Lab Class 6 Exercise Sheet

Enrico Caprioglio

March 2025

Introduction: In general, community detection algorithms can be divided into two categories: "**descriptive**" or "**inferential**". The former aims at describing the network community structure in the best possible way given some definition of modularity (modularity index). The latter instead aims at explaining the possible latent community structure that, by driving edge formation, gave rise to the observed network (generative model) [1].

In the following exercises, you will explore these differences in two distinct scenarios: one in which the network construction is fully known and one in which it is not.

1 Question 1

A network scientist wants to study the social network of a small school of 200 Blue Tang fish. According to the movie "Finding Nemo", Blue Tang fish have a really short memory [2][1]. Assuming that these fish have short term memory, the network scientist uses a model in which each fish is connected with only 4 other fish at **random** (undirected connections). What null model would best represent the network drawn by the network scientist? Would you expect this network to have a modular structure?

Write the code to produce the null model the network scientist picked and visualize the generated network. Do you notice any modular structure in the network you have generated?

The network scientist then decides to use a modularity optimization algorithm to search for communities. However, the result doesn't seem to match their expectations. Can you replicate what the network scientist observed using the following functions?

- `nx.community.greedy_modularity_communities`
- `nx.community.louvain_communities`

To remove any doubts, they install `graph-tool` following the instructions here and write the following piece of code:

```python
import graph_tool.all as gt

# convert networkx graph (nx_g) to graph-tool graph (gt_g)
gt_g = gt.Graph(nx.adjacency_matrix(nx_g), directed = False)
gt_g_state = gt.BlockState(gt_g)
print("State is initialized with: {:.0f} block ".format(gt_g_state.get_nonempty_B()))

# Next we call the MCMC function:
dS, nattempts, nmoves = gt_g_state.multiflip_mcmc_sweep(niter=1000)
print("Change in description length:", dS)
print("Number of moves attempted:", nattempts)
print("Number of accepted vertex moves:", nmoves)

print('\nUsing Baysian inference we get {:.0f} block(s) instead'.format(g_gt_state.get_nonempty_B())
print('and the modularity index is {:.0f}'.format(gt.modularity(g_gt, g_gt_state.get_blocks())))
```

After replicating what the network scientist did, what do you think happened when you used modularity optimization to search for the optimal network partition? Does the partition found by modularity optimization reflect the underlying network model used to generate the network?

Hint: see slide 14 from lecture 6!

[1] This is of course not true

2 Question 2

In this exercise, we are going to compare "descriptive" versus "inferential" community detection methods using a real world network generated using email data from an European research institution. The question we would like to answer here is the following: *what underlying network partition most likely gave rise to the observed network?*

NOTE: the dataset, which can be downloaded here, comes with a "ground truth" partition of the nodes, reflecting the actual community membership of each node. We can use this information to test whether it is preferable to use modularity optimization or Bayesian inference to answer to our question.

After downloading the data, you can construct two equivalent graphs using `networkx` and `graph-tool` as follows:

```python
path_to_edges_data = '<your_path_to>/email-Eu-core.txt'
path_to_labels_data = '<your_path_to>/email-Eu-core-department-labels.txt'
edge_list_email_g = np.loadtxt(path_to_edges_data, delimiter=' ', dtype=int)
labels_email_g = np.loadtxt(path_to_labels_data, delimiter=' ', dtype=int)
ground_truth_b = np.array(labels_email_g[:,1])

# create Graph using networkx
email_nx = nx.DiGraph()
email_nx.add_edges_from(edge_list_email_g)
N = email_nx.number_of_nodes()
L = email_nx.number_of_edges()

# create Graph using graph-tool
email_gt = gt.Graph(directed = True)
email_gt.add_edge_list(edge_list_email_g)

print('This is the networkx graph object:')
print(email_nx)
print('\nThis is the graph-tool graph object:')
print(email_gt)
```

When dealing with real world data, it is always a good idea to perform an initial summary network analysis to familiarize oneself with the data. What is the density of the network? Is it sparse or dense? Is the graph strongly or weakly connected? (note the email network is directed)

Having familiarized yourselves with the network, you start testing both modularity optimization algorithms and inference algorithms using `graph-tool`. You can refer to the `graph-tool` documentation here to learn how to infer modular network structures using Bayesian inference. Here is an example of how to do this:

```python
# we first only include in the analysis the strongly connected network using the following function
email_gt_gcc = gt.GraphView(email_gt, vfilt = gt.label_largest_component(email_gt))
print(email_gt_gcc)

# initialize state
email_gt_gcc_state = gt.BlockState(email_gt_gcc)
print("State is initialized with: {:.0f} block ".format(email_gt_gcc_state.get_nonempty_B()))
# this one block is the initial partition guess.

#Next we are going to call the MCMC function and propose some changes:
dS, nattempts, nmoves = email_gt_gcc_state.multiflip_mcmc_sweep(niter=1000)

print("Change in description length:", dS)
print("Number of moves attempted:", nattempts)
print("Number of accepted vertex moves:", nmoves)

print('\nNumber of communities found: ', email_gt_gcc_state.get_nonempty_B())
gt_b = email_gt_gcc_state.get_state()
```

What community detection method works best? How do you interpret your results?

A few extra questions:

- modularity optimization algorithms accept a parameter called resolution, which changes the preference towards detecting large or small communities. Does the algorithm output better results if you change this parameter?
- in this particular case, you have the actual ground truth partition of the network nodes. What is the modularity index of this partition? Is it higher or lower than that of the partitions found using the modularity maximisation algorithms?

- one way to quantify the algorithms' success at recovering the correct partition is to use information theoretic measures such as the normalized mutual information. You can find an implementation of this measure from scikit learn and import it using:[2]

  ```
  from sklearn.metrics import adjusted_mutual_info_score.
  ```

  Use this function to compare the partition you have obtained from the various methods you have tested against the actual network partition. Which one gives you the best result? Would you have reached the same conclusion if you didn't know about the ground truth partition?

References

1. Peixoto TP. Descriptive vs. Inferential Community Detection in Networks: Pitfalls, Myths and Half-Truths. Cambridge University Press, 2023. DOI: 10.1017/9781009118897. Available from: http://dx.doi.org/10.1017/9781009118897
2. Finding Nemo. Animated film. 2003. Available from: https://www.youtube.com/watch?v=KuvF113uty4&t=89s

[2] You may not have this installed, to install it see the scikit-learn documentation. Using `pip install scikit-learn` should work.