ECE 232E Lecture 13 notes

Professor Vwani Roychowdhury

June 8, 2018

In this set of lecture notes, we will explore all-to-all shortest path algorithms,the relationship between max-flow and min-cut, and various graph theory problems that can be posed as max-flow problems.

1 Shortest path algorithm (all-to-all)

In the previous set of lecture notes, we covered Dijkstra's algorithm for solving the one-to-all shortest path problem. In this section, we will explore algorithm for solving the all-to-all shortest path problem. Floyd-Warshall algorithm finds the shortest path between all pairs of nodes. The algorithm takes as input an $|V| \times |V|$ array of numbers d_{ij}, initially set to the edge weights w_{ij}, of the directed graph $G = (V, E, W)$. The crux of this algorithm lies in the triangle operation, which replaces, for all i and k and a fixed j, the d_{ik} entry with the distance $d_{ij} + d_{jk}$ if $d_{ik} > d_{ij} + d_{jk}$

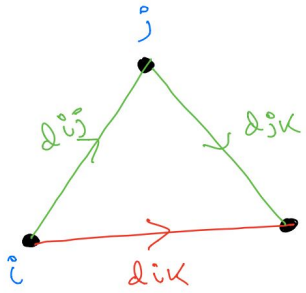

Figure 1: Triangle operation

The algorithm is based on the following theorem:

Theorem: If we perform a triangle operation for successive values $j = 1, 2, \cdots, |V|$, then each entry d_{ik} becomes equal to the length of the shortest path from i to k, assuming the weights $w_{ij} \geq 0$.

There exists a simple inductive proof to the above theorem in the textbook. The above theorem then gives the following Floyd-Warshall algorithm:

1. **Initialization step**
   (a) $\forall i \neq j, d_{ij} = w_{ij}$
   (b) for $i = 1, 2, \cdots, |V| \quad d_{ii} = \infty$
2. **Triangle operation** for $j = 1, \cdots, |V|$, $i = 1, \cdots, |V|$ $i \neq j$, $k = 1, \cdots, |V|$ $k \neq j$
   (a) $d_{ik} = min\{d_{ik}, d_{ij} + d_{jk}\}$

2 Relationship between max-flow and min-cut

A cut of a network with respect to two nodes s and t (denoted as s-t cut) is a partition of the vertex set V into two disjoint sets W_1 and W_2 such that

$$W_1 \cup W_2 = V$$
$$W_1 \cap W_2 = \emptyset$$
$$s \in W_1, t \in W_2$$

We define the capacity of the s-t cut, denoted as $C(W_1, W_2)$, as

$$C(W_1, W_2) = \sum_{\substack{e=(u,v) \in E \\ u \in W_1, v \in W_2}} b_e \tag{1}$$

where b_e is the capacity of edge e. An s-t cut is shown in figure 2

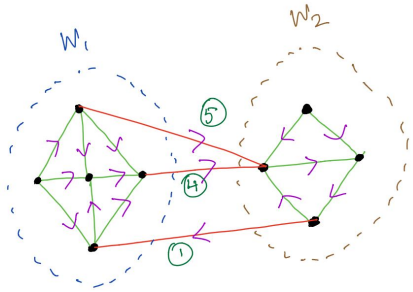

Figure 2: s-t cut

The capacity of the s-t shown in figure 2 is

$$C(W_1, W_2) = 5 + 4 = 9$$

From the above figure, it can be observed that the capacity of a cut is the sum of the capacities of the forward arcs: those which go from nodes in W_1 to nodes in W_2. Therefore, we would expect that the value of an s-t flow cannot exceed

the capacity of an s-t cut, since all the flow from s to t must flow through the forward arcs of a cut. Mathematically, we have

$$d_{max} \leq \min_{W_i, W_j} C(W_i, W_j) \tag{2}$$

The augmenting path algorithm terminates when the forward arcs have been saturated and there is zero flow on the backward arcs. Hence,at termination we have

$$d_f = C(W_1', W_2') \tag{3}$$

where the partition (W_1', W_2') is determined by the algorithm. Combining equations 2 and 3 we have

$$\begin{aligned} d_{max} &\leq d_f \\ \Rightarrow d_{max} &= d_f \end{aligned}$$

Therefore,we have the following relationship between max-flow and min-cut

$$\text{Max flow} = \text{Min-cut capacity}$$

3 Variations of max-flow problem

In this section, we will formulate some graph theory problems as variations of the max-flow problem.

3.1 Edge disjoint paths from s to t

We can find the number of edge disjoint paths from s to t by solving the max-flow problem in the directed network where each edge have unit capacity.

$$b_e = 1, \quad \forall e \in E$$

Each unit of flow defines an edge disjoint path from s to t, and therefore we have

$$\text{max-flow} = \text{number of edge disjoint paths from } s \text{ to } t$$

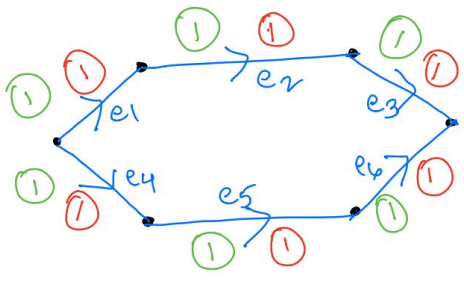

Figure 3: Edge disjoint paths

In the network shown in 3, the edge capacities are marked in red and the optimal flows are marked in green. From the figure, it can be observed that the max-flow from s to t is 2 and there are also 2 edge disjoint paths from s to t.

$$P_1 = \{e_1, e_2, e_3\}$$
$$P_2 = \{e_4, e_5, e_6\}$$

3.2 Maximum matching in bipartite networks

A matching M is a subset of the edge set E, such that the edges in M are node disjoint. Mathematically,

$$\text{If } (u,v),(w,z) \in M, \text{ then } u \neq w, z \text{ and } v \neq w, z$$

There are two types of matching problem:

- **Unweighted maximum matching problem**: Find a matching M with maximum size
- **Weighted maximum matching problem**: If $w(e)$ are non-negative weights on edges, then find a matching M that maximizes $\sum_{e \in E} w(e)$

We will restrict the matching problems to bipartite networks. In a bipartite network, $V = U \cup V$, where U might be applicants and W might represent jobs. In this bipartite network, a matching M is assignments of applicants to jobs. By solving the maximum matching problem, we can assign the maximum number of applicants to jobs. We can formulate the maximum matching problem in bipartite networks to a max-flow problem by performing the following modification:

$\forall u \in U$ add a directed edge from s to u $(s \rightarrow u)$ and $\forall w \in W$ add a directed edge from w to t $(w \rightarrow t)$. Also, for every edge $(u, v) \in E$ make it a directed edge $(u \rightarrow w)$. We also make the capacity of the edges to be 1 $(b_e = 1)$.

In the modified network(described above), any feasible flow defines a matching and maximizing the flow from s to t we can get the maximum matching. The edges (u, v) that has a flow of 1 in the max-flow solution will be in the matching set M.

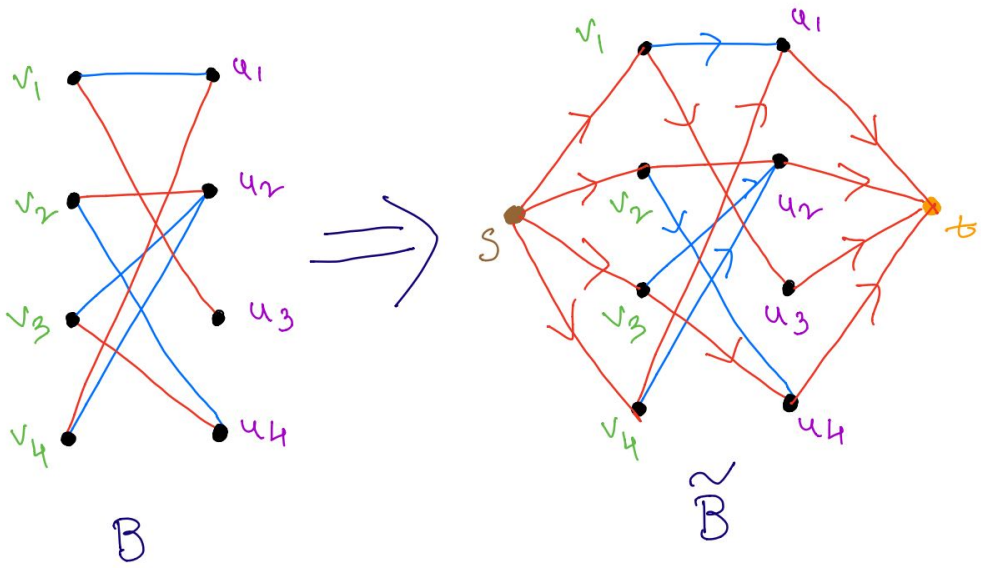

Figure 4: Maximum matching in bipartite networks

In figure 4, we have a bipartite network B and the maximum matching of size 4 is marked in red. We modify the bipartite network to get a directed network $\tilde{B}$ and solve the max-flow problem to get a max-flow of 4 and the paths from s to t are marked in red (each edge-disjoint path pushing unit flow from s to t).

We don't cover weighted maximum matching problems in this set of lecture notes, but you can refer to the textbook for Hungarian algorithms which are used to solve weighted maximum matching problems.