ICS 260—Fall, 2000–Class Notes 3: Network Flows: Intro. and Applications, Ford-Fulkerson Algorithm

Notes Originally Developed by Sandy Irani
Modified by Michael B. Dillencourt

These notes are based on material from [CLR Sections 27.1 and 27.2] and [AMO, Section 6.2].

1 Definitions

A *flow network* consists of a graph $G = (V, E)$ such that each edge $(u, v) \in E$ has a non-negative capacity $c(u, v) \geq 0$. If (u, v) is not an edge, we can assume that $c(u, v) = 0$. There are two distinguished vertices, the *source* s and the *sink* t.

Let $G = (V, E)$ be a flow network with capacity function c. A *flow* in G is a function $f : V \times V \rightarrow \mathbf{R}$ such that

1. *Capacity Constraints:* For all $u, v \in V$, $f(u, v) \leq c(u, v)$.
2. *Symmetry:* For all $u, v \in V$, $f(u, v) = -f(v, u)$.
3. *Flow Conservation:* For all $u \in V - \{s, t\}$, $\sum_{v \in V} f(u, v) = 0$.

$f(u, v)$ is the *net flow* from u to v. The *value* of the flow is $\sum_{u \in V} f(s, u)$. We will sometimes denote the value of flow f by $|f|$.

The *maximum flow problem* is given a flow network to find the flow with the maximum value from s to t.

Notice that there is no flow from u to v unless (u, v) or (v, u) is an edge in the graph.

The *positive net flow* entering a vertex u is defined as

$$\sum_{v \in V f(v,u) > 0} f(v, u).$$

The positive net flow leaving a vertex is defined symetrically.

- Example.

2 Cuts in Flow Networks

An $s - t$ cut is a partition of the nodes (S, T), $S \subseteq V$, $s \in S$, $T = V - S$, $t \in T$.

The *value of the flow across the cut*, denoted by $f(S, T)$, is defined to be $\sum_{u \in S, v \in T} f(u, v)$.

The *capacity of the cut*, denoted $c(S, T)$, is defined to be $\sum_{u \in S, v \in T} c(u, v)$.

The Min Flow Max Cut Theorem which we will prove later states that the minimum value cut is equal to the maximum value flow.

3 Applications

[AMO, Section 6.2]

3.1 Scheduling on Uniform Parallel Machines

We are given a set of jobs J to schedule on a set of m identical machines. Each job has a processing requirement p_j, a release time r_j and a deadline d_j. A job cannot be started before its release time and must be completed before its deadline. A machine can only work on one job at a time and a job can only be worked on by one machine at a time. However, preemptions are allowed. That is, we can suspend work on a job by one machine and continue working on that job on another machine at some later point in time.

We are just trying to solve the *feasibility* problem. That is, we want to know whether it is possible to complete all jobs by their deadines. This can be used as a subroutine for other problems, like determining the schedule in which the maximum lateness over all jobs is minimized, or determining the schedule with the minimum completion or the maximum utilization.

We will transform this into a flow problem as follows: sort all the release and deadline times. Determine all the mutually disjoint intervals between these milestones. Let $T_{k,l}$ denote the interval that starts at the beginning of time unit k and ends at the end of time unit l.

For example, we will denote each job by a triple $\langle p_j, r_j, d_j \rangle$. Suppose that we have jobs

$$\langle 1.5, 3, 5 \rangle, \langle 1.25, 1, 4 \rangle, \langle 2.1, 3, 7 \rangle, \langle 3.6, 5, 9 \rangle.$$

The ordered list of milestones is $1, 3, 4, 5, 7, 9$. This gives us the following set of intervals: $T_{1,2}, T_{3,3}, T_{4,4}, T_{5,6}, T_{7,8}$

Now we have a network G with a designated source node s, a designated sink node t, a node for every job and a node for every interval. There is an arc from s to every job node j with capacity p_j. There is an arc from every interval node $T_{k,l}$ to t with capacity $(l-k+1)m$. Also, for every job j and every interval $T_{k,l}$ such that $r_j \leq k$ and $d_j > l$, there will be an arce from the node representing j to the node representing $T_{k,l}$ with capacity $l-k+1$.

There is a one to one correspondence between flows with value $\sum_j p_j$ and feasible schedules.

3.2 Tanker Scheduling Problem

This application involves the *minimum flow problem* which is very closely related to the maximum flow problem. In the min flow problem there is a lower bound and an upper bound (capacity) on the flow which must be sent through each arc. The goal is to find the flow of minimum value which satisfies these constraints.

We will solve the following scheduling problem using an algorithm which solves the minimum flow problem. In the tanker scheduling problem, perishable goods must be delivered to various locations on particular days. Since the goods are perishable, they can not arrive late or early. Each delivery has a port of origin and a destination port. Shipment transit times and return times are known between each pair of ports. Thus, we can determine for each pair of jobs, whether it is possible to perform the delivery for the two jobs using the same ship.

The goal is to find a way to make all deliveries using the minimum number of ships.

Consider the graph where there is a node for every shipment and an edge from shipment i to shipment j if shipment i can be delivered before shipment j by the same tanker. Any path in this network denotes a feasible schedule for one ship.

Now we change the network into the framwork for a min flow problem. First add a source node s and a sink node t, We split each job node i into two nodes i' and i''. There is an edge (i', i'') with a lower bound of 1 for the flow which must go from i' to i''. There is an arc from s to every node i'. There is an arc from every node i'' to t. All arcs in the network are given a capacity of 1.

The flow with minimum value corresponds to the schedule which uses the minimum number of ships.

4 Cuts in Flow Networks

An $s-t$ cut is a partition of the nodes (S, T), $S \subseteq V$, $s \in S$, $T = V - S$, $t \in T$.

The *value of the flow across the cut*, denoted by $f(S, T)$, is defined to be $\sum_{u \in S, v \in T} f(u, v)$. We will extend this definition to any sets $X, Y \subseteq V$: $f(X, Y) = \sum_{x \in X, y \in Y} f(x, y)$.

The *capacity of the cut*, denoted $c(S, T)$, is defined to be $\sum_{u \in S, v \in T} c(u, v)$.

The first lemma that we will use says that the value of any flow is equal to the value of the flow across any (S, T) cut.

Lemma 1 *Consider network G with source s and sink t. Let f be a flow on G and let (S, T) be an $s-t$ cut. Then $|f| = f(S, T)$.*

Proof.

$$|f| = f(s, V) = f(s, V) + f(S - \{s\}, V) = f(S, V) = f(S, V) - f(S, S) = f(S, T).$$

■

The next lemma says that the value of any flow is at most the capacity of any $s-t$ cut.

Lemma 2 *Consider network G with source s and sink t. Let f be a flow on G and let (S, T) be an $s-t$ cut. Then $|f| \leq c(S, T)$.*

Proof.

$$|f| = f(S,T) = \sum_{u\in S}\sum_{v\in T} f(u,v) \le \sum_{u\in S}\sum_{v\in T} c(u,v) = c(S,T).$$

■

The Max-Flow Min-Cut Theorem which we prove below says that the inequality in Lemma 2 is actually an equality.

5 Residual Networks

For an edge (u,v) with flow $f(u,v)$ and capacity $c(u,v)$, we define residual capacities as $c_f = c - f$.

- If the flow along (u,v) is positive, then this is additional forward capacity, or the amount of additional flow the edge can carry.
- If the flow from u to v is negative (i.e. flow is going from v to u), it is the amount by which the flow from v to u can be reduced.

Note that for every edge e, $c_f(e) \ge 0$. If $c_f(e) = 0$, then we remove that edge from G_f.

- Example.

The *residual network* with respect to G and f, denoted by G_f is the network with capacities c_f. Residual networks are important because they capture the amount by which the current flow can be increased. In particular, any flow f on a network can be augmented by any flow found in G_f. This is made more precise in the following lemma:

Lemma 3 *Let f be a flow on network G. Let f' be a flow on network G_f. Then $f + f'$ is a flow on network G whose value is $|f|$.*

Proof. Left as an exercise. ■

The first class of algorithms which we will consider starts with a zero flow and keeps augmenting the current flow by finding a flow in the residual graph. The flow in the residual graph has a simple form in that it is always a simple path. This path will be called the *augmenting path*. Thus, we can take any path from s to t in G_f and push as much flow as possible along this path and this will be our augmenting path.

Thus, if we define $c_f(p) = \min_{e\in p} c_f(e)$, The following is a valid flow in G_f:

$$f'(u,v) = \begin{cases} c_f(p) & \text{if } (u,v) \in p \\ -c_f(p) & \text{if } (v,u) \in p \\ 0 & \text{otherwise} \end{cases}$$

- Example continued.

Theorem 1 *If f is a flow on network G, with source s and sink t, then the following conditions are equivalent:*

1. *f is a maximum flow in G.*
2. *The residual network G_f contains no augmenting path.*
3. *$|f| = c(S,T)$ for some cut (S,T) of G*

Proof. (1) $\Rightarrow$ (2): Suppose by contradiction that f is a maximum flow and there is a path from s to t in the residual network. If there is a path p from s to t in G_f, then there is a flow f' in G_f which can be obtained by pushing $\min_{e \in p} c_f(e)$ units of flow along p. By definition $c_f(e) > 0$ for all edges e in G_f which means that $|f'| > 0$. Thus, by lemma 3, we can augment the flow in G by $|f'|$.

(2) $\Rightarrow$ (3): Let S be all the nodes that are reachable from s. Let $T = V - S$. Since t can not be reached from s, it must be the case that $t \in T$ and that there are no edges from any vertex in S to any vertex in T in G_f. This means that for each edge $(u,v) \in G$ such that $u \in S$ and $v \in T$, $c(u,v) = f(u,v)$. Thus, $|f| = f(S,T) = C(S,T)$.

(3) $\Rightarrow$ (1): There can not be a flow f such that $|f| = f(S,T) > c(S,T)$ due to Lemma 2. Thus, if $|f| = c(S,T)$, then f is a flow with maximum value.

■

6 The Ford-Fulkerson Algorithm

The basic Ford-Fulkerson algorithm attempts to increase the flow along any augmenting path until no augmenting path remains:

```
Algorithm FordFulkerson(G, s, t):
Initialization:
for each edge (u, v) ∈ E
      f(u, v) ← 0
      f(v, u) ← 0

Main Loop:
while there is a path p from s to t in G_f
      c_f(p) ← min{c_f(e) | e ∈ p}
      for each edge (u, v) ∈ E
            f(u, v) ← f(u, v) + c_f(p)
            f(v, u) ← −f(u, v)
```

The fact that the Ford-Fulkerson algorithm works follows from the Max-Flow Min-Cut Theorem.

6.1 Running time of Ford-Fulkerson

Let us first analyze the Ford-Fulkerson algorithm when the capacities are integers. In this case, the value of the flow along any augmenting path will be at least 1. Thus, we are guaranteed that in each iteration, the value of the flow increases by at least 1. Thus, the number of iterations is bounded by the value of the maximum flow. Since the value of the maximum flow is bounded by thevalue of the minimum cut, we know that this can be at most mc_{max}, where c_{max} is the maximum capacity of any edge. This is because there can be at most m edges crossing the cut and each such edge has capacity at most c_{max}. Each iteration of For-Fulkerson takes $O(m)$ time. Thus, the running time is bounded by $O(m^2 c_{max})$.

The Ford-Fulkerson algorithm can take as much time as $\Theta(mf^*)$, where f^* is the value of the maximum flow.

- Example

There have been many variations of this algorithm which select the augmenting path more carefully so that better bounds can be proven for the running time.