# 26 Maximum Flow

Just as we can model a road map as a directed graph in order to find the shortest path from one point to another, we can also interpret a directed graph as a "flow network" and use it to answer questions about material flows. Imagine a material coursing through a system from a source, where the material is produced, to a sink, where it is consumed. The source produces the material at some steady rate, and the sink consumes the material at the same rate. The "flow" of the material at any point in the system is intuitively the rate at which the material moves. Flow networks can model many problems, including liquids flowing through pipes, parts through assembly lines, current through electrical networks, and information through communication networks.

We can think of each directed edge in a flow network as a conduit for the material. Each conduit has a stated capacity, given as a maximum rate at which the material can flow through the conduit, such as 200 gallons of liquid per hour through a pipe or 20 amperes of electrical current through a wire. Vertices are conduit junctions, and other than the source and sink, material flows through the vertices without collecting in them. In other words, the rate at which material enters a vertex must equal the rate at which it leaves the vertex. We call this property "flow conservation," and it is equivalent to Kirchhoff's current law when the material is electrical current.

In the maximum-flow problem, we wish to compute the greatest rate at which we can ship material from the source to the sink without violating any capacity constraints. It is one of the simplest problems concerning flow networks and, as we shall see in this chapter, this problem can be solved by efficient algorithms. Moreover, we can adapt the basic techniques used in maximum-flow algorithms to solve other network-flow problems.

This chapter presents two general methods for solving the maximum-flow problem. Section 26.1 formalizes the notions of flow networks and flows, formally defining the maximum-flow problem. Section 26.2 describes the classical method of Ford and Fulkerson for finding maximum flows. An application of this method, finding a maximum matching in an undirected bipartite graph, appears in Section 26.3. Section 26.4 presents the push-relabel method, which underlies many of the fastest algorithms for network-flow problems. Section 26.5 covers the "relabel-to-front" algorithm, a particular implementation of the push-relabel method that runs in time $O(V^3)$. Although this algorithm is not the fastest algorithm known, it illustrates some of the techniques used in the asymptotically fastest algorithms, and it is reasonably efficient in practice.

## 26.1 Flow networks

In this section, we give a graph-theoretic definition of flow networks, discuss their properties, and define the maximum-flow problem precisely. We also introduce some helpful notation.

### Flow networks and flows

A ***flow network*** $G = (V, E)$ is a directed graph in which each edge $(u, v) \in E$ has a nonnegative ***capacity*** $c(u, v) \geq 0$. We further require that if $E$ contains an edge $(u, v)$, then there is no edge $(v, u)$ in the reverse direction. (We shall see shortly how to work around this restriction.) If $(u, v) \notin E$, then for convenience we define $c(u, v) = 0$, and we disallow self-loops. We distinguish two vertices in a flow network: a ***source*** $s$ and a ***sink*** $t$. For convenience, we assume that each vertex lies on some path from the source to the sink. That is, for each vertex $v \in V$, the flow network contains a path $s \leadsto v \leadsto t$. The graph is therefore connected and, since each vertex other than $s$ has at least one entering edge, $|E| \geq |V| - 1$. Figure 26.1 shows an example of a flow network.

We are now ready to define flows more formally. Let $G = (V, E)$ be a flow network with a capacity function $c$. Let $s$ be the source of the network, and let $t$ be the sink. A ***flow*** in $G$ is a real-valued function $f : V \times V \to \mathbb{R}$ that satisfies the following two properties:

**Capacity constraint:** For all $u, v \in V$, we require $0 \leq f(u, v) \leq c(u, v)$.

**Flow conservation:** For all $u \in V - \{s, t\}$, we require

$$\sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v) .$$

When $(u, v) \notin E$, there can be no flow from $u$ to $v$, and $f(u, v) = 0$.

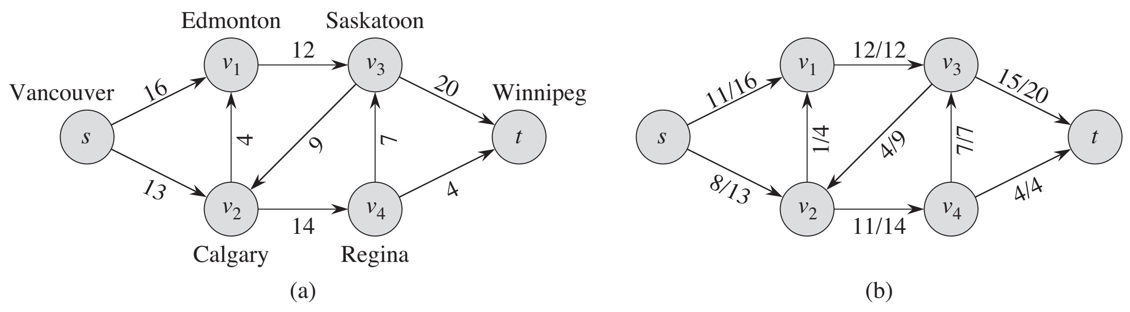

**Figure 26.1** **(a)** A flow network $G = (V, E)$ for the Lucky Puck Company's trucking problem. The Vancouver factory is the source $s$, and the Winnipeg warehouse is the sink $t$. The company ships pucks through intermediate cities, but only $c(u, v)$ crates per day can go from city $u$ to city $v$. 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.

We call the nonnegative quantity $f(u, v)$ the flow from vertex $u$ to vertex $v$. The ***value*** $|f|$ of a flow $f$ is defined as

$$|f| = \sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, s) , \tag{26.1}$$

that is, the total flow out of the source minus the flow into the source. (Here, the $|\cdot|$ notation denotes flow value, not absolute value or cardinality.) Typically, a flow network will not have any edges into the source, and the flow into the source, given by the summation $\sum_{v \in V} f(v, s)$, will be 0. We include it, however, because when we introduce residual networks later in this chapter, the flow into the source will become significant. In the ***maximum-flow problem***, we are given a flow network $G$ with source $s$ and sink $t$, and we wish to find a flow of maximum value.

Before seeing an example of a network-flow problem, let us briefly explore the definition of flow and the two flow properties. The capacity constraint simply says that the flow from one vertex to another must be nonnegative and must not exceed the given capacity. The flow-conservation property says that the total flow into a vertex other than the source or sink must equal the total flow out of that vertex—informally, "flow in equals flow out."

## An example of flow

A flow network can model the trucking problem shown in Figure 26.1(a). The Lucky Puck Company has a factory (source $s$) in Vancouver that manufactures hockey pucks, and it has a warehouse (sink $t$) in Winnipeg that stocks them. Lucky

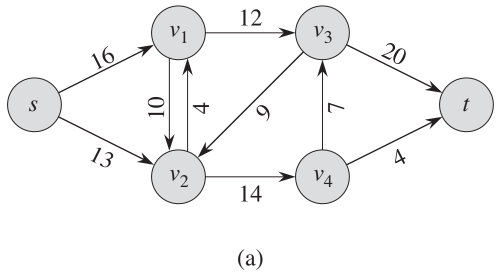

**Figure 26.2** Converting a network with antiparallel edges to an equivalent one with no antiparallel edges. **(a)** A flow network containing both the edges $(v_1, v_2)$ and $(v_2, v_1)$. **(b)** An equivalent network with no antiparallel edges. We add the new vertex $v'$, and we replace edge $(v_1,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

Puck leases space on trucks from another firm to ship the pucks from the factory to the warehouse. Because the trucks travel over specified routes (edges) between cities (vertices) and have a limited capacity, Lucky Puck can ship at most $c(u, v)$ crates per day between each pair of cities $u$ and $v$ in Figure 26.1(a). Lucky Puck has no control over these routes and capacities, and so the company cannot alter the flow network shown in Figure 26.1(a). They need to determine the largest number $p$ of crates per day that they can ship and then to produce this amount, since there is no point in producing more pucks than they can ship to their warehouse. Lucky Puck is not concerned with how long it takes for a given puck to get from the factory to the warehouse; they care only that $p$ crates per day leave the factory and $p$ crates per day arrive at the warehouse.

We can model the "flow" of shipments with a flow in this network because the number of crates shipped per day from one city to another is subject to a capacity constraint. Additionally, the model must obey flow conservation, for in a steady state, the rate at which pucks enter an intermediate city must equal the rate at which they leave. Otherwise, crates would accumulate at intermediate cities.

## Modeling problems with antiparallel edges

Suppose that the trucking firm offered Lucky Puck the opportunity to lease space for 10 crates in trucks going from Edmonton to Calgary. It would seem natural to add this opportunity to our example and form the network shown in Figure 26.2(a). This network suffers from one problem, however: it violates our original assumption that if an edge $(v_1, v_2) \in E$, then $(v_2, v_1) \notin E$. We call the two edges $(v_1, v_2)$ and $(v_2, v_1)$ ***antiparallel***. Thus, if we wish to model a flow problem with antiparallel edges, we must transform the network into an equivalent one containing no antiparallel edges. Figure 26.2(b) displays this equivalent network. We choose one of the two antiparallel edges, in this case $(v_1, v_2)$, and split it by adding a new vertex $v'$ and replacing edge $(v_1, v_2)$ with the pair of edges $(v_1, v')$ and $(v', v_2)$. We also set the capacity of both new edges to the capacity of the original edge. The resulting network satisfies the property that if an edge is in the network, the reverse edge is not. Exercise 26.1-1 asks you to prove that the resulting network is equivalent to the original one.

Thus, we see that a real-world flow problem might be most naturally modeled by a network with antiparallel edges. It will be convenient to disallow antiparallel edges, however, and so we have a straightforward way to convert a network containing antiparallel edges into an equivalent one with no antiparallel edges.

## Networks with multiple sources and sinks

A maximum-flow problem may have several sources and sinks, rather than just one of each. The Lucky Puck Company, for example, might actually have a set of $m$ factories $\{s_1, s_2, \ldots, s_m\}$ and a set of $n$ warehouses $\{t_1, t_2, \ldots, t_n\}$, as shown in Figure 26.3(a). Fortunately, this problem is no harder than ordinary maximum flow.

We can reduce the problem of determining a maximum flow in a network with multiple sources and multiple sinks to an ordinary maximum-flow problem. Figure 26.3(b) shows how to convert the network from (a) to an ordinary flow network with only a single source and a single sink. We add a ***supersource*** $s$ and add a directed edge $(s, s_i)$ with capacity $c(s, s_i) = \infty$ for each $i = 1, 2, \ldots, m$. We also create a new ***supersink*** $t$ and add a directed edge $(t_i, t)$ with capacity $c(t_i, t) = \infty$ for each $i = 1, 2, \ldots, n$. Intuitively, any flow in the network in (a) corresponds to a flow in the network in (b), and vice versa. The single source $s$ simply provides as much flow as desired for the multiple sources $s_i$, and the single sink $t$ likewise consumes as much flow as desired for the multiple sinks $t_i$. Exercise 26.1-2 asks you to prove formally that the two problems are equivalent.

## Exercises

***26.1-1***

Show that splitting an edge in a flow network yields an equivalent network. More formally, suppose that flow network $G$ contains edge $(u, v)$, and we create a new flow network $G'$ by creating a new vertex $x$ and replacing $(u, v)$ by new edges $(u, x)$ and $(x, v)$ with $c(u, x) = c(x, v) = c(u, v)$. Show that a maximum flow in $G'$ has the same value as a maximum flow in $G$.

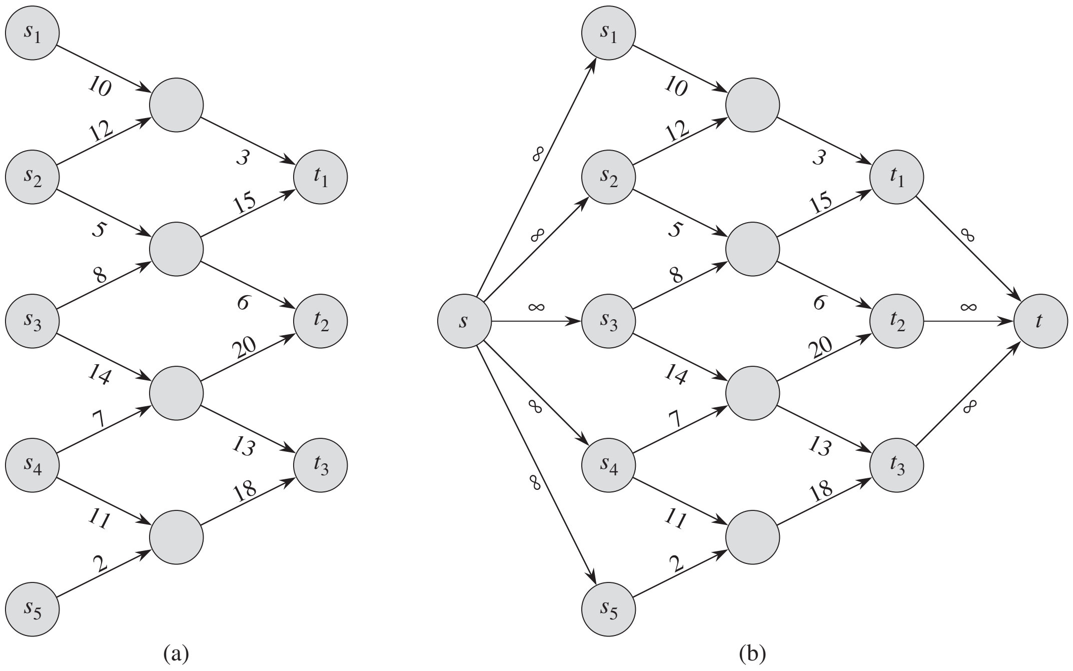

**Figure 26.3** Converting a multiple-source, multiple-sink maximum-flow problem into a problem with a single source and a single sink. **(a)** A flow network with five sources $S = \{s_1, s_2, s_3, s_4, s_5\}$ and three sinks $T = \{t_1, t_2, t_3\}$. **(b)** An equivalent single-source, single-sink flow network. We add a supersource $s$ and an edge with infinite capacity from $s$ to each of the multiple sources. We also add a supersink $t$ and an edge with infinite capacity from each of the multiple sinks to $t$.

***26.1-2***

Extend the flow properties and definitions to the multiple-source, multiple-sink problem. Show that any flow in a multiple-source, multiple-sink flow network corresponds to a flow of identical value in the single-source, single-sink network obtained by adding a supersource and a supersink, and vice versa.

***26.1-3***

Suppose that a flow network $G = (V, E)$ violates the assumption that the network contains a path $s \leadsto v \leadsto t$ for all vertices $v \in V$. Let $u$ be a vertex for which there is no path $s \leadsto u \leadsto t$. Show that there must exist a maximum flow $f$ in $G$ such that $f(u, v) = f(v, u) = 0$ for all vertices $v \in V$.

***26.1-4***

Let $f$ be a flow in a network, and let $\alpha$ be a real number. The ***scalar flow product***, denoted $\alpha f$, is a function from $V \times V$ to $\mathbb{R}$ defined by

$$(\alpha f)(u, v) = \alpha \cdot f(u, v) .$$

Prove that the flows in a network form a ***convex set***. That is, show that if $f_1$ and $f_2$ are flows, then so is $\alpha f_1 + (1 - \alpha) f_2$ for all $\alpha$ in the range $0 \le \alpha \le 1$.

***26.1-5***

State the maximum-flow problem as a linear-programming problem.

***26.1-6***

Professor Adam has two children who, unfortunately, dislike each other. The problem is so severe that not only do they refuse to walk to school together, but in fact each one refuses to walk on any block that the other child has stepped on that day. The children have no problem with their paths crossing at a corner. Fortunately both the professor's house and the school are on corners, but beyond that he is not sure if it is going to be possible to send both of his children to the same school. The professor has a map of his town. Show how to formulate the problem of determining whether both his children can go to the same school as a maximum-flow problem.

***26.1-7***

Suppose that, in addition to edge capacities, a flow network has ***vertex capacities***. That is each vertex $v$ has a limit $l(v)$ on how much flow can pass though $v$. Show how to transform a flow network $G = (V, E)$ with vertex capacities into an equivalent flow network $G' = (V', E')$ without vertex capacities, such that a maximum flow in $G'$ has the same value as a maximum flow in $G$. How many vertices and edges does $G'$ have?

## 26.2 The Ford-Fulkerson method

This section presents the Ford-Fulkerson method for solving the maximum-flow problem. We call it a "method" rather than an "algorithm" because it encompasses several implementations with differing running times. The Ford-Fulkerson method depends on three important ideas that transcend the method and are relevant to many flow algorithms and problems: residual networks, augmenting paths, and cuts. These ideas are essential to the important max-flow min-cut theorem (Theorem 26.6), which characterizes the value of a maximum flow in terms of cuts of the flow network. We end this section by presenting one specific implementation of the Ford-Fulkerson method and analyzing its running time.

The Ford-Fulkerson method iteratively increases the value of the flow. We start with $f(u, v) = 0$ for all $u, v \in V$, giving an initial flow of value 0. At each iteration, we increase the flow value in $G$ by finding an "augmenting path" in an associated "residual network" $G_f$. Once we know the edges of an augmenting path in $G_f$, we can easily identify specific edges in $G$ for which we can change the flow so that we increase the value of the flow. Although each iteration of the Ford-Fulkerson method increases the value of the flow, we shall see that the flow on any particular edge of $G$ may increase or decrease; decreasing the flow on some edges may be necessary in order to enable an algorithm to send more flow from the source to the sink. We repeatedly augment the flow until the residual network has no more augmenting paths. The max-flow min-cut theorem will show that upon termination, this process yields a maximum flow.

```
FORD-FULKERSON-METHOD($G, s, t$)
1  initialize flow $f$ to 0
2  while there exists an augmenting path $p$ in the residual network $G_f$
3      augment flow $f$ along $p$
4  return $f$
```

In order to implement and analyze the Ford-Fulkerson method, we need to introduce several additional concepts.

## Residual networks

Intuitively, given a flow network $G$ and a flow $f$, the residual network $G_f$ consists of edges with capacities that represent how we can change the flow on edges of $G$. An edge of the flow network can admit an amount of additional flow equal to the edge's capacity minus the flow on that edge. If that value is positive, we place that edge into $G_f$ with a "residual capacity" of $c_f(u, v) = c(u, v) - f(u, v)$. The only edges of $G$ that are in $G_f$ are those that can admit more flow; those edges $(u, v)$ whose flow equals their capacity have $c_f(u, v) = 0$, and they are not in $G_f$.

The residual network $G_f$ may also contain edges that are not in $G$, however. As an algorithm manipulates the flow, with the goal of increasing the total flow, it might need to decrease the flow on a particular edge. In order to represent a possible decrease of a positive flow $f(u, v)$ on an edge in $G$, we place an edge $(v, u)$ into $G_f$ with residual capacity $c_f(v, u) = f(u, v)$—that is, an edge that can admit flow in the opposite direction to $(u, v)$, at most canceling out the flow on $(u, v)$. These reverse edges in the residual network allow an algorithm to send back flow it has already sent along an edge. Sending flow back along an edge is equivalent to *decreasing* the flow on the edge, which is a necessary operation in many algorithms.

More formally, suppose that we have a flow network $G = (V, E)$ with source $s$ and sink $t$. Let $f$ be a flow in $G$, and consider a pair of vertices $u, v \in V$. We define the ***residual capacity*** $c_f(u, v)$ by

$$c_f(u, v) = \begin{cases} c(u, v) - f(u, v) & \text{if } (u, v) \in E \,, \\ f(v, u) & \text{if } (v, u) \in E \,, \\ 0 & \text{otherwise} \,. \end{cases} \tag{26.2}$$

Because of our assumption that $(u, v) \in E$ implies $(v, u) \notin E$, exactly one case in equation (26.2) applies to each ordered pair of vertices.

As an example of equation (26.2), if $c(u, v) = 16$ and $f(u, v) = 11$, then we can increase $f(u, v)$ by up to $c_f(u, v) = 5$ units before we exceed the capacity constraint on edge $(u, v)$. We also wish to allow an algorithm to return up to 11 units of flow from $v$ to $u$, and hence $c_f(v, u) = 11$.

Given a flow network $G = (V, E)$ and a flow $f$, the ***residual network*** of $G$ induced by $f$ is $G_f = (V, E_f)$, where

$$E_f = \{(u, v) \in V \times V : c_f(u, v) > 0\} \,. \tag{26.3}$$

That is, as promised above, each edge of the residual network, or ***residual edge***, can admit a flow that is greater than 0. Figure 26.4(a) repeats the flow network $G$ and flow $f$ of Figure 26.1(b), and Figure 26.4(b) shows the corresponding residual network $G_f$. The edges in $E_f$ are either edges in $E$ or their reversals, and thus

$$|E_f| \le 2\,|E| \,.$$

Observe that the residual network $G_f$ is similar to a flow network with capacities given by $c_f$. It does not satisfy our definition of a flow network because it may contain both an edge $(u, v)$ and its reversal $(v, u)$. Other than this difference, a residual network has the same properties as a flow network, and we can define a flow in the residual network as one that satisfies the definition of a flow, but with respect to capacities $c_f$ in the network $G_f$.

A flow in a residual network provides a roadmap for adding flow to the original flow network. If $f$ is a flow in $G$ and $f'$ is a flow in the corresponding residual network $G_f$, we define $f \uparrow f'$, the ***augmentation*** of flow $f$ by $f'$, to be a function from $V \times V$ to $\mathbb{R}$, defined by

$$(f \uparrow f')(u, v) = \begin{cases} f(u, v) + f'(u, v) - f'(v, u) & \text{if } (u, v) \in E \,, \\ 0 & \text{otherwise} \,. \end{cases} \tag{26.4}$$

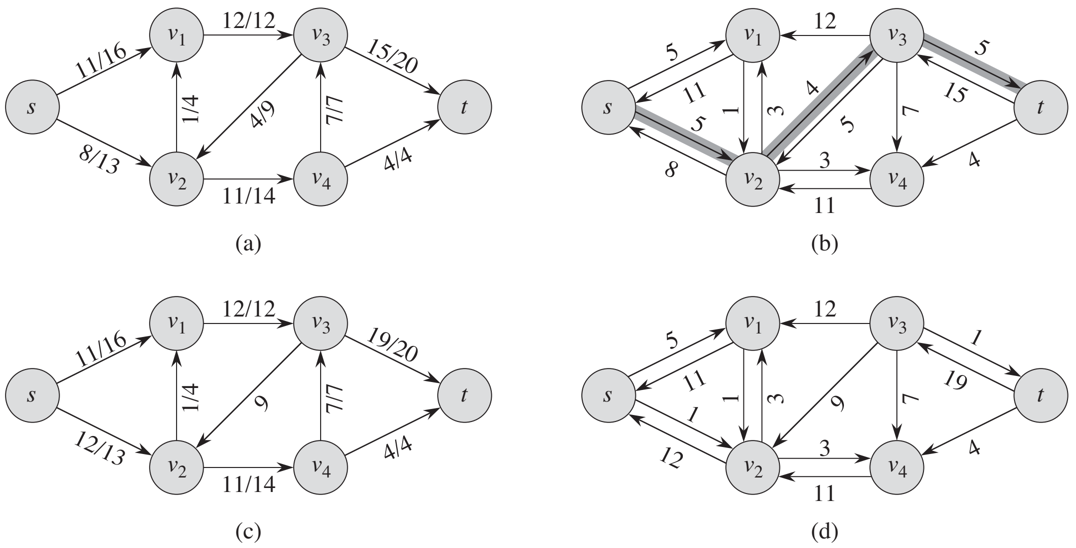

**Figure 26.4** **(a)** The flow network $G$ and flow $f$ of Figure 26.1(b). **(b)** The residual network $G_f$ with augmenting path $p$ shaded; its residual capacity is $c_f(p) = c_f(v_2, v_3) = 4$. Edges with residual capacity equal to 0, such as $(v_1, v_3)$, are not shown, a convention we follow 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).

The intuition behind this definition follows the definition of the residual network. We increase the flow on $(u, v)$ by $f'(u, v)$ but decrease it by $f'(v, u)$ because pushing flow on the reverse edge in the residual network signifies decreasing the flow in the original network. Pushing flow on the reverse edge in the residual network is also known as ***cancellation***. For example, if we send 5 crates of hockey pucks from $u$ to $v$ and send 2 crates from $v$ to $u$, we could equivalently (from the perspective of the final result) just send 3 creates from $u$ to $v$ and none from $v$ to $u$. Cancellation of this type is crucial for any maximum-flow algorithm.

***Lemma 26.1***
Let $G = (V, E)$ be a flow network with source $s$ and sink $t$, and let $f$ be a flow in $G$. Let $G_f$ be the residual network of $G$ induced by $f$, and let $f'$ be a flow in $G_f$. Then the function $f \uparrow f'$ defined in equation (26.4) is a flow in $G$ with value $|f \uparrow f'| = |f| + |f'|$.

***Proof*** We first verify that $f \uparrow f'$ obeys the capacity constraint for each edge in $E$ and flow conservation at each vertex in $V - \{s, t\}$.

For the capacity constraint, first observe that if $(u,v) \in E$, then $c_f(v,u) = f(u,v)$. Therefore, we have $f'(v,u) \le c_f(v,u) = f(u,v)$, and hence

$$\begin{aligned}
(f \uparrow f')(u,v) &= f(u,v) + f'(u,v) - f'(v,u) && \text{(by equation (26.4))} \\
&\ge f(u,v) + f'(u,v) - f(u,v) && \text{(because } f'(v,u) \le f(u,v)\text{)} \\
&= f'(u,v) \\
&\ge 0 .
\end{aligned}$$

In addition,

$$\begin{aligned}
&(f \uparrow f')(u,v) \\
&\quad = f(u,v) + f'(u,v) - f'(v,u) && \text{(by equation (26.4))} \\
&\quad \le f(u,v) + f'(u,v) && \text{(because flows are nonnegative)} \\
&\quad \le f(u,v) + c_f(u,v) && \text{(capacity constraint)} \\
&\quad = f(u,v) + c(u,v) - f(u,v) && \text{(definition of } c_f\text{)} \\
&\quad = c(u,v) .
\end{aligned}$$

For flow conservation, because both $f$ and $f'$ obey flow conservation, we have that for all $u \in V - \{s,t\}$,

$$\begin{aligned}
\sum_{v \in V} (f \uparrow f')(u,v) &= \sum_{v \in V} (f(u,v) + f'(u,v) - f'(v,u)) \\
&= \sum_{v \in V} f(u,v) + \sum_{v \in V} f'(u,v) - \sum_{v \in V} f'(v,u) \\
&= \sum_{v \in V} f(v,u) + \sum_{v \in V} f'(v,u) - \sum_{v \in V} f'(u,v) \\
&= \sum_{v \in V} (f(v,u) + f'(v,u) - f'(u,v)) \\
&= \sum_{v \in V} (f \uparrow f')(v,u) ,
\end{aligned}$$

where the third line follows from the second by flow conservation.

Finally, we compute the value of $f \uparrow f'$. Recall that we disallow antiparallel edges in $G$ (but not in $G_f$), and hence for each vertex $v \in V$, we know that there can be an edge $(s,v)$ or $(v,s)$, but never both. We define $V_1 = \{v : (s,v) \in E\}$ to be the set of vertices with edges from $s$, and $V_2 = \{v : (v,s) \in E\}$ to be the set of vertices with edges to $s$. We have $V_1 \cup V_2 \subseteq V$ and, because we disallow antiparallel edges, $V_1 \cap V_2 = \emptyset$. We now compute

$$\begin{aligned}
|f \uparrow f'| &= \sum_{v \in V} (f \uparrow f')\,(s,v) - \sum_{v \in V} (f \uparrow f')\,(v,s) \\
&= \sum_{v \in V_1} (f \uparrow f')\,(s,v) - \sum_{v \in V_2} (f \uparrow f')\,(v,s) , \qquad (26.5)
\end{aligned}$$

where the second line follows because $(f \uparrow f')(w,x)$ is 0 if $(w,x) \notin E$. We now apply the definition of $f \uparrow f'$ to equation (26.5), and then reorder and group terms to obtain

$$\begin{aligned}
|f \uparrow f'| \\
&= \sum_{v \in V_1} (f(s,v) + f'(s,v) - f'(v,s)) - \sum_{v \in V_2} (f(v,s) + f'(v,s) - f'(s,v)) \\
&= \sum_{v \in V_1} f(s,v) + \sum_{v \in V_1} f'(s,v) - \sum_{v \in V_1} f'(v,s) \\
&\qquad - \sum_{v \in V_2} f(v,s) - \sum_{v \in V_2} f'(v,s) + \sum_{v \in V_2} f'(s,v) \\
&= \sum_{v \in V_1} f(s,v) - \sum_{v \in V_2} f(v,s) \\
&\qquad + \sum_{v \in V_1} f'(s,v) + \sum_{v \in V_2} f'(s,v) - \sum_{v \in V_1} f'(v,s) - \sum_{v \in V_2} f'(v,s) \\
&= \sum_{v \in V_1} f(s,v) - \sum_{v \in V_2} f(v,s) + \sum_{v \in V_1 \cup V_2} f'(s,v) - \sum_{v \in V_1 \cup V_2} f'(v,s) \; . \qquad (26.6)
\end{aligned}$$

In equation (26.6), we can extend all four summations to sum over $V$, since each additional term has value 0. (Exercise 26.2-1 asks you to prove this formally.) We thus have

$$\begin{aligned}
|f \uparrow f'| &= \sum_{v \in V} f(s,v) - \sum_{v \in V} f(v,s) + \sum_{v \in V} f'(s,v) - \sum_{v \in V} f'(v,s) \qquad (26.7) \\
&= |f| + |f'| \; .
\end{aligned}$$

■

## Augmenting paths

Given a flow network $G = (V,E)$ and a flow $f$, an ***augmenting path*** $p$ is a simple path from $s$ to $t$ in the residual network $G_f$. By the definition of the residual network, we may increase the flow on an edge $(u,v)$ of an augmenting path by up to $c_f(u,v)$ without violating the capacity constraint on whichever of $(u,v)$ and $(v,u)$ is in the original flow network $G$.

The shaded path in Figure 26.4(b) is an augmenting path. Treating the residual network $G_f$ in the figure as a flow network, we can increase the flow through each edge of this path by up to 4 units without violating a capacity constraint, since the smallest residual capacity on this path is $c_f(v_2,v_3) = 4$. We call the maximum amount by which we can increase the flow on each edge in an augmenting path $p$ the ***residual capacity*** of $p$, given by

$$c_f(p) = \min\{c_f(u,v) : (u,v) \text{ is on } p\} \; .$$

The following lemma, whose proof we leave as Exercise 26.2-7, makes the above argument more precise.

***Lemma 26.2***
Let $G = (V, E)$ be a flow network, let $f$ be a flow in $G$, and let $p$ be an augmenting path in $G_f$. Define a function $f_p : V \times V \to \mathbb{R}$ by

$$f_p(u, v) = \begin{cases} c_f(p) & \text{if } (u, v) \text{ is on } p \, , \\ 0 & \text{otherwise} \, . \end{cases} \qquad (26.8)$$

Then, $f_p$ is a flow in $G_f$ with value $|f_p| = c_f(p) > 0$. ■

The following corollary shows that if we augment $f$ by $f_p$, we get another flow in $G$ whose value is closer to the maximum. Figure 26.4(c) shows the result of augmenting the flow $f$ from Figure 26.4(a) by the flow $f_p$ in Figure 26.4(b), and Figure 26.4(d) shows the ensuing residual network.

***Corollary 26.3***
Let $G = (V, E)$ be a flow network, let $f$ be a flow in $G$, and let $p$ be an augmenting path in $G_f$. Let $f_p$ be defined as in equation (26.8), and suppose that we augment $f$ by $f_p$. Then the function $f \uparrow f_p$ is a flow in $G$ with value $|f \uparrow f_p| = |f| + |f_p| > |f|$.

***Proof*** Immediate from Lemmas 26.1 and 26.2. ■

## Cuts of flow networks

The Ford-Fulkerson method repeatedly augments the flow along augmenting paths until it has found a maximum flow. How do we know that when the algorithm terminates, we have actually found a maximum flow? The max-flow min-cut theorem, which we shall prove shortly, tells us that a flow is maximum if and only if its residual network contains no augmenting path. To prove this theorem, though, we must first explore the notion of a cut of a flow network.

A ***cut*** $(S, T)$ of flow network $G = (V, E)$ is a partition of $V$ into $S$ and $T = V - S$ such that $s \in S$ and $t \in T$. (This definition is similar to the definition of "cut" that we used for minimum spanning trees in Chapter 23, except that here we are cutting a directed graph rather than an undirected graph, and we insist that $s \in S$ and $t \in T$.) If $f$ is a flow, then the ***net flow*** $f(S, T)$ across the cut $(S, T)$ is defined to be

$$f(S, T) = \sum_{u \in S} \sum_{v \in T} f(u, v) - \sum_{u \in S} \sum_{v \in T} f(v, u) \, . \qquad (26.9)$$

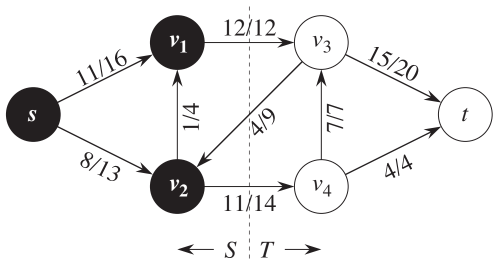

**Figure 26.5** A cut $(S, T)$ in the flow network of Figure 26.1(b), where $S = \{s, v_1, v_2\}$ and $T = \{v_3, v_4, t\}$. The vertices in $S$ are black, and the vertices in $T$ are white. The net flow across $(S, T)$ is $f(S, T) = 19$, and the capacity is $c(S, T) = 26$.

The ***capacity*** of the cut $(S, T)$ is

$$c(S, T) = \sum_{u \in S} \sum_{v \in T} c(u, v) . \tag{26.10}$$

A ***minimum cut*** of a network is a cut whose capacity is minimum over all cuts of the network.

The asymmetry between the definitions of flow and capacity of a cut is intentional and important. For capacity, we count only the capacities of edges going from $S$ to $T$, ignoring edges in the reverse direction. For flow, we consider the flow going from $S$ to $T$ minus the flow going in the reverse direction from $T$ to $S$. The reason for this difference will become clear later in this section.

Figure 26.5 shows the cut $(\{s, v_1, v_2\}, \{v_3, v_4, t\})$ in the flow network of Figure 26.1(b). The net flow across this cut is

$$\begin{aligned} f(v_1, v_3) + f(v_2, v_4) - f(v_3, v_2) &= 12 + 11 - 4 \\ &= 19 , \end{aligned}$$

and the capacity of this cut is

$$\begin{aligned} c(v_1, v_3) + c(v_2, v_4) &= 12 + 14 \\ &= 26 . \end{aligned}$$

The following lemma shows that, for a given flow $f$, the net flow across any cut is the same, and it equals $|f|$, the value of the flow.

***Lemma 26.4***

Let $f$ be a flow in a flow network $G$ with source $s$ and sink $t$, and let $(S, T)$ be any cut of $G$. Then the net flow across $(S, T)$ is $f(S, T) = |f|$.

***Proof*** We can rewrite the flow-conservation condition for any node $u \in V - \{s, t\}$ as

$$\sum_{v \in V} f(u, v) - \sum_{v \in V} f(v, u) = 0 \, . \qquad (26.11)$$

Taking the definition of $|f|$ from equation (26.1) and adding the left-hand side of equation (26.11), which equals 0, summed over all vertices in $S - \{s\}$, gives

$$|f| = \sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, s) + \sum_{u \in S - \{s\}} \left( \sum_{v \in V} f(u, v) - \sum_{v \in V} f(v, u) \right) .$$

Expanding the right-hand summation and regrouping terms yields

$$\begin{aligned} |f| &= \sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, s) + \sum_{u \in S - \{s\}} \sum_{v \in V} f(u, v) - \sum_{u \in S - \{s\}} \sum_{v \in V} f(v, u) \\ &= \sum_{v \in V} \left( f(s, v) + \sum_{u \in S - \{s\}} f(u, v) \right) - \sum_{v \in V} \left( f(v, s) + \sum_{u \in S - \{s\}} f(v, u) \right) \\ &= \sum_{v \in V} \sum_{u \in S} f(u, v) - \sum_{v \in V} \sum_{u \in S} f(v, u) \, . \end{aligned}$$

Because $V = S \cup T$ and $S \cap T = \emptyset$, we can split each summation over $V$ into summations over $S$ and $T$ to obtain

$$\begin{aligned} |f| &= \sum_{v \in S} \sum_{u \in S} f(u, v) + \sum_{v \in T} \sum_{u \in S} f(u, v) - \sum_{v \in S} \sum_{u \in S} f(v, u) - \sum_{v \in T} \sum_{u \in S} f(v, u) \\ &= \sum_{v \in T} \sum_{u \in S} f(u, v) - \sum_{v \in T} \sum_{u \in S} f(v, u) \\ &\qquad + \left( \sum_{v \in S} \sum_{u \in S} f(u, v) - \sum_{v \in S} \sum_{u \in S} f(v, u) \right) . \end{aligned}$$

The two summations within the parentheses are actually the same, since for all vertices $x, y \in V$, the term $f(x, y)$ appears once in each summation. Hence, these summations cancel, and we have

$$\begin{aligned} |f| &= \sum_{u \in S} \sum_{v \in T} f(u, v) - \sum_{u \in S} \sum_{v \in T} f(v, u) \\ &= f(S, T) \, . \end{aligned}$$

■

A corollary to Lemma 26.4 shows how we can use cut capacities to bound the value of a flow.

***Corollary 26.5***
The value of any flow $f$ in a flow network $G$ is bounded from above by the capacity of any cut of $G$.

***Proof*** Let $(S, T)$ be any cut of $G$ and let $f$ be any flow. By Lemma 26.4 and the capacity constraint,

$$\begin{aligned} |f| &= f(S,T) \\ &= \sum_{u \in S} \sum_{v \in T} f(u,v) - \sum_{u \in S} \sum_{v \in T} f(v,u) \\ &\le \sum_{u \in S} \sum_{v \in T} f(u,v) \\ &\le \sum_{u \in S} \sum_{v \in T} c(u,v) \\ &= c(S,T) . \end{aligned}$$

■

Corollary 26.5 yields the immediate consequence that the value of a maximum flow in a network is bounded from above by the capacity of a minimum cut of the network. The important max-flow min-cut theorem, which we now state and prove, says that the value of a maximum flow is in fact equal to the capacity of a minimum cut.

***Theorem 26.6 (Max-flow min-cut theorem)***
If $f$ is a flow in a flow network $G = (V, E)$ with source $s$ and sink $t$, then the following conditions are equivalent:

1. $f$ is a maximum flow in $G$.
2. The residual network $G_f$ contains no augmenting paths.
3. $|f| = c(S, T)$ for some cut $(S, T)$ of $G$.

***Proof*** (1) $\Rightarrow$ (2): Suppose for the sake of contradiction that $f$ is a maximum flow in $G$ but that $G_f$ has an augmenting path $p$. Then, by Corollary 26.3, the flow found by augmenting $f$ by $f_p$, where $f_p$ is given by equation (26.8), is a flow in $G$ with value strictly greater than $|f|$, contradicting the assumption that $f$ is a maximum flow.

(2) $\Rightarrow$ (3): Suppose that $G_f$ has no augmenting path, that is, that $G_f$ contains no path from $s$ to $t$. Define

$S = \{v \in V : \text{there exists a path from } s \text{ to } v \text{ in } G_f\}$

and $T = V - S$. The partition $(S, T)$ is a cut: we have $s \in S$ trivially and $t \notin S$ because there is no path from $s$ to $t$ in $G_f$. Now consider a pair of vertices

$u \in S$ and $v \in T$. If $(u, v) \in E$, we must have $f(u, v) = c(u, v)$, since otherwise $(u, v) \in E_f$, which would place $v$ in set $S$. If $(v, u) \in E$, we must have $f(v, u) = 0$, because otherwise $c_f(u, v) = f(v, u)$ would be positive and we would have $(u, v) \in E_f$, which would place $v$ in $S$. Of course, if neither $(u, v)$ nor $(v, u)$ is in $E$, then $f(u, v) = f(v, u) = 0$. We thus have

$$\begin{aligned} f(S, T) &= \sum_{u \in S} \sum_{v \in T} f(u, v) - \sum_{v \in T} \sum_{u \in S} f(v, u) \\ &= \sum_{u \in S} \sum_{v \in T} c(u, v) - \sum_{v \in T} \sum_{u \in S} 0 \\ &= c(S, T) . \end{aligned}$$

By Lemma 26.4, therefore, $|f| = f(S, T) = c(S, T)$.

(3) $\Rightarrow$ (1): By Corollary 26.5, $|f| \leq c(S, T)$ for all cuts $(S, T)$. The condition $|f| = c(S, T)$ thus implies that $f$ is a maximum flow. ■

## The basic Ford-Fulkerson algorithm

In each iteration of the Ford-Fulkerson method, we find *some* augmenting path $p$ and use $p$ to modify the flow $f$. As Lemma 26.2 and Corollary 26.3 suggest, we replace $f$ by $f \uparrow f_p$, obtaining a new flow whose value is $|f| + |f_p|$. The following implementation of the method computes the maximum flow in a flow network $G = (V, E)$ by updating the flow attribute $(u, v).f$ for each edge $(u, v) \in E$.[1] If $(u, v) \notin E$, we assume implicitly that $(u, v).f = 0$. We also assume that we are given the capacities $c(u, v)$ along with the flow network, and $c(u, v) = 0$ if $(u, v) \notin E$. We compute the residual capacity $c_f(u, v)$ in accordance with the formula (26.2). The expression $c_f(p)$ in the code is just a temporary variable that stores the residual capacity of the path $p$.

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

[1] Recall from Section 22.1 that we represent an attribute $f$ for edge $(u, v)$ with the same style of notation—$(u, v).f$—that we use for an attribute of any other object.

The FORD-FULKERSON algorithm simply expands on the FORD-FULKERSON-METHOD pseudocode given earlier. Figure 26.6 shows the result of each iteration in a sample run. Lines 1–2 initialize the flow $f$ to 0. The **while** loop of lines 3–8 repeatedly finds an augmenting path $p$ in $G_f$ and augments flow $f$ along $p$ by the residual capacity $c_f(p)$. Each residual edge in path $p$ is either an edge in the original network or the reversal of an edge in the original network. Lines 6–8 update the flow in each case appropriately, adding flow when the residual edge is an original edge and subtracting it otherwise. When no augmenting paths exist, the flow $f$ is a maximum flow.

## Analysis of Ford-Fulkerson

The running time of FORD-FULKERSON depends on how we find the augmenting path $p$ in line 3. If we choose it poorly, the algorithm might not even terminate: the value of the flow will increase with successive augmentations, but it need not even converge to the maximum flow value.[2] If we find the augmenting path by using a breadth-first search (which we saw in Section 22.2), however, the algorithm runs in polynomial time. Before proving this result, we obtain a simple bound for the case in which we choose the augmenting path arbitrarily and all capacities are integers.

In practice, the maximum-flow problem often arises with integral capacities. If the capacities are rational numbers, we can apply an appropriate scaling transformation to make them all integral. If $f^*$ denotes a maximum flow in the transformed network, then a straightforward implementation of FORD-FULKERSON executes the **while** loop of lines 3–8 at most $|f^*|$ times, since the flow value increases by at least one unit in each iteration.

We can perform the work done within the **while** loop efficiently if we implement the flow network $G = (V, E)$ with the right data structure and find an augmenting path by a linear-time algorithm. Let us assume that we keep a data structure corresponding to a directed graph $G' = (V, E')$, where $E' = \{(u, v) : (u, v) \in E$ or $(v, u) \in E\}$. Edges in the network $G$ are also edges in $G'$, and therefore we can easily maintain capacities and flows in this data structure. Given a flow $f$ on $G$, the edges in the residual network $G_f$ consist of all edges $(u, v)$ of $G'$ such that $c_f(u, v) > 0$, where $c_f$ conforms to equation (26.2). The time to find a path in a residual network is therefore $O(V + E') = O(E)$ if we use either depth-first search or breadth-first search. Each iteration of the **while** loop thus takes $O(E)$ time, as does the initialization in lines 1–2, making the total running time of the FORD-FULKERSON algorithm $O(E\,|f^*|)$.

[2]The Ford-Fulkerson method might fail to terminate only if edge capacities are irrational numbers.

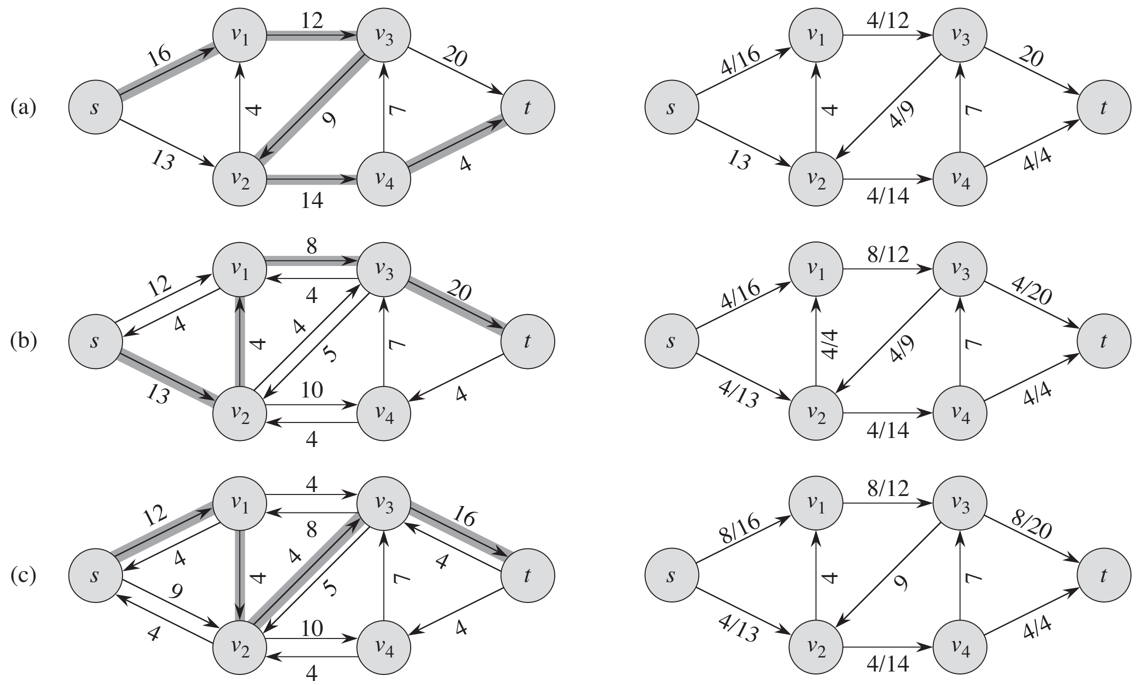

**Figure 26.6** The execution of the basic Ford-Fulkerson algorithm. **(a)–(e)** Successive iterations of the **while** loop. The left side of each part shows the residual network $G_f$ from line 3 with a shaded augmenting path $p$. The right side of each part shows the new flow $f$ that results from augmenting $f$ by $f_p$. The residual network in (a) is the input network $G$.

When the capacities are integral and the optimal flow value $|f^*|$ is small, the running time of the Ford-Fulkerson algorithm is good. Figure 26.7(a) shows an example of what can happen on a simple flow network for which $|f^*|$ is large. A maximum flow in this network has value 2,000,000: 1,000,000 units of flow traverse the path $s \to u \to t$, and another 1,000,000 units traverse the path $s \to v \to t$. If the first augmenting path found by FORD-FULKERSON is $s \to u \to v \to t$, shown in Figure 26.7(a), the flow has value 1 after the first iteration. The resulting residual network appears in Figure 26.7(b). If the second iteration finds the augmenting path $s \to v \to u \to t$, as shown in Figure 26.7(b), the flow then has value 2. Figure 26.7(c) shows the resulting residual network. We can continue, choosing the augmenting path $s \to u \to v \to t$ in the odd-numbered iterations and the augmenting path $s \to v \to u \to t$ in the even-numbered iterations. We would perform a total of 2,000,000 augmentations, increasing the flow value by only 1 unit in each.

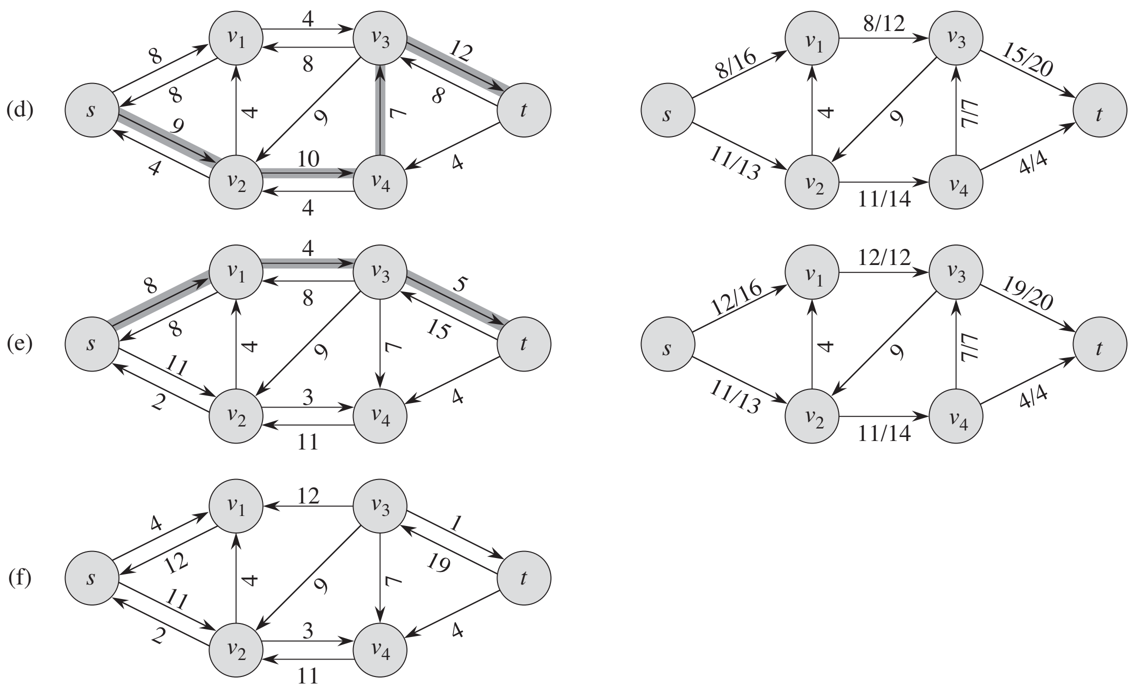

**Figure 26.6, continued** **(f)** The residual network at the last **while** loop test. It has no augmenting paths, and the flow $f$ shown in (e) is therefore a maximum flow. The value of the maximum flow found is 23.

## The Edmonds-Karp algorithm

We can improve the bound on FORD-FULKERSON by finding the augmenting path $p$ in line 3 with a breadth-first search. That is, we choose the augmenting path as a *shortest* path from $s$ to $t$ in the residual network, where each edge has unit distance (weight). We call the Ford-Fulkerson method so implemented the ***Edmonds-Karp algorithm***. We now prove that the Edmonds-Karp algorithm runs in $O(VE^2)$ time.

The analysis depends on the distances to vertices in the residual network $G_f$. The following lemma uses the notation $\delta_f(u, v)$ for the shortest-path distance from $u$ to $v$ in $G_f$, where each edge has unit distance.

***Lemma 26.7***

If the Edmonds-Karp algorithm is run on a flow network $G = (V, E)$ with source $s$ and sink $t$, then for all vertices $v \in V - \{s, t\}$, the shortest-path distance $\delta_f(s, v)$ in the residual network $G_f$ increases monotonically with each flow augmentation.

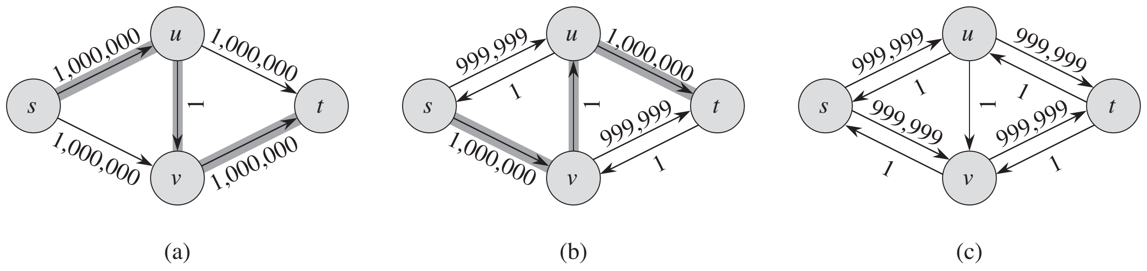

**Figure 26.7** **(a)** A flow network for which FORD-FULKERSON can take $\Theta(E\,|f^*|)$ time, where $f^*$ is a maximum flow, shown here with $|f^*| = 2{,}000{,}000$. The shaded path is an augmenting path with residual capacity 1. **(b)** The resulting residual network, with another augmenting path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

***Proof*** We will suppose that for some vertex $v \in V - \{s,t\}$, there is a flow augmentation that causes the shortest-path distance from $s$ to $v$ to decrease, and then we will derive a contradiction. Let $f$ be the flow just before the first augmentation that decreases some shortest-path distance, and let $f'$ be the flow just afterward. Let $v$ be the vertex with the minimum $\delta_{f'}(s,v)$ whose distance was decreased by the augmentation, so that $\delta_{f'}(s,v) < \delta_f(s,v)$. Let $p = s \rightsquigarrow u \to v$ be a shortest path from $s$ to $v$ in $G_{f'}$, so that $(u,v) \in E_{f'}$ and

$$\delta_{f'}(s,u) = \delta_{f'}(s,v) - 1\ . \tag{26.12}$$

Because of how we chose $v$, we know that the distance of vertex $u$ from the source $s$ did not decrease, i.e.,

$$\delta_{f'}(s,u) \geq \delta_f(s,u)\ . \tag{26.13}$$

We claim that $(u,v) \notin E_f$. Why? If we had $(u,v) \in E_f$, then we would also have

$$\begin{aligned} \delta_f(s,v) &\leq \delta_f(s,u) + 1 && \text{(by Lemma 24.10, the triangle inequality)} \\ &\leq \delta_{f'}(s,u) + 1 && \text{(by inequality (26.13))} \\ &= \delta_{f'}(s,v) && \text{(by equation (26.12))}\ , \end{aligned}$$

which contradicts our assumption that $\delta_{f'}(s,v) < \delta_f(s,v)$.

How can we have $(u,v) \notin E_f$ and $(u,v) \in E_{f'}$? The augmentation must have increased the flow from $v$ to $u$. The Edmonds-Karp algorithm always augments flow along shortest paths, and therefore the shortest path from $s$ to $u$ in $G_f$ has $(v,u)$ as its last edge. Therefore,

$$\begin{aligned} \delta_f(s,v) &= \delta_f(s,u) - 1 \\ &\leq \delta_{f'}(s,u) - 1 && \text{(by inequality (26.13))} \\ &= \delta_{f'}(s,v) - 2 && \text{(by equation (26.12))}\ , \end{aligned}$$

which contradicts our assumption that $\delta_{f'}(s, v) < \delta_f(s, v)$. We conclude that our assumption that such a vertex $v$ exists is incorrect. ■

The next theorem bounds the number of iterations of the Edmonds-Karp algorithm.

***Theorem 26.8***
If the Edmonds-Karp algorithm is run on a flow network $G = (V, E)$ with source $s$ and sink $t$, then the total number of flow augmentations performed by the algorithm is $O(VE)$.

***Proof*** We say that an edge $(u, v)$ in a residual network $G_f$ is ***critical*** on an augmenting path $p$ if the residual capacity of $p$ is the residual capacity of $(u, v)$, that is, if $c_f(p) = c_f(u, v)$. After we have augmented flow along an augmenting path, any critical edge on the path disappears from the residual network. Moreover, at least one edge on any augmenting path must be critical. We will show that each of the $|E|$ edges can become critical at most $|V|/2$ times.

Let $u$ and $v$ be vertices in $V$ that are connected by an edge in $E$. Since augmenting paths are shortest paths, when $(u, v)$ is critical for the first time, we have

$$\delta_f(s, v) = \delta_f(s, u) + 1 .$$

Once the flow is augmented, the edge $(u, v)$ disappears from the residual network. It cannot reappear later on another augmenting path until after the flow from $u$ to $v$ is decreased, which occurs only if $(v, u)$ appears on an augmenting path. If $f'$ is the flow in $G$ when this event occurs, then we have

$$\delta_{f'}(s, u) = \delta_{f'}(s, v) + 1 .$$

Since $\delta_f(s, v) \le \delta_{f'}(s, v)$ by Lemma 26.7, we have

$$\begin{aligned} \delta_{f'}(s, u) &= \delta_{f'}(s, v) + 1 \\ &\ge \delta_f(s, v) + 1 \\ &= \delta_f(s, u) + 2 . \end{aligned}$$

Consequently, from the time $(u, v)$ becomes critical to the time when it next becomes critical, the distance of $u$ from the source increases by at least 2. The distance of $u$ from the source is initially at least 0. The intermediate vertices on a shortest path from $s$ to $u$ cannot contain $s$, $u$, or $t$ (since $(u, v)$ on an augmenting path implies that $u \ne t$). Therefore, until $u$ becomes unreachable from the source, if ever, its distance is at most $|V| - 2$. Thus, after the first time that $(u, v)$ becomes critical, it can become critical at most $(|V| - 2)/2 = |V|/2 - 1$ times more, for a total of at most $|V|/2$ times. Since there are $O(E)$ pairs of vertices that can have an edge between them in a residual network, the total number of critical edges during the entire execution of the Edmonds-Karp algorithm is $O(VE)$. Each augmenting path has at least one critical edge, and hence the theorem follows. ■

Because we can implement each iteration of FORD-FULKERSON in $O(E)$ time when we find the augmenting path by breadth-first search, the total running time of the Edmonds-Karp algorithm is $O(VE^2)$. We shall see that push-relabel algorithms can yield even better bounds. The algorithm of Section 26.4 gives a method for achieving an $O(V^2E)$ running time, which forms the basis for the $O(V^3)$-time algorithm of Section 26.5.

## Exercises

***26.2-1***
Prove that the summations in equation (26.6) equal the summations in equation (26.7).

***26.2-2***
In Figure 26.1(b), what is the flow across the cut $(\{s, v_2, v_4\}, \{v_1, v_3, t\})$? What is the capacity of this cut?

***26.2-3***
Show the execution of the Edmonds-Karp algorithm on the flow network of Figure 26.1(a).

***26.2-4***
In the example of Figure 26.6, what is the minimum cut corresponding to the maximum flow shown? Of the augmenting paths appearing in the example, which one cancels flow?

***26.2-5***
Recall that the construction in Section 26.1 that converts a flow network with multiple sources and sinks into a single-source, single-sink network adds edges with infinite capacity. Prove that any flow in the resulting network has a finite value if the edges of the original network with multiple sources and sinks have finite capacity.

***26.2-6***
Suppose that each source $s_i$ in a flow network with multiple sources and sinks produces exactly $p_i$ units of flow, so that $\sum_{v \in V} f(s_i, v) = p_i$. Suppose also that each sink $t_j$ consumes exactly $q_j$ units, so that $\sum_{v \in V} f(v, t_j) = q_j$, where $\sum_i p_i = \sum_j q_j$. Show how to convert the problem of finding a flow $f$ that obeys

these additional constraints into the problem of finding a maximum flow in a single-source, single-sink flow network.

***26.2-7***
Prove Lemma 26.2.

***26.2-8***
Suppose that we redefine the residual network to disallow edges into $s$. Argue that the procedure FORD-FULKERSON still correctly computes a maximum flow.

***26.2-9***
Suppose that both $f$ and $f'$ are flows in a network $G$ and we compute flow $f \uparrow f'$. Does the augmented flow satisfy the flow conservation property? Does it satisfy the capacity constraint?

***26.2-10***
Show how to find a maximum flow in a network $G = (V, E)$ by a sequence of at most $|E|$ augmenting paths. (*Hint:* Determine the paths *after* finding the maximum flow.)

***26.2-11***
The ***edge connectivity*** of an undirected graph is the minimum number $k$ of edges that must be removed to disconnect the graph. For example, the edge connectivity of a tree is 1, and the edge connectivity of a cyclic chain of vertices is 2. Show how to determine the edge connectivity of an undirected graph $G = (V, E)$ by running a maximum-flow algorithm on at most $|V|$ flow networks, each having $O(V)$ vertices and $O(E)$ edges.

***26.2-12***
Suppose that you are given a flow network $G$, and $G$ has edges entering the source $s$. Let $f$ be a flow in $G$ in which one of the edges $(v, s)$ entering the source has $f(v, s) = 1$. Prove that there must exist another flow $f'$ with $f'(v, s) = 0$ such that $|f| = |f'|$. Give an $O(E)$-time algorithm to compute $f'$, given $f$, and assuming that all edge capacities are integers.

***26.2-13***
Suppose that you wish to find, among all minimum cuts in a flow network $G$ with integral capacities, one that contains the smallest number of edges. Show how to modify the capacities of $G$ to create a new flow network $G'$ in which any minimum cut in $G'$ is a minimum cut with the smallest number of edges in $G$.

## 26.3 Maximum bipartite matching

Some combinatorial problems can easily be cast as maximum-flow problems. The multiple-source, multiple-sink maximum-flow problem from Section 26.1 gave us one example. Some other combinatorial problems seem on the surface to have little to do with flow networks, but can in fact be reduced to maximum-flow problems. This section presents one such problem: finding a maximum matching in a bipartite graph. In order to solve this problem, we shall take advantage of an integrality property provided by the Ford-Fulkerson method. We shall also see how to use the Ford-Fulkerson method to solve the maximum-bipartite-matching problem on a graph $G = (V, E)$ in $O(VE)$ time.

### The maximum-bipartite-matching problem

Given an undirected graph $G = (V, E)$, a ***matching*** is a subset of edges $M \subseteq E$ such that for all vertices $v \in V$, at most one edge of $M$ is incident on $v$. We say that a vertex $v \in V$ is ***matched*** by the matching $M$ if some edge in $M$ is incident on $v$; otherwise, $v$ is ***unmatched***. A ***maximum matching*** is a matching of maximum cardinality, that is, a matching $M$ such that for any matching $M'$, we have $|M| \geq |M'|$. In this section, we shall restrict our attention to finding maximum matchings in bipartite graphs: graphs in which the vertex set can be partitioned into $V = L \cup R$, where $L$ and $R$ are disjoint and all edges in $E$ go between $L$ and $R$. We further assume that every vertex in $V$ has at least one incident edge. Figure 26.8 illustrates the notion of a matching in a bipartite graph.

The problem of finding a maximum matching in a bipartite graph has many practical applications. As an example, we might consider matching a set $L$ of machines with a set $R$ of tasks to be performed simultaneously. We take the presence of edge $(u, v)$ in $E$ to mean that a particular machine $u \in L$ is capable of performing a particular task $v \in R$. A maximum matching provides work for as many machines as possible.

### Finding a maximum bipartite matching

We can use the Ford-Fulkerson method to find a maximum matching in an undirected bipartite graph $G = (V, E)$ in time polynomial in $|V|$ and $|E|$. The trick is to construct a flow network in which flows correspond to matchings, as shown in Figure 26.8(c). We define the ***corresponding flow network*** $G' = (V', E')$ for the bipartite graph $G$ as follows. We let the source $s$ and sink $t$ be new vertices not in $V$, and we let $V' = V \cup \{s, t\}$. If the vertex partition of $G$ is $V = L \cup R$, the

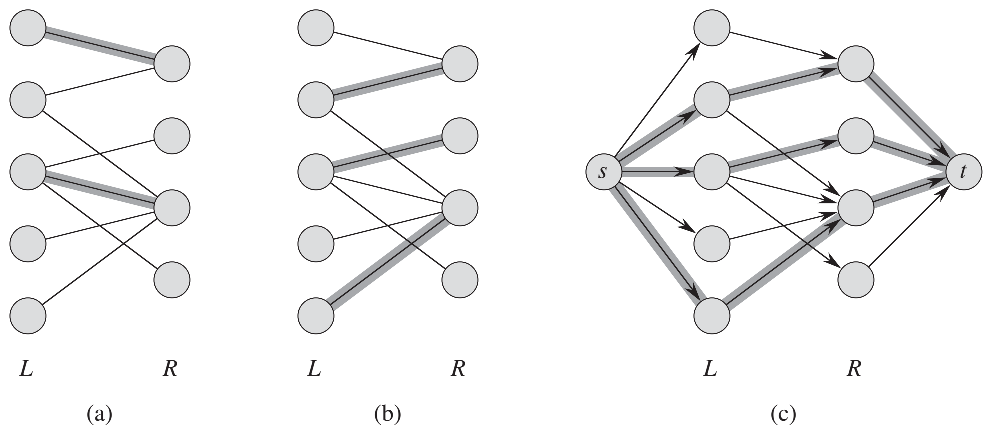

**Figure 26.8** A bipartite graph $G = (V, E)$ with vertex partition $V = L \cup R$. **(a)** A matching with cardinality 2, indicated by shaded edges. **(b)** A maximum matching with cardinality 3. **(c)** The corresponding flow network $G'$ with a maximum flow shown. Each edge has unit capacity. Shaded edges have a flow of 1, and all other edges carry no flow. The shaded edges from $L$ to $R$ correspond to those in the maximum matching from (b).

directed edges of $G'$ are the edges of $E$, directed from $L$ to $R$, along with $|V|$ new directed edges:

$$E' = \{(s, u) : u \in L\} \cup \{(u, v) : (u, v) \in E\} \cup \{(v, t) : v \in R\} .$$

To complete the construction, we assign unit capacity to each edge in $E'$. Since each vertex in $V$ has at least one incident edge, $|E| \geq |V|/2$. Thus, $|E| \leq |E'| = |E| + |V| \leq 3|E|$, and so $|E'| = \Theta(E)$.

The following lemma shows that a matching in $G$ corresponds directly to a flow in $G$'s corresponding flow network $G'$. We say that a flow $f$ on a flow network $G = (V, E)$ is ***integer-valued*** if $f(u, v)$ is an integer for all $(u, v) \in V \times V$.

***Lemma 26.9***
Let $G = (V, E)$ be a bipartite graph with vertex partition $V = L \cup R$, and let $G' = (V', E')$ be its corresponding flow network. If $M$ is a matching in $G$, then there is an integer-valued flow $f$ in $G'$ with value $|f| = |M|$. Conversely, if $f$ is an integer-valued flow in $G'$, then there is a matching $M$ in $G$ with cardinality $|M| = |f|$.

***Proof*** We first show that a matching $M$ in $G$ corresponds to an integer-valued flow $f$ in $G'$. Define $f$ as follows. If $(u, v) \in M$, then $f(s, u) = f(u, v) = f(v, t) = 1$. For all other edges $(u, v) \in E'$, we define $f(u, v) = 0$. It is simple to verify that $f$ satisfies the capacity constraint and flow conservation.

Intuitively, each edge $(u, v) \in M$ corresponds to one unit of flow in $G'$ that traverses the path $s \to u \to v \to t$. Moreover, the paths induced by edges in $M$ are vertex-disjoint, except for $s$ and $t$. The net flow across cut $(L \cup \{s\}, R \cup \{t\})$ is equal to $|M|$; thus, by Lemma 26.4, the value of the flow is $|f| = |M|$.

To prove the converse, let $f$ be an integer-valued flow in $G'$, and let

$$M = \{(u, v) : u \in L,\ v \in R, \text{ and } f(u, v) > 0\} .$$

Each vertex $u \in L$ has only one entering edge, namely $(s, u)$, and its capacity is 1. Thus, each $u \in L$ has at most one unit of flow entering it, and if one unit of flow does enter, by flow conservation, one unit of flow must leave. Furthermore, since $f$ is integer-valued, for each $u \in L$, the one unit of flow can enter on at most one edge and can leave on at most one edge. Thus, one unit of flow enters $u$ if and only if there is exactly one vertex $v \in R$ such that $f(u, v) = 1$, and at most one edge leaving each $u \in L$ carries positive flow. A symmetric argument applies to each $v \in R$. The set $M$ is therefore a matching.

To see that $|M| = |f|$, observe that for every matched vertex $u \in L$, we have $f(s, u) = 1$, and for every edge $(u, v) \in E - M$, we have $f(u, v) = 0$. Consequently, $f(L \cup \{s\}, R \cup \{t\})$, the net flow across cut $(L \cup \{s\}, R \cup \{t\})$, is equal to $|M|$. Applying Lemma 26.4, we have that $|f| = f(L \cup \{s\}, R \cup \{t\}) = |M|$. ■

Based on Lemma 26.9, we would like to conclude that a maximum matching in a bipartite graph $G$ corresponds to a maximum flow in its corresponding flow network $G'$, and we can therefore compute a maximum matching in $G$ by running a maximum-flow algorithm on $G'$. The only hitch in this reasoning is that the maximum-flow algorithm might return a flow in $G'$ for which some $f(u, v)$ is not an integer, even though the flow value $|f|$ must be an integer. The following theorem shows that if we use the Ford-Fulkerson method, this difficulty cannot arise.

***Theorem 26.10 (Integrality theorem)***
If the capacity function $c$ takes on only integral values, then the maximum flow $f$ produced by the Ford-Fulkerson method has the property that $|f|$ is an integer. Moreover, for all vertices $u$ and $v$, the value of $f(u, v)$ is an integer.

***Proof*** The proof is by induction on the number of iterations. We leave it as Exercise 26.3-2. ■

We can now prove the following corollary to Lemma 26.9.

***Corollary 26.11***
The cardinality of a maximum matching $M$ in a bipartite graph $G$ equals the value of a maximum flow $f$ in its corresponding flow network $G'$.

***Proof*** We use the nomenclature from Lemma 26.9. Suppose that $M$ is a maximum matching in $G$ and that the corresponding flow $f$ in $G'$ is not maximum. Then there is a maximum flow $f'$ in $G'$ such that $|f'| > |f|$. Since the capacities in $G'$ are integer-valued, by Theorem 26.10, we can assume that $f'$ is integer-valued. Thus, $f'$ corresponds to a matching $M'$ in $G$ with cardinality $|M'| = |f'| > |f| = |M|$, contradicting our assumption that $M$ is a maximum matching. In a similar manner, we can show that if $f$ is a maximum flow in $G'$, its corresponding matching is a maximum matching on $G$. ■

Thus, given a bipartite undirected graph $G$, we can find a maximum matching by creating the flow network $G'$, running the Ford-Fulkerson method, and directly obtaining a maximum matching $M$ from the integer-valued maximum flow $f$ found. Since any matching in a bipartite graph has cardinality at most $\min(L, R) = O(V)$, the value of the maximum flow in $G'$ is $O(V)$. We can therefore find a maximum matching in a bipartite graph in time $O(VE') = O(VE)$, since $|E'| = \Theta(E)$.

For construction of flow network

## Exercises

***26.3-1***
Run the Ford-Fulkerson algorithm on the flow network in Figure 26.8(c) and show the residual network after each flow augmentation. Number the vertices in $L$ top to bottom from 1 to 5 and in $R$ top to bottom from 6 to 9. For each iteration, pick the augmenting path that is lexicographically smallest.

***26.3-2***
Prove Theorem 26.10.

***26.3-3***
Let $G = (V, E)$ be a bipartite graph with vertex partition $V = L \cup R$, and let $G'$ be its corresponding flow network. Give a good upper bound on the length of any augmenting path found in $G'$ during the execution of FORD-FULKERSON.

***26.3-4*** ★
A ***perfect matching*** is a matching in which every vertex is matched. Let $G = (V, E)$ be an undirected bipartite graph with vertex partition $V = L \cup R$, where $|L| = |R|$. For any $X \subseteq V$, define the ***neighborhood*** of $X$ as

$$N(X) = \{y \in V : (x, y) \in E \text{ for some } x \in X\} ,$$