Instructors:
Michael Ritter
Hugo K. Kasuya Rosado

Exercise sheet #7
Combinatorial Optimization
SuSe 2024

Exercises with an asterisk mark "*" are optional.

1 Minimum Mean Cycle

Consider the following instance of an edge-weighted directed graph on $n = |V| = 6$ vertices that is depicted in Figure 1.

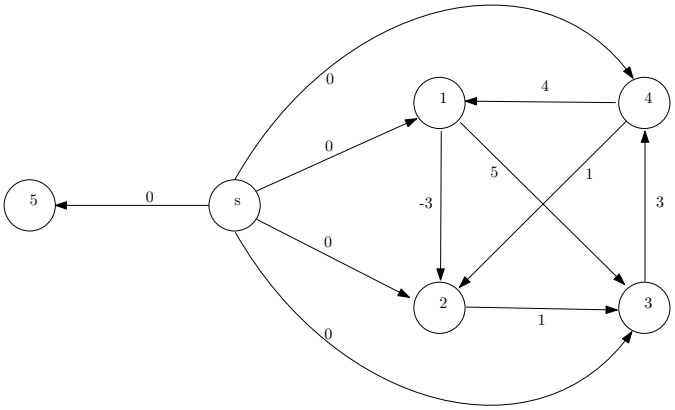

Figure 1: An edge-weighted directed graph.

a. For each $k \in \{0, \dots, n\}$, let $F_k(x)$ denote the shortest edge progression from vertex s to vertex x with exactly k edges. Compute $F_k(x)$ for each $k \in \{0, \dots, n\}$ and each $x \in V$. Because the graph is small, you don't have to formally execute an algorithm for this.
b. Let $c(F_i(x))$ denote the total weight of $F_i(x)$. For each $x \in V$, compute

$$F_{max}(x) := \max_{0 \le k \le n-1 : c(F_k(x)) < \infty} \left\{ \frac{F_n(x) - F_k(x)}{n-k} \right\}.$$

c. Compute $\mu := \min_{x \in V} \{F_{max}(x)\}$.
d. Let x^* be the vertex such that $F_{max}(x^*) = \mu$. What is the cycle in the edge progression $F_n(x^*)$? What is its mean weight?

2 Non-Negative Edges

Consider the special case of the MINIMUM COST FLOW PROBLEM where $c(e) \geq 0$ for all $e \in E(G)$. Let $F \subseteq E(G)$ be the set of edges $e \in E(G)$ for which an optimum solution f with $f(e) > 0$ exists.

a. Show that if there is a circuit C in $(V(G), F)$, then there is an optimal solution with $f(e) > 0$ for all $e \in E(C)$.
b. Show that every circuit C in $(V(G), F)$ consists only of edges e with $c(e) = 0$.

3 Coding the Minimum Mean Cycle Algorithm

In this exercise, we will write a program to run the Minimum Mean Cycle Algorithm.

a. Download from Moodle the following files and place them directory/folder: main.py, and graph_module.py.
   (a) file main.py is the file we will be executing in the terminal and it calls functions and methods defined in the other file;

(b) graph_module.py is the file we will be working on in this exercise. It contains an incomplete implementation of the Minimum Mean Cycle Algorithm.

With Python 3 installed, to run the code, simply open the terminal, navigate to the folder containing the files, then run the following command to run the code

```
python3 main.py
```

b. Open graph_module.py. This file contains the methods `minimum_mean_cycle()` and `find_cycle()`. The method `find_cycle()` has one argument:

(a) `x_star` denotes a vertex for which a shortest edge progression P from the source to `x_star` contains a cycle.

This method should return the corresponding cycle. Method `minimum_mean_cycle()` has no arguments and it should return a tuple of a minimum mean cycle and its total weight if a cycle exists, otherwise, it should return `None`.

The source vertex `src` is defined on line 6 and there should always be an out-edge with weight 0 from it to all other vertices. In the file main.py, on line 7, one can manually write the `adjacency_matrix` which is used as the input graph for the code. Optionally, line 19 can be uncommented to use an input graph generated randomly (details are in the comments in the file).

c. Implement the remaining of `find_cycle()` - there is no need to optimize the time complexity of your code. Here is a list of useful variables you could use for the implementation:

(a) `self.vertex_num` returns the number of vertices and `self.vertex_list` returns a list of vertices (each vertex is an integer between 0 and `self.vertex_num`-1)
(b) `self.f` is a matrix that such that `self.f[i][x]["weight"]` returns the weight of a shortest edge progression P from the source to vertex `x` with exactly `i` edges, and `self.f[i][x]["parent"]` returns the vertex that is the predecessor of `x` in P.

This method should return a cycle defined as a list of tuples `(u,v, weight(u,v))`, where `(u,v)` is a directed edge in the cycle, and `weight(u,v)` is its weight.

d. Implement the remaining of `minimum_mean_cycle()` - there is no need to optimize the time complexity of your code. Make use of `self.f` to compute both `mu` and `x_star` accordingly. After computing `x_star`, the method should call `self.find_cycle(x_star)` to obtain the corresponding minimum mean cycle. Then, it should return a tuple `(C,mu)` of the obtained minimum mean cycle `C` and its mean weight `mu` if such a cycle exists, otherwise, it should return `None`.

e. Uncomment line 19 to generate random graphs, and modify the parameters to see how it affects the output and the running times of the algorithm.