# Lab 03: Community Detection

CSC14114 Big Data Application 19KHMT

nhom 1

2023-05-10

# Contents

| Task | Completed |
|---|---|
| From raw data to graph | **Percent**: 100 |
| Betweenness calculation | **Percent**: 100 |
| Betweenness calculation comparison with `networkx` | **Percent**: 100 |
| Community detection | **Percent**: 100 |
| Community detection comparision with `networkx` | **Percent**: 100 |

# 1 Lab 03: Community Detection

## 1.1 The one do this version is Nguyen Ngoc Phuoc - 19127519

## 1.2 From raw data to graphs

### 1.2.1 Graph

You may find immplementation of this in `graph.py` file. This is a simple abstraction of this class. This is the data structure that will store all data point through out this lab.

```
class Graph:
    # add and edge to the graph
    def add_edge(self, u: str, v: str):
        pass
    # remove and edge from the graph
    def remove_edge(self, u: str, v: str):
        pass
    # get connected components in the graph
    def get_communities(self) -> list[tuple[str]]:
        pass
    # return copy of the graph
    def copy(self):
        pass
    # number of nodes
    def count_nodes(self) -> int:
        return len(self.nodes)
    # number of edges
    def count_edges(self) -> int:
        pass
    # whether node_u and node_v is connected by and edge
    def connected(self, node_u, node_v) -> bool:
```

```
        pass
```

### 1.2.2 GraphBuilder

This is where I encounter a problem. The graph construction is actually performed on 2 difference objects:

- `self-implemmented` graph
- `networkx` graph

To actually keep the business logic compact, not duplicate, I use builder design pattern to do the graph construction phases. You may find this in `graph_builder.py`.

This is the interface of all builder classes.

```
class GraphBuilder:
    # add and edge to the graph
    @abstractmethod
    def add_edge(self, node_u: str, node_v: str):
        raise NotImplementedError()
    # reset the builder result instance
    @abstractmethod
    def reset(self):
        raise NotImplementedError()
```

There are 2 implementation for this interface, which is :

```
class SelfImplementedGraphBuilder(GraphBuilder):
    pass
```

and:

```
class NetworkXGraphBuilder(GraphBuilder):
    pass
```

### 1.2.3 GraphDirector

This is a `Director` class that would perform on `GraphBuilder` for graph construction, business logic for graph construction is defined inside this class.

```
class GraphBuilderDirector:
    @staticmethod
    def build_graph_from_csv(file_name: str, graph_builder: GraphBuilder, thresh
        pass
```

Then in the `main.py` file, you will see the way I use these classes to perform graph constructions.

```
    # self implemented_graph
    implemented_graph_builder = SelfImplementedGraphBuilder()
    GraphBuilderDirector.build_graph_from_csv(data_file, implemented_graph_build
    implemented_graph = implemented_graph_builder.build()

    # networkx graph
    networkx_graph_builder = NetworkXGraphBuilder()
    GraphBuilderDirector.build_graph_from_csv(data_file, networkx_graph_builder,
    networkx_graph = networkx_graph_builder.build()
```

There's the graph construction phase, the output is 2 graph:

- `self-implemented` graph
- `networkx` graph

## 1.3 Betweenness Calculation

Again, there would be 2 different implemmentation in this phase:

- `self-implemented`
- `networkx`

The reason for this is because I want to compare my calculation with `networkx` one.

### 1.3.1 Tree

For the betweenness calculation, I go with the most obvious approach: because in the slide, lectures, books use a `Tree` to do it, so I will do it like that too.

As I present classes, which you can find in `tree.py`.

```
class TreeNode:
    pass
class Tree:
    pass
```

As each tree is constructed from a graph, the between calculation is done for that node which is the root.

This mean for each graph, I will construct n tree with n is the number of nodes within the graph. The betweenness calculation for each tree (each node as root) is done in a recursive manner, with the intuition from the slide, from top-down requested (bottom-up calculation).

I'm quite proud of this actually. I think that this lab is both hard and interesting compare to the others 2 because of this.

### 1.3.2 BetweennessCalculator

This is an interface that would perform calculation logic and save it to files. It can also perform benchmark between 2 files for comparison, I use `mse` for this benchmark. You can find these code in `betweenness_calculator.py`

```
class BetweennessCalculator:
    def calculate_betweenness(self, file_name: str = None) -> list[tuple[tuple[s
        pass

    @staticmethod
    def mse_benchmark_betweenness(betweenness_file_a: str, betweenness_file_b: s
> float:
        pass
```

And there are 2 classes that implement this:

- `class SelfImplementedBetweennessCalculator(BetweennessCalculator):`
  - Use `Tree` data structure to perform edge betweenness fusion for all nodes in graph.
- `class NetworkXBetweennessCalculator(BetweennessCalculator):`
  - Just call library function.

I run the code in `main.py`, then save the result to file and perform benchmark. The algorithm output 2 txt file:

- `output/edge_betweenness.txt`: for self implemented
- `output/edge_betweenness_lib.txt`: for the library version

The file is almost identical, just with a few different in calculation at a very small fraction level, as the `mse` score output is:

```
============================================
betweenness mse: 1.7621061666021155e-28
============================================
```

You may find the code call these function in `main.py`

```
    # betweenness self implemented
    implemented_betweenness_calculator = SelfImplementedBetweennessCalculator(im
    implemented_betweenness_calculator.calculate_betweenness(betweenness_out_fil

    # betweenness lib
    lib_edge_betweenness_calculator = NetworkXBetweennessCalculator(networkx_gra
    lib_edge_betweenness_calculator.calculate_betweenness(betweenness_lib_out_fi

    # benchmark betweenness
    BetweennessCalculator.mse_benchmark_betweenness(betweenness_out_file, betwee
```

## 1.4 Community Detection

For the last part, you may find source code in `community_detector.py`.

With the same approach used, I present you the base classs:

- It can perform communities detection and save the result to files
- It can perfrom benchmark between 2 different files.

```
class CommunityDetector:
    def detect(self, file_name: str = None) -> list[tuple[str]]:
        pass

    @staticmethod
    def benchmark_communities(community_a_file: str, community_b_file: str):
        pass
```

As for the benchmark in this class, you may find it quite strict, as it actually read both file and compare the content with a single == operator. But nonetheless, it works ;). My implementation and library yeild exactly the same result.

There're 2 classes that implement this base:

- `class SelfImplementedCommunityDetector(CommunityDetector):`
  - Modularity is calculated by the algorithm presented in the assignment pdf.
  - Girvan newman is `self-implemented` (quite simple, just remove edges till there's new community).
- `class NetworkxCommunityDetector(CommunityDetector):`
  - Just use modularity and girvan-newman in the library

You may find the code run for this in `main.py`:

```
# detection self implemented
community_detector = SelfImplementedCommunityDetector(implemented_graph)
community_detector.detect(community_detection_out_file)

# detection networkx
community_detector_lib = NetworkxCommunityDetector(networkx_graph)
community_detector_lib.detect(community_detection_lib_out_file)

# benchmark communities
CommunityDetector.benchmark_communities(community_detection_out_file, commun
```

This output 2 files:

- `output/communities.txt`: for the one I implemented
- `output/communities_lib.txt`: for the library one.

And the benchmark result with == operator used between 2 files:

```
==========================================
detected communities identical
==========================================
```

## 1.5 Reflection

- This lab is very interesting, and hard too than the other 2 lab.
- But nonetheless it bring me satisfaction when completed
- I learn more about girvan-newman, community detection, feel proud because I just wrote some sotisphicated algorithm in about a day or two

## 1.6 References

- Slides
- Assignment PDF