Exercise set #8 (17 pts)

- The deadline for handing in your solutions is November 19th 2018 23:55.
- Return your solutions (one `.pdf` file and one `.zip` file containing Python code) in MyCourses (Assignments tab). Additionally, submit your pdf file also to the Turnitin plagiarism checker in MyCourses.
- Check also the course practicalities page in MyCourses for more details on writing your report.

1. Community detection overview (9 pts)

In this exercise, we will get some hands-on experience on community detection. You will be studying two networks, the network of dolphins `dolphins.edg` and an artificial graph `lfr100.edg` with a built-in community structure `lfr100.cmtys`.

The actual community detections are performed using a online community detection service Jako CD. As a first step, open your favorite browser and find your way to `http://jako.complex.cs.aalto.fi`.

Part 1: Detecting communities in the dolphin network

a) (1 pt) Upload first the dolphin network to Jako, and **visualize** the bare network (In Jako CD, you can visualize networks by applying the `NullCD` community on the network.) **Comment** the visualization based on your personal impression: are there any well defined communities in this network? How would you partition the network into different communities using only your own intuition?

Let's next find out whether some algorithms are able to discover similar communities as you identified. Do this by applying four different community detection methods on the graph:

1. **Infomap (Edler and Rosvall)**:

   Infomap optimizes the average description length of a path taken by a random walker [1].

   Use the following options:

   `directed=False, hierarchical=False, trials=10, full_lower_levels=False, include_self_links=False`

2. **Girvan-Newman**:

   An old community detection method, which obtains communities by removing links with highest betweenness centrality values [2]. This procedure produces a hierarchical division of nodes to communities, and the "best" level in this hierarchy is selected as the one that maximises modularity.

   No options to be specified.

3. **Louvain (original code)**:

   A more recent modularity optimization method, which uses label propagation heuristic to find a (local) maximum for the modularity function [3].

   Use the following options:

   `trials=10, weighted=False, stop_dQ=None, which_partition=-1`

   Note: The Louvain method returns a hierarchical set of layers. In Jako, all of these are returned. Use the one with the least number of communities – this one has the highest modularity. Or, in case only one layer is returned, then it is the layer of maximal modularity (Jako is still in development phase, and things may change slightly over time.)

4. **Stochastic Block Model**:

   Fits a stochastic block model to the data. The block structure in the model that best describes the data is used as "communities" [4, 5].

   Use the following options:

   `deg_corr=True, nested=True, directed=False, weighted=False, B_min=None, B_max=None,`

Feel free to also try out some other community detection methods found in the Jako CD; but notice that some of the methods may *e.g.* yield overlapping communities that can not be visualized within the service (yet).

b) (1 pt) Use the built-in visualization tool in `Jako CD` to **visualize** the community structures obtained using the above methods.

c) (1 pt) Do the different methods give out same results? Do the results correspond to your intuition of what the modules should be as you described before?

Part 2: Comparing communities and benchmarking

Now, let's put the algorithms to a test! One approach for ranking the algorithms based on their performance is to use artificially generated networks where the underlying community structure is known.

Examples of such graphs are the Lancichinetti-Fortunato-Radicchi (LFR) graphs [6], see also Fig. 1. In LFR graphs the community sizes and degree distributions are both approximate power-laws. Further, 1-μ fraction of each node's links connect to nodes that belong to the same community, and fraction μ connects nodes between communities. The parameter μ plays thus the role of a mixing parameter: with μ values close to zero the communities are just isolated components and with large values of μ no clear community structure is present.

The graph `lfr100.edg` (consisting of 100 nodes) has mean degree ≈ 10 and a mixing parameter value of $\mu = 0.5$. In practice, this network corresponds to a case when the communities are still detectable, but the task of finding them is not trivial for all algorithms.

d) (1 pt) Apply the first three algorithms of Part 1 to the LFR network in Jako CD and **visualize** the community structures. **Comment** on how easy it is to assess the performance of the algorithms visually.

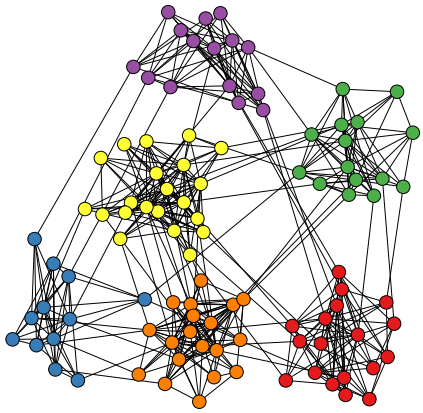

Figure 1: An example of a LFR-graph, with clear network modules. Note that this is not the network used in the exercise!

Next, the goal is to compute the variation of information (VI) between each of the computed network partitions in Jako CD and the ground-truth network partition given in file `lfr100.cmtys`. VI is a distance between network partitions: two partitions are more similar when VI is close to zero, and different when VI is large.

$VI(\mathcal{P}, \mathcal{P}')$ between two partitions $\mathcal{P}, \mathcal{P}'$ measures the information content of the partitions which is not shared by them:

$$VI(\mathcal{P}, \mathcal{P}') = H(\mathcal{P}) + H(\mathcal{P}') - 2I(\mathcal{P}, \mathcal{P}') \in [0, \log_2 N], \tag{1}$$

where N equals the total number of nodes in the network. If we denote the size of a community C by $|C|$, we can write the entropy of a partition (i.e., the expected amount of information that is needed to transmit the community identity of a randomly selected node) as

$$H(\mathcal{P}) = -\sum_{C_i \in P} P(C_i) \log_2(P(C_i)) = -\sum_{C_i \in P} \frac{|C_i|}{N} \log_2 \left(\frac{|C_i|}{N} \right) \tag{2}$$

and mutual information (describing the amount of information shared by the two partitions) as

$$I(\mathcal{P}, \mathcal{P}) = \sum_{C_i \in P} \sum_{C'_j \in P'} P(C_i, C'_j) \log_2 \left(\frac{P(C_i, C'_j)}{P(C_i) P(C'_j)} \right) \tag{3}$$

$$= \sum_{C_i \in P} \sum_{C'_j \in P'} \frac{|C_i \cap C'_j|}{N} \log_2 \left(\frac{|C_i \cap C'_j| \times N}{|C_i||C'_j|} \right) \tag{4}$$

In the above definition of mutual information, all terms of the form $0 \log_2 0$ are evaluated as 0.

e) (2 pts) To get familiar with VI, let's first consider two partitions of 12 nodes:
A = {1: [1,2,3,4,5,6,7,8,9], 2: [10,11,12]}

and
B = {1: [1,2,3,4,5,6,7], 2: [8,9,10], 3: [11,12]}.
Calculate VI between these partitions manually (pen and paper).

f) (1 pt) Next, **write** a Python function for calculating VI between two partitions. Use the functions for calculating entropies and mutual information in the code template (`com_det_benchmarking.py`) available in MyCourses. Before applying your code on the actual community structures we are interested in, **test** your function by making sure that your code for the VI produces the following results:

i) VI between same partitions equals to zero

ii) VI between a partition in which all nodes belong to the same and a partition in which all nodes belong to different communities equals $\log_2 N$.

iii) VI between the partitions in e) equals the result you calculated manually.

Report your testing results!

g) (2 pts) **Compute** the pairwise distances between all the network partitions (Infomap, Louvain, Girvan-Newman, Stochastic block model, ground-truth). **Comment** on the results you have obtained. Which method performs best in terms of the VI measure?

Hint: From Jako CD, you can download the communities in a variety of formats – you may use any format you wish for the computation of the VI measure. There are also many ways to represent network partitions. In this exercise, it may be easiest to take an approach where the network partitions are represented as dictionaries: each key gives the label of a community and the corresponding value is a list of nodes in the community.

2. Modularity (8 pts, pen and paper)

When nodes of a network are partitioned into communities (or clusters), *modularity* can be used to measure how well the given partition catches network's community structure. The modularity Q of a partition into communities can be written as

$$Q = \sum_{c \in \mathcal{P}} \left[\frac{l_c}{L} - \left(\frac{d_c}{2L} \right)^2 \right], \tag{5}$$

where the sum runs over all clusters/modules/communities, L is the number of links in the whole network, l_c is the number of links internal to cluster c (internal = both endpoint nodes are in c), and d_c is the total degree of nodes in cluster c (sum of the degrees of nodes in the cluster, such that the degrees count all links attached to the nodes irrespective of if they go inside or outside of the community). In other words, modularity is defined as the difference between the relative number of links inside the communities and the expected relative number of links inside the communities in a randomized network (usually configuration model) without clear community structure.

a) (2 pts) **Calculate** the value of modularity for the two partitions shown in Fig. 2 (in the first, there are two clusters, and in the second, the whole network is taken as a single cluster).

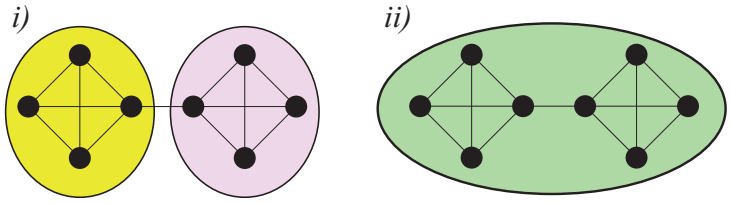

Figure 2: Two module configurations for a)

In the rest of this exeercise, we will show that the modularity Q defined in the above Eq. (5) contains an intrinsic scale. This means that the size of communities that are favored by modularity optimization depends on the number of links L in the network. Therefore, sometimes subgraphs which should by common sense be labeled as separate modules, such as full cliques attached to the rest of the network with a single link, get merged when Q is maximized.

Consider now a very general case – a network where there are altogether L links. Suppose that we have somehow *a priori* identified two groups of nodes a and b which we consider as modules (say, because they are cliques). Let us also assume that there is a single link connecting these two modules[1]. Instead, we make no assumptions about the number of links that connect a and b to the rest of the network. For a schematic illustration, see Figure 3.

Let's now consider two alternative ways of parcellating this network into communities: either 1) consider the two real modules as two modules, or 2) merge them into a single module ab. In community detection based on modularity optimization, we select the partition that yields higher modularity. So, the idea now is to calculate the difference in modularity $\Delta Q = Q_2 - Q_1$ between these two partitions using the formulation of Eq. 5 for modularity. When this difference is positive modularity optimization will merge the two "physical" modules.

b) (2 pts) **Write** ΔQ as a function of L, d_a, d_b, d_{ab}, l_a, l_b, and l_{ab}.
Hint: Note that the part of the modularity that comes from any other community than a, b or their merger ab is cancelled out when Q_1 is substracted from Q_2. This can be seen easily by writing in both formulas Q_1 and Q_2 the part of the sum that deals with communities in the rest of the network as $Q_R = \sum_{c\in\mathcal{P}'}\left[\frac{l_c}{L} - \left(\frac{d_c}{2L}\right)^2\right]$, where $\mathcal{P}'$ is the set of communties in the rest of the network.

c) (2 pts) **Write** d_{ab} as the function of d_a and d_b, and l_{ab} as the function of l_a and l_b. Using these substitutions **show** that the option of merging two clusters becomes favored by modularity optimization when $L > \frac{d_a d_b}{2}$. **Explain** why this result indicates that modularity optimization must have a *resolution limit*, or a minimum size of community that can be found that depends on the size of the network (where size is the number of links). You may assume that the network is sparse so d_a, d_b, and d_{ab} are constants when L increases.

d) (2 pts) Next, **provide an argument** that the resolution limit of modularity has a form such that in a network with L links it is not possible to find communities of size smaller than $n \propto \sqrt{L}$. **Use** the following steps in your argument:

i) Assume that community a is a clique of n nodes and that there is one single link

[1]Similar, although slightly more general, calculations could be made by assuming that there are $l_{a\leftrightarrow b}$ links connecting the two communities without much difference in the results. Here we, however, concentrate on the limiting case of only a single connecting link.

connecting a to the rest of the network. Write the formula for d_a as function of n. To make things slightly simpler, you can approximate $d_a \approx n^2$ for the rest of this exercise.

ii) Using the results of c) show that the community a is always merged to some other community when $n < C\sqrt{L}$, where C is some constant that doesn't depend on L.

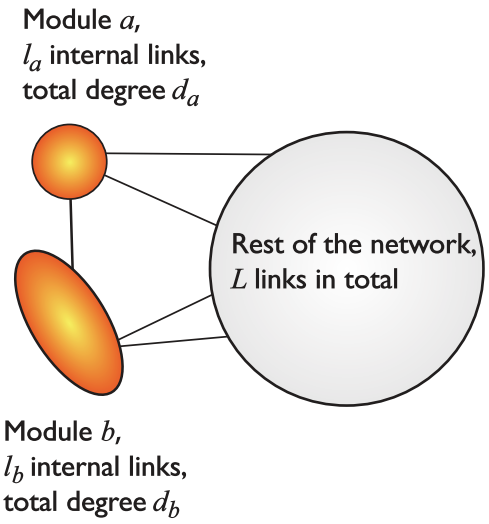

Figure 3: Schematic figure of two modules a and b, connected with a single link. Note that in b) and c) we do not make any assumptions of how many links are connecting the two modules to the rest of the network.

Feedback (1 pt)

To earn one bonus point, give feedback on this exercise set and the corresponding lecture latest two day after the report's submission deadline.
Link to the feedback form: `nourlyet`.

References

[1] M. Rosvall and C. T. Bergstrom, "Maps of random walks on complex networks reveal community structure," *Proceedings of the National Academy of Sciences*, vol. 105, no. 4, pp. 1118–1123, 2008.

[2] M. Girvan and M. E. Newman, "Community structure in social and biological networks," *Proceedings of the national academy of sciences*, vol. 99, no. 12, pp. 7821–7826, 2002.

[3] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. P10008, 2008.

[4] B. Karrer and M. E. J. Newman, "Stochastic blockmodels and community structure in networks," *Phys. Rev. E*, vol. 83, p. 016107, 2011.

[5] T. P. Peixoto, "Hierarchical block structures and high-resolution model selection in large networks," *Phys. Rev. X*, vol. 4, p. 011047, 2014.

[6] A. Lancichinetti, S. Fortunato, and F. Radicchi, "Benchmark graphs for testing community detection algorithms," *Physical Review E*, vol. 78, no. 4, p. 046110, 2008.