MATH 566: Lecture 1 (08/20/2024)

This is Math 466/566 Network Optimization
I'm Bala Krishnamoorthy (call me Bala).

Today

* syllabus, logistics
* The Königsberg bridges problem
* Network flow problems

A field that is closely related to network optimization is graph theory. While we will use several results that might also typically be presented in a graph theory course (Math 453/553), we will concentrate more on the "network" aspects.

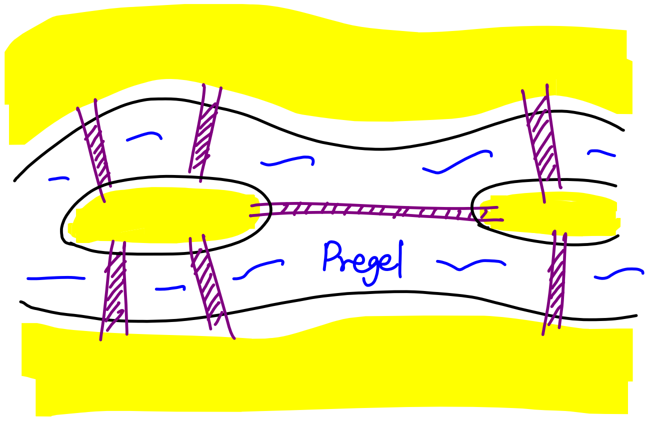
One key difference between "graphs" and "networks" as used conventionally, is that the arcs (or edges) in networks have capacities. In other words, one could think of them as pipes with limits on how much fluid could be sent through them at a time. On the other hand, edges in "graphs" are typically "lines" drawn between the points representing the nodes (or vertices). Thus, we will adopt a "flow" point of view.

Another difference between our approach and that of a typical graph theory class is that we will emphasize efficient algorithms to solve the optimization problems on networks. We will also explore numerous applications from various fields, including science, engineering, and business, of the different network problems we will study.

We will also emphasize **implementation of several algorithms**. Thus, there will be a sizeable programming component in this class.

The Königsberg Bridges Problem

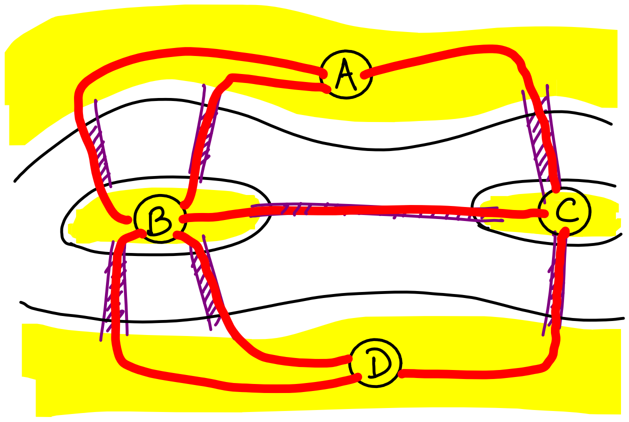
We start will a historical note. The first papor in graph theory (and, by extension, in network optimization) was published by Euler in 1736. He was visiting Königsberg in Prussia (now in Kaliningrad) The river Pregel flows through the town, and had seven bridges across it.

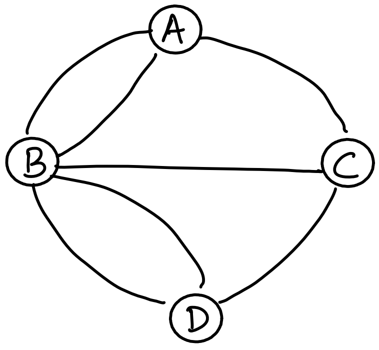


The citizens asked him if one could cross each of the seven bridges exactly once, and return to the starting point?

It was generally considered impossible to do. Euler characterized when it would be possible.

Here is a model for this problem. We represent each land mass by a node, and each bridge by an edge connecting the nodes representing the two land masses in question.





graph

We can restate the question as follows. In the graph, is it possible to start at A, say, traverse each edge exactly once, and return to A?

Since we return to where we started, such a "walk" is a cycle. Such a cycle, which traverses each edge exactly once, is called an Eulerian cycle. Hence we can ask "Does there exist an Eulerian cycle in the graph"?

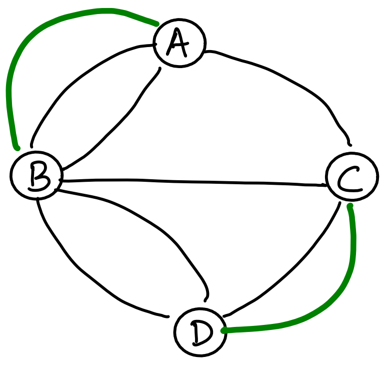
Theorem An undirected graph has an Eulerian cycle iff

* every node has an even degree, and
* the graph is connected, i.e., every pair of nodes has a "path" between them.

The degree of a node is the number of edges connected to it. We will not get into the details of this result right now. But one could get the intuition behind the even degree requirement. We should be able to "come in" to a node on an edge, and "go out" on another edge to a different node.

Many results (theorems) we present in this course will have a similar nature – intuitive to grasp and understand, and easy to state (may be not always easy to prove ☺!).

In the present case, if there were two more bridges as shown here, there would exist an Eulerian cycle.



Notice that all nodes have even degrees now (A, C, D: 4, B: 6).

A Related problem (Hamiltonian cycle problem): Does there exist a "cycle" that visits every node exactly once? This is the traveling salesman problem (TSP).

There are many applications of the TSP. It is one of the most well-studied network optimization problems. One typical application is in chip design, where the robotic arm printing the circuit pattern on the chip has to be programmed to finish printing the entire circuit in an optimal fashion in order to minimize the time spent.

We will now introduce several network optimization problems, which we will study in detail.

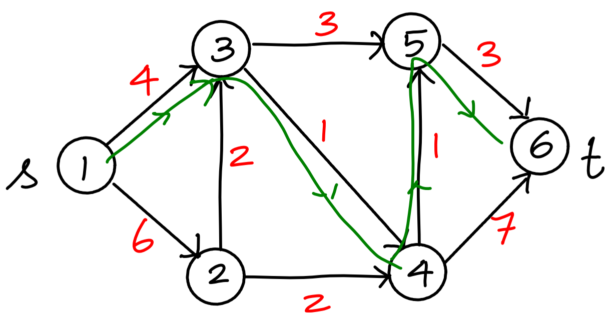
One more point about notation. In graph theory, it is common to call a graph as $G=(V,E)$, where V is the set of vertices and E the set of edges (undirected, by default). We will use the notation $G=(N,A)$ to denote a network, where N is the set of nodes and A is the set of arcs, which are directed by default.

Network Flow Problems

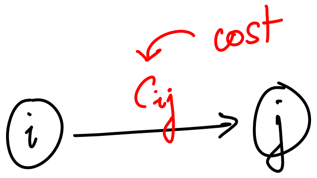
① Shortest path problem

Objective: find a path of minimum cost (length) from an origin (or source) node s to a destination (or sink) node t in a network with node set N and arc set A.

Here is an example:



Notation:



optimal or shortest path is shown in green

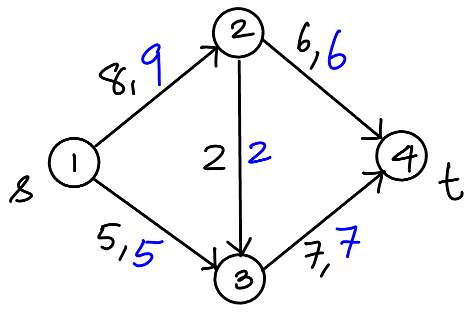
A typical application is driving directions on Google Maps. The costs c_{ij} could just be the distances between the cities modeled by the nodes, or could capture other relevant info such as traffic, tolls, etc.

Notice that the only parameter specified for the arcs is the cost, and the nodes themselves have no additional attributes (except for two of them being specified as s and t).

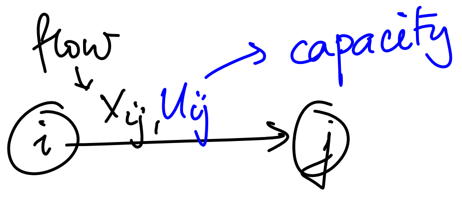
② Maximum flow problem (or max-flow)

Objective: Find a "feasible" flow (i.e., a solution that honors capacity limits on arcs) that sends maximum flow from a source node s to a sink node t.

Example:



notation:



max flow value $v = 13$.

Typical examples include traffic flow management - in road transportation networks or in computer networks.

Notice that we are working with directed arcs This will be the default mode going forward, unless mentioned otherwise.

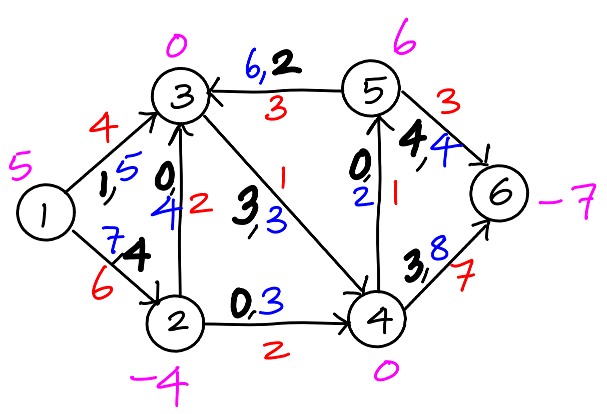
In the above instance, notice that the total flow coming into nodes 2 and 3 is equal to the total flow going out of that node. These nodes are "transshipment nodes", i.e., there is no loss or addition of material in these nodes. For the node s, we assume there is an infinite supply of material to be sent to node t, honoring the capacities.

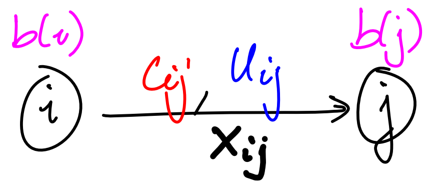
We point out that the set of parameters specified in max flow include u_{ij}'s, the capacities on arcs. Notice there are no costs (c_{ij}) specified (unlike in the shortest path case).

③ Minimum cost flow (min-cost flow problem)

Objective : determine a least-cost shipment of commodity through a network in order to satisfy demands at certain nodes using supply at certain other nodes.

An illustration:





b(i): supply/demand

Nodes 1,5 : supply nodes (S)
2,6 : demand nodes (D)
3,4 : transshipment nodes.

We assume $\sum_{i \in N} b(i) = 0$ by default. In other words, the total supply in all supply nodes (S) is equal to the total demand in all demand nodes (D). The goal is to ship the supply from S nodes to D nodes at the minimum total cost while honoring capacity limits on arcs.

In min-cost flow, we specify costs (c_{ij}), capacities (u_{ij}), and supply/demand values at nodes (b_i).