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.

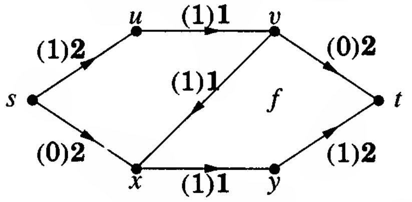
4.3.1. Definition. A **network** is a digraph with a nonnegative **capacity** $c(e)$ on each edge e and a distinguished **source vertex** s and **sink vertex** t. Vertices are also called **nodes**. A **flow** f assigns a value $f(e)$ to each edge e. We write $f^+(v)$ for the total flow on edges leaving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

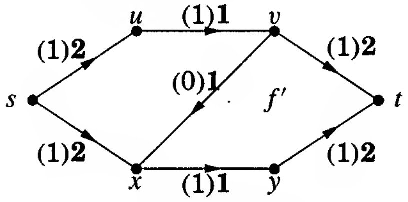
MAXIMUM NETWORK FLOW

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 $\min_{e \in E(P)} \epsilon(e)$.

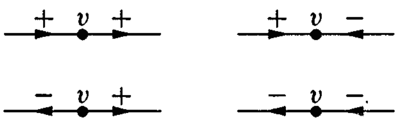
As in Example 4.3.3, an f-augmenting path leads to a flow with larger value. The definition of f-augmenting path ensures that the tolerance is positive; this amount is the increase in the flow value.

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 $\text{val}(f') = \text{val}(f) + z$.

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'.

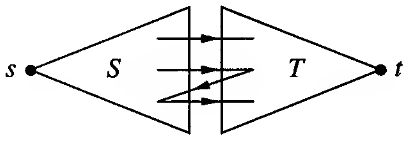
Finally, the net flow into the sink t increases by z. ■



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 $\mathrm{cap}(S, T)$, 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 $\mathrm{cap}(S, T)$. 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 $\mathrm{val}(f)$.

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 $-\mathrm{val}(f)$. 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 $\mathrm{val}(f) \le \mathrm{cap}(S, T)$.

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

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

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

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 $v \in R - S$.

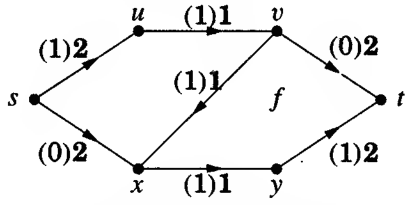
For each exiting edge vw with $f(vw) < c(vw)$ and $w \notin R$, add w to R.

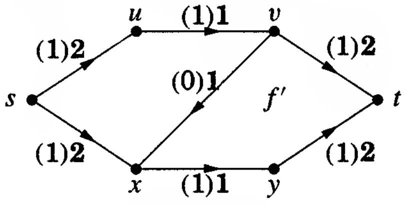
For each entering edge uv with $f(uv > 0)$ and $u \notin R$, add u to R.

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 $a \leq b$ 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

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

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.

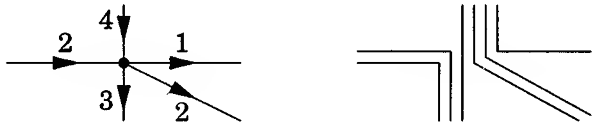
INTEGRAL FLOWS

In combinatorial applications, we typically have integer capacities and want a solution in which the flow on each edge is an integer.

4.3.12. Corollary. (Integrality Theorem) If all capacities in a network are integers, then there is a maximum flow assigning integral flow to each edge. Furthermore, some maximum flow can be partitioned into flows of unit value along paths from source to sink.

Proof: In the labeling algorithm of Ford and Fulkerson, the change in flow value when an augmenting path is found is always a flow value or the difference between a flow value and a capacity. When these are integers, the difference is also an integer. Starting with the zero flow, this implies that there is no first time when a noninteger flow appears.

The algorithm thus produces a maximum flow with integer flow on each edge. At each internal node, we now match units of entering flow to units of exiting flow. This forms s, t-paths and perhaps cycles. If a cycle arises, then we decrease flow on its edges by 1 to eliminate it without changing the flow value. This leaves val(f) paths from s to t, each corresponding to a unit of flow. ■



The integrality theorem yields paths of unit flow. In applications, we build networks where these units of flow have meaning.

The next two remarks show that the Max-flow Min-cut Theorem for networks with integer capacities is almost the same statement as Menger's Theorem for edge-disjoint paths in digraphs.

4.3.13. Remark. *Menger from Max-flow Min-cut.* When x, y are vertices in a digraph D, we can view D as a network with source x and sink y and capacity 1 on every edge. Capacity 1 ensures that units of flow from x to y correspond to pairwise edge-disjoint x, y-paths in D. Thus a flow of value k yields a set of k such paths.

Similarly, every source/sink partition S, T defines a set of edges whose deletion makes y unreachable from x: the set $[S, T]$. Since every capacity is 1, the size of this set is cap(S, T).

The paths and the edge cut we have obtained might not be optimal, but by the Max-flow Min-cut Theorem we have

$$\lambda'_D(x, y) \geq \max \text{val}(f) = \min \text{cap}(S, T) \geq \kappa'_D(x, y).$$

Since always $\kappa'(x, y) \geq \lambda'(x, y)$, equality now holds. ■

4.3.14. Remark. *Max-flow Min-cut from Menger.* To show that Menger's Theorem implies the Max-flow Min-cut Theorem for rational capacities, we take an arbitrary network and transform it into a digraph where we apply Menger's Theorem. By multiplying all capacities by the least common denominator, we may assume that the capacities are integers.

Given a network N with integer capacities, we form a digraph D by splitting each edge of capacity j into j edges with the same endpoints. For N, duality yields $\max \mathrm{val}(f) \le \min \mathrm{cap}(S, T)$. This time we want to use Menger's Theorem on D to obtain the reverse inquality, so in contrast to Remark 4.3.13 our desired computation is

$$\max \mathrm{val}(f) \ge \lambda'_D(s,t) = \kappa'_D(s,t) \ge \min \mathrm{cap}(S,T).$$

A set of $\lambda'(s,t)$ pairwise edge-disjoint s,t-paths in D collapses into a flow of value $\lambda'(s,t)$ in N, since the number of copies of each edge in D equals the capacity of the edge in N. Thus $\max \mathrm{val}(f) \ge \lambda'(s,t)$.

Now, let F be a set of $\kappa'(s,t)$ edges disconnecting t from s in D. If $e \in F$, then the minimality of F implies that $D - (F - e)$ has an s,t-path P through e. If some other copy e' of the edge $e = uv$ is not in F, then P can be rerouted along e' to obtain an s,t-path in $D - F$. Therefore, F contains all copies or no copies of each multiple edge in D. Hence $\kappa'(s,t)$ is the sum of the capacities on a set of edges that disconnects t from s in N. Letting S be the set of vertices reachable from s in $D - F$, we have $\mathrm{cap}(S,T) = \kappa'(s,t)$. The minimum cut has at most this capacity, so $\min \mathrm{cap}(S,T) \le \kappa'(s,t)$, and we have proved all the needed inequalities. ■

For combinatorial applications, Menger's Theorem may yield simpler proofs than the Max-flow Min-cut Theorem (compare Theorem 4.2.25 with ??). Nevertheless, our proof of Menger's Theorem in Section 4.2 is awkward to implement algorithmically. For large-scale computations, network flow and the Ford–Fulkerson labeling algorithm are more appropriate. Indeed, most algorithms that compute connectivity in graphs and digraphs use network flow methods (Stoer–Wagner [1994] presents a different approach).

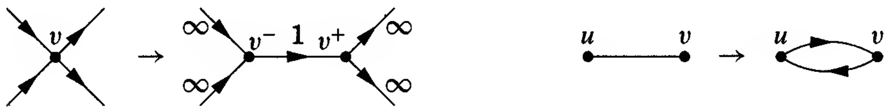
We present other network models for combinatorial problems. For example, the other local versions of Menger's Theorem can also be obtained directly.

4.3.15. Remark. *Other transformations.* For each version of Menger's Theorem, we encode the path problem using network flows with integer capacities.

To obtain a network model for the problem of internally disjoint paths in a digraph D, we must prevent two units of flow from passing through a vertex. This can be done by replacing each vertex v with two vertices v^-, v^+ that inherit the entering and exiting edges at v. By adding an edge of unit capacity from v^- to v^+, we obtain the effect of limiting flow through v to one unit. By putting very large capacity (essentially infinite) on the edges that were in D, we ensure that a minimum cut will count only edges of the form v^-v^+

To obtain a network model for the problem of edge-disjoint paths in a graph G, we must permit flow to pass either way in an edge. This can be done by replacing each edge uv with two directed edges uv and vu. When the network sends unit flow in both directions, in effect the edge is not being used at all.

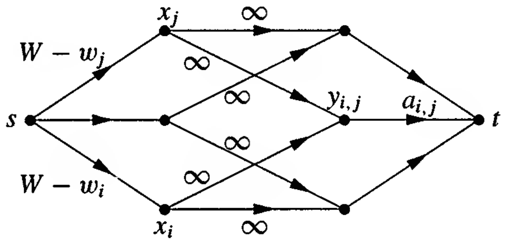
In each case, a flow in the network provides a set of paths, and a minimum cut leads to a separating set of vertices or edges. As in Remark 4.3.13, duality then gives us the desired equality in Menger's Theorem. To model the problem of internally disjoint paths in a graph, we need both of these transformations. Exercises 5–7 request the details of these proofs. ■



4.3.16. Application. *Baseball Elimination Problem* (Schwartz [1966]). At some time during the season, we may wonder whether team X can still win the championship. In other words, can winners be assigned for the remaining games so that no team ends with more victories than X? If so, then such an assignment exists with X winning all its remaining games, reaching W wins. We want to know whether winners can be chosen for other games so that no team obtains more than W wins. To test this, we create a network where units of flow correspond to the remaining games.

Let $X_1, \dots, X_n$ be the other teams. Include nodes $x_1, \dots, x_n$ for the n teams, nodes $y_{i,j}$ for the $\binom{n}{2}$ pairs of teams, and a source s and sink t. Put an edge from s to each team node and an edge from each pair node to t. Each pair node $y_{i,j}$ is entered by edges from x_i and x_j.

The capacities model the constraints. The capacity on edge $y_{i,j}t$ is $a_{i,j}$, the number of remaining games between X_i and X_j. Given that X_i has won w_i games already, the capacity on edge sx_i is $W - w_i$ to keep X in contention. The capacity on edges $x_i y_{i,j}$ and $x_j y_{i,j}$ is ∞ (the number of games x_i can win from x_j is constrained by the capacity on $y_{i,j}t$).



By the integrality theorem, a maximum flow breaks into flow units. Each unit corresponds to one game; the first edge specifies the winner, and the last edge specifies the pair. The network has a flow of value $\sum_{i,j} a_{i,j}$ if and only if *all* remaining games can be played with no team exceeding W wins; this is the condition for X remaining in contention.

By the Max-flow Min-cut Theorem, there is a flow of value $\sum a_{i,j}$ if and only if every cut has capacity at least $\sum a_{i,j}$. Let S, T be a cut with finite capacity, and let $Z = \{i\colon x_i \in T\}$. Since $c(x_i y_{i,j}) = \infty$, we cannot have $x_i \in S$ and $y_{i,j} \in T$; thus $y_{i,j} \in S$ whenever i or j is not in Z. To minimize capacity, we put $y_{i,j} \in T$ whenever $\{i, j\} \subseteq Z$. Now $\text{cap}(S, T) = \sum_{i \in Z}(W - w_i) + \sum_{\{i,j\} \not\subseteq Z} a_{i,j}$. The condition that every cut have capacity at least $\sum a_{i,j}$ becomes

$$\sum_{i \in Z}(W - w_i) \ge \sum_{\{i,j\} \subseteq Z} a_{i,j} \qquad \text{for all } Z \subseteq [n].$$

Note that this condition is obviously necessary; it states that we need enough leeway in the total wins among teams indexed by Z in order to accommodate winners for all the games among these teams. We have proved TONCAS. ■

Combinatorial applications of network flow usually involve showing that the desired configuration exists if and only if a related network has a large enough flow. As in Application 4.3.16, the Max-flow Min-cut Theorem then yields a necessary and sufficient condition for its existence. Other examples include most of Exercises 5– and also Exercise 13 and Theorems 4.3.17–4.3.18.