CMSC 451: Lecture 12
Network Flows: Basic Concepts

Network Flow: The term "network flow" refers to a variety of related graph optimization problems, which are of fundamental value. In general, we are given a *flow network*, which is essentially a directed graph with nonnegative edge weights. We can think of the edges as "pipes" that are capable of carrying some sort of "stuff." In applications, this stuff can be any measurable quantity, such as fluid, megabytes of network traffic, commodities, currency, and so on. Each edge of the network has a given *capacity*, that limits the amount of stuff it is able to carry. The idea is to find out how much flow we can push from a designated source node to a designated sink node.

Although the network flow problem is defined in terms of the metaphor of pushing fluids, this problem and its many variations find remarkably diverse applications. These are often studied in the area of operations research. The network flow problem is also of interest because it is a restricted version of a more general optimization problem, called *linear programming*. A good understanding of network flows is helpful in obtaining a deeper understanding of linear programming.

Flow Networks: A *flow network* is a directed graph $G = (V, E)$ in which each edge $(u, v) \in E$ has a nonegative *capacity* $c(u, v) \geq 0$. (In our book, the capacity of edge e is denoted by c_e.) If $(u, v) \notin E$ we model this by setting $c(u, v) = 0$. There are two special vertices: a *source* s, and a *sink* t (see Fig. 1).

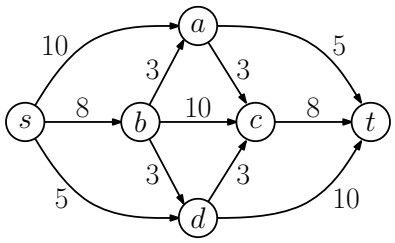


Fig. 1: A flow network.

We assume that there is no edge entering s and no edge leaving t. Such a network is sometimes called an *s-t network*. We also assume that every vertex lies on some path from the source to the sink.[1] This implies that $m \geq n-1$, where $n = |V|$ and $m = |E|$. It will also be convenient to assume that all capacities are integers. (We can assume more generally that the capacities are rational numbers, since we can convert them to integers by multiplying them by the least common multiple of the denominators.)

Flows, Capacities, and Conservation: Given an s-t network, a *flow* (also called an *s-t flow*) is a function f that maps each edge to a nonnegative real number and satisfies the following properties:

[1] Neither of these is an essential requirement. Given a network that fails to satisfy these assumptions, we can easily generate an equivalent one that satisfies both.

Capacity Constraint: For all $(u, v) \in E$, $f(u, v) \leq c(u, v)$.

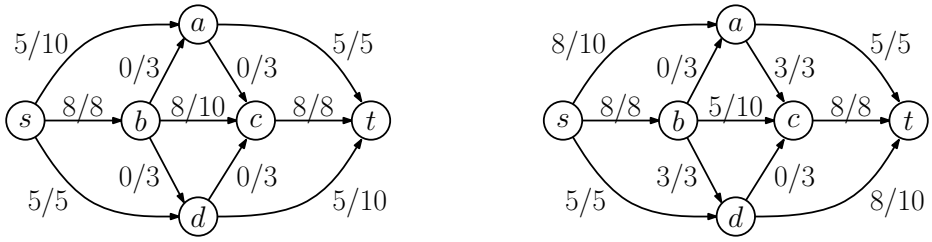
Flow conservation (or flow balance): For all $v \in V \setminus \{s, t\}$, the sum of flow along edges into v equals the sum of flows along edges out of v.

We can state flow conservation more formally as follows. First off, let us make the assumption that if (u, v) is *not* an edge of E, then $f(u, v) = 0$. We then define the total flow into v and total flow out of v as:

$$f^{\text{in}}(v) = \sum_{(u,v)\in E} f(u, v) \qquad \text{and} \qquad f^{\text{out}}(v) = \sum_{(v,w)\in V} f(v, w).$$

Then flow conservation states that $f^{\text{in}}(v) = f^{\text{out}}(v)$, for all $v \in V \setminus \{s, t\}$. Note that flow conservation *does not* apply to the source and sink, since we think of ourselves as pumping flow from s to t.

Two examples are shown in Fig. 2, where we use the notation f/c on each edge to denote the flow f and capacity c for this edge.



A valid flow ($|f| = 18$) (a)

A maximum flow ($|f| = 21$) (b)

Fig. 2: A valid flow and a maximum flow.

The quantity $f(u, v)$ is called the *flow* along edge (u, v). We are interested in defining the total flow, that is, the total amount of fluid flowing from s to t. The *value* of a flow f, denoted $|f|$, is defined as the sum of flows out of s, that is,

$$|f| = f^{\text{out}}(s) = \sum_{w\in V} f(s, w),$$

(For example, the value of the flow shown in Fig. 2(a) is 5+8+5 = 18.) From flow conservation, it follows easily that this is also equal to the flow into t, that is, $f^{\text{in}}(t)$. We will prove this later.

Maximum Flow: Given an s-t network, an obvious optimization problem is to determine a flow of maximum value. More formally, the *maximum-flow problem* is, given a flow network $G = (V, E)$, and source and sink vertices s and t, find the flow of maximum value from s to t. (For example, in Fig. 2(b) we show flow of value $8 + 8 + 5 = 21$, which can be shown to be the maximum flow for this network.) Note that, although the value of the maximum flow is unique, there may generally be many different flow functions that achieve this value.

This problem has been the subject of a great deal of study. Here is a history of some of some of the highlight results. Recall that $n = |V|$ and $m = |E|$, and let C denote the sum of the edge capacities (assumed to be integers).

- Ford-Fulkerson (1956) - Did not analyze the running time, but roughly $O((n+m)C)$. The same Ford of DP fame and the Bellman-Ford algorithm.
- Dinitz (1970) - $O(n^2m)$. This is often called "Dinic's algorithm", due to a misspelling of the name.
- Edmonds-Karp (1972) - $O(nm^2)$. This incremental algorithm can also be used to compute optimal weighted matchings of all sizes in bipartite graphs. Richard Karp is famous for being one of the founders of NP-completeness (with Stephen Cook).
- Gabow (1985) - $O(nm \log C)$. This paper introduced an interesting optimization technique called *scaling*, where problems are solved at increasingly higher levels of accuracy.
- Goldberg-Tarjan (1986) - $O(nm \log(n^2/m))$ time. This is considered the asymptotically fastest algorithm for Network Flows. This is the same Tarjan known for DFS, Union-Find, and the most efficient version of Dijkstra's algorithm.

Path-Based Flows: The definition of flow we gave above is sometimes call the *edge-based* definition of flows. An alternative, but mathematically equivalent, definition is called the *path-based* definition of flows. Define an *s-t path* to be any simple path from s to t. For example, in Fig. 1, $\langle s, a, t\rangle$, $\langle s, b, a, c, t\rangle$ and $\langle s, d, c, t\rangle$ are all examples of s-t paths. There may generally be an exponential number of such paths (but that is alright, since this just a mathematical definition).

A *path-based flow* is a function that assigns each s-t path a nonnegative real number such that, for every edge $(u, v) \in E$, the sum of the flows on all the paths containing this edge is at most $c(u, v)$. Note that there is no need to provide a flow conservation constraint, because each path that carries a flow into a vertex (excluding s and t), carries an equivalent amount of flow out of that vertex. For example, in Fig. 3(b) we show a path-based flow that is equivalent to the edge-based flow of Fig. 3(a). The paths carrying zero flow are not shown.

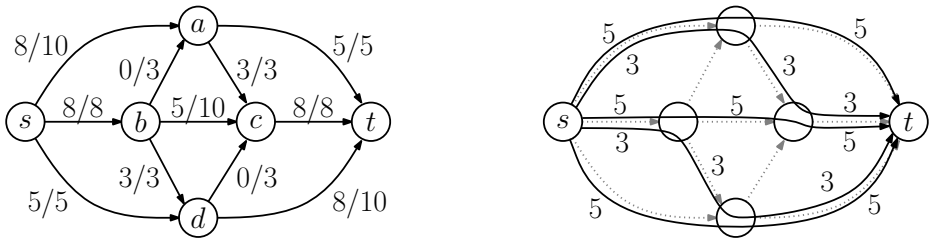


Fig. 3: (a) An edge-based flow and (b) its path-based equivalent.

The *value* of a path-based flow is defined to be the total sum of all the flows on all the s-t paths of the network. Although we will not prove it, the following claim is an easy consequence of the above definitions.

Claim: Given an s-t network G, under the assumption that there are no edges entering s or leaving t, G has an edge-based flow of value x if and only if G has a path-based flow of value x.

Multi-source, multi-sink networks: It may seem overly restrictive to require that there is only a single source and a single sink vertex. Many flow problems have situations in which many source vertices $s_1, \ldots, s_k$ and many sink vertices $t_1, \ldots, t_l$. This can easily be modeled by just adding a special *super-source* s' and a *super-sink* t', and attaching s' to all the s_i and attach all the t_j to t'. We let these edges have infinite capacity (see Fig. 4). Now by pushing the maximum flow from s' to t' we are effectively producing the maximum flow from all the s_i's to all the t_j's.

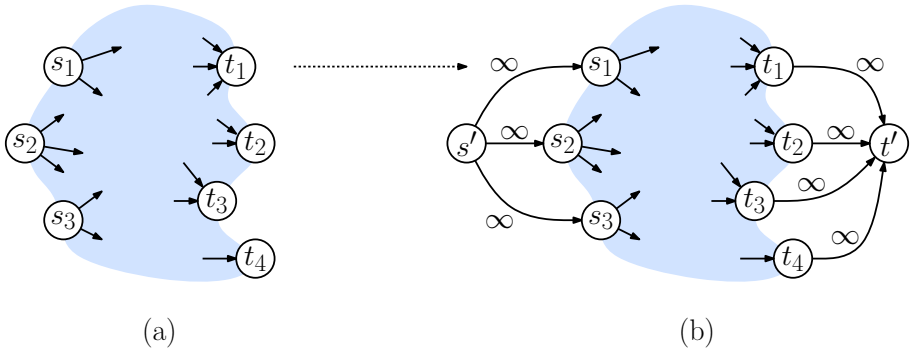


Fig. 4: Reduction from (a) multi-source/multi-sink to (b) single-source/single-sink.

Note that we don't assume any correspondence between flows leaving source s_i and entering t_j. Flows from one source may flow into *any* sink vertex. In some cases, you would like to specify the flow from a certain source must arrive at a designated sink vertex. For example, imagine that the sources are manufacturing production centers and sinks are retail outlets, and you are told the amount of commodity from s_i to arrive at t_j. This variant of the flow problem, called the *multi-commodity flow problem*, is a much harder problem to solve (in fact, some formulations are NP-hard).

Why Greedy Fails: Before considering algorithms for solving network flows, let's first consider why a simple greedy scheme for computing the maximum flow fails. The idea behind the greedy algorithm is motivated by the path-based notion of flow. (Recall this from the previous lecture.) Initially the flow on each edge is set to zero. Next, find any path P from s to t, such that the edge capacities on this path are all strictly positive. Let $c_{\min}$ be the minimum capacity of any edge on this path. This quantity is called the *bottleneck capacity* of the path. Push $c_{\min}$ units through this path. For each edge $(u, v) \in P$, set $f(u, v) \leftarrow c_{\min} + f(u, v)$, and decrease the capacity of (u, v) by $c_{\min}$. Repeat this until no s-t path (of positive capacity edges) remains in the network.

While this may seem to be a very reasonable algorithm, and will generally produce a valid flow, it may fail to compute the maximum flow. To see why, consider the network shown in Fig. 5(a). Suppose we push 5 units along the topmost path, 8 units along the middle path, and 5 units along the bottommost path. We have a flow of value 18. After adjusting the

capacities (see Fig. 5(b)) we see that there is no path of positive capacity from s to t. Thus, greedy gets stuck.

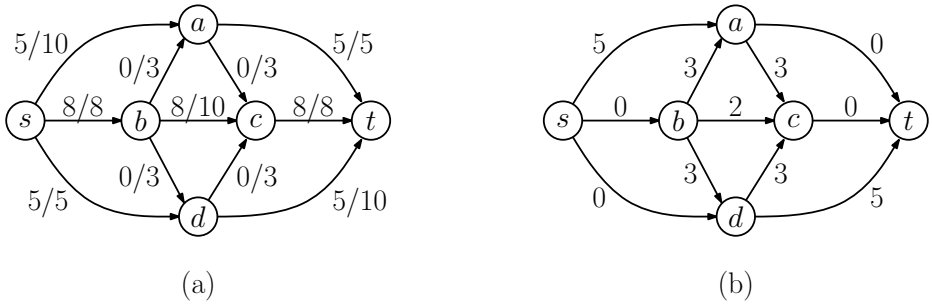


Fig. 5: The greedy flow algorithm can get stuck before finding the maximum flow.

Residual Network: The key insight to overcoming the problem with the greedy algorithm is to observe that, in addition to increasing flows on edges, it is possible to *decrease* flows on edges that already carry flow (as long as the flow never becomes negative). It may seem counterintuitive that this would help, but we shall see that it is exactly what is needed to obtain an optimal solution.

To make this idea clearer, we first need to define the notion of the residual network and augmenting paths. Given a flow network G and a flow f, define the *residual network*, denoted G_f, to be a network having the same vertex set and same source and sink, and whose edges are defined as follows:

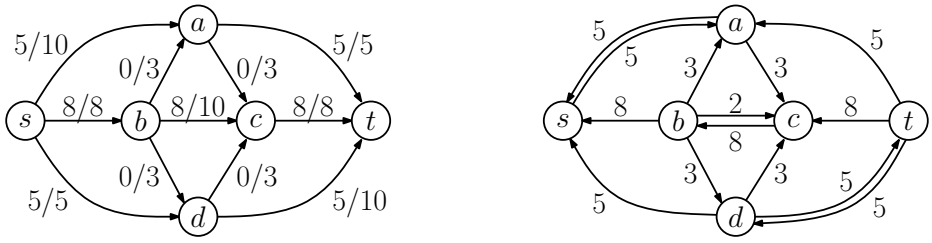
Forward edges: For each edge (u, v) for which $f(u, v) < c(u, v)$, create an edge (u, v) in G_f and assign it the capacity $c_f(u, v) = c(u, v) - f(u, v)$. Intuitively, this edge signifies that we can *increase* the flow along this edge by up to $c_f(u, v)$ units without violating the original capacity constraint.

Backward edges: For each edge (u, v) for which $f(u, v) > 0$, create an edge (v, u) in G_f and assign it a capacity of $c_f(v, u) = f(u, v)$. Intuitively, this edge signifies that we can *decrease* the existing flow by as much as $c_f(u, v)$ units. Conceptually, by pushing positive flow along the reverse edge (v, u) we are decreasing the flow along the original edge (u, v).

Observe that every edge of the residual network has *strictly positive* capacity. (This will be important later on.) Note that each edge in the original network may result in the generation of up to two new edges in the residual network. Thus, the residual network is of the same asymptotic size as the original network.

An example of a flow and the associated residual network are shown in Fig. 6(a) and (b), respectively. For example, the edge (b, c) of capacity 2 signifies that we can add up to 2 more units of flow to edge (b, c) and the edge (c, b) of capacity 8 signifies that we can cancel up to 8 units of flow from the edge (b, c).

The capacity of each edge in the residual network is called its *residual capacity*. The key observation about the residual network is that if we can push flow through the residual



(a): A flow f in network G (b): Residual network G_f

Fig. 6: A flow f and the residual network G_f.

network then we can push this additional amount of flow through the original network. This is formalized in the following lemma. Given two flows f and f', we define their *sum*, $f + f'$, in the natural way, by summing the flows along each edge. If $f'' = f + f'$, then $f''(u, v) = f(u, v) + f'(u, v)$. Clearly, the value of $f + f'$ is equal to $|f| + |f'|$.

Lemma: Let f be a flow in G and let f' be a flow in G_f. Then $(f + f')$ is a flow in G.

Proof: (Sketch) To show that the resulting flow is valid, we need to show that it satisfies both the capacity constraints and flow conservation. It is easy to see that the capacities of G_f were exactly designed so that any flow along an edge of G_f when added to the flow f of G will satisfy G's capacity constraints. Also, since both flows satisfy flow conservation, it is easy to see that their sum will as well. (More generally, any linear combination $\alpha f + \beta f'$ will satisfy flow conservation.)

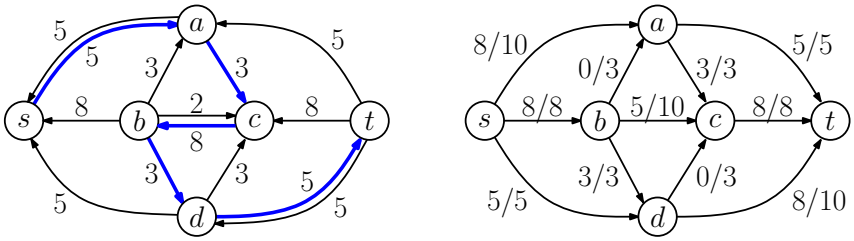
This lemma suggests that all we need to do to increase the flow is to find any flow in the residual network. This leads to the notion of an augmenting path.

Augmenting Paths and Ford-Fulkerson: Consider a network G, let f be a flow in G, and let G_f be the associated residual network. An *augmenting path* is a simple path P from s to t in G_f. The *residual capacity* (also called the *bottleneck capacity*) of the path is the minimum capacity of any edge on the path. It is denoted $c_f(P)$. (Recall that all the edges of G_f are of strictly positive capacity, so $c_f(P) > 0$.) By pushing $c_f(P)$ units of flow along each edge of the path, we obtain a valid flow in G_f, and by the previous lemma, adding this to f results in a valid flow in G of strictly higher value.

For example, in Fig. 7(a) we show an augmenting path of capacity 3 in the residual network for the flow given earlier in Fig. 6. In (b), we show the result of adding this flow to every edge of the augmenting path. Observe that because of the backwards edge (c, b), we have decreased the flow along edge (b, c) by 3, from 8 to 5.

How is this different from the greedy algorithm? The greedy algorithm only increases flow on edges. Since an augmenting path may increase flow on a backwards edge, it may actually *decrease* the flow on some edge of the original network.

This observation naturally suggests an algorithm for computing flows of ever larger value. Start with a flow of weight 0, and then repeatedly find an augmenting path. Repeat this until no such path exists. This, in a nutshell, is the simplest and best known algorithm



(a): Augmenting path of capacity 3 (b): The flow after augmentation

Fig. 7: Augmenting path and augmentation.

for computing flows, called the *Ford-Fulkerson method*. (We do not call it an "algorithm," since the method of selecting the augmenting path is not specified. We will discuss various strategies in future lectures.) It is summarized in the code fragment below and an example is shown in Fig. 8.

Ford-Fulkerson Network Flow

```
ford-fulkerson-flow(G = (V, E, s, t)) {
    f = 0 (all edges carry zero flow)
    while (true) {
        G' = the residual-network of G for f
        if (G' has no s-t augmenting path)
            break                                   // no augmentations left
        P = any-augmenting-path of G'               // augmenting path
        c = minimum capacity edge of P              // augmentation amount
        augment f by adding c to the flow on every edge of P
    }
    return f
}
```

There are three issues to consider before declaring this a reasonable algorithm.

- How efficiently can we perform augmentation?
- How many augmentations might be required until converging?
- If no more augmentations can be performed, have we found the max-flow?

Let us consider first the question of how to perform augmentation. First, given G and f, we need to compute the residual network, G_f. This is easy to do in $O(n+m)$ time, where $n = |V|$ and $m = |E|$. We assume that G_f contains only edges of strictly positive capacity. Next, we need to determine whether there exists an augmenting path from s to t G_f. We can do this by performing either a DFS or BFS in the residual network starting at s and terminating as soon (if ever) t is reached. Let P be the resulting path. Clearly, this can be done in $O(n+m)$ time as well. Finally, we compute the minimum cost edge along P, and increase the flow f by this amount for every edge of P.

Two questions remain: What is the best way to select the augmenting path, and is this correct in the sense of converging to the maximum flow? Next, we consider the issue of correctness.

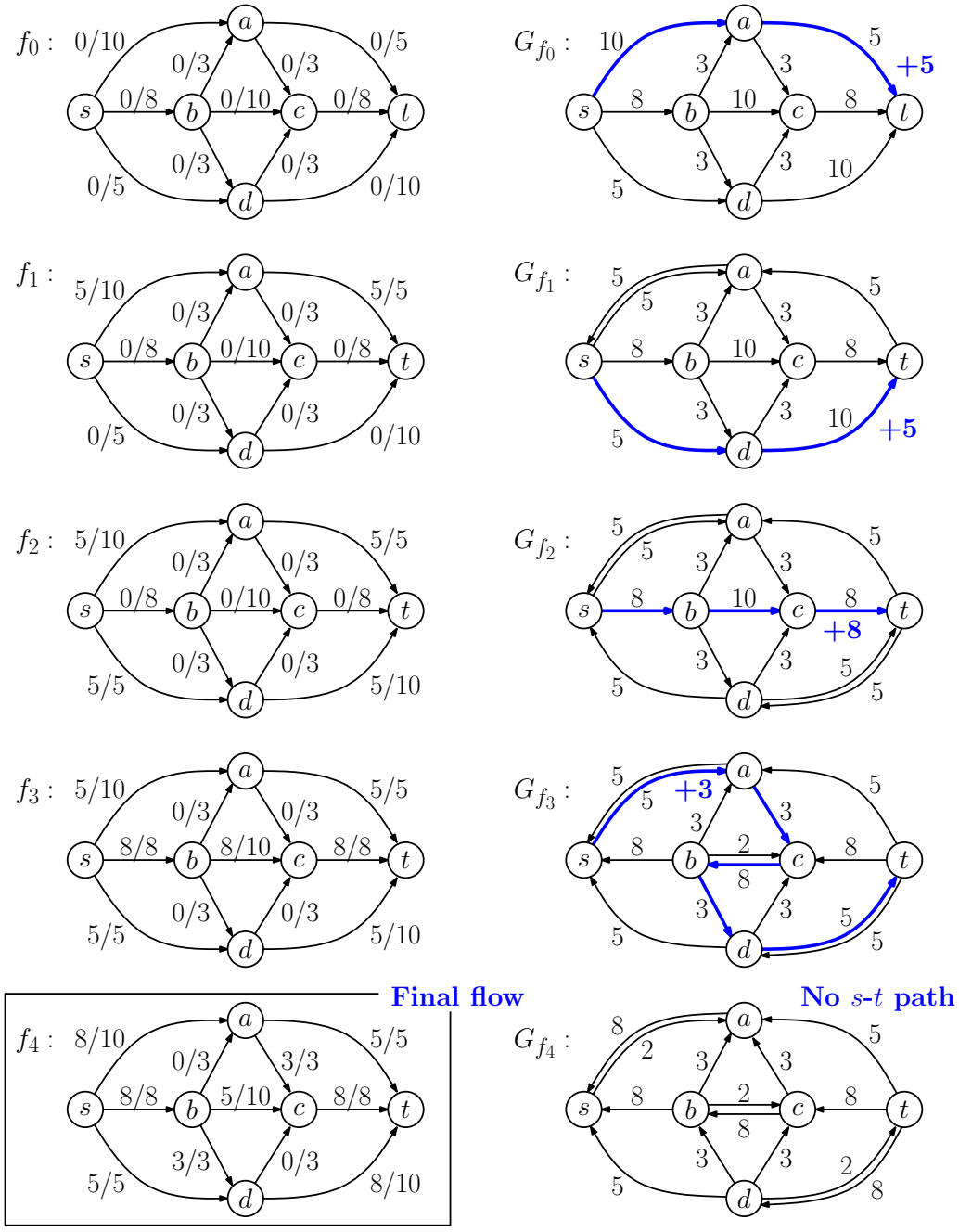


Fig. 8: The Ford-Fulkerson Algorithm. The final total flow is $5 + 5 + 8 + 3 = 21$.

Before doing this, we will need to introduce the concept of a cut.

Cuts: In order to show that Ford-Fulkerson leads to the maximum flow, we need to formalize the notion of a "bottleneck" in the network. Intuitively, the flow cannot be increased forever, because there is some subset of edges, whose capacities eventually become saturated with flow. Every path from s to t must cross one of these saturated edges, and so the sum of capacities of these edges imposes an upper bound on size of the maximum flow. Thus, these edges form a bottleneck.

We want to make this concept mathematically formal. Since such a set of edges lie on every path from s from t, their removal defines a partition separating the vertices that s can reach from the vertices that s cannot reach. This suggests the following concept.

Given a network G, define a *cut* (also called an *s-t cut*) to be a partition of the vertex set into two disjoint subsets X and Y (that is, $X \cup Y = V$ and $X \cap Y = \emptyset$) such that $s \in X$ and $t \in Y$. Define the *capacity* of the cut (X, Y) to be the sum of the capacities of the edges leading from X to Y, that is,

$$c(X, Y) \;=\; \sum_{x \in X} \sum_{y \in Y} c(x, y).$$

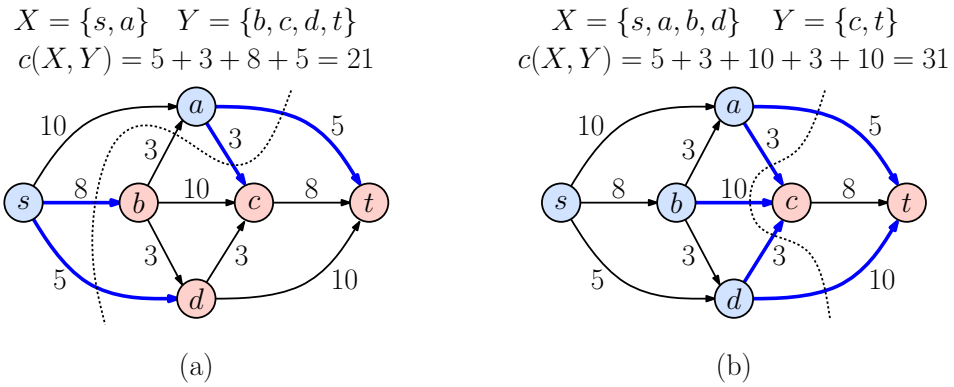


Fig. 9: (a) A cut $X = \{s, a\}$, $Y = \{b, c, d, t\}$) of capacity $5 + 3 + 8 + 5 = 21$ and (b) another cut $X = \{s, a, b, d\}$, $Y = \{c, t\}$ of capacity $5 + 3 + 10 + 3 + 10 = 31$.

Given a flow f in G, define the *net flow* across the cut (X, Y) to be the sum of flows from X to Y minus the sum of flows from Y to X, that is,

$$f(X, Y) \;=\; \sum_{x \in X} \sum_{y \in Y} f(x, y) - \sum_{y \in Y} \sum_{x \in X} f(y, x)$$

(see Fig. 10). Observe that $f(X, Y) = -f(Y, X)$.

There is a notable asymmetry between net flows and cut capacities. The capacity considers only edges directed from X to Y, whereas the net flow considers both directions. Fig. 10 shows that the net flow across two different cuts is the same, even though these cuts have different capacities. The following lemma shows that this is generally true for all cuts.

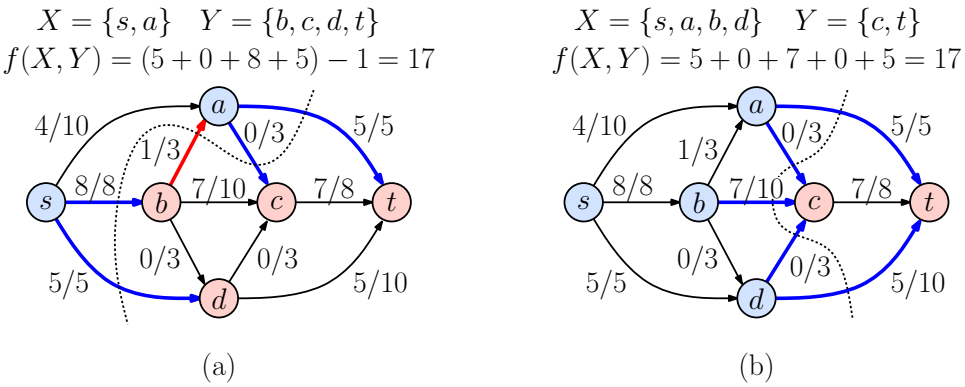


Fig. 10: The flow f across both cuts has the same net flow value, which is equal to $|f| = 17$.

Lemma: Given any network G, any flow f, and any cut (X, Y), the flow value is equal to the net flow across the cut, that is, $f(X, Y) = |f|$.

Proof: Recall that there are no edges leading into s, and so we have $|f| = f^{\text{out}}(s) = f^{\text{out}}(s) - f^{\text{in}}(s)$. Since all the other nodes of X must satisfy flow conservation it follows that

$$|f| = \sum_{x \in X} (f^{\text{out}}(x) - f^{\text{in}}(x))$$

Now, observe that every edge (u, v) where both u and v are in X contributes one positive term and one negative term of value $f(u, v)$ to the above sum, and so all of these cancel out. The only terms that remain are the edges that either go from X to Y (which contribute positively) and those from Y to X (which contribute negatively). Thus, it follows that the value of the sum is exactly $f(X, Y)$, and therefore $|f| = f(X, Y)$.

In the net flow, we consider both the flows from X to Y and (negated) from Y to X. In contrast, in the definition of cut capacity, we only consider capacities from X to Y. To understand why this asymmetry exists, suppose that there there are two edges (x, y) and (y, x), where $x \in X$ and $y \in Y$, where the edges have the same capacities and both carry the same flow. The flow from y to x effectively cancels out the flow from x to y (as if you have an "eddy" in the middle of a river, where the flow cycles around). Thus, this cycle does not contribute to the total flow value, and so it is important to consider both in the net flow. On the other hand, the capacity of the edge from y to x does not contribute to nor detract from the maximum amount of flow we can push from the X-side to the Y-side of the cut. Therefore, we do not need to consider it in computing the cut's total capacity.

It is easy to see that it is not possible to push more flow through a cut than its capacity. Combining this with the above lemma we have:

Lemma: Given any network G, any flow f, and any cut (X, Y), the flow value cannot exceed the cut's capacity, that is, $|f| \leq c(X, Y)$.

The optimality of the Ford-Fulkerson method is based on the following famous theorem, called the *Max-Flow/Min-Cut Theorem*. Basically, it states that in any flow network the minimum

capacity cut acts like a bottleneck to limit the maximum amount of flow. The Ford-Fulkerson method terminates when it finds this bottleneck, and hence on termination, it finds both the minimum cut and the maximum flow.

Max-Flow/Min-Cut Theorem: The following three conditions are equivalent.

(i) f is a maximum flow in G,
(ii) The residual network G_f contains no augmenting paths,
(iii) $|f| = c(X, Y)$ for some cut (X, Y) of G.

Proof:

- (i) $\Rightarrow$ (ii): (by contradiction) If f is a max flow and there were an augmenting path in G_f, then by pushing flow along this path we would have a larger flow, a contradiction.
- (ii) $\Rightarrow$ (iii): If there are no augmenting paths then s and t are not connected in the residual network (see Fig. 11(a)). Let X be those vertices reachable from s in the residual network, and let Y be the rest. Since there is no augmenting path, $t \in Y$, which implies that (X, Y) forms a cut (see Fig. 11(b)). This also implies that each forward edge $(x, y) \in X \times Y$ is saturated, meaning that $f(x, y) = c(x, y)$ (blue edges in Fig. 11(c)), and each backward edge $(y, x) \in Y \times X$ carries no flow, that is, $f(y, x) = 0$ (red edges in Fig. 11(c)). It follows that the flow across this cut equals the capacity of the cut, that is, $f(X, Y) = c(X, Y)$. By the previous lemma $|f| = f(X, Y)$, and hence we have $|f| = c(X, Y)$.

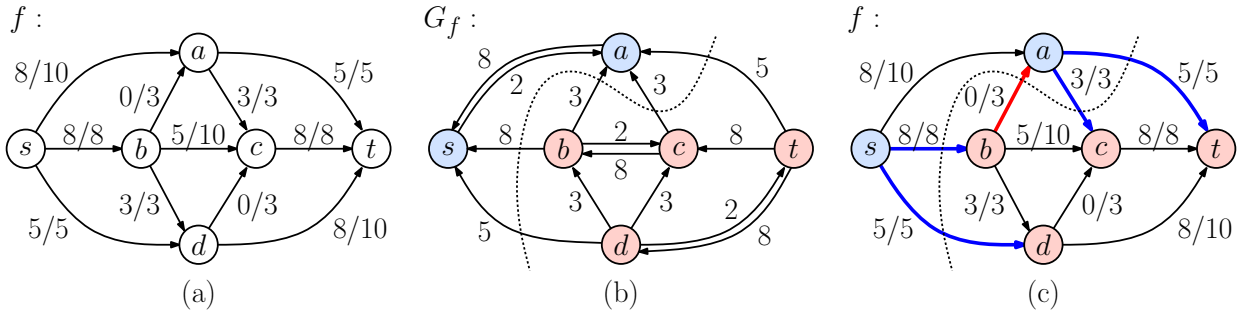


Fig. 11: Proof of the Max-Flow/Min-Cut Theorem.

- (iii) $\Rightarrow$ (i): This follows directly from the previous lemma. No flow value can exceed any cut capacity, and so if *any* flow's value matches *any* cut's capacity, this flow must be maximum.

We have established that, on termination, Ford-Fulkerson generates the maximum flow. But, is it guaranteed to terminate and, if so, how long does it take? We will consider this question next time.