries), which creates a similar effect. *For the future, keep in mind that this is not always a good idea because it depends on the specific underlying space!*

Problem 5.1: Finding Groups in Social Networks with K-Means

Task 1: Reading The Data

The first thing you are supposed to do is reading in the data set (*NetworkActivityData.csv*). As mentioned above, the file's rows contain a social network user with their activity types (posts, comments, likes and friends). One possibility is to save the data as a dictionary, the user ID as a key and the list of activity types as its values.

Implement a function that reads in a .csv file given a certain path so you can call it later on the data set.

Task 2: Starting Centroids

In order to use k-means clustering, we need k starting centroids to begin with. There are different ways to do it, one of them is randomly assigning the initial values. For each dimension of the centroid, the random values should fall between the dimension's maximum and minimum value of all the points in the data set.
For example in our case, if the dimension "number of comments" has a maximum value of 49 and a minimum value of 0, the value assigned to the third component of one centroid should be randomly assigned between 0 and 49. The same for the other features and centroids, of course.

Implement a function that computes k random starting centroids, bounded within the minimum and maximum dimension of the input features.

Problem 5.1: Finding Groups in Social Networks with K-Means

Task 3: Updating The Clusters

The core functionality of the k-means algorithm is to assign each observation (user and their properties) to its new cluster and afterwards to recompute the centroid average. These two steps are repeated until the centroids do not change anymore, meaning that the clusters are finally set.

Implement two functions, one that **updates the cluster affiliation for each observation by determining the closest centroid** and **another to update the centroid values by averaging their data points**, but only for one iteration in both cases.

Task 4: k-Means Loop

As a last step, k-means uses the functionality that was implemented in the tasks beforehand in a loop and terminates after a few iterations when the centroid values do not change anymore.

Implement k-means clustering by iteratively using the functions defined above until the centroids are set. Afterwards, run your implementation with k=4 using the given starting centroids {0: [9, 33, 29, 25], 1: [4, 44, 12, 41], 2: [10, 13, 44, 65], 3: [10, 44, 48, 70]} instead of the random initialization.

Hint: You can verfiy your solution's correctness by comparing your final cluster observations to the pre-defined function's output:

```
from sklearn.cluster import KMeans
from numpy import genfromtxt

observations = genfromtxt('NetworkActivityData.csv', delimiter=',')[:, 1 : 5]
centers = np.asarray([[9, 33, 29, 25], [4, 44, 12, 41], [10, 13, 44, 65], [10, 44, 48, 70]])
kmeans = KMeans(n_clusters=4, init=centers).fit(observations)
result = kmeans.predict(observations)
print(sum(result == 0), sum(result == 1), sum(result == 2), sum(result == 3))
```

Output the observation counts in each cluster. Finally, have a look at the results and describe the common properties of the four groups. What information do we have on activity patterns for different groups and for different features? How much did they contribute in the social network? Don't write more than 5 sentences.

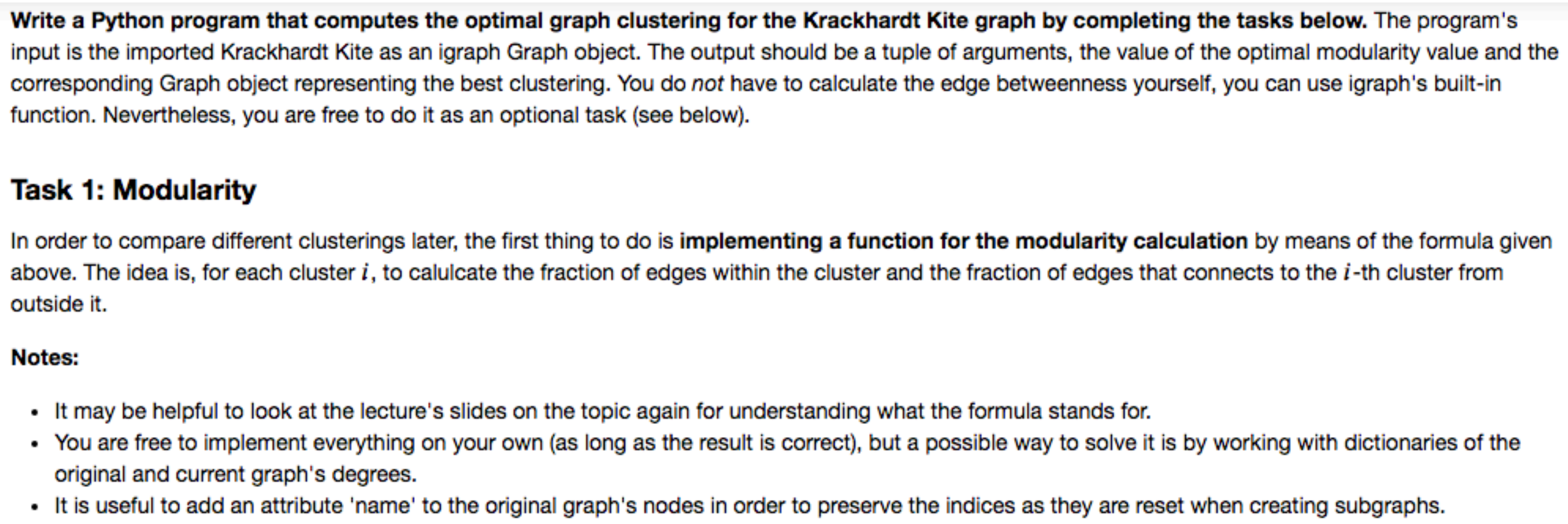
Problem 5.2: Girvan-Newman Clustering

- Cluster connectivity matrix E
  → Modularity $Q = \sum_i e_{ii} - a_i^2$
  where $a_i = \sum_j e_{ij}$

- Algorithm:
  - Calculate the edge betweenness for all edges in the graph
  - Remove the edge with highest betweenness
  - Calculate (and update) the modularity for the resulting clustering
  - Repeat steps 1 to 3 until there are no edges left

- Final clustering:
  calculate Modularity for all levels of the dendrogram, use level where Modularity is largest

Problem 5.2: Girvan-Newman Clustering



Write a Python program that computes the optimal graph clustering for the Krackhardt Kite graph by completing the tasks below. The program's input is the imported Krackhardt Kite as an igraph Graph object. The output should be a tuple of arguments, the value of the optimal modularity value and the corresponding Graph object representing the best clustering. You do *not* have to calculate the edge betweenness yourself, you can use igraph's built-in function. Nevertheless, you are free to do it as an optional task (see below).

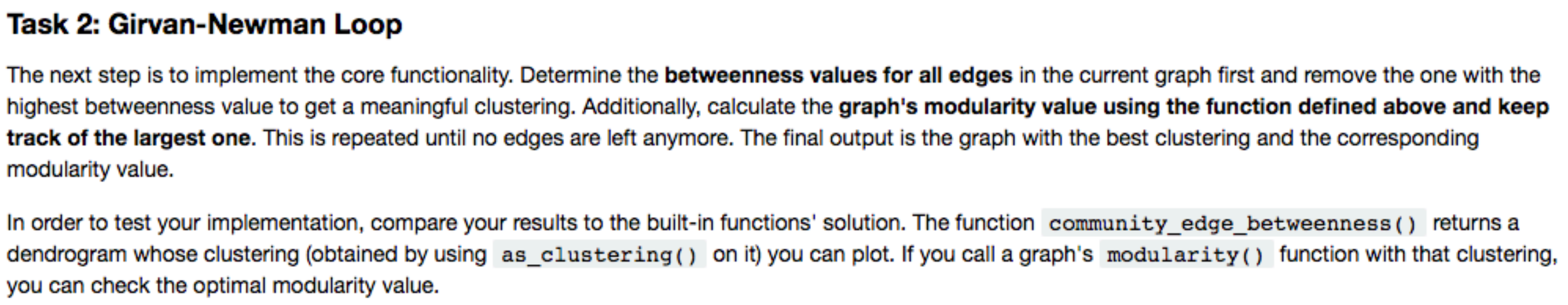
Task 1: Modularity

In order to compare different clusterings later, the first thing to do is **implementing a function for the modularity calculation** by means of the formula given above. The idea is, for each cluster i, to calulcate the fraction of edges within the cluster and the fraction of edges that connects to the i-th cluster from outside it.

Notes:

- It may be helpful to look at the lecture's slides on the topic again for understanding what the formula stands for.
- You are free to implement everything on your own (as long as the result is correct), but a possible way to solve it is by working with dictionaries of the original and current graph's degrees.
- It is useful to add an attribute 'name' to the original graph's nodes in order to preserve the indices as they are reset when creating subgraphs.

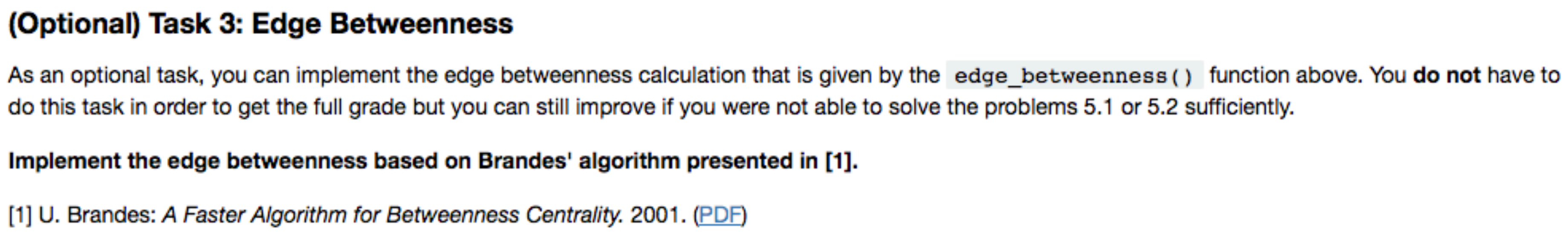
Problem 5.2: Girvan-Newman Clustering



Task 2: Girvan-Newman Loop

The next step is to implement the core functionality. Determine the **betweenness values for all edges** in the current graph first and remove the one with the highest betweenness value to get a meaningful clustering. Additionally, calculate the **graph's modularity value using the function defined above and keep track of the largest one**. This is repeated until no edges are left anymore. The final output is the graph with the best clustering and the corresponding modularity value.

In order to test your implementation, compare your results to the built-in functions' solution. The function `community_edge_betweenness()` returns a dendrogram whose clustering (obtained by using `as_clustering()` on it) you can plot. If you call a graph's `modularity()` function with that clustering, you can check the optimal modularity value.



(Optional) Task 3: Edge Betweenness

As an optional task, you can implement the edge betweenness calculation that is given by the `edge_betweenness()` function above. You **do not** have to do this task in order to get the full grade but you can still improve if you were not able to solve the problems 5.1 or 5.2 sufficiently.

Implement the edge betweenness based on Brandes' algorithm presented in [1].

[1] U. Brandes: *A Faster Algorithm for Betweenness Centrality*. 2001. (PDF)

Submitting Your Solution

- work by expanding the .ipynb iPython notebook for the exercise that you downloaded from Moodle.
- save your expanded .ipynb iPython notebook in your working directory. Submit your .ipynb iPython notebook via Moodle (nothing else)
- remember: working in groups is not permitted. Each student must submit her own ipynb notebook!
- we check for plagiarism. Each detected case will have the consequence of 5.0 for the whole exercise grade.
- deadline: please check slide nr 2 (this slide-set)

Citations

1. Newman, M., Girvan, M.: Finding and evaluating community structure in networks, arXiv:cond-mat/0308217v1, 2003

2. Ulrik Brandes: *A Faster Algorithm for Betweenness Centrality.* 2001. (PDF)