1 Maximum Flows and Minimum Cuts

A classic result in graph theory is known as the max-flow/min-cut theorem. The basic idea is that if we have a weighted graph G and pair of vertices s, t, which represent the *source* and *target*, how much flow can we push through the network, if all flow is produced at s and all flow must be consumed at t? If we assume that individual nodes cannot store flow and that otherwise flow is *conserved*, i.e., not created or destroyed anywhere except at s and t, the maximum amount of flow is given by the minimum weight s-t cut of the network, i.e., a division of the network into two pieces, one containing s and the other containing t. That is, the max-flow problem asks what is the largest amount of stuff we can move from s to t through G, while the min-cut problem asks what is the minimum amount of damage necessary to separate s and t.

When is finding a max flow or min cut useful? Aside from the obvious applications to flows of commodities (or weapons, or oil, or information, or whatever)[1] through distribution networks, we can also use max flow to solve problems that we can *reduce* to a max flow problem. For instance, if we have a weighted bipartite graph, we can use a max flow algorithm to find a maximum matching on this graph, i.e., a collection of edges such that no two edges share a vertex.

Here is the reduction to max flow: let G be the given (undirected) bipartite graph with vertex sets U and V, such that every edge $(u, v) \in E$ has $u \in U$ and $v \in V$. We create a new directed graph G' by orienting every edge from U to V, adding two new vertices s and t, adding edges from s to every vertex in U and from every vertex in V to t. Finally, we assign every edge in G' a capacity of 1. Because any matching on G can be transformed into a max flow on G', a solution on G' corresponds to a solution on G (do you see how?).

1.1 Flows

Recall that under our model, flow is *conserved*, that is, for each vertex (except s and t), the flux into a vertex v must equal the flux out of v; mathematically, we say

$$\forall_v \qquad \sum_u f(u \to v) = \sum_w f(v \to w)$$

where f is a function that maps edges into non-negative real values ($f : E \to \mathbb{R}_{\geq 0}$) and we assume that $f(u \to v) = 0$ if $(u, v) \notin E$. That is, flows are only allowed on edges in the graph and negative

[1] As an interesting point of history, it turns out that the max-flow / min-cut problem was relevant during the Cold War: in the mid-1950s, two researchers in the US Air Force published a classified report studying the railroad network of the Soviet Union and Eastern Europe. The network had 44 vertices and 105 edges, with each vertex representing a geographic region and edges representing railway links between them. Through laborious trial-and-error, they determined the maximum amount of material that could be transported from Russia to Europe as well as the cheapest way to disrupt that flow (by blowing up train tracks). Their results were declassified in 1999. See A. Schrijver's "On the history of combinatorial optimization (till 1960)."

flows are forbidden. A useful analogy is to think of the graph G as being a set of pipes connected together in a network, with water flowing from the source s through the pipes to the sink t.

Flows on such a network are meaningless unless edges have *capacities*, that is, we have another function c that gives the maximum flow possible, also as a non-negative real value, on a particular edge ($c : E \to \mathbb{R}_{\geq 0}$). And, like flows, we require $c(u, v) = 0$ if $(u, v) \notin E$

A flow is called *feasible* with respect to c if for every edge $e \in E$, the capacity is not exceeded by the flow $f(e) \leq c(e)$. Typically, capacities of all edges are fixed and we only consider feasible flows with respect to the specified c. Continuing our analogy with water and pipes, a large pipe has a large capacity, and because water is an incompressible fluid, if a pipe has a capacity of k units of water flow, no flow through the network can push more than k units through that pipe.

Finally, we say that a flow *saturates* an edge e if $f(e) = c(e)$ (the largest possible flow on e) and a flow *avoids* an edge if $f(e) = 0$ (the smallest possible flow on e).

The maximum flow problem is then to compute a feasible (s, t)-flow on G, given c such that the total value of the flow $\sum_{e \in E} f(e)$ is as large as possible. (Because s and t are not required to respect the conservation property, the *value* of the flow is equal to the net flow out of s or the net flow into t, which must be equal. Do you see why?)

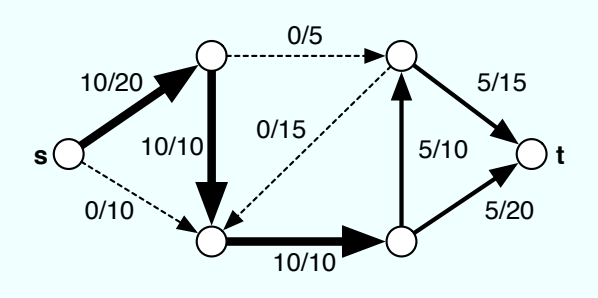

Figure 1: An example graph showing an (s, t)-flow with value $|f| = 10$. Edge labels: numerator is the flow on the edge; denominator is the capacity. f is avoiding the dashed edges.

1.2 Cuts

A *cut* on G is a bipartitioning of the vertices V into subsets S and T (meaning $S \cup T = V$ and $S \cap T = \{\}$), with $s \in S$ and $t \in T$.

If we are given some cut and a capacity function c (as above), the *capacity* of a cut is the sum of the capacities of the edges "in" the cut, i.e., the edges that start in S and end in T:

$$||S,T|| = \sum_{v \in S} \sum_{w \in T} c(v \to w) \ .$$

Note: the capacity of $||S,T||$ is not necessarily equal to the capacity of $||T,S||$.

The minimum cut problem is to find an (s,t)-cut whose capacity is the smallest of all possible cuts. The edges in this cut are thus the minimum-cost edges required to disconnect s and t.

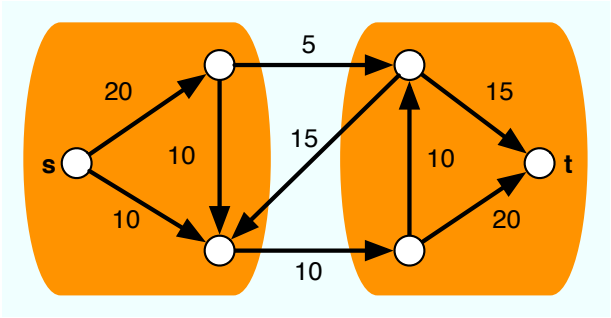

Figure 2: The same graph from Figure 1, now showing an (s,t)-cut with capacity $||S,T|| = 15$. Edge labels give the capacity only. (In this case $||S,T|| = ||T,S||$.) Note that the size of the cut here is larger than the size of the flow f in Figure 1; this implies that f in Figure 1 was not a maximum flow.

1.3 Max-Flow Min-Cut Theorem

It is not hard to prove that the value of any feasible (s,t)-flow is at most the capacity of any (s,t)-cut. In fact, an even stronger statement holds, which we call the *equality condition*: the value of a flow $|f| = ||S,T||$ if and only if the flow f saturates every edge from S to T and avoids every edge from T to S. (Why is avoiding edges from T to S important?) Moreover, for a flow f and a cut (S,T) that satisfies this equality condition, f must be a maximum flow and (S,T) must be a minimum cut.

For any weighted directed graph, there is always some flow f and some cut (S,T) that satisfy the equality condition. Stated more precisely:

Max-Flow Min-Cut Theorem:
The value of the maximum flow equals the capacity of the minimum cut.

Proof: Consider a fixed, directed graph $G = (V, E)$, vertices $s, t \in V$, and a capacity function c. To make some of the accounting easier later on, it will be useful to first derive a *reduced graph* G'. In G', we require that for all pairs of vertices u and v, either $c(u \to v) = 0$ or $c(v \to u) = 0$, i.e., if an edge appears in G', then its reversal does not. We do this because we will use the reverse edge to store some extra information about our current solution (and it makes the figures easier to understand).

Deriving a reduced graph G', which contains no bidirectional arcs, from the input graph G, which may contain them, can be done by taking each pair of non-reduced vertices $\{(u, v), (v, u)\}$ and replacing them with a small subgraph $\{(u, v), (v, x), (x, u)\}$, where $c(v \to x) = c(x \to u) = c(v \to u)$. This inserts a new vertex x into one of the two offending edges and thereby eliminates the redundancy. The max flow and min cut of G' will be the same as G (do you see why?).

Suppose we have some feasible flow f. Now, define a new function called *residual capacity* c_f that also maps pairs of vertices to positive real values, i.e., $c_f : V \times V \to \mathbb{R}_{\geq 0}$. We use this function to measure the used and remaining capacity on an edge, given our current flow f:

$$c_f(u \to v) = \begin{cases} c(u \to v) - f(u \to v) & \text{if } (u, v) \in E \\ f(u \to v) & \text{if } (v, u) \in E \\ 0 & \text{otherwise .} \end{cases}$$

The first case corresponds to edges $u \to v$ that currently carry some flow in f; c_f assigns them a weight equal to their unused capacity. The second case assigns a weight to the edge $v \to u$ equal to the flow on the edge $u \to v$. The third case assigns zero weight to all other pairs of vertices. Because $0 \leq f(u \to v) \leq c(u \to v)$, residual capacities are always non-negative.

This residual capacity function lets us define a *residual graph* $G_f = (V, E_f)$, where E_f is the set of edges whose residual capacity is positive, i.e., the set of edges that are not saturated. For example, see Figure 3. In the residual graph, a "back" edge $v \to u$ stores the flow along $u \to v$; this lets us store the unused capacity of $u \to v$ as a "forward" edge. In short, the residual graph G_f is a fancy kind of graph data structure that stores used and unused capacity in a way that makes it easy to find incremental increases to the total flow, which is how we will find maximum flows on G.

To see why, suppose there is no path from s to t in the residual graph G_f. Let S be the set of all vertices that are reachable in G_f from s, and let $T = V - S$ (the rest of the graph). The partition (S, T) is then an (s, t)-cut, and for every pair of vertices $u \in S$, $v \in T$ (the edges in the cut), we

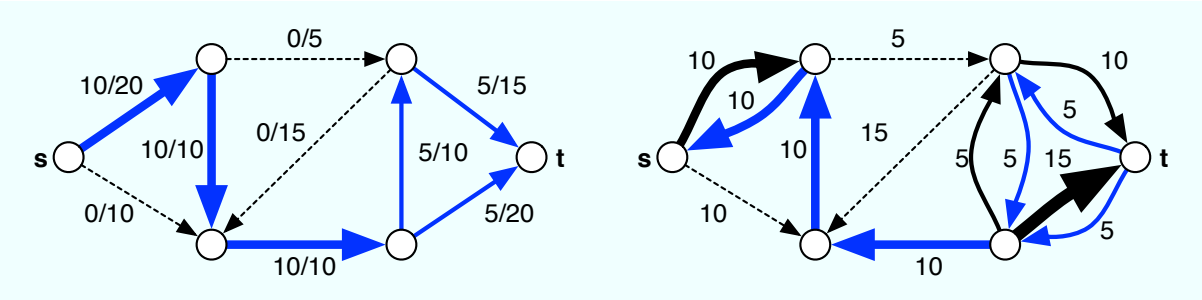

Figure 3: A (non-max) flow f on our example weighted graph G and the corresponding residual graph G_f. The blue edges represent the flow edges; in G, they point forward, while in G_f, they point backward, representing used capacity. Black edges in G_f represent unused capacity along edges currently carrying flow. Dashed edges carried no flow in f and thus point forward in G_f. Can you spot the one augmenting path in G_f?

have

$$c_f(u \to v) = \Big(c(u \to v) - f(u \to v)\Big) + f(v \to u) = 0 \ ,$$

which implies that each of these edges from S to T is saturated (i.e., $c(u \to v) - f(u \to v) = 0$) and each edge from T to S is avoided (i.e., $f(v \to u) = 0$). Thus, f is a max flow and (S, T) is a min cut.

Suppose there exists a path $\sigma_{st} = \{s = v_0 \to v_1 \to \cdots \to v_r = t\}$ in the residual graph G_f, i.e., a path from s to t. Call σ_{st} an *augmenting path*. On this path, define $F = \min_i c_f(v_i \to v_{i+1})$, i.e., F is the maximum amount of residual flow along the path σ_{st}. This implies that if we can find some augmenting path σ_{st}, then we can improve our current flow f by pushing F additional units of flow through G along this path. We can update f to include this flow in the following way: let f' be the updated flow, and let

$$f'(u \to v) = \begin{cases} f(u \to v) + F & \text{if } (u,v) \in \sigma_{st} \\ f(u \to v) - F & \text{if } (v,u) \in \sigma_{st} \\ f(u \to v) & \text{otherwise} \ . \end{cases}$$

In the first case, there is unused capacity on an existing edge $u \to v$, and we can increase the flow from s to t by using F of that capacity. In the second case, we move some flow off some edge $u \to v$, because we have moved it elsewhere. The third case treats the case of an edge not in the augmenting path, in which case we leave its flow unchanged.

We now prove that f' is feasible with respect to the original capacities c, which simply requires us to verify that $0 \leq f' \leq c$ for each edge. Since we only modify edges in the augmenting path

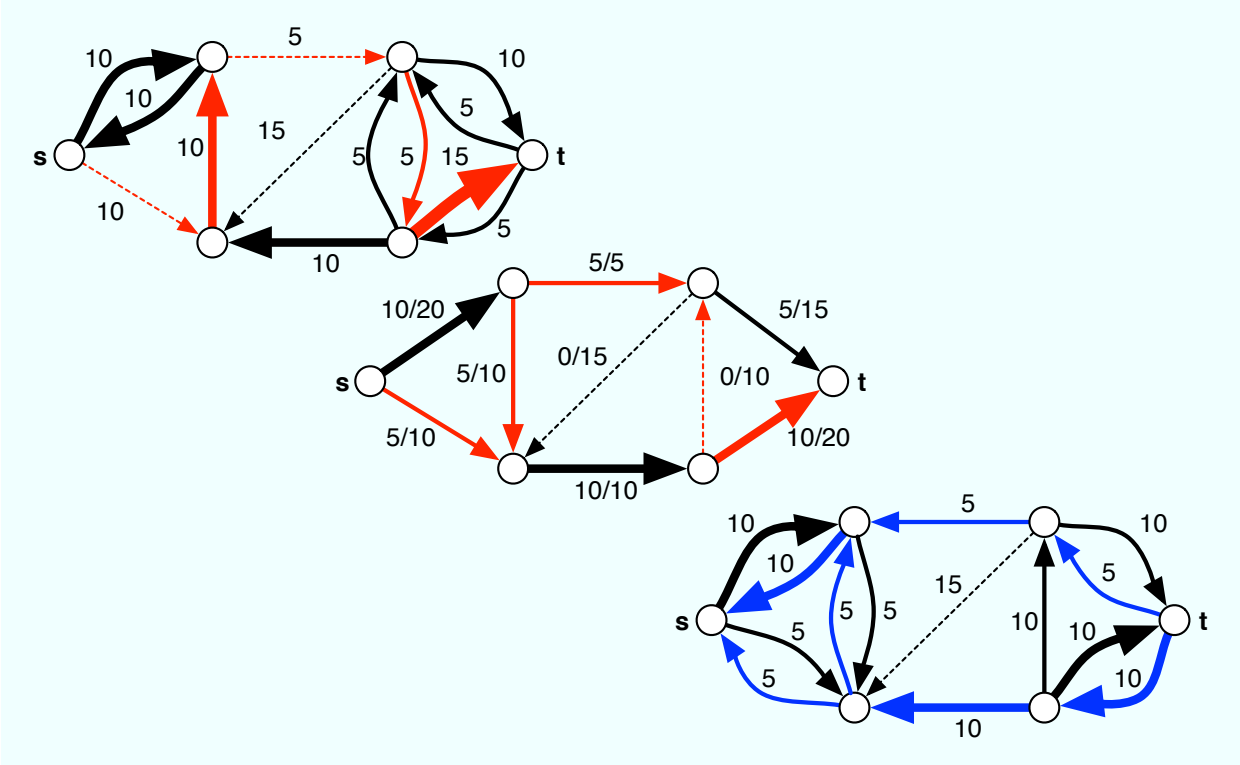

Figure 4: (upper) An augmenting path σ_{st} (shown in red) in G_f with additional flux $F = 5$, (middle) the corresponding augmented flow f' on G, and (lower) the residual graph $G_{f'}$ after the update. In the final residual graph, the fluxes of f' are shown in blue and residual capacities in black; note that now there exists no additional σ_{st}, implying that f' is a max flow. (There are other max flow solutions to this graph, including one that uses only 6 edges. Do you see it?)

itself, we must only check whether we have violated any of their capacities. If the forward edge $(u, v) \in \sigma_{st}$, then we increase its flow and $f'(u \to v) > f(u \to v) \geq 0$ and thus

$$\begin{aligned} f'(u \to v) &= f(u \to v) + F \\ &\leq f(u \to v) + c_f(u \to v) \\ &= f(u \to v) + c(u \to v) - f(u \to v) \\ &= c(u \to v) \end{aligned}$$

On the other hand, if the backward edge $(v, u) \in \sigma_{st}$, then we decrease its flow and $f'(u \to v) <$

$f(u \to v) \leq c(u \to v)$ and thus

$$\begin{aligned} f'(u \to v) &= f(u \to v) - F \\ &\geq f(u \to v) - c_f(u \to v) \\ &= f(u \to v) - f(u \to v) \\ &= 0 \end{aligned}$$

Finally, since $|f'| = |f| + F > 0$, f was not a maximum flow. □

Thus, we have proved that if f is a maximum flow, there will exist no augmenting path in G_f and (S, T) is a minimum cut. The neat thing about this proof is that it is constructive, i.e., it not only proves that the maximum flow is the minimum cut, it tells us how to find it: repeatedly find an augmenting path in G_f until no such path exists.

1.4 Ford-Fulkerson algorithm

Most max-flow min-cut algorithms use this observation about augmenting paths to construct a solution to the max-flow min-cut problem.

The Ford-Fulkerson algorithm[2] does exactly this: it starts with $f = 0$ and then continues finding *some* augmenting path in G_f (not necessarily the path with the largest F), adding the corresponding flow to f, until no such path can be found.

If all the edge capacities are integers, then each augmentation step increases $|f|$ by a positive integer which is at least 1. Thus, the algorithm must halt after at most $|f^*|$ iterations, where $|f^*|$ is the maximum flow. Each iteration of this algorithm requires $O(E)$ time, which accounts both for creating the residual graph G_f and for running a search-tree algorithm to find some augmenting path (it doesn't matter what kind of tree-search algorithm; try DFS). Thus, Ford-Fulkerson takes $O(E\,|f^*|)$ time.

It also holds that Ford-Fulkerson will halt if the edge capacities are rational numbers because we can convert any set of rational numbers into a set of integers by multiplying by a sufficiently large integer (do you see why? what integer would do the trick?). However, if any edge capacity is irrational (and represented using infinite precision), then Ford-Fulkerson fails to halt because the algorithm can always find an augmenting path with smaller and small additional flux F, and worse, this infinite sequence of augmentations may not even converge on the maximum flow.[3] This point is

[2] Due to Lester R. Ford, Jr. and Delbert R. Fulkerson, published in 1954.

[3] Can you construct an example of such pathological behavior? Uri Zwick identified one such graph with 6 nodes and 9 edges where six edges have some large integer capacity X, two have capacity 1 and one has capacity $\phi_- = (\sqrt{5} - 1)/2 = 0.61803....$

not entirely academic: because modern computers represent rational numbers using floating point structures, a careless implementation of Ford-Fulkerson could enter into an infinite loop simply because of a round-off error.

1.5 Edmonds-Karp (1)

We can be smarter about converging on the maximum flow if we are smarter about how we choose which augmenting path to use, if there are multiple choices. One approach, called Edmonds-Karp,[4] is to be greedy about it, i.e., to choose the augmenting path with the largest F. In other words, when given a choice, fill the fattest open pipe first.[5]

The maximum-bottleneck (s,t)-path in a directed graph can be computed in $O(E \log V)$ time using Prim's MST algorithm or Dijkstra's SSSP algorithm: start growing a directed spanning tree S rooted at s by repeatedly finding and adding to S the highest-capacity edge within the cut (S,T) (i.e., among the edges with exactly one vertex in S) until the tree contains t (at which point, the tree must also contain a path from s to t).

The analysis of this algorithm is similar to that of Ford-Fulkerson. A simple upper bound on the running time is to observe that we find, again, at most $|f^*|$ augmenting paths and it takes $O(E \log V)$ time to find each one and update our residual graph data structure. This yields $O(|f^*|\, E \log V)$ time. Note that we can make this running time arbitrarily slow by choosing some edge capacities to be very very large; this changes $|f^*|$, which may not be a desirable property (and could lead to non-polynomial running times).

1.6 Edmonds-Karp (2)

An alternative version of Edmonds-Karp (which was independently discovered by Dinit) chooses the augmenting path with the fewest edges, i.e., to find the shortest open pipe to fill.[6] This version, which I won't go into here,[7] runs in time $O(VE^2)$, which can be improved to $O(V^2E)$ using an insight due to Dinit.

2 For Next Time

1. Reminder: solutions to PS2 due February 25, via email, by 11:59pm

[4] First published by Yefim Dinic in 1970 and independently published by Jack Edmonds and Richard Karp in 1972.
[5] In the spirit of FIFO, LIFO and OSPF, you might call this "FOPF."
[6] Perhaps "SOPF"?
[7] Briefly: this is like asking for the shortest augmenting path, which means running breadth-first search on the residual graph. Surprisingly, the running time of this approach can be shown to be independent of the edge capacities, which can be a desirable property.