4.3. Network Flow Problems

Consider a network of pipes where valves allow flow in only one direction. Each pipe has a capacity per unit time. We model this with a vertex for each junction and a (directed) edge for each pipe, weighted by the capacity. We also assume that flow cannot accumulate at a junction. Given two locations s, t in the network, we may ask "what is the maximum flow (per unit time) from s to t?"

This question arises in many contexts. The network may represent roads with traffic capacities, or links in a computer network with data transmission capacities, or currents in an electrical network. There are applications in industrial settings and to combinatorial min-max theorems. The seminal book on the subject is Ford–Fulkerson [1962]. More recently, Ahuja–Magnanti–Orlin [1993] presents a thorough treatment of network flow problems.

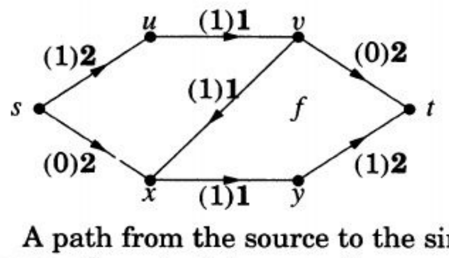
4.3.1. Definition. A **network** is a digraph with a nonnegative **capacity** $c(e)$ on each edge e and a distinguished **source vertex** s and **sink vertex** t. Vertices are also called **nodes**. A **flow** f assigns a value $f(e)$ to each edge e. We write $f^+(v)$ for the total flow on edges leaving v and $f^-(v)$ for the total flow on edges entering v. A flow is **feasible** if it satisfies the **capacity constraints** $0 \le f(e) \le c(e)$ for each edge and the **conservation constraints** $f^+(v) = f^-(v)$ for each node $v \notin \{s, t\}$.

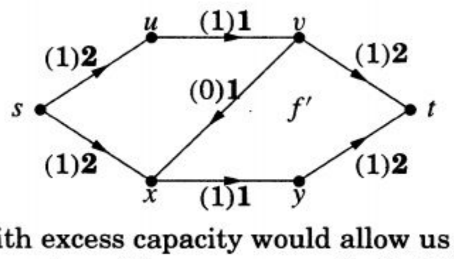
MAXIMUM NETWORK FLOW

We consider first the problem of maximizing the net flow into the sink.

4.3.2. Definition. The **value** val(f) of a flow f is the net flow $f^-(t) - f^+(t)$ into the sink. A **maximum flow** is a feasible flow of maximum value.

4.3.3. Example. The **zero flow** assigns flow 0 to each edge; this is feasible. In the network below we illustrate a nonzero feasible flow. Each capacities are shown in bold, flow values in parentheses. Our flow f assigns $f(sx) = f(vt) = 0$, and $f(e) = 1$ for every other edge e. This is a feasible flow of value 1.





A path from the source to the sink with excess capacity would allow us to increase flow. In this example, no path remains with excess capacity, but the flow f' with $f'(vx) = 0$ and $f'(e) = 1$ for $e \neq vx$ has value 2. The flow f is "maximal" in that no other feasible flow can be found by increasing the flow on some edges, but f is not a maximum flow.

We need a more general way to increase flow. In addition to traveling forward along edges with excess capacity, we allow traveling backward (against the arrow) along edges where the flow is nonzero. In this example, we can travel from s to x to v to t. Increasing the flow by 1 on sx and vt and decreasing it by one on vx changes f into f'. ■

4.3.4. Definition. When f is a feasible flow in a network N, an f**-augmenting path** is a source-to-sink path P in the underlying graph G such that for each $e \in E(P)$,

a) if P follows e in the forward direction, then $f(e) < c(e)$.

b) if P follows e in the backward direction, then $f(e) > 0$.

Let $\epsilon(e) = c(e) - f(e)$ when e is forward on P, and let $\epsilon(e) = f(e)$ when e is backward on P. The **tolerance** of P is $\min_{e \in E(P)} \epsilon(e)$.

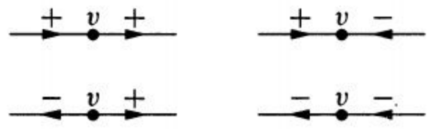
As in Example 4.3.3, an f-augmenting path leads to a flow with larger value. The definition of f-augmenting path ensures that the tolerance is positive; this amount is the increase in the flow value.

4.3.5. Lemma. If P is an f-augmenting path with tolerance z, then changing flow by $+z$ on edges followed forward by P and by $-z$ on edges followed backward by P produces a feasible flow f' with $\mathrm{val}(f') = \mathrm{val}(f) + z$.

Proof: The definition of tolerance ensures that $0 \leq f'(e) \leq c(e)$ for every edge e, so the capacity constraints hold. For the conservation constraints we need only check vertices of P, since flow elsewhere has not changed.

The edges of P incident to an internal vertex v of P occur in one of the four ways shown below. In each case, the change to the flow out of v is the same as the change to the flow into v, so the net flow out of v remains 0 in f'.

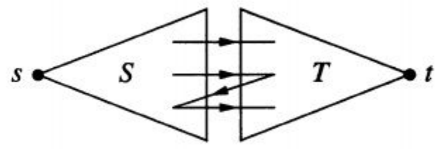
Finally, the net flow into the sink t increases by z. ■



The flow on backward edges did not disappear; it was redirected. In effect, the augmentation in Example 4.3.3 cuts the flow path and extends each portion to become a new flow path. We will soon describe an algorithm to find augmenting paths.

Meanwhile, we would like a quick way to know when our present flow is a maximum flow. In Example 4.3.3, the central edges seem to form a "bottleneck"; we only have capacity 2 from the left half of the network to the right half. This observation will give us a PROOF that the flow value can be no larger.

4.3.6. Definition. In a network, a **source/sink cut** $[S, T]$ consists of the edges from a **source set** S to a **sink set** T, where S and T partition the set of nodes, with $s \in S$ and $t \in T$. The **capacity** of the cut $[S, T]$, written $\text{cap}(S, T)$, is the total of the capacities on the edges of $[S, T]$.



Keep in mind that in a digraph $[S, T]$ denotes the set of edges with tail in S and head in T. Thus the capacity of a cut $[S, T]$ is completely unaffected by edges from T to S.

Given a cut $[S, T]$, every s, t-path uses at least one edge of $[S, T]$, so intuition suggests that the value of a feasible flow should be bounded by $\text{cap}(S, T)$. To make this precise, we extend the notion of net flow to sets of nodes. Let $f^+(U)$ denote the total flow on edges leaving U, and let $f^-(U)$ be the total flow on edges entering U. The net flow out of U is then $f^+(U) - f^-(U)$.

4.3.7. Lemma. If U is a set of nodes in a network, then the net flow out of U is the sum of the net flows out of the nodes in U. In particular, if f is a feasible flow and $[S, T]$ is a source/sink cut, then the net flow out of S and net flow into T equal $\text{val}(f)$.

Proof: The stated claim is that

$$f^+(U) - f^-(U) = \textstyle\sum_{v \in U} [f^+(v) - f^-(v)].$$

We consider the contribution of the flow $f(xy)$ on an edge xy to each side of the formula. If $x, y \in U$, then $f(xy)$ is not counted on the left, but it contributes positively (via $f^+(x)$) and negatively (via $f^-(y)$) on the right. If $x, y \notin U$, then $f(xy)$ contributes to neither sum. If $xy \in [U, \overline{U}]$, then it contributes positively to each sum. If $xy \in [\overline{U}, U]$, then it contributes negatively to each sum. Summing over all edges yields the equality.

When $[S, T]$ is a source/sink cut and f is a feasible flow, net flow from nodes of S sums to $f^+(s) - f^-(s)$, and net flow from nodes of T sums to $f^+(t) - f^-(t)$, which equals $-\text{val}(f)$. Hence the net flow across any source/sink cut equals both the net flow out of s and the net flow into t. ∎

4.3.8. Corollary. (Weak duality) If f is a feasible flow and $[S, T]$ is a source/sink cut, then $\text{val}(f) \le \text{cap}(S, T)$.

Proof: By the lemma, the value of f equals the net flow out of S. Thus

$$\text{val}(f) = f^+(S) - f^-(S) \le f^+(S),$$

since the flow into S is no less than 0. Since the capacity constraints require $f^+(S) \le \text{cap}(S, T)$, we obtain $\text{val}(f) \le \text{cap}(S, T)$. ∎

Among source/sink cuts, one with minimum capacity yields the best bound on the value of a flow. This defines the **minimum cut** problem. The max flow and min cut problems on a network are dual optimization problems.[†] Given a flow with value α and a cut with value α, the duality inequality in Corollary 4.3.8 PROVES that the cut is a minimum cut and the flow is a maximum flow.

If every instance has solutions with the same value to both the max problem and the min problem ("strong duality"), then a short proof of optimality always exists. This does not hold for all dual pairs of problems (recall matching and covering in general graphs), but it holds for max flow and min cut.

The Ford–Fulkerson algorithm seeks an augmenting path to increase the flow value. If it does not find such a path, then it finds a cut with the same value (capacity) as this flow; by Corollary 4.3.8, both are optimal. If no infinite sequence of augmentations is possible, then the iteration leads to equality between the maximum flow value and the minimum cut capacity.

4.3.9. Algorithm. (Ford–Fulkerson labeling algorithm)
Input: A feasible flow f in a network.
Output: An f-augmenting path or a cut with capacity val(f).
Idea: Find the nodes reachable from s by paths with positive tolerance. Reaching t completes an f-augmenting path. During the search, R is the set of nodes labeled *Reached*, and S is the subset of R labeled *Searched*.
Initialization: $R = \{s\}$, $S = \emptyset$.
Iteration: Choose $v \in R - S$.

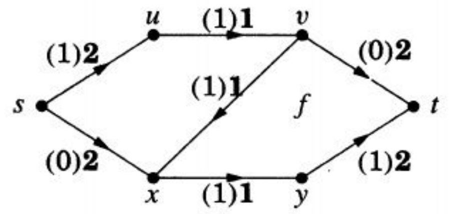
For each exiting edge vw with $f(vw) < c(vw)$ and $w \notin R$, add w to R.

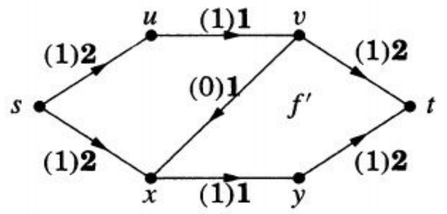
For each entering edge uv with $f(uv > 0)$ and $u \notin R$, add u to R.

Label each vertex added to R as "reached", and record v as the vertex reaching it. After exploring all edges at v, add v to S.

If the sink t has been reached (put in R), then trace the path reaching t to report an f-augmenting path and terminate. If $R = S$, then return the cut $[S, \overline{S}]$ and terminate. Otherwise, iterate. ■

4.3.10. Example. On the left below is the network of Example 4.3.3 with the flow f. We run the labeling algorithm. First we search from s and find excess capacity to u and x, labeling them reached. Now we have $u, v \in R - S$. There is no excess capacity on uv or xy, so searching from u reaches nothing, and also





[†]The precise notion of "dual problem" comes from linear programming. For our purposes, dual problems are a maximization problem and a minimization problem such that $a \le b$ whenever a and b are the values of feasible solutions to the max problem and min problem, respectively. See Section 8.1 for further discussion.

searching from x does not reach y. However, there is nonzero flow on vx. Thus we label v from x. Now v is the only element of $R - S$, and searching from v reaches t. We labeled t from v, v from x, and x from s, so we have found the augmenting path s, x, v, t.

The tolerance on this path is 1, so the augmentation increases the flow value by 1. In the new flow f' shown on the right, every edge has unit flow except $f'(vx) = 0$. When we run the labeling algorithm again, we have excess capacity on su and sx and can label $\{u, x\}$, but from these nodes we can label no others. We terminate with $R = S = \{s, u, x\}$. The capacity of the resulting cut $[S, \overline{S}]$ is 2, which equals val(f') and proves that f' is a maximum flow. ■

Repeated use of the labeling algorithm allows us to solve the maximum flow problem and prove the strong duality relationship.

4.3.11. Theorem. (Max-flow Min-cut Theorem—Ford and Fulkerson [1956]) In every network, the maximum value of a feasible flow equals the minimum capacity of a source/sink cut.

Proof: In the max-flow problem, the zero flow ($f(e) = 0$ for all e) is always a feasible flow and gives us a place to start. Given a feasible flow, we apply the labeling algorithm. It iteratively adds vertices to S (each vertex at most once) and terminates with $t \in R$ ("breakthrough") or with $S = R$.

In the breakthrough case, we have an f-augmenting path and increase the flow value. We then repeat the labeling algorithm. When the capacities are rational, each augmentation increases the flow by a multiple of $1/a$, where a is the least common multiple of the denominators, so after finitely many augmentations the capacity of some cut is reached. The labeling algorithm then terminates with $S = R$.

When terminating this way, we claim that $[S, T]$ is a source/sink cut with capacity val(f), where $T = \overline{S}$ and f is the present flow. It is a cut because $s \in S$ and $t \notin R = S$. Since applying the labeling algorithm to the flow f introduces no node of T into R, no edge from S to T has excess capacity, and no edge from T to S has nonzero flow in f. Hence $f^+(S) = \text{cap}(S, T)$ and $f^-(S) = 0$.

Since the net flow out of any set containing the source but not the sink is val(f), we have proved

$$\text{val}(f) = f^+(S) - f^-(S) = f^+(S) = \text{cap}(S, T). \quad ■$$

This proof of Theorem 4.3.11 requires rational capacities; otherwise, Algorithm 4.3.9 may yield augmenting paths forever! Ford and Fulkerson provided an example of this with only ten vertices (see Papadimitriou–Steiglitz [1982, p126-128]). Edmonds and Karp [1972] modified the labeling algorithm to use at most $(n^3 - n)/4$ augmentations in an n-vertex network and work for all real capacities. As in the bipartite matching problem (Theorem 3.2.22), this is done by searching always for shortest augmenting paths. Faster algorithms are now known; again we cite Ahuja–Magnanti–Orlin [1993] for a thorough discussion.

"Introduction to Graph Theory" - West – 2da. Ed.