

A process cannot be understood by stopping it. Understanding must move with the flow of the process, must join it and flow with it.

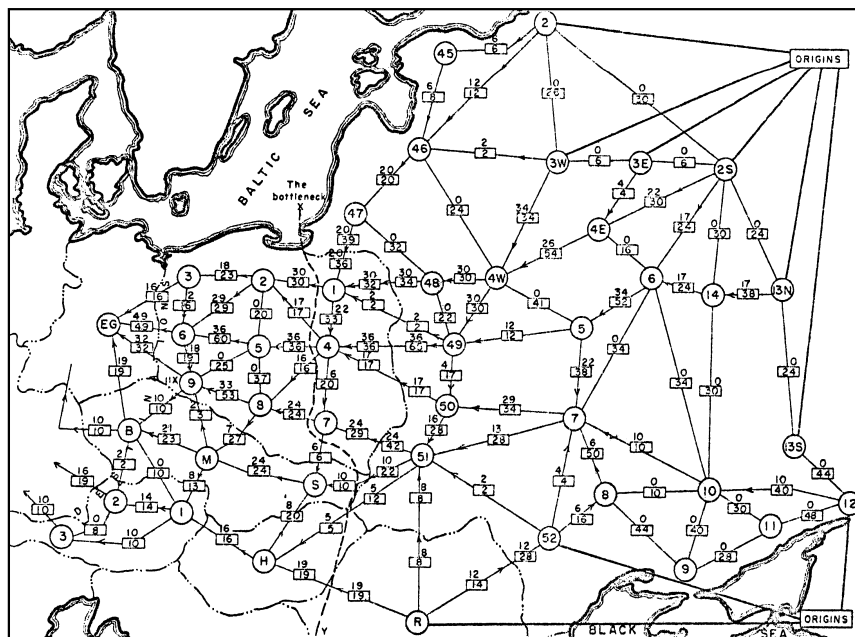
— The First Law of Mentat, in Frank Herbert's *Dune* (1965)

There's a difference between knowing the path and walking the path.

— Morpheus [Laurence Fishburne], *The Matrix* (1999)

22 Maximum Flows and Minimum Cuts

In the mid-1950s, Air Force researcher Theodore E. Harris and retired army general Frank S. Ross published a classified report studying the rail network that linked the Soviet Union to its satellite countries in Eastern Europe. The network was modeled as a graph with 44 vertices, representing geographic regions, and 105 edges, representing links between those regions in the rail network. Each edge was given a weight, representing the rate at which material could be shipped from one region to the next. Essentially by trial and error, they determined both the maximum amount of stuff that could be moved from Russia into Europe, as well as the cheapest way to disrupt the network by removing links (or in less abstract terms, blowing up train tracks), which they called 'the bottleneck'. Their results, including the drawing of the network below, were only declassified in 1999.¹



Harris and Ross's map of the Warsaw Pact rail network

This one of the first recorded applications of the *maximum flow* and *minimum cut* problems. For both problems, the input is a directed graph $G = (V, E)$, along with special vertices s and t called the *source* and *target*. As in the previous lectures, I will use $u \rightarrow v$ to denote the directed edge from vertex u to vertex v . Intuitively, the maximum flow problem asks for the largest amount of material that can be transported from s to t ; the minimum cut problem asks for the minimum damage needed to separate s from t .

¹Both the map and the story were taken from Alexander Schrijver's fascinating survey 'On the history of combinatorial optimization (till 1960)'.

22.1 Flows

An (s, t) -flow (or just a flow if the source and target are clear from context) is a function $f : E \rightarrow \mathbb{R}_{\geq 0}$ that satisfies the following **conservation constraint** at every vertex v except possibly s and t :

$$\sum_u f(u \rightarrow v) = \sum_w f(v \rightarrow w).$$

In English, the total flow into v is equal to the total flow out of v . To keep the notation simple, we define $f(u \rightarrow v) = 0$ if there is no edge $u \rightarrow v$ in the graph. The **value** of the flow f , denoted $|f|$, is the total net flow out of the source vertex s :

$$|f| := \sum_w f(s \rightarrow w) - \sum_u f(u \rightarrow s).$$

It's not hard to prove that $|f|$ is also equal to the total net flow into the target vertex t , as follows. To simplify notation, let $\partial f(v)$ denote the total net flow out of any vertex v :

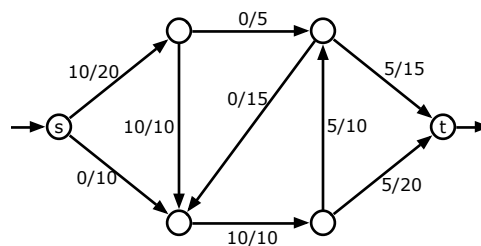
$$\partial f(v) := \sum_u f(u \rightarrow v) - \sum_w f(v \rightarrow w).$$

The conservation constraint implies that $\partial f(v) = 0$ for every vertex v except s and t , so

$$\sum_v \partial f(v) = \partial f(s) + \partial f(t).$$

On the other hand, any flow that leaves one vertex must enter another vertex, so we must have $\sum_v \partial f(v) = 0$. It follows immediately that $|f| = \partial f(s) = -\partial f(t)$.

Now suppose we have another function $c : E \rightarrow \mathbb{R}_{\geq 0}$ that assigns a non-negative **capacity** $c(e)$ to each edge e . We say that a flow f is **feasible** (with respect to c) if $f(e) \leq c(e)$ for every edge e . Most of the time we will consider only flows that are feasible with respect to some fixed capacity function c . We say that a flow f **saturates** edge e if $f(e) = c(e)$, and **avoids** edge e if $f(e) = 0$. The **maximum flow problem** is to compute a feasible (s, t) -flow in a given directed graph, with a given capacity function, whose value is as large as possible.



An (s, t) -flow with value 10. Each edge is labeled with its flow/capacity.

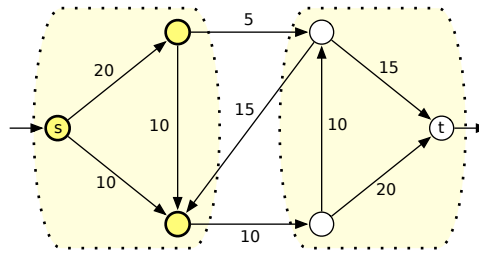
22.2 Cuts

An (s, t) -cut (or just cut if the source and target are clear from context) is a partition of the vertices into disjoint subsets S and T —meaning $S \cup T = V$ and $S \cap T = \emptyset$ —where $s \in S$ and $t \in T$.

If we have a capacity function $c : E \rightarrow \mathbb{R}_{\geq 0}$, the **capacity** of a cut is the sum of the capacities of the edges that start in S and end in T :

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

(Again, if $v \rightarrow w$ is not an edge in the graph, we assume $c(v \rightarrow w) = 0$.) Notice that the definition is asymmetric; edges that start in T and end in S are unimportant. The *minimum cut problem* is to compute an (s, t) -cut whose capacity is as large as possible.



An (s, t) -cut with capacity 15. Each edge is labeled with its capacity.

Intuitively, the minimum cut is the cheapest way to disrupt all flow from s to t . Indeed, it is not hard to show that **the value of any feasible (s, t) -flow is at most the capacity of any (s, t) -cut**. Choose your favorite flow f and your favorite cut (S, T) , and then follow the bouncing inequalities:

$$\begin{aligned}
 |f| &= \sum_w f(s \rightarrow w) - \sum_u f(u \rightarrow s) && \text{by definition} \\
 &= \sum_{v \in S} \left(\sum_w f(v \rightarrow w) - \sum_u f(u \rightarrow v) \right) && \text{by the conservation constraint} \\
 &= \sum_{v \in S} \left(\sum_{w \in T} f(v \rightarrow w) - \sum_{u \in T} f(u \rightarrow v) \right) && \text{removing duplicate edges} \\
 &\leq \sum_{v \in S} \sum_{w \in T} f(v \rightarrow w) && \text{since } f(u \rightarrow v) \geq 0 \\
 &\leq \sum_{v \in S} \sum_{w \in T} c(v \rightarrow w) && \text{since } f(u \rightarrow v) \leq c(v \rightarrow w) \\
 &= \|S, T\| && \text{by definition}
 \end{aligned}$$

Our derivation actually implies the following stronger observation: $|f| = \|S, T\|$ **if and only if f saturates every edge from S to T and avoids every edge from T to S** . Moreover, if we have 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.

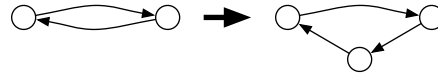
22.3 The Maxflow Mincut Theorem

Surprisingly, for any weighted directed graph, there is always a flow f and a cut (S, T) that satisfy the equality condition. This is the famous *max-flow min-cut theorem*, first proved by Lester Ford (of shortest path fame) and Delbert Ferguson in 1954 and independently by Peter Elias, Amiel Feinstein, and Claude Shannon (of information theory fame) in 1956.

The Maxflow Mincut Theorem. *In any flow network with source s and target t , the value of the maximum (s, t) -flow is equal to the capacity of the minimum (s, t) -cut.*

Ford and Fulkerson proved this theorem as follows. Fix a graph G , vertices s and t , and a capacity function $c: E \rightarrow \mathbb{R}_{\geq 0}$. The proof will be easier if we assume that the capacity function is *reduced*: For any vertices u and v , either $c(u \rightarrow v) = 0$ or $c(v \rightarrow u) = 0$, or equivalently, if an edge appears in G , then its

reversal does not. This assumption is easy to enforce. Whenever an edge $u \rightarrow v$ and its reversal $v \rightarrow u$ are both the graph, replace the edge $u \rightarrow v$ with a path $u \rightarrow x \rightarrow v$ of length two, where x is a new vertex and $c(u \rightarrow x) = c(x \rightarrow v) = c(u \rightarrow v)$. The modified graph has the same maximum flow value and minimum cut capacity as the original graph.

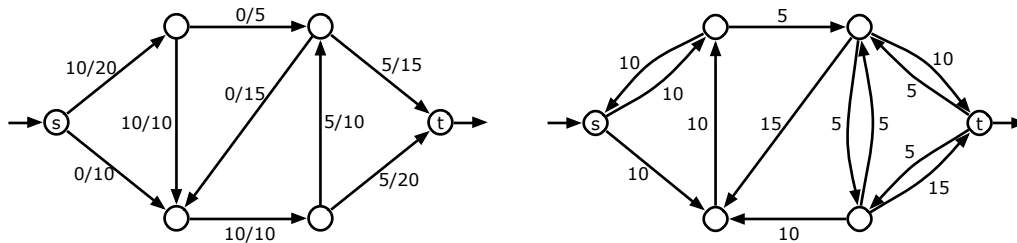


Enforcing the one-direction assumption.

Let f be a feasible flow. We define a new capacity function $c_f : V \times V \rightarrow \mathbb{R}$, called the **residual capacity**, as follows:

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

Since $f \geq 0$ and $f \leq c$, the residual capacities are always non-negative. It is possible to have $c_f(u \rightarrow v) > 0$ even if $u \rightarrow v$ is not an edge in the original graph G . Thus, we define the **residual graph** $G_f = (V, E_f)$, where E_f is the set of edges whose residual capacity is positive. Notice that the residual capacities are *not* necessarily reduced; it is quite possible to have both $c_f(u \rightarrow v) > 0$ and $c_f(v \rightarrow u) > 0$.

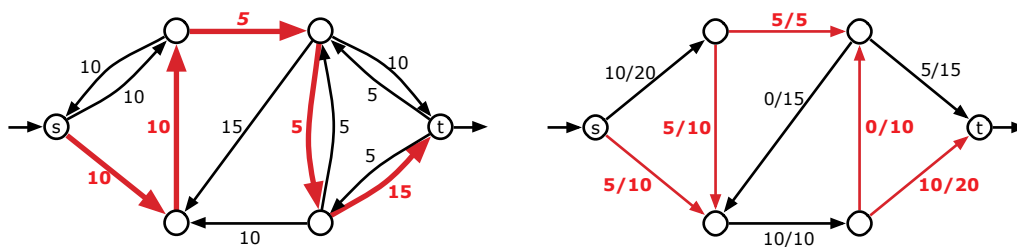


A flow f in a weighted graph G and the corresponding residual graph G_f .

Suppose there is no path from the source s to the target t in the residual graph G_f . Let S be the set of vertices that are reachable from s in G_f , and let $T = V \setminus S$. The partition (S, T) is clearly an (s, t) -cut. For every vertex $u \in S$ and $v \in T$, we have

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

which implies that $c(u \rightarrow v) - f(u \rightarrow v) = 0$ and $f(v \rightarrow u) = 0$. In other words, our flow f saturates every edge from S to T and avoids every edge from T to S . It follows that $|f| = ||S, T||$. Moreover, f is a maximum flow and (S, T) is a minimum cut.



An augmenting path in G_f with value $F = 5$ and the augmented flow f' .

On the other hand, suppose there is a path $s = v_0 \rightarrow v_1 \rightarrow \dots \rightarrow v_r = t$ in G_f . We refer to $v_0 \rightarrow v_1 \rightarrow \dots \rightarrow v_r$ as an **augmenting path**. Let $F = \min_i c_f(v_i \rightarrow v_{i+1})$ denote the maximum amount of flow that we can

push through the augmenting path in G_f . We define a new flow function $f': E \rightarrow \mathbb{R}$ as follows:

$$f'(u \rightarrow v) = \begin{cases} f(u \rightarrow v) + F & \text{if } u \rightarrow v \text{ is in the augmenting path} \\ f(u \rightarrow v) - F & \text{if } v \rightarrow u \text{ is in the augmenting path} \\ f(u \rightarrow v) & \text{otherwise} \end{cases}$$

To prove that the flow f' is feasible with respect to the original capacities c , we need to verify that $f' \geq 0$ and $f' \leq c$. Consider an edge $u \rightarrow v$ in G . If $u \rightarrow v$ is in the augmenting path, then $f'(u \rightarrow v) > f(u \rightarrow v) \geq 0$ and

$$\begin{aligned} f'(u \rightarrow v) &= f(u \rightarrow v) + F && \text{by definition of } f' \\ &\leq f(u \rightarrow v) + c_f(u \rightarrow v) && \text{by definition of } F \\ &= f(u \rightarrow v) + c(u \rightarrow v) - f(u \rightarrow v) && \text{by definition of } c_f \\ &= c(u \rightarrow v) && \text{Duh.} \end{aligned}$$

On the other hand, if the reversal $v \rightarrow u$ is in the augmenting path, then $f'(u \rightarrow v) < f(u \rightarrow v) \leq c(u \rightarrow v)$, which implies that

$$\begin{aligned} f'(u \rightarrow v) &= f(u \rightarrow v) - F && \text{by definition of } f' \\ &\geq f(u \rightarrow v) - c_f(v \rightarrow u) && \text{by definition of } F \\ &= f(u \rightarrow v) - f(u \rightarrow v) && \text{by definition of } c_f \\ &= 0 && \text{Duh.} \end{aligned}$$

Finally, we observe that (without loss of generality) only the first edge in the augmenting path leaves s , so $|f'| = |f| + F > 0$. In other words, f is *not* a maximum flow.

This completes the proof!

22.4 Ford and Fulkerson's augmenting-path algorithm

Ford and Fulkerson's proof of the Maxflow-Mincut Theorem translates immediately to an algorithm to compute maximum flows: Starting with the zero flow, repeatedly augment the flow along **any** path from s to t in the residual graph, until there is no such path.

This algorithm has an important but straightforward corollary:

Integrality Theorem. *If all capacities in a flow network are integers, then there is a maximum flow such that the flow through every edge is an integer.*

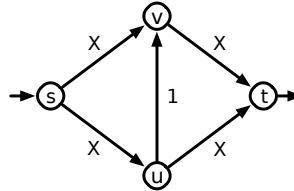
Proof: We argue by induction that after each iteration of the augmenting path algorithm, all flow values and residual capacities are integers. Before the first iteration, residual capacities are the original capacities, which are integral by definition. In each later iteration, the induction hypothesis implies that the capacity of the augmenting path is an integer, so augmenting changes the flow on each edge, and therefore the residual capacity of each edge, by an integer.

In particular, the algorithm increases the overall value of the flow by a positive integer, which implies that the augmenting path algorithm halts and returns a maximum flow. \square

If every edge capacity is an integer, the algorithm halts after $|f^*|$ iterations, where f^* is the actual maximum flow. In each iteration, we can build the residual graph G_f and perform a whatever-first-search

to find an augmenting path in $O(E)$ time. Thus, for networks with integer capacities, the Ford-Fulkerson algorithm runs in $O(E|f^*|)$ time in the worst case.

The following example shows that this running time analysis is essentially tight. Consider the 4-node network illustrated below, where X is some large integer. The maximum flow in this network is clearly $2X$. However, Ford-Fulkerson might alternate between pushing 1 unit of flow along the augmenting path $s \rightarrow u \rightarrow v \rightarrow t$ and then pushing 1 unit of flow along the augmenting path $s \rightarrow v \rightarrow u \rightarrow t$, leading to a running time of $\Theta(X) = \Omega(|f^*|)$.



A bad example for the Ford-Fulkerson algorithm.

Ford and Fulkerson's algorithm works quite well in many practical situations, or in settings where the maximum flow value $|f^*|$ is small, but without further constraints on the augmenting paths, this is *not* an efficient algorithm in general. The example network above can be described using only $O(\log X)$ bits; thus, the running time of Ford-Fulkerson is actually *exponential* in the input size.

22.5 Irrational Capacities

If we multiply all the capacities by the same (positive) constant, the maximum flow increases everywhere by the same constant factor. It follows that if all the edge capacities are *rational*, then the Ford-Fulkerson algorithm eventually halts, although still in exponential time.

However, if we allow *irrational* capacities, the algorithm can actually loop forever, always finding smaller and smaller augmenting paths! Worse yet, this infinite sequence of augmentations may not even converge to the maximum flow, or even to a significant fraction of the maximum flow! Perhaps the simplest example of this effect was discovered by Uri Zwick.

Consider the six-node network shown on the next page. Six of the nine edges have some large integer capacity X , two have capacity 1, and one has capacity $\phi = (\sqrt{5} - 1)/2 \approx 0.618034$, chosen so that $1 - \phi = \phi^2$. To prove that the Ford-Fulkerson algorithm can get stuck, we can watch the residual capacities of the three horizontal edges as the algorithm progresses. (The residual capacities of the other six edges will always be at least $X - 3$.)

Suppose the Ford-Fulkerson algorithm starts by choosing the central augmenting path, shown in the large figure on the next page. The three horizontal edges, in order from left to right, now have residual capacities 1, 0, and ϕ . Suppose inductively that the horizontal residual capacities are ϕ^{k-1} , 0, ϕ^k for some non-negative integer k .

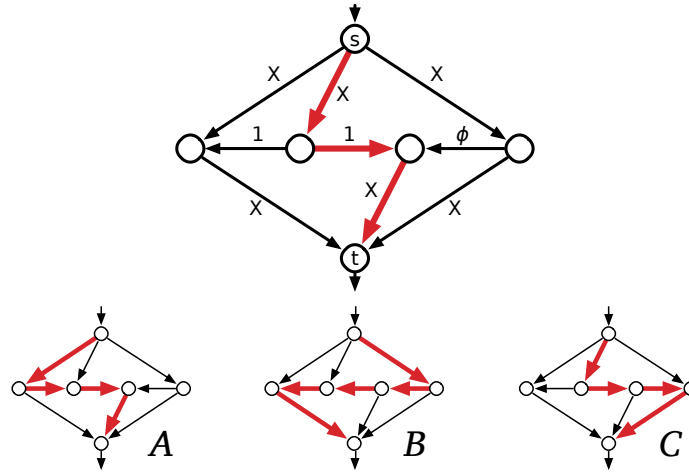
1. Augment along B , adding ϕ^k to the flow; the residual capacities are now $\phi^{k+1}, \phi^k, 0$.
2. Augment along C , adding ϕ^k to the flow; the residual capacities are now $\phi^{k+1}, 0, \phi^k$.
3. Augment along B , adding ϕ^{k+1} to the flow; the residual capacities are now $0, \phi^{k+1}, \phi^{k+2}$.
4. Augment along A , adding ϕ^{k+1} to the flow; the residual capacities are now $\phi^{k+1}, 0, \phi^{k+2}$.

It follows by induction that after $4n + 1$ augmentation steps, the horizontal edges have residual capacities

$\phi^{2n-2}, 0, \phi^{2n-1}$. As the number of augmentations grows to infinity, the value of the flow converges to

$$1 + 2 \sum_{i=1}^{\infty} \phi^i = 1 + \frac{2}{1 - \phi} = 4 + \sqrt{5} < 7,$$

even though the maximum flow value is clearly $2X + 1 \gg 7$.



Uri Zwick’s non-terminating flow example, and three augmenting paths.

Picky students might wonder at this point why we care about irrational capacities; after all, computers can’t represent anything but (small) integers or (dyadic) rationals exactly. Good question! One reason is that the integer restriction is literally *artificial*; it’s an *artifact* of actual computational hardware², not an inherent feature of the abstract mathematical problem. Another reason, which is probably more convincing to most practical computer scientists, is that the behavior of the algorithm with irrational inputs tells us something about its worst-case behavior *in practice* given floating-point capacities—terrible! Even with very reasonable capacities, a careless implementation of Ford-Fulkerson could enter an infinite loop simply because of round-off error.

22.6 Edmonds and Karp’s Algorithms

Ford and Fulkerson’s algorithm does not specify which path in the residual graph to augment, and the poor behavior of the algorithm can be blamed on poor choices for the augmenting path. In the early 1970s, Jack Edmonds and Richard Karp analyzed two natural rules for choosing augmenting paths, both of which led to more efficient algorithms.

22.6.1 Fat Pipes

Edmonds and Karp’s first rule is essentially a greedy algorithm:

Choose the augmenting path with largest bottleneck value.

It’s a fairly easy to show that the maximum-bottleneck (s, t) -path in a directed graph can be computed in $O(E \log V)$ time using a variant of Jarník’s minimum-spanning-tree algorithm, or of Dijkstra’s shortest path algorithm. Simply grow a directed spanning tree T , rooted at s . Repeatedly find the highest-capacity

²...or perhaps the laws of physics. Yeah, whatever. Like *reality* actually matters in this class.

edge leaving T and add it to T , until T contains a path from s to t . Alternately, one could emulate Kruskal's algorithm—insert edges one at a time in decreasing capacity order until there is a path from s to t —although this is less efficient, at least when the graph is directed.

We can now analyze the algorithm in terms of the value of the maximum flow f^* . Let f be any flow in G , and let f' be the maximum flow in the current residual graph G_f . (At the beginning of the algorithm, $G_f = G$ and $f' = f^*$.) Let e be the bottleneck edge in the next augmenting path. Let S be the set of vertices reachable from s through edges in G_f with capacity greater than $c_f(e)$ and let $T = V \setminus S$. By construction, T is non-empty, and every edge from S to T has capacity at most $c_f(e)$. Thus, the capacity of the cut (S, T) is at most $c_f(e) \cdot E$. On the other hand, the maxflow-mincut theorem implies that $\|S, T\| \geq |f'|$. We conclude that $c(e) \geq |f'|/E$.

The preceding argument implies that augmenting f along the maximum-bottleneck path in G_f multiplies the maximum flow value in G_f by a factor of at most $1 - 1/E$. In other words, the residual maximum flow value *decays exponentially* with the number of iterations. After $E \cdot \ln|f^*|$ iterations, the maximum flow value in G_f is at most

$$|f^*| \cdot (1 - 1/E)^{E \cdot \ln|f^*|} < |f^*| e^{-\ln|f^*|} = 1.$$

(That's Euler's constant e , not the edge e . Sorry.) In particular, *if all the capacities are integers*, then after $E \cdot \ln|f^*|$ iterations, the maximum capacity of the residual graph is *zero* and f is a maximum flow.

We conclude that for graphs with integer capacities, the Edmonds-Karp 'fat pipe' algorithm runs in $O(E^2 \log E \log|f^*|)$ time, which is actually a polynomial function of the input size.

22.6.2 Short Pipes

The second Edmonds-Karp rule was actually proposed by Ford and Fulkerson in their original max-flow paper; a variant of this rule was independently considered by the Russian mathematician Yefim Dinits around the same time as Edmonds and Karp.

Choose the augmenting path with the smallest number of edges.

The shortest augmenting path can be found in $O(E)$ time by running breadth-first search in the residual graph. Surprisingly, the resulting algorithm halts after a polynomial number of iterations, independent of the actual edge capacities!

The proof of this polynomial upper bound relies on two observations about the evolution of the residual graph. Let f_i be the current flow after i augmentation steps, let G_i be the corresponding residual graph. In particular, f_0 is zero everywhere and $G_0 = G$. For each vertex v , let $level_i(v)$ denote the unweighted shortest path distance from s to v in G_i , or equivalently, the *level* of v in a breadth-first search tree of G_i rooted at s .

Our first observation is that these levels can only increase over time.

Lemma 1. $level_{i+1}(v) \geq level_i(v)$ for all vertices v and integers i .

Proof: The claim is trivial for $v = s$, since $level_i(s) = 0$ for all i . Choose an arbitrary vertex $v \neq s$, and let $s \rightarrow \dots \rightarrow u \rightarrow v$ be a shortest path from s to v in G_{i+1} . (If there is no such path, then $level_{i+1}(v) = \infty$, and we're done.) Because this is a shortest path, we have $level_{i+1}(v) = level_{i+1}(u) + 1$, and the inductive hypothesis implies that $level_{i+1}(u) \geq level_i(u)$.

We now have two cases to consider. If $u \rightarrow v$ is an edge in G_i , then $level_i(v) \leq level_i(u) + 1$, because the levels are defined by breadth-first traversal.

On the other hand, if $u \rightarrow v$ is not an edge in G_i , then $v \rightarrow u$ must be an edge in the i th augmenting path. Thus, $v \rightarrow u$ must lie on the shortest path from s to t in G_i , which implies that $level_i(v) = level_i(u) - 1 \leq level_i(u) + 1$.

In both cases, we have $level_{i+1}(v) = level_{i+1}(u) + 1 \geq level_i(u) + 1 \geq level_i(v)$. \square

Whenever we augment the flow, the bottleneck edge in the augmenting path disappears from the residual graph, and some other edge in the *reversal* of the augmenting path may (re-)appear. Our second observation is that an edge cannot appear or disappear too many times.

Lemma 2. *During the execution of the Edmonds-Karp short-pipe algorithm, any edge $u \rightarrow v$ disappears from the residual graph G_f at most $V/2$ times.*

Proof: Suppose $u \rightarrow v$ is in two residual graphs G_i and G_{j+1} , but not in any of the intermediate residual graphs G_{i+1}, \dots, G_j , for some $i < j$. Then $u \rightarrow v$ must be in the i th augmenting path, so $level_i(v) = level_i(u) + 1$, and $v \rightarrow u$ must be on the j th augmenting path, so $level_j(v) = level_j(u) - 1$. By the previous lemma, we have

$$level_j(u) = level_j(v) + 1 \geq level_i(v) + 1 = level_i(u) + 2.$$

In other words, the distance from s to u increased by at least 2 between the disappearance and reappearance of $u \rightarrow v$. Since every level is either less than V or infinite, the number of disappearances is at most $V/2$. \square

Now we can derive an upper bound on the number of iterations. Since each edge can disappear at most $V/2$ times, there are at most $EV/2$ edge disappearances overall. But at least one edge disappears on each iteration, so the algorithm must halt after at most $EV/2$ iterations. Finally, since each iteration requires $O(E)$ time, this algorithm runs in $O(VE^2)$ time overall.

22.7 Further Progress

This is nowhere near the end of the story for maximum-flow algorithms. Decades of further research have led to a number of even faster algorithms, some of which are summarized in the table below.³ All of the algorithms listed below compute a maximum flow in several iterations. Each algorithm has two variants: a simpler version that performs each iteration by brute force, and a faster variant that uses sophisticated data structures to maintain a spanning tree of the flow network, so that each iteration can be performed (and the spanning tree updated) in logarithmic time. There is no reason to believe that the best algorithms known so far are optimal; indeed, maximum flows are still a very active area of research.

The fastest known maximum flow algorithm, announced by James Orlin in 2012, runs in $O(VE)$ time. The details of Orlin's algorithm are far beyond the scope of this course; in addition to his own new techniques, Orlin uses several existing algorithms and data structures as black boxes, most of which are themselves quite complicated. Nevertheless, for purposes of analyzing algorithms that use maximum flows, this is the time bound you should cite. So write the following sentence on your cheat sheets and cite it in your homeworks:

Maximum flows can be computed in $O(VE)$ time.

³To keep the table short, I have deliberately omitted algorithms whose running time depends on the maximum capacity, the sum of the capacities, or the maximum flow value. Even with this restriction, the table is incomplete!

Technique	Direct	With dynamic trees	Sources
Blocking flow	$O(V^2E)$	$O(VE \log V)$	[Dinitz; Sleator and Tarjan]
Network simplex	$O(V^2E)$	$O(VE \log V)$	[Dantzig; Goldfarb and Hao; Goldberg, Grigoriadis, and Tarjan]
Push-relabel (generic)	$O(V^2E)$	—	[Goldberg and Tarjan]
Push-relabel (FIFO)	$O(V^3)$	$O(V^2 \log(V^2/E))$	[Goldberg and Tarjan]
Push-relabel (highest label)	$O(V^2 \sqrt{E})$	—	[Cheriy and Maheshwari; Tunçel]
Pseudoflow	$O(V^2E)$	$O(VE \log V)$	[Hochbaum]
Compact abundance graphs		$O(VE)$	[Orlin 2012]

Several purely combinatorial maximum-flow algorithms and their running times.

Exercises

1. Suppose you are given a directed graph $G = (V, E)$, two vertices s and t , a capacity function $c: E \rightarrow \mathbb{R}^+$, and a second function $f: E \rightarrow \mathbb{R}$. Describe an algorithm to determine whether f is a maximum (s, t) -flow in G .
2. Let (S, T) and (S', T') be minimum (s, t) -cuts in some flow network G . Prove that $(S \cap S', T \cup T')$ and $(S \cup S', T \cap T')$ are also minimum (s, t) -cuts in G .
3. Suppose (S, T) is the *unique* minimum (s, t) -cut in some flow network. Prove that (S, T) is also a minimum (x, y) -cut for all vertices $x \in S$ and $y \in T$.
4. Cuts are sometimes defined as subsets of the edges of the graph, instead of as partitions of its vertices. In this problem, you will prove that these two definitions are *almost* equivalent.

We say that a subset X of (directed) edges *separates* s and t if every directed path from s to t contains at least one (directed) edge in X . For any subset S of vertices, let δS denote the set of directed edges leaving S ; that is, $\delta S := \{u \rightarrow v \mid u \in S, v \notin S\}$.

- (a) Prove that if (S, T) is an (s, t) -cut, then δS separates s and t .
 - (b) Let X be an arbitrary subset of edges that separates s and t . Prove that there is an (s, t) -cut (S, T) such that $\delta S \subseteq X$.
 - (c) Let X be a *minimal* subset of edges that separates s and t . (Such a set of edges is sometimes called a **bond**.) Prove that there is an (s, t) -cut (S, T) such that $\delta S = X$.
5. A flow f is **acyclic** if the subgraph of directed edges with positive flow contains no directed cycles.
 - (a) Prove that for any flow f , there is an acyclic flow with the same value as f . (In particular, this implies that some maximum flow is acyclic.)
 - (b) A *path flow* assigns positive values only to the edges of one simple directed path from s to t . Prove that every acyclic flow can be written as the sum of $O(E)$ path flows.
 - (c) Describe a flow in a directed graph that *cannot* be written as the sum of path flows.
 - (d) A *cycle flow* assigns positive values only to the edges of one simple directed cycle. Prove that every flow can be written as the sum of $O(E)$ path flows and cycle flows.

- (e) Prove that every flow with value 0 can be written as the sum of $O(E)$ cycle flows. (Zero-value flows are also called *circulations*.)
6. Suppose instead of capacities, we consider networks where each edge $u \rightarrow v$ has a non-negative **demand** $d(u \rightarrow v)$. Now an (s, t) -flow f is *feasible* if and only if $f(u \rightarrow v) \geq d(u \rightarrow v)$ for every edge $u \rightarrow v$. (Feasible flow values can now be arbitrarily large.) A natural problem in this setting is to find a feasible (s, t) -flow of *minimum* value.
- (a) Describe an efficient algorithm to compute a feasible (s, t) -flow, given the graph, the demand function, and the vertices s and t as input. [Hint: Find a flow that is non-zero everywhere, and then scale it up to make it feasible.]
- (b) Suppose you have access to a subroutine MAXFLOW that computes *maximum* flows in networks with edge capacities. Describe an efficient algorithm to compute a *minimum* flow in a given network with edge demands; your algorithm should call MAXFLOW exactly once.
- (c) State and prove an analogue of the max-flow min-cut theorem for this setting. (Do minimum flows correspond to maximum cuts?)
7. For any flow network G and any vertices u and v , let $bottleneck_G(u, v)$ denote the maximum, over all paths π in G from u to v , of the minimum-capacity edge along π .
- (a) Describe and analyze an algorithm to compute $bottleneck_G(s, t)$ in $O(E \log V)$ time.
- (b) Describe an algorithm to construct a spanning tree T of G such that $bottleneck_T(u, v) = bottleneck_G(u, v)$ for all vertices u and v . (Edges in T inherit their capacities from G .)
8. Describe an efficient algorithm to determine whether a given flow network contains a *unique* maximum flow.
9. Suppose you have already computed a maximum flow f^* in a flow network G with **integer** edge capacities.
- (a) Describe and analyze an algorithm to update the maximum flow after the capacity of a single edge is increased by 1.
- (b) Describe and analyze an algorithm to update the maximum flow after the capacity of a single edge is decreased by 1.
- Both algorithms should be significantly faster than recomputing the maximum flow from scratch.
10. Let G be a network with integer edge capacities. An edge in G is *upper-binding* if increasing its capacity by 1 also increases the value of the maximum flow in G . Similarly, an edge is *lower-binding* if decreasing its capacity by 1 also decreases the value of the maximum flow in G .
- (a) Does every network G have at least one upper-binding edge? Prove your answer is correct.
- (b) Does every network G have at least one lower-binding edge? Prove your answer is correct.
- (c) Describe an algorithm to find all upper-binding edges in G , given both G and a maximum flow in G as input, in $O(E)$ time.

- (d) Describe an algorithm to find all lower-binding edges in G , given both G and a maximum flow in G as input, in $O(EV)$ time.
11. A given flow network G may have more than one minimum (s, t) -cut. Let's define the *best* minimum (s, t) -cut to be any minimum cut with the smallest number of edges.
- (a) Describe an efficient algorithm to determine whether a given flow network contains a *unique* minimum (s, t) -cut.
- (b) Describe an efficient algorithm to find the best minimum (s, t) -cut when the capacities are integers.
- (c) Describe an efficient algorithm to find the best minimum (s, t) -cut for *arbitrary* edge capacities.
- (d) Describe an efficient algorithm to determine whether a given flow network contains a unique *best* minimum (s, t) -cut.
12. A new assistant professor, teaching maximum flows for the first time, suggests the following greedy modification to the generic Ford-Fulkerson augmenting path algorithm. Instead of maintaining a residual graph, just reduce the capacity of edges along the augmenting path! In particular, whenever we saturate an edge, just remove it from the graph.

```

GREEDYFLOW( $G, c, s, t$ ):
  for every edge  $e$  in  $G$ 
     $f(e) \leftarrow 0$ 

  while there is a path from  $s$  to  $t$ 
     $\pi \leftarrow$  an arbitrary path from  $s$  to  $t$ 
     $F \leftarrow$  minimum capacity of any edge in  $\pi$ 
    for every edge  $e$  in  $\pi$ 
       $f(e) \leftarrow f(e) + F$ 
      if  $c(e) = F$ 
        remove  $e$  from  $G$ 
      else
         $c(e) \leftarrow c(e) - F$ 

  return  $f$ 

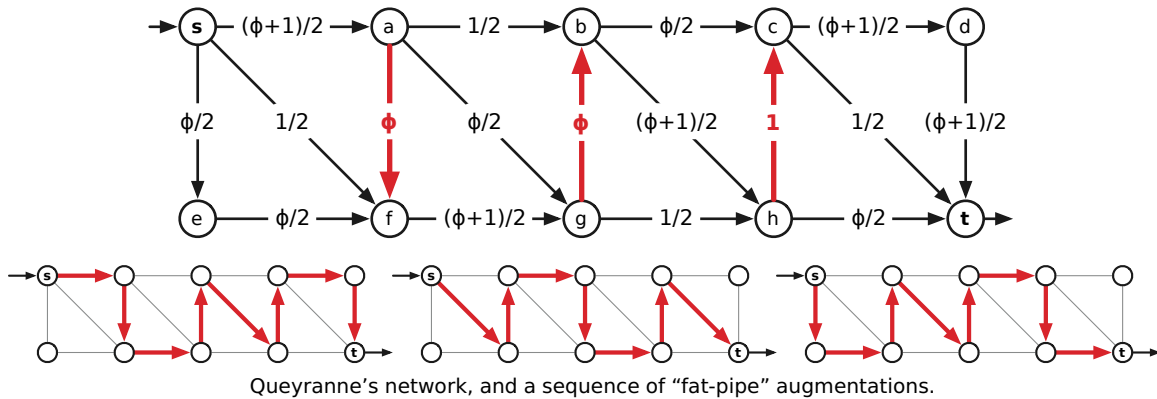
```

- (a) Show that this algorithm does *not* always compute a maximum flow.
- (b) Prove that for any flow network, if the Greedy Path Fairy tells you precisely which path π to use at each iteration, then GREEDYFLOW does compute a maximum flow. (Sadly, the Greedy Path Fairy does not actually exist.)
13. We can speed up the Edmonds-Karp 'fat pipe' heuristic, at least for integer capacities, by relaxing our requirements for the next augmenting path. Instead of finding the augmenting path with maximum bottleneck capacity, we find a path whose bottleneck capacity is at least half of maximum, using the following *capacity scaling* algorithm.

The algorithm maintains a bottleneck threshold Δ ; initially, Δ is the maximum capacity among all edges in the graph. In each *phase*, the algorithm augments along paths from s to t in which every edge has residual capacity at least Δ . When there is no such path, the phase ends, we set $\Delta \leftarrow \lfloor \Delta/2 \rfloor$, and the next phase begins.

- (a) How many phases will the algorithm execute in the worst case, if the edge capacities are integers?
- (b) Let f be the flow at the end of a phase for a particular value of Δ . Let S be the nodes that are reachable from s in the residual graph G_f using only edges with residual capacity at least Δ , and let $T = V \setminus S$. Prove that the capacity (with respect to G 's original edge capacities) of the cut (S, T) is at most $|f| + E \cdot \Delta$.
- (c) Prove that in each phase of the scaling algorithm, there are at most $2E$ augmentations.
- (d) What is the overall running time of the scaling algorithm, assuming all the edge capacities are integers?

14. In 1980 Maurice Queyranne published the following example of a flow network where Edmonds and Karp's "fat pipe" heuristic does not halt. Here, as in Zwick's bad example for the original Ford-Fulkerson algorithm, ϕ denotes the inverse golden ratio $(\sqrt{5} - 1)/2$. The three vertical edges play essentially the same role as the horizontal edges in Zwick's example.



(a) Show that the following infinite sequence of path augmentations is a valid execution of the Edmonds-Karp algorithm. (See the figure above.)

```

QUEYRANNEFATPIPES:
for  $i \leftarrow 1$  to  $\infty$ 
  push  $\phi^{3i-2}$  units of flow along  $s \rightarrow a \rightarrow f \rightarrow g \rightarrow b \rightarrow h \rightarrow c \rightarrow d \rightarrow t$ 
  push  $\phi^{3i-1}$  units of flow along  $s \rightarrow f \rightarrow a \rightarrow b \rightarrow g \rightarrow h \rightarrow c \rightarrow t$ 
  push  $\phi^{3i}$  units of flow along  $s \rightarrow e \rightarrow f \rightarrow a \rightarrow g \rightarrow b \rightarrow c \rightarrow h \rightarrow t$ 
forever
    
```

(b) Describe a sequence of $O(1)$ path augmentations that yields a maximum flow in Queyranne's network.

15. An (s, t) -series-parallel graph is an directed acyclic graph with two designated vertices s (the source) and t (the target or sink) and with one of the following structures:

- **Base case:** A single directed edge from s to t .
- **Series:** The union of an (s, u) -series-parallel graph and a (u, t) -series-parallel graph that share a common vertex u but no other vertices or edges.
- **Parallel:** The union of two smaller (s, t) -series-parallel graphs with the same source s and target t , but with no other vertices or edges in common.

Describe an efficient algorithm to compute a maximum flow from s to t in an (s, t) -series-parallel graph with arbitrary edge capacities.