CSC 373 H 1 Y — Summer 2007

University of Toronto — St. George Campus

Lecture Summary for Week 5

This summary is not a replacement for the lecture. If you miss a class, please arrange with a friend to take note for you.

3 NETWORK FLOW [K&T Chapter 7, CRLS Chapter 26]

Consider the following example (from CRSL Section 26.1): The Lucky Puck Company has a factory in Vancouver that manufactures hockey pucks and a warehouse in Winnipeg that stocks them. The company wants to ship pucks from the factory to the warehouse using trucks. Because the trucks travel over specified routes between cities and have a limited capacity, Lucky Puck can ship at most $c(u, v)$ crates perday between each pair of cities u and v in the following Figure 2 ($c(u, v)$ is shown next to the edge (u, v)).

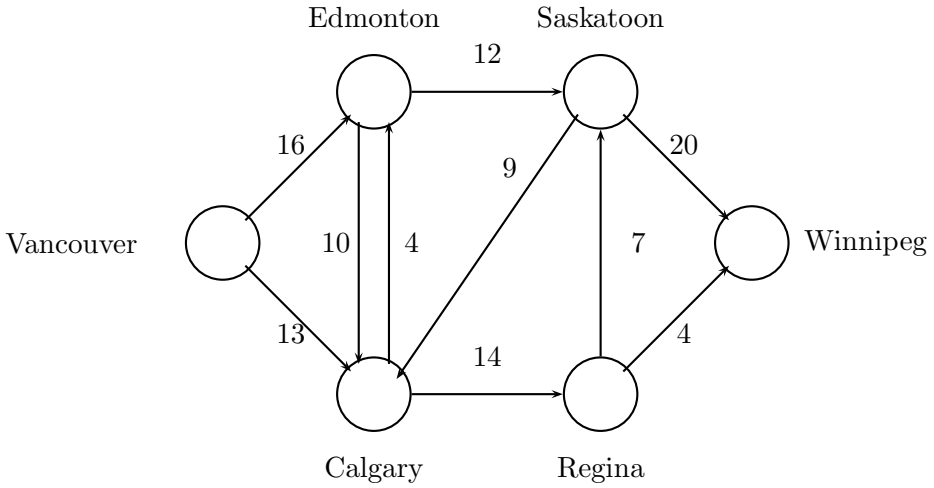

Figure 1: A "flow network"

Their goal is to determine the largest number p of crates per day that can be shipped (they will produce exactly this many pucks at the factory). The company is not concerned with how long it takes for a given truck to get from the factory to the warehouse; they care only that p crates per day leave the factory and p crates per day arrive at the warehouse.

A "flow" of shipments with total 19 crates per day is shown in Figure 2.

3.1 Flow Network

A flow network is defined formally as follows:

Definition: A *flow network* is a directed graph $G = (V, E)$ where

- associated with each edge e is a capacity $c(e) > 0$;

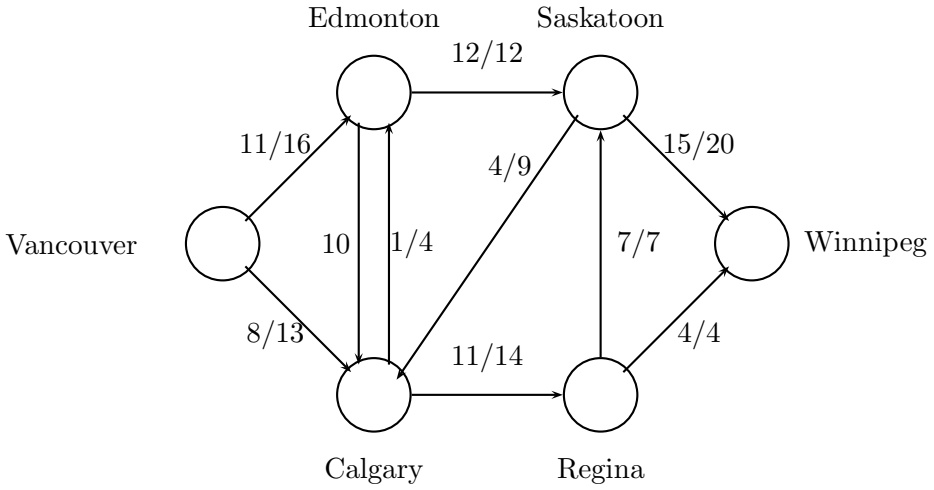

Figure 2: A "flow" of value 19

- there are two designated nodes in V: s (the source) and t (the sink) (nodes other than s, t are called *internal nodes*);
- there is no edge entering the source s, and no edge leaving the sink t;
- for every internal node v, there is a path from s to v to t.

A flow f in the flow network G is a real function on the set of edges ($f : E \to \mathbb{R}^+$) that satisfies:

1. **Capacity condition**: $0 \leq f(e) \leq c(e)$ for every $e \in E$
2. **Conservative condition**: *the total flow into an internal node is the same as the total flow outgoing from it*, i.e.,
$$\sum_{e \text{ goes into } v} f(e) = \sum_{e \text{ leaves } v} f(e)$$
for every internal node v.

Value of a flow: The value Value(f) of a flow f is defined to be the total flow that leaves the source:
$$Value(f) = \sum_{e \text{ leaves } s} f(e)$$

***st*-Cut**: An *st*-cut (or just a cut) is a partition (A, B) of V where $s \in A$ and $t \in B$.
Examples of cuts include $(\{s\}, V - \{s\})$, $(V - \{t\}, \{t\})$.

Capacity of a cut: The capacity of a cut (A, B) is defined to be
$$c(A, B) = \sum_{e \text{ leaves } A} c(e)$$

Lemma: For any cut (A, B):
$$Value(f) = f^{out}(A) - f^{in}(A)$$

where

$$f^{out}(A) = \sum_{e \text{ leaves } A} f(e), \qquad f^{in}(A) = \sum_{e \text{ goes into } A} f(e)$$

Proof

$$\begin{aligned}
Value(f) &= f^{out}(s) \\
&= f^{out}(s) - f^{in}(s) \\
&= \sum_{v \in A} (f^{out}(v) - f^{in}(v)) \qquad \text{(by conservation: } f^{out}(v) - f^{in}(v) = 0 \text{ for } v \in A, v \neq s) \\
&= \left(\sum_{v \in A} f^{out}(v) \right) - \left(\sum_{v \in A} f^{in}(v) \right) \\
&= \left(\sum_{e \text{ leaves } A} f(e) + \sum_{e \text{ inside } A} f(e) \right) - \left(\sum_{e \text{ goes into } A} f(e) + \sum_{e \text{ inside } A} f(e) \right) \\
&= \sum_{e \text{ leaves } A} f(e) - \sum_{e \text{ goes into } A} f(e) \\
&= f^{out}(A) - f^{in}(A) \qquad \text{QED.}
\end{aligned}$$

Corollary 1: $Value(f) = f^{in}(B) - f^{out}(B)$.

Proof: The corollary follows from the fact that $f^{in}(B) = f^{out}(A)$ and $f^{out}(B) = f^{in}(A)$. QED

Corollary 2: $Value(f) = f^{in}(t)$.

Proof: The corollary follows from Corollary 1 by taking $B = \{t\}$. QED

Corollary 3: $Value(f) \leq c(A, B)$.

Proof: The corollary follows from the lemma, because $f^{out}(A) \leq c(A, B)$ (by capacity condition for each edge leaving A) and $f^{in}(A) \geq 0$ (by capacity condition for each edge going into A). QED

Corollary 3 implies that the maximum value of a flow is bounded above by the minimum capacity of a cut. We will prove later that these two are actually equal.

3.2 The Maximum Flow Problem

The Maximum Flow Problem: Given a flow network, the problem is to find a flow of maximum value.

It is important to note that when all capacities are integers, there is a maximum flow f_{max} where $f_{max}(e)$ is an integer for every edge e. Furthermore, there are polytime algorithms (e.g., Ford-Fulkerson, Karp-Edmond, etc) that produce such maximum flow. (Note that even when all capacities are integers, there are maximum flows where the flows on the edges are not integers.) This is a topic for next week.

Here we focus on how to use an existing algorithm for the Maximum Flow Problem to solve some other problems. The idea is to set up a flow network that can be used to solve a given problem, obtain a maximum flow of the network and use this maximum flow to obtain an (optimal) solution to the original problem. Of course we have to prove that the solution obtained this way is indeed correct (or optimal).

In general, we follow the following steps:

1. Construct the flow network.
2. Specify the algorithm to find the maximum flow in the network (Ford-Fulkerson algorithm or other algorithms to be mentioned later). Describe the output.
3. Construct a solution to the original problem from the maximum flow.
4. Argue that the above solution is correct/optimal.

Running time: The running time is the total running time of steps 1, 2 and 3.

3.3 Bipartite Matching Problem [K&T 7.5]

Consider the following scenario: There are n girls and n boys in a dancing party. Each girl has a list of boys that she wants to dance with. A girl is "matched" if she is paired up with a boy on her list. The problem is to pair up boys and girls so that as many girls are matched as possible.

The problem is called Bipartite Matching because it is essentially a graph theoretic problem. Here a graph is bipartite if the set of vertices can be partitioned into two disjoint subsets A, B such that every edge has an endpoint in each subset. Also, a matching is a set of edges that do not share any common endpoint. The size of a matching is the number of edges in the matching.

The Bipartite Matching problem is, given a bipartite graph, to find a matching of maximum size. For the dancing scenario above, we can represent each girl by a node a_i and each boy by a node b_j ($1 \le i \le n, 1 \le j \le n$) and connect a_i to b_j by an edge if b_j is on a_i's list. (Generally, in the Bipartite Matching problem we do not require the two subsets A, B to have equal number of vertices.)

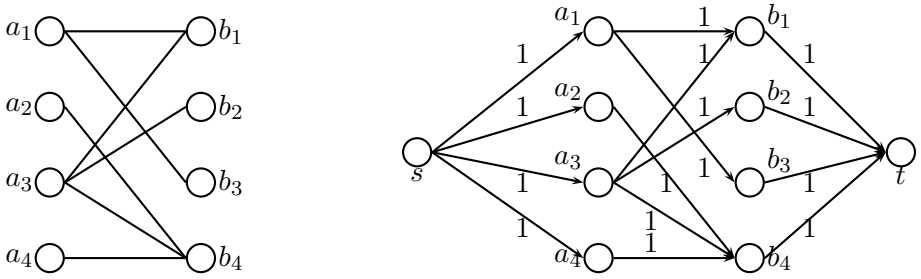

Figure 3: Left: A bipartite graph. Right: The corresponding flow network.

An example of the Bipartite Matching problem is depicted on the left in Figure 3 above. There are four girls represented by a_1, a_2, a_3, a_4 and four boys b_1, b_2, b_3, b_4.

To solve the Bipartite Matching problem with undirected graph $G = (V, E)$, we set up a flow network as follows: the directed graph $G' = (V', E')$ where

$$V' = V \cup \{s, t\}$$
$$E' = \{(s, a_i) \ : \ 1 \leq i \leq n\} \cup \{(b_j, t) \ : \ 1 \leq j \leq n\} \cup \{(a_i, b_j) \ : \ (a_i, b_j) \in E\}$$

(In other words, G' is obtained from G by adding a source s, a target t, an edge (s, a_i) for every a_i, an edge (b_j, t) for every b_j, and directing all edges in G to go from a_i to b_j.) The capacity of each edge is 1.

The flow network on the right in Figure 3 corresponds to the bipartite graph given on the left.

Now a maximum flow algorithm (such as Ford-Fulkerson, will be studied later) return a maximum flow f_{max} where the flow on each edge is either 0 or 1 (because it is a nonnegative integer ≤ 1).

A matching is constructed using f_{max} as follows:

$$\text{match } a_i \text{ with } b_j \text{ if and only if } f_{max}((a_i, b_j)) = 1.$$

We have to prove that we actually obtain a matching, and that the matching is of maximum size.

First, suppose that $f_{max}((a_i, b_j)) = 1$, then since the incoming flow to a_i is atmost 1, for all $k \neq j$ we have $f_{max}((a_i, b_k)) = 0$. So each a_i is matched with at most one b_j. By similar arguments, each b_j is matched with atmost one a_i. Thus the output of the above algorithm is indeed a matching.

Next we show that the matching obtained above has maximum size. For this we show

Claim The size of any matching is bounded above by the maximum flow value.

Then, since the above matching has size $Value(f_{max})$ its size must be maximum.

To prove the claim, we proceed by defining for each matching M (whose size is $|M|$) a flow f of value $Value(f) = |M|$.

The flow f is constructed as follows: We let $f((a_i, b_j)) = 1$ if and only if a_i is matched to b_j in M. Also, let $f((s, a_i)) = 1$ if a_i is matched to some b_j, and $f((b_j, t)) = 1$ if b_j is matched to some a_i.

We need to verify that f is a flow: Here we have to check that the Capacity Condition and Conservation Condition hold. The Capacity condition holds because each edge has capacity 1 and we have set the flow to atmost 1. The Conservation Condition can be verified by considering for each a_i (respectively b_j) whether a_i (resp. b_j) is matched. If a_i is matched, then both the incoming flow and outgoing flow are one. If a_i is not matched then both flows are 0. Similarly for b_j.

Finally, $Value(f) = |M|$ because the $f^{out}(s)$ is the number of a_i in the matching.