ECE 232E Lecture 11 and 12 notes

Professor Vwani Roychowdhury

June 7, 2018

In this set of lecture notes, we will study one-to-all shortest path algorithm, edge-betweeness community detection algorithm, and max flow problems in graphs.

1 Shortest path algorithm

Given a graph, $G = (V, E, W)$, we can find the shortest path between the nodes in the network. If the weights on all the edges are 1, then the shortest path finding is equivalent to finding the path with the minimum number of edges. In this lecture, we will explore the algorithm for solving the one-to-all shortest path problem. In one-to-all shortest path problem, we have a source node s and we want to find

$$\delta(s, v_i), \;\; \forall v_i \in V \setminus s$$

where $\delta(s, v_i)$ is the shortest path between s and v_i. If we assume that the weights on the edges are non-negative, then we can solve the one-to-all shortest path problem using Dijkstra's algorithm.

1.1 Dijkstra's one-to-all shortest path algorithm

We first explain the idea behind the Dijkstra's algorithm and then state the algorithm. At any stage in the Dijkstra's algorithm, we have a set $W \subset V$ of nodes and distances $\rho(x)$ for all $x \in V$ with the property

$$\rho(x) = \text{shortest path length from } s \text{ to } x\text{, using only intermediate nodes in } W$$

Now suppose we consider a node $x \notin W$ with smallest $\rho(x)$

$$x = arg \min_{u \notin W} \rho(u)$$

Then, we can add x to W and update the distances of the nodes $\rho(y)$ for $y \notin W$ by

$$\rho(y) = min\{\rho(y), \rho(x) + c_{xy}\}, \;\; \forall y \notin W \tag{1}$$

where c_{xy} is the weight on the edge e_{xy}. From equation 1 we can observe that the distance of the nodes $y \notin W$ are either unaffected by the addition of x to W (the shortest path from s to y does not pass through x) or is given by the shortest distance from s to x via nodes in W plus the distance between x and y

(the shortest path from s to y passes through x). We repeat the above procedure of continually adding nodes to W until $W = V$. The Dijkstra's algorithm is stated below

1. Initialization step
   - $W = \{s\}$
   - $\rho(s) = 0$
   - $\rho(y) = c_{sy}, \ \ \forall y \in V \setminus \{s\}$
2. Computation and update step
   - $x = arg \min_{u \notin W} \rho(u)$
   - $W = W \cup \{x\}$
   - $\rho(y) = min\{\rho(y), \rho(x) + c_{xy}\}, \ \ \forall y \in V \setminus W$
3. Repeat step 2 until

$$V = W$$

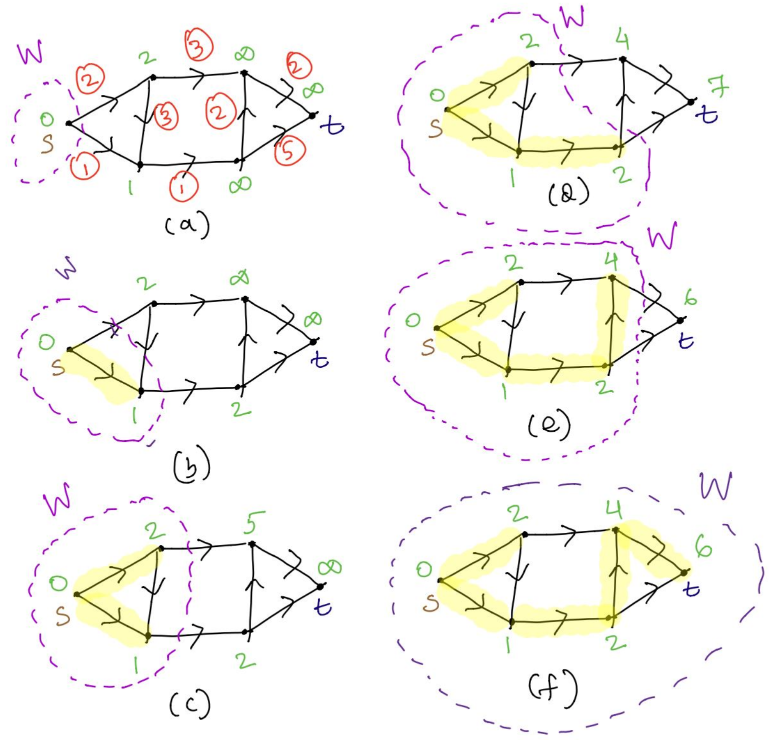

Figure 1: Successive stages in Dijkstra's algorithm

In the above figure, we can observe that every time a node is added to W, the corresponding edge is thatched (highlighted in yellow). This results in a tree rooted at s with the shortest paths from s to all other nodes.

2 Edge-betweeness community detection algorithm

Before stating the algorithm, we will define the notion of edge-betweeness in networks. We compute the shortest path between each pair of nodes in the network ($\binom{n}{2}$ pairs in an undirected network). Then we define, the edge-betweeness of an edge e denoted as $w(e)$

$$w(e) = \text{Number of shortest paths } e \text{ is a part of} \tag{2}$$

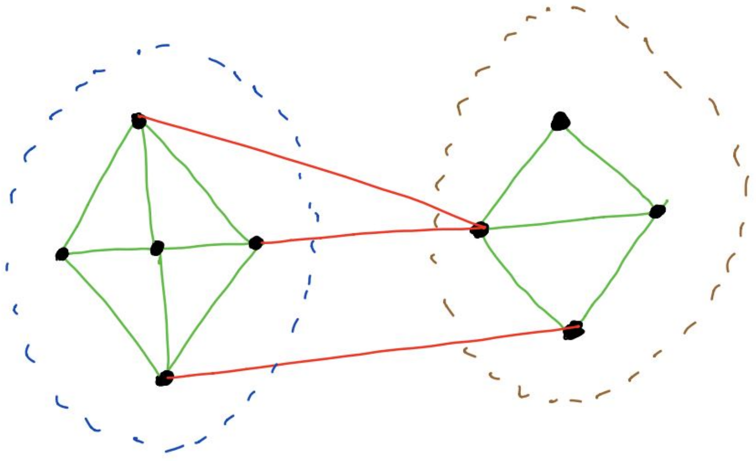

Figure 2: Edge-betweeness community detection

From figure 2, we can observe that edges which act as bridges between clusters (inter cluster edges marked in red) have high $w(e)$ because all the shortest paths between nodes belonging to different clusters have these edges as a part.

Having defined the notion of edge-betweeness, now we will state the algorithm:

1. Compute $w(e)$ for all the edges in the network. Delete the edge with the highest $w(e)$.
2. Recompute $w(e)$ for all the edges in the modified network and delete the edge with the highest $w(e)$.
3. Repeat step 2 until you get disjoint clusters. To be specific, you compute the modularity index after execution of step 2 and if there is an increase in the modularity index then you repeat step 2. If there is a decrease in the modularity index, then you terminate the algorithm.

3 Finding paths in networks using node-edge incidence matrix

In this section, we will show that how we can find paths in networks using node-edge incidence matrix. We first define the node-edge incidence matrix and then use it to find paths in the network.

3.1 Node-edge incidence matrix

Node-edge incidence matrix, denoted as C, has nodes as rows and edges as columns ($C \in \mathbb{R}^{|V| \times |E|}$). The entries of the matrix are

$$C_{ik} = \begin{cases} 1, & \text{if edge } e_k \text{ is directed out of node } v_i \\ -1, & \text{if edge } e_k \text{ is directed in to node } v_i \\ 0, & \text{otherwise} \end{cases}$$

Let's consider the directed network given in figure 2.

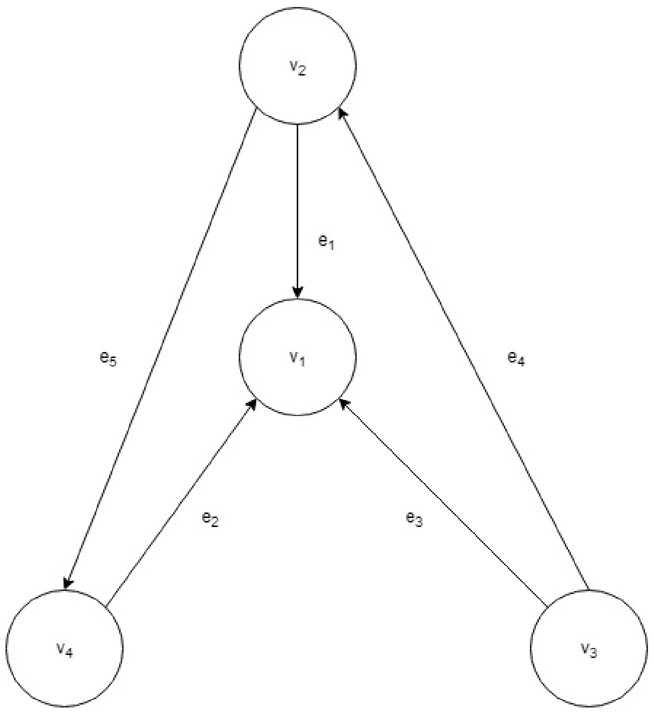

Figure 3: Directed acyclic network

The node-edge incidence matrix for the directed network is given below

$$C = \begin{bmatrix} -1 & -1 & -1 & 0 & 0 \\ 1 & 0 & 0 & -1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & -1 \end{bmatrix}$$

3.2 Paths in directed acyclic networks

In this section, we will formulate the path finding problem in a directed acyclic network as a LP feasibility problem. To facilitate the formulation, let's define the path vector, denoted as $Q \in \mathbb{R}^{|E|}$, in the following manner

$$Q(i) = \begin{cases} 1, & e_i \in \text{path } P \\ 0, & e_i \notin \text{path } P \end{cases}$$

Then with the above notation, there exists a directed path from node v_i to v_j if the following linear program has a feasible solution

$$\begin{aligned} \min_{Q} \quad & 0 \\ \text{subject to} \quad & CQ = F, \;\; Q \geq 0 \end{aligned} \tag{3}$$

where,

$$F = \begin{bmatrix} 0 \\ 0 \\ . \\ . \\ 1 \\ 0 \\ . \\ -1 \\ 0 \\ . \\ 0 \end{bmatrix}$$

The i^{th} entry of F is 1, j^{th} entry is -1, and all the other entries are 0.

3.2.1 Shortest paths in directed acyclic networks

We can use a linear program (LP) to find the shortest path from node v_i to v_j. If we define the weight vector $w \in \mathbb{R}^{|E|}$ as

$$w(i) = \text{weight of } e_i$$

then the solution to the following linear program gives the shortest path from node v_i to v_j

$$\begin{aligned} \min_{Q} \quad & w^T Q \\ \text{subject to} \quad & CQ = F, \;\; Q \geq 0 \end{aligned} \tag{4}$$

where,

$$F = \begin{bmatrix} 0 \\ 0 \\ . \\ . \\ 1 \\ 0 \\ . \\ -1 \\ 0 \\ . \\ 0 \end{bmatrix}$$

The i^{th} entry of F is 1, j^{th} entry is -1, and all the other entries are 0.

3.3 Paths in directed cyclic networks

In directed cyclic network, we define the path vector $Q \in \mathbb{R}^{|E|}$ in the following manner

$$Q(i) = \begin{cases} k, & k \text{ is the number of times edge } e_i \text{ is traversed in path } P \\ 0, & e_i \notin \text{path } P \end{cases}$$

Then with the above notation, there exists a directed path from node v_i to v_j if the following linear program has a feasible solution

$$\begin{aligned} \min_{Q} \quad & 0 \\ \text{subject to} \quad & CQ = F, \;\; Q \geq 0 \end{aligned} \tag{5}$$

where,

$$F = \begin{bmatrix} 0 \\ 0 \\ . \\ . \\ 1 \\ 0 \\ . \\ -1 \\ 0 \\ . \\ 0 \end{bmatrix}$$

The i^{th} entry of F is 1, j^{th} entry is -1, and all the other entries are 0.

In the LP feasibility problem for both the cyclic and acyclic case, we do not put integer constraints on the path vector Q. The unimodulairity of the node-edge incidence matrix C ensures that the basic feasible solutions of the LP are integer valued.

3.3.1 Detecting cycles in directed networks

We can use LP to detect cycles in directed networks. If the LP given by equation is unbounded below, then the directed network has a cycle

$$\begin{aligned} \max_{Q} \quad & \mathbf{1}^T Q \\ \text{subject to} \quad & CQ = 0, \;\; Q \geq 0 \end{aligned} \tag{6}$$

From the above formulation, it can be observed that if there is a cycle in the network then we can keep increasing the number of times an edge (part of a cycle) is traversed in a path and still get a feasible Q. This causes the cost function to approach infinity.

Existence of cycles in a directed network can cause the shortest path problem to be unbounded below. The cost function is unbounded below if and only if there exists a negative weight cycle that intersects an $i \to j$ path.

4 Max-flow in directed networks

In this section of the lecture notes, we study a very important problem in graph theory, known as the max-flow problem. We have a weighted directed network $G = (V, E, b)$ where b is the vector of edge capacities. Also, in this network, the vertex set V has two special nodes:

- Source node s
- Sink node t

In max-flow problem, the objective is to push maximum flow from the source to the sink node. Since the edges in the network have capacity constraints and the intermediate nodes have no storage, so we have the following constraints:

- Capacity constraint : $f_e < b_e, \;\; \forall e \in E$
- Conservation of flow: $\sum_{e \in In(i)} f_e = \sum_{e \in Out(i)} f_e, \;\; \forall i \in V \setminus (s,t)$

where f_e is the flow along edge e and b_e is the capacity of edge e. We also, define the net flow into the network, denoted by d as

$$\sum_{e \in Out(s)} f_e - \sum_{e \in In(s)} f_e = \sum_{e \in In(t)} f_e - \sum_{e \in Out(t)} f_e = d \tag{7}$$

With the above notation, the max-flow LP is given by equation

$$\begin{aligned} \max_{f_e} \quad & \sum_{e \in Out(s)} f_e - \sum_{e \in In(s)} f_e \\ \text{subject to} \quad & f_e < b_e, \;\; \forall e \in E \\ & \sum_{e \in In(i)} f_e = \sum_{e \in Out(i)} f_e, \;\; \forall i \in V \setminus (s,t) \end{aligned} \tag{8}$$

Since C is totally unimodular (TUM), so if the edge capacity vector b is integer valued then the feasible flow vector f is guaranteed to be integer valued.

4.1 Augmenting path algorithm

The augmenting path algorithm or the labeling algorithm solves the max-flow problem in directed networks by finding directed paths from s to t along which flow can be augmented. The steps in the algorithm are stated below:

1. Start with a feasible flow, $f = 0$
2. Modify the directed network G in the following manner
   - Replace the capacity of the forward edges (i, j) in the directed path P (from s to t) with excess capacity δ_{ij} where
   $$\delta_{ij} = b_{ij} - f_{ij}$$
   - Replace the capacity of the backward edges (j, i) in the directed path P (from s to t) with capacity δ_{ji} where
   $$\delta_{ji} = f_{ij}$$

   We will denote this modified network as $\tilde{G}$
3. Find a directed path P from s to t in $\tilde{G}$. Augment the flow in this path by δ, where
   $$\delta = \min_{(i,j) \in P} \delta_{ij}$$
4. Repeat steps 2 and 3 until s and t is disconnected