B(E)3M33UI — Scheduling: Critical Path Method

Radek Mařík, Jiří Spilka

May 21, 2019

1 Critical Path Method (CPM)

The goal of this task is to become familiar with scheduling, specifically with Critical Path Method (CPM) used heavily to schedule project activities. A critical path is determined by identifying the longest stretch of dependent activities and measuring the time required to complete them from start to finish.

A model of the project requires the following list:

- A list of all activities required to complete the project (typically categorized within a work breakdown structure)
- The time (duration) that each activity will take to complete
- The dependencies between the activities

Task 1: Design a non-trivial process with the following constraints.

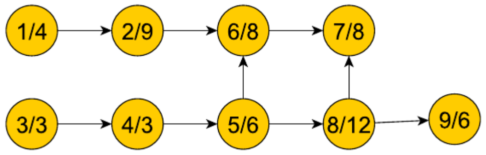
Constraints on the process:

- The process is non-trivial.
- Contains at least 10 activities.
- The sequence of activities is not linear.

Examples:

- Bike, PC, building construction
- Rock climbing
- Cooking recipe
- Software development plan.
- Soldier's operation schedule

First, we will use the simple example from lectures to compute the critical path for jobs processing times given by the following graph with node encoding (jobId/p_j):



jobs	1	2	3	4	5	6	7	8	9
p_j	4	9	3	3	6	8	8	12	6

Notation

- Forward procedure:
  - p_j – procedure time of jobs j
  - S'_j – the earliest possible start time of job j
  - C'_j – the earliest possible completion time of job j
  - $C'_j = S'_j + p_j$
  - $\{\text{all } k \rightarrow j\}$ jobs that are predecessors of job j
- Backward procedure:
  - S''_j – the latest possible start of job j
  - C''_j – the latest possible completion time of job j
  - $\{j \rightarrow \text{all } k\}$ jobs that are successors of job j

We will use the `networkx` to represent a graph. The nodes of the graph can be any object, e.g. `cpm.add_node(1,p=5)`,`cpm.add_node(2,p=5)`. This creates nodes with integer indices that can be further accessed as `cpm.node[1]`. The edges are then defined using `cpm.add_edges_from([(1,2)])`.

Task 2: Implement the forward and backward procedure in the module `cpm.py`

Hints:

- Use `nx.topological_sort`
- Use method `predecessors/successors` to get predecessors/successors nodes for node n.
- Use `nx.get_node_attributes` to get nodes attributes from all nodes, e.g. from C_j.

Task 3: Implement function `_compute_critical_path` in `cpm.py`

Hints:

- Save the nodes that are on critical path and then use the function `subgraph` (`self.subgraph`).

Task 4: Use the implemented CPM on your own non-trivial process.

2 Have fun!

Complete the exercise as a homework, ask questions on the forum, and upload the solution via the BRUTE!