1 Max flow

Consider a directed graph G with positive edge weights c that define the "capacity" of each edge. We can identify two special nodes in G: the source node s and the sink node t. We want to find a max flow from s to t (we will explain what that is in a bit).

Real-life example: Suppose we want to send trucks from city s to city t, and the capacity of each edge represents the number of trucks that can go on that road every hour. What is the maximum number of trucks that we can send from s to t every hour?

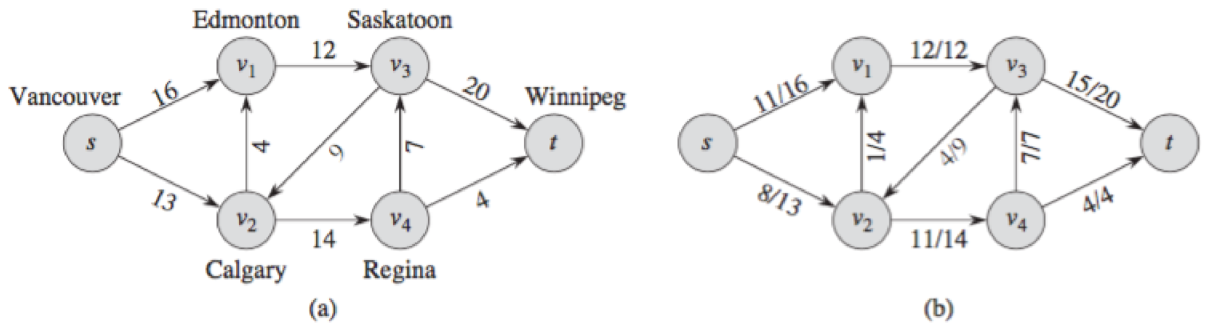

Figure 26.1 **(a)** A flow network $G = (V, E)$ for the Lucky Puck Company's trucking problem. The Vancouver factory is the source s, and the Winnipeg warehouse is the sink t. The company ships pucks through intermediate cities, but only $c(u, v)$ crates per day can go from city u to city v. Each edge is labeled with its capacity. **(b)** A flow f in G with value $|f| = 19$. Each edge (u, v) is labeled by $f(u, v)/c(u, v)$. The slash notation merely separates the flow and capacity; it does not indicate division.

1.0.1 Formal definition of flow

A flow f is a function defined on the edges of a graph, where

1. For all edges, the flow on that edge is at least 0, and no greater than the capacity of the edge. That is, $0 \leq f(u, v) \leq c(u, v)$ for all edges (u, v).
2. For all vertices (other than s and t), the flow into the vertex has to equal the flow out of the vertex. That is,
$$\sum_{x \in N_{in}(v)} f(x, v) = \sum_{y \in N_{out}(v)} f(v, y)$$
where N_{in} is the set of nodes that have edges pointing to v, and N_{out} is the set of nodes that v points to.

In the truck example, this means that we cannot have any more trucks on a road than the road can hold, and the number of trucks going into a city has to equal the number of trucks coming out of the city (which makes sense because otherwise the city would run out of trucks or the trucks would build up).

The value $|f|$ of the flow is defined as the total flow coming out of the source (which is the same as the total flow coming into the sink). That is,

$$|f| = \sum_{x \in N_{out}(s)} f(s,x) - \sum_{y \in N_{in}(s)} f(y,s)$$

In the max-flow problem, our goal is to find the flow with the maximum value. That is, we want to find the best way to send trucks from s to t.

2 Max flow / min cut

An s-t cut is a partition of the vertices into two disjoint sets S and T, such that $s \in S$ and $t \in T$. The size (or cost) of the cut is the sum of the capacities of the edges:

$$||S,T|| = \sum_{x \in S, y \in T} c(x,y)$$

Note that when computing the size of a cut, we only consider edges going from S to T, not the other way around.

The min s-t cut is the s-t cut with the smallest size.

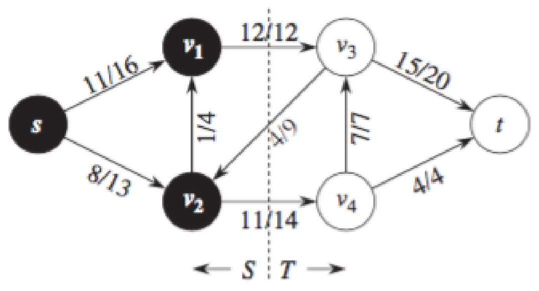

Figure 26.5 A cut (S,T) in the flow network of Figure 26.1(b), where $S = \{s, v_1, v_2\}$ and $T = \{v_3, v_4, t\}$. The vertices in S are black, and the vertices in T are white. The net flow across (S,T) is $f(S,T) = 19$, and the capacity is $c(S,T) = 26$.

2.1 Max flow min cut lemma

Theorem: For any graph G, source s, and destination t, the value of the max flow is equal to the size of the min cut.

First we will prove that the value of the max flow is less than or equal to the size of the min cut. Then we will find an algorithm that finds a flow whose value is equal to the size of the min cut, proving the theorem.

Lemma: For any flow f and any s-t cut (S, T), we have $|f| \leq ||S, T||$. (This proves that the value of the max flow is less than or equal to the size of the min cut.)

Proof: By definition, we have

$$|f| = \sum_{x \in N_{out}(s)} f(s, x) - \sum_{y \in N_{in}(s)} f(y, s)$$

To simplify this expression, we can add a bunch of extraneous terms that sum to zero, to get

$$\sum_{v \in S} \left(\sum_{x \in N_{out}(v)} f(v, x) - \sum_{y \in N_{in}(v)} f(y, v) \right)$$

Now we can separate out vertices in S and T, getting

$$\sum_{v \in S} \left(\sum_{x \in N_{out}(v) \cap S} f(v, x) - \sum_{y \in N_{in}(v) \cap S} f(y, v) \right) + \sum_{v \in S} \left(\sum_{x \in N_{out}(v) \cap T} f(v, x) - \sum_{y \in N_{in}(v) \cap T} f(y, v) \right)$$

Observe that the first two sums cancel out, because we are summing over all vertices in S, and each edge from a vertex in S to another vertex in S is represented exactly twice: once in the term $\sum_{v \in S} \sum_{x \in N_{out}(v) \cap S} f(v, x)$, and once in the term $\sum_{v \in S} \sum_{x \in N_{in}(v) \cap S} f(y, v)$. So this actually just equals

$$\sum_{v \in S} \left(\sum_{x \in N_{out}(v) \cap T} f(v, x) - \sum_{y \in N_{in}(v) \cap T} f(y, v) \right)$$

which is less than or equal to

$$\sum_{v \in S, x \in N_{out}(v) \cap T} f(v, x)$$

which is less than or equal to

$$\sum_{v \in S, x \in N_{out}(v) \cap T} c(v, x) = ||S, T||$$

3 Ford Fulkerson

This is the method we will use to find a flow whose value equals the value of a cut (i.e. the max flow). Key to this method is the notion of a "residual graph", which is based on the flow that is currently going through the graph, and says how much more flow can be pushed through the graph on each edge.

3.1 Residual networks

Specifically, the residual network G_f is defined as follows:

Assume that our graph does not have edges pointing in opposite directions between the same two vertices. That is, if (u, v) is an edge, (v, u) must not be an edge. (If (u, v) and (v, u) are both edges, we can split the edge (u, v) by introducing a new node x, and replacing (u, v) with edges (u, x) and (x, v). This makes the number of nodes at most $m + n$.)

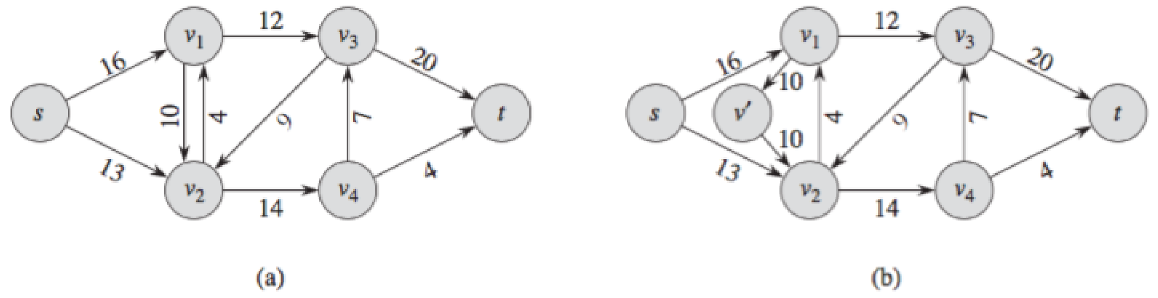

Figure 26.2 Converting a network with antiparallel edges to an equivalent one with no antiparallel edges. **(a)** A flow network containing both the edges (v_1, v_2) and (v_2, v_1). **(b)** An equivalent network with no antiparallel edges. We add the new vertex v', and we replace edge (v_1, v_2) by the pair of edges (v_1, v') and (v', v_2), both with the same capacity as (v_1, v_2).

Let the residual capacity c_f be defined on each edge as

$$c_f(u, v) = \begin{cases} c(u, v) - f(u, v) & \text{if } (u, v) \text{ is an edge in the original graph} \\ f(v, u) & \text{if } (v, u) \text{ is an edge in the original graph} \\ 0 & \text{otherwise} \end{cases}$$

The logic behind this definition is that if $u \to v$ is an edge in the original graph, and we are only pushing $f(u, v)$ units of flow through it right now, we can potentially push $c(u, v) - f(u, v)$ more units through that edge. However, if the edge points from $v \to u$, we can push $f(u, v)$ more units in the $u \to v$ direction by just removing the flow in the $v \to u$ direction.

We define G_f as the graph that contains the same vertices as the original graph, but only contains the edges where $c_f > 0$.

3.2 Ford Fulkerson approach

First we will show that if t is not reachable from s in G_f, then f is a maximum flow. Then, if G_f has a path from s to t, we will show how to "augment" f to get a flow f' with greater value.

Based on this, we know that we can find a maximum flow by repeatedly augmenting the flow until we can no longer find a path in the residual graph from s to t.

3.2.1 If t is not reachable from s in G_f, then f is a maximum flow

Proof: Define an s-t cut $S \cup T$ in the original graph G, where S is the set of nodes that are reachable from s in G_f, and T is the set of nodes that are not. Observe that there are no edges in G_f between nodes in S and nodes in T, because otherwise one of the nodes in T would be reachable from s. So for all $u \in S, v \in T$, we have $c_f(u, v) = 0$.

This means there are three cases:

1. If (u, v) is an edge in the original graph, then $c_f(u, v) = c(u, v) - f(u, v) = 0$, so $c(u, v) = f(u, v)$.
2. If (v, u) is an edge in the original graph, then $c_f(u, v) = f(v, u) = 0$.
3. If neither (u, v) nor (v, u) are edges, we can completely disregard this pair of vertices since they do not appear in any flow or cut.

Therefore,

$$\begin{aligned}
|f| &= \sum_{u \in S} \left(\sum_{x \in N_{out}(u) \cap T} f(u, x) - \sum_{y \in N_{in}(u) \cap T} f(y, u) \right) \quad \text{(from the previous proof)} \\
&= \sum_{u \in S} \left(\sum_{x \in N_{out}(u) \cap T} c(u, x) - \sum_{y \in N_{in}(u) \cap T} 0 \right) \\
&= ||S, T||
\end{aligned}$$

Since the flow is equal to the cut, by the previous lemma, f must be a max flow.

3.2.2 If G_f has a path from s to t, we can augment the path

Let $P = x_0 \to x_1 \to \cdots \to x_k$ be the path from s to t in the residual graph (note that this may not necessarily correspond to a path in the real graph), and consider the edge of minimum capacity in the residual graph. Let F be the weight of this edge, i.e. $\min_i c_f(x_i, x_{i+1})$.

We may increase the overall flow in the graph by augmenting the flow by F. That is, if there is an edge from $x_i \to x_{i+1}$ in the original graph, we increase the flow on that edge by F. If there is an edge from x_{i+1} to x_i, we decrease the flow on that edge by F. Note that we can always do this because of the definition of capacity in the residual graph.

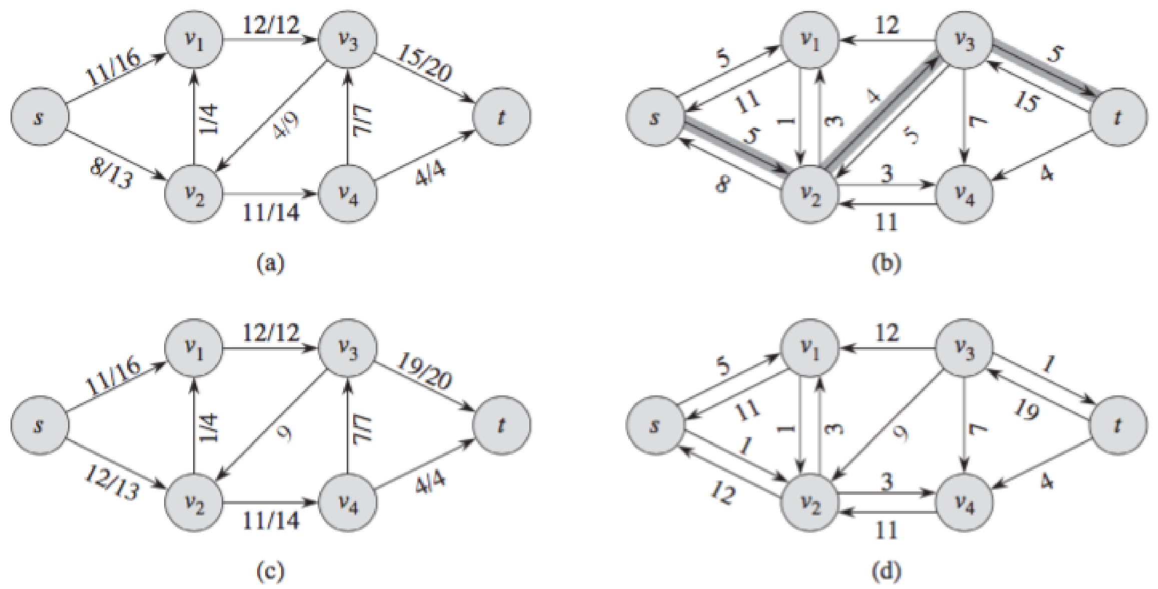

Figure 26.4 **(a)** The flow network G and flow f of Figure 26.1(b). **(b)** The residual network G_f with augmenting path p shaded; its residual capacity is $c_f(p) = c_f(v_2, v_3) = 4$. Edges with residual capacity equal to 0, such as (v_1, v_3), are not shown, a convention we follow in the remainder of this section. **(c)** The flow in G that results from augmenting along path p by its residual capacity 4. Edges carrying no flow, such as (v_3, v_2), are labeled only by their capacity, another convention we follow throughout. **(d)** The residual network induced by the flow in (c).

That is, define a new flow f' where

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

(Note that f' is only defined on edges that were in the original graph, so the $(v,u) \in P$ case is the case where the edge in the residual graph points in the opposite direction of the original edge, and the $(u,v) \in P$ case is the case where they are pointing in the same direction.)

We need to show that f' is a flow with greater value than f.

Capacity constraints:

1. If $(u,v) \in P$, then $0 \leq f(u,v) + F \leq f(u,v) + c_f(u,v) = f(u,v) + c(u,v) - f(u,v) = c(u,v)$.
2. If $(v,u) \in P$, then $f(u,v) - F \leq f(u,v) \leq c(u,v)$ and $f(u,v) - F \geq f(u,v) - c_f(u,v) = 0$.
3. Otherwise, $f'(u,v) = f(u,v)$, which is between 0 and $c(u,v)$.

Conservation constraints: Suppose that P is a simple path. Then for every vertex v other than s and t, v either touches 0 edges of P or 2 edges of P. If it touches 0 edges, the flow is conserved. If it touches 2 edges, then we consider four cases, depending on whether the edge in G_f is the same direction as the corresponding edge in G, or whether it is in the opposite direction.

Exercise: Verify that in all these cases, the flow is conserved.

Finally, $|f'| > f$ because we are increasing our flow by F, which is greater than 0.

3.3 Ford Fulkerson pseudocode

Therefore, we can create a maximum flow by repeatedly augmenting paths in the flow network until there is no longer any path from s to t in the residual network. We can find the path in the residual network using some path finding method, like breadth-first search.

```
FORD-FULKERSON($G, s, t$)
for each edge $(u, v) \in G.E$
    $(u, v).f = 0$
while there exists a path $p$ from $s$ to $t$ in the residual network $G_f$
    $c_f(p) = \min\{c_f(u, v) : (u, v) \text{ is in } p\}$
    for each edge $(u, v)$ in $p$
        if $(u, v) \in E$
            $(u, v).f = (u, v).f + c_f(p)$
        else $(v, u).f = (v, u).f - c_f(p)$
```

Runtime for arbitrary path finding method: If all the capacities are integers, we must augment the flow by at least 1 every time, so the runtime is bounded by $O(m|f|)$, where $|f|$ is the value of the maximum flow. However, this is inefficient when the flow is large (see the figure), or the capacities are not integers. In fact, if the capacities are irrational numbers, the algorithm is not guaranteed to terminate.

Good path finding method: However, if we choose the path using breadth-first search (each time choosing the shortest unweighted path from s to t in the residual network), the algorithm is guaranteed to run in polynomial time. Specifically, it runs in $O(nm^2)$ time. When the Ford Fulkerson method is implemented using breadth-first search, it is called the Edmonds-Karp algorithm.

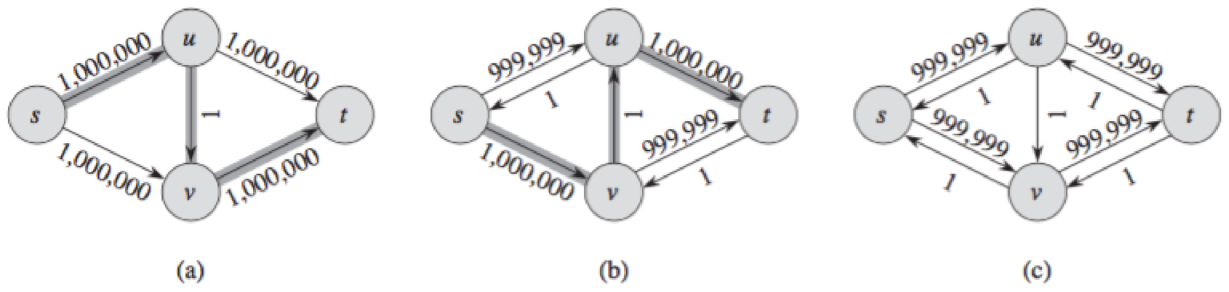

Figure 26.7 **(a)** A flow network for which FORD-FULKERSON can take $\Theta(E\,|f^*|)$ time, where f^* is a maximum flow, shown here with $|f^*| = 2{,}000{,}000$. The shaded path is an augmenting path with residual capacity 1. **(b)** The resulting residual network, with another augmenting path whose residual capacity is 1. **(c)** The resulting residual network.

4 Bipartite perfect matching

Let G be an undirected, unweighted bipartite graph. That is, we may partition the vertices into sets V_1 and V_2, where there are no edges with two endpoints entirely in V_1 or entirely in V_2. A matching in G is a collection of edges M, no two of which share an endpoint.

For example, G might represent a set of people, where the nodes in V_1 are women and the nodes in V_2 are men. A matching on G would pair women with men in such a way that no woman can be paired with more than one man, or vice versa.

A perfect matching is a matching that includes all the vertices of the graph. So in the example, all men and women would be paired.

Given a bipartite graph G with n nodes on each side, how can we tell if it has a perfect matching? We can solve the problem as follows:

Convert G into a directed graph, where all the edges between V_1 and V_2 now point from V_1 to V_2. Add a source s that points to all the nodes in V_1, and a sink t that all the nodes in V_2 point to. Let all the edge capacities be 1, and run the Ford Fulkerson method to compute the max flow. If the value of the max flow is n, then G has a perfect matching; otherwise it doesn't.

Proof: If G has a perfect matching M, then we can set all the edges in M to have flow 1, and all the edges out of s and into t also have flow 1. All other flow values are 0. This produces a flow of value n.

The flow conservation constraints are satisfied since for every $x \in V_1$ there is exactly one edge (s, x) into x that has flow 1, and exactly one edge $(x, y) \in M$ with flow 1. Similarly, for every $y \in V_2$ there is exactly one edge $(x, y) \in M$ into y with flow 1 and exactly one edge (y, t) out of y with flow 1. The capacity constraints are satisfied since all edges have flow ≤ 1.

Now suppose Ford-Fulkerson returns a flow of value n, so that $f(s, x) = f(y, t) = 1$ for all $x \in V_1$ and $y \in V_2$. Since Ford-Fulkerson causes all flow values on the edges to be integers (when the capacities are all integers), the flow values on all edges are either 1 or 0. Therefore, every node $x \in V_1$ gets a flow of 1 going into it and a flow of 1 needs to come out, so there is a single edge (x, y) that has flow value 1 and all other edges out of x have flow value 0. Similarly, for every $y \in V_2$ there is a unique edge into y with positive flow value 1. Then the edges in $V_1 \cup V_2$ with positive flow form a perfect matching.