

Lecture 4: Network Flow

Instructor: Dieter van Melkebeek

Scribe: Greg Quinn, Nolan Salzmann

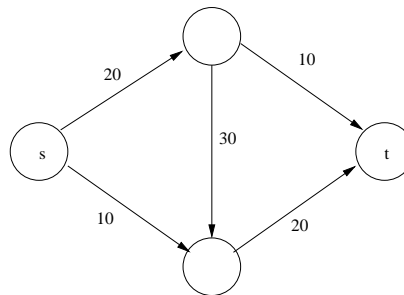
This lecture provides an overview of concepts that would typically be discussed in an undergraduate level treatment of network flow. We first review definitions and concepts relevant to the problem of maximizing flow in a network. We then build towards the Ford-Fulkerson scheme for maximizing flow. The important issues of partial correctness and termination for this scheme are discussed in some detail. This lecture concludes with a presentation of some perhaps unexpected applications of network flow.

1 Network Flow Concepts

This portion of lecture covers the fundamental concepts in discussing network flow problems. In particular, we define *network*, *flow*, and *cut*; we also present some useful basic results.

Definition 1 (Network). A network is a directed graph, $G = (V, E)$. A capacity function, $c : E \mapsto [0, \infty)$, maps each edge to how much “traffic” it can carry. Two vertices in the graph, s and t , are distinguished as the source and sink, respectively. The source vertex has only outgoing edges and the sink has only incoming edges.

Example: The following figure shows a sample network. The capacities are shown next to each edge and the source and sink vertices are labeled as s and t .



⊠

Definition 2 (Flow). Given a network $G = (V, E)$, a flow is a function $f : E \mapsto [0, \infty)$ such that:

1. $\forall e \in E : f(e) \leq c(e)$
2. $\forall v \in V \setminus \{s, t\} : \sum_{(u,v) \in E} f(u, v) = \sum_{(v,u) \in E} f(v, u)$

A flow function f maps each edge to the amount of traffic it carries. Requirement 1 in the preceding definition states that each edge must not carry any more than its capacity. Requirement 2 is a statement of conservation; all vertices except the source and sink must have equal inflow and outflow. The source is allowed to have net outflow, and the sink may have net inflow.

An intuitive way to think of a flow is as specifying some way of traffic being routed from the source vertex to the sink vertex according to the capacity of the network. The value, ν , of a flow quantifies the amount of traffic being routed from source to sink. One way to formalize this is as the total amount flowing on the outgoing edges of the source:

$$\nu = \sum_{(s,u) \in E} f(s,u).$$

The last concept we introduce in this part of lecture is the cut.

Definition 3 (Cut). A cut of a network is any partition (S, T) of the vertices into two sets such that $s \in S$ and $t \in T$.

The idea of capacity can be extended from single edges to a cut. In particular, the capacity of a cut is defined as:

$$c(S, T) = \sum_{(u,v) \in E, u \in S, v \in T} c(u, v)$$

Note that given any cut (S, T) and any flow f , the value of the flow is equal to the net amount of flow from S into T of the given cut.

2 Weak Duality

A simple observation regarding network flow leads to the idea of weak duality. Here we make this observation and also present strong duality, which will be proven later. Note that these concepts relate strongly to duality in linear programming and will be revisited later in the course.

With regard to the structures presented previously, the following observation may be made:

Observation 1 Given any flow f and any cut (S, T) on the same network, the value of the flow cannot exceed the capacity of the cut:

$$\nu(f) \leq c(S, T)$$

This observation follows from the fact that the value of flow f is equal to the amount of flow carried by edges that cross from S into T minus the amount carried by edges from T into S :

$$\nu(f) = \sum_{(u,v) \in E, u \in S, v \in T} f(u, v) - \sum_{(u,v) \in E, u \in T, v \in S} f(u, v)$$

Observation 1 is true because the first term in the above equation is upper bounded by the capacity of cut (S, T) and the second term is nonnegative. Therefore, notice that the following two conditions must hold in order for the strong duality equality to be applicable to a particular flow and cut:

1. No “backward” edges from T into S may carry any traffic.
2. All “forward” edges from S into T must be fully utilized.

Since Observation 1 is true for any flow and any cut, the minimum value of the the capacity over all cuts in the network provides an upper bound for the value of any flow. This relationship is known as *weak duality* for network flow problems:

$$\text{MAXFLOW} \leq \text{MINCUT}$$

We later prove the condition of *strong duality*:

$$\text{MAXFLOW} = \text{MINCUT}$$

3 Ford-Fulkerson Scheme

Toward our goal of creating an algorithm for maximizing network flow, we present two additional concepts in order to create an iterative process to continually improve our results: the *residual network* and *augmenting path*.

Definition 4 (Residual network). Given a flow f on graph $G = (V, E)$, the residual network G_f is a graph on the same vertices but with a possibly different edge set. That edge set can contain both edges from the original network G as well as reverse edges representing traffic in f that may be undone. Specifically,

$$G_f = (V, E_f)$$

where

$$E_f = \{e \in E \mid f(e) < c(e)\} \cup \{e \in E^{-1} \mid f(e) > 0\}$$

The capacities of the residual graph edges are calculated by subtracting the flow on an edge from the capacity of that edge in G (if that edge is in E), or by the flow of e^{-1} if e is the reverse of an edge in E .

$$c_f(e) = \begin{cases} c(e) - f(e) & e \in E_f \cap E \\ f(e^{-1}) & e^{-1} \in E_f^{-1} \cap E \end{cases}$$

(In short, whenever an edge in G bears positive flow, G_f will include a reverse edge for which the capacity equals the amount of that flow.)

Definition 5 (Augmenting path). Given a residual network G_f with source s and sink t , an *augmenting path* P is a path from s to t in G_f .

The edges in a residual network either indicate flow that is still under an original edge's capacity, or flow that has already been used. The existence of an augmenting path indicates that more flow can be achieved without violating capacity limitations by increasing traffic pushed along edges with capacity remaining and/or decreasing traffic along a currently used edge.

The idea of the Ford-Fulkerson scheme is to keep finding augmenting paths, using each to its capacity, recalculating the residual network, and repeating until no more augmenting paths exist. The scheme works as follows:

FORD-FULKERSON SCHEME($G = (V, E), s, v \in V$)

$f \leftarrow 0$

$G_f \leftarrow G$ (since the residual network is the original when there is no flow)

While there exists an augmenting path P in G_f :

$f \leftarrow f + f_P$ (where f_P is the maximal additional flow that can be pushed through P)

For all $e \in P$:

$f(e) \leftarrow f(e) + f_P$

$c_f(e) \leftarrow c_f(e) - f_P$

$c_f(e^{-1}) \leftarrow c_f(e^{-1}) + f_P$

Naturally, two questions arise about this scheme: “Does it halt?” and “If it does halt, will it return the maximum flow?”. We will start by proving the second. This will provide proof of partial correctness.

Theorem 1. *The following are equivalent for any flow f :*

- (1) $\nu(f)$ is a maximum.
- (2) There is no augmenting path in G_f .
- (3) There exists a cut (S, T) in G of capacity $\nu(f)$.

Proof. We argue using cyclic implications.

(1) \Rightarrow (2). As shown, if there is an augmenting path P in G_f , then there is as yet unused capacity in all the edges of P , and we can push flow into these edges, increasing our maximum. Hence, the presence of an augmenting path implies that f is not maximal. The contrapositive proves our claim.

(2) \Rightarrow (3). If there is no augmenting path from s to t in G_f , then there is a group of points that can be reached from s , but from which we cannot reach any new points. Any augmenting path is blocked within these points, as their outward edges in G are used to capacity and any inward edges in G are carrying no flow. Define the set of reachable points as S in cut (S, T) . The capacity of this cut in G is the sum of all outgoing edge capacities, which equals our flow.

$$S = \{u \mid u \text{ is reachable from } s \text{ in } G_f\}$$

By the fact that there is no augmenting path in G_f , $t \notin S$.

$$f(u, v) = \begin{cases} c(u, v) & u \in S, v \in T, (u, v) \in E \\ 0 & u \in T, v \in S, (u, v) \in E \end{cases}$$

$$\nu(f) = c(S, T)$$

(3) \Rightarrow (1). This can be proven by simple application of weak duality, $MAXFLOW \leq MINCUT$. Since our flow value, $\nu(f)$ is equal to the capacity of some cut by (3), we know that it is maximal. \square

Two corollaries are implied in the proof of the above theorem.

Corollary 1 (Strong duality). *The relationship*

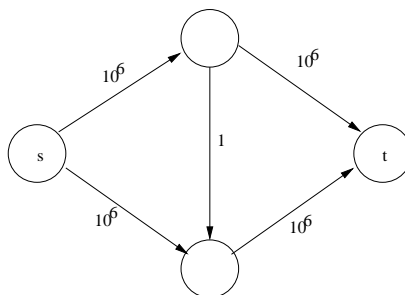
$$MAXFLOW = MINCUT$$

holds for network flow problems.

Corollary 2 (Partial correctness of Ford-Fulkerson). *If the Ford-Fulkerson halts, it returns a maximal flow.*

So we now know that if the scheme halts, we will get a maximum flow. We now turn to the question of whether the scheme will actually halt, as desired. Observe that if all the capacities in G are integer-valued, then f will also be integer valued, and the Ford-Fulkerson scheme will increase the flow by a minimum of 1 at each iteration. Hence, we must have a number of iterations bounded by the maximal flow of the network. Therefore in this case, the scheme does halt.

However, the maximal flow is not a desirable bound, as in the case of a network like the one shown below. If the scheme repeatedly makes a poor choice for the augmenting path, it can improve the flow by only one at each iteration. If the maximal flow were a value like two million as shown, what could simply be a two iteration job may be 6 orders of magnitude worse!



This implies that we need to consider more carefully how the augmenting paths are chosen. Note that we do not call the Ford-Fulkerson scheme an algorithm because it does not specify how augmenting paths are chosen. We present two alternatives for choosing the augmenting path here.

- *Alternative 1: Largest Capacity*

One alternative is to choose the augmenting path for which we can increase our flow by a maximal amount. We know that if the network is given as an adjacency matrix, this choice can be made in linear time (it is left as an exercise to show how). This method causes the Ford-Fulkerson scheme to run for $O(m \log(C))$ iterations (assuming that all capacities are integral), where $C = \sum_{e \in E} c(e)$.

- *Alternative 2: Shortest Path*

Another alternative is to choose the augmenting path with the least number of edges. This can also be done in linear time, using a breadth-first search. The number of iterations for the scheme using this method is $O(mn)$, and the scheme will halt even with non-integer capacities in the network.

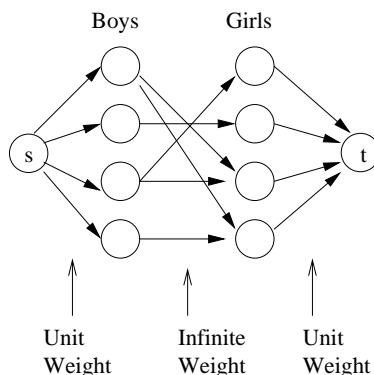
There are alternative approaches to solving network flow problems, and some are more efficient than the augmenting path solution presented here on particular classes of inputs. We still do not know of anything close to linear time, however.

4 Applications of Network Flow

This section presents some interesting ways of applying the network flow approach. The first, bipartite matching, is discussed in detail here, while a couple of others will be discussed in the next lecture.

4.1 Bipartite Matching

The problem of matching n boys and n girls into pairs in which the boy and girl both like each other can be reduced to a network flow problem if the set of boys B and the set of girls G are graphed so that each edge connects a boy b to girl g with infinite weight if they like each other (unit weights will actually be fine for this application, but we'll see one later in which infinite weights are more appropriate). We need to create a source s with unit-weight edges to all boys, and a sink t with unit-weight edges from all girls. An example of such a network is shown below.



Claim 1. *There is a one-to-one correspondence between integer flows in a graph constructed in the above manner and matchings in the original problem. The value of the flow corresponds to the number of pairs matched.*

Now, consider a cut in the solution network. If $c(S, T)$ is finite, no infinite-weight edge may be cut. Hence, for each boy $b \in S$, all girls he likes must also be in S . So, the capacity of our cut is the sum of all boys in T and girls in S (as each represents a unit-weight edge that was cut).

That is,

$$c(S, T) = |B \cap T| + |G \cap S|.$$

Now, observe that

$$|B \cap T| = n - |B \cap S|$$

(since there are exactly n boys), and

$$|G \cap S| \geq |\Gamma(B \cap S)|,$$

where Γ means “every girl liked by a boy in this set” (since otherwise the cut would have infinite capacity).

Now, suppose that $n > c(S, T)$ (i.e., no perfect matching is possible). Then, it must be true that $|\Gamma(B \cap S)| < |B \cap S|$ —that is, the set of girls liked by a boy in S is smaller than the set of boys in S . Clearly there can be no perfect matching in such a case!

In this case the MINCUT problem is equivalent to the minimum vertex cover problem. Although there is no efficient solution to this problem in general, it can be solved efficiently for bipartite graphs.

4.2 Project Selection

Consider a set of projects, each with a corresponding benefit and a set of tools, each with a corresponding cost. We wish to maximize our profit by choosing a set of projects to work on so as to minimize the cost of the required tools. This problem may be solved as a min cut problem, and will be discussed in the next lecture.

4.3 Image Segmentation

The problem of separating an image into its background and foreground components can also be solved using min cut, and will be covered in the next lecture.