Goals, assumptions and recommendations

The purpose of the assignment is to design the experiment and execute all the steps to evaluate the following general problem statement:

Given a directed graph, the *Maximum Flow Problem* consists of finding the maximum flow that can sent from a source vertex to a target vertex, without exceeding the capacity of each arc. We assume that a capacity of each arc is a positive integer. This problem arises in many real-life applications, for instance, to find a set of disjoint paths in a communications network taking into account the maximum bandwidth between every two network nodes.

There are several algorithms to solve this problem for a given input. For this project, we will consider three algorithms for the maximum flow problem: *Edmond-Karp* (EK), *Dinic*, and *Malhotra, Pramodh-Kumar and Maheshwari* (MPM). For a given graph *G=(V, A)*, where *V* is the vertex set and *A* is the arc set, EK Algorithm has a time complexity of $O(|V||A|^2)$, whereas Dinic has $O(|A||V|^2)$ and MPM has $O(|V|^3)$. Therefore, the first is better suited for graphs with few arcs, while the last two are better for graphs with fewer vertices. However, these are theoretical results that only apply to the worst case and it is hard to extrapolate them for a given particular problem input.

Your goal is to investigate the runtime performance of these three algorithms in practice. You believe that several problem features may play a role on performance, such as, the number of vertices, the number of arcs, the graph topology, the capacity at each arc, etc.. Although this is a somewhat artificial problem statement (the main goal is pedagogic and not to represent a realistic research question), it is worth noting that it requires all the steps of a real experiment. Furthermore, the type of measurements required by this experiment (runtime performance) is highly representative of experiments with computers.

The following assumptions and recommendations should be taken into account:

- There are three programs (EK.exe, Dinic.exe, MPM.exe) that implement algorithms above.
- All implementations require two arguments: the maximum CPU-time in seconds to run the program and the name of the file that contains the input graph. All programs output the following information for each input file: the value of the maximum flow found, or value -1 in case the program was not able to find the solution within the maximum time defined by the user, and the CPU time taken (in both cases). Since some experiments may take too much time, define the maximum CPU-time at your choice.
- There is an input data generator in python (gen.py) I do not have the permission to share the code. This generator creates randomly generated input data (a file) that can be read by the two implementations mentioned above. You need to define the number of vertices (n), the probability of generating an arc between two vertices (p), the maximum capacity (r) that can be assigned to an arc, a random seed and the name of the file that will contain the input data. The code will generate a random graph and assign a capacity to each arc that is taken randomly from the interval [1,c] (according to an uniform distribution). Each input file that is generated contains two integers (n and m) in the first row, and each of the next m lines contains three integers, the indices of two vertices that are connected by an arc and the maximum capacity of this arc. The vertices are numbered from 1 to n. We assume that the source vertex has index 1 and the target vertex has index n. You may consider other generators with different features. Note that if p is too small, the graph may not be feasible for the problem, that is, there exists no path between vertex 1 and vertex n. In that case, the file will only contain value -1.