

Lecture 10: Max-Flow Min-Cut Theorem

*Lecturer: Abraham Ladha**Scribe(s): Richard Zhang*

This is the last lecture on this unit of graph algorithms, and one of the most important: Max-Flow Min-Cut. It comes from many real world problems, and can actually be applied to many more real world problems. In this lecture, we'll discuss the problem setting, an algorithm to solve it, and go through some real world examples.

Throughout the lecture, we'll talk about the algorithm at a high level; it's not terribly important you understand the internals of the algorithm, but it is important you understand what it does and you can apply it.

1 Max-Flow

A flow network $G = (V, E)$ is a subset of weighted graphs with a single source s and a single sink t . Every edge in a flow network has a non-negative capacity $0 \leq c(u, v)$. We then make the following assumptions, if there is an edge from u to v , there is no edge from v to u , and all vertices lies in a path between the source and the sink ($s \rightarrow \dots v \rightarrow t$).

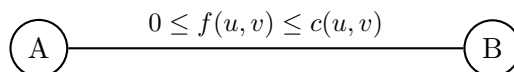


Figure 1: Flow is less than or equal to capacity.

Now, to define our problem, we have two constraints. The capacity constraint: each edge u, v in a flow network has some flow $f(u, v)$ attached such $0 \leq f(u, v) \leq c(u, v)$. The flow conservation that the flow entering a node must be the same as the flow exiting a node ($\sum_{v \in V} f(u, v) = \sum_{v \in V} f(v, u)$). Then, in the maximum-flow problem, given G , s , t , and c we try to maximize the flow moving across the network (or the total flow moving out of the source or into the sink).

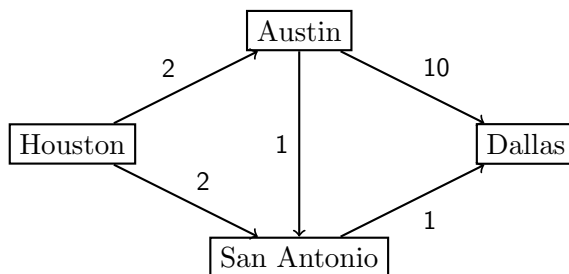


Figure 2: A simple flow network illustrating oil movement between cities in Texas.

If you're a little confused hopefully the following example can help; consider you're shipping oil through pipes from an oil refinery to a consumer. Say your oil refiner is in

Houston, Texas and you want to transfer oil to a consumer in Dallas. You have pipes each with individual capacity (barrels/hour) connecting Houston to Austin (2 barrels/hour), Houston to San Antonio (2 barrels per hour), Austin to San Antonio (1 barrel/hour), Austin to Dallas (10 barrel/hour), and San Antonio to Dallas (1 barrel/hour).

You don't have to actually move this amount of oil through these pipes though, you're simply limited by this capacity. Since Austin and San Antonio are pass through cities, the can not produce or consume oil. Therefore, if more oil flows into a city than it can move out, that oil is wasted, so it makes no sense to move in and out unequal amounts of oil.

There are many flows which could go through this network. The most obvious is that we could move no oil through the network. Alternatively, you could move oil only through the top path (from Houston to Austin and Austin to Dallas), getting two barrels per hour. However these aren't particularly helpful because moving nothing or moving a little of something does not make full use of the network.

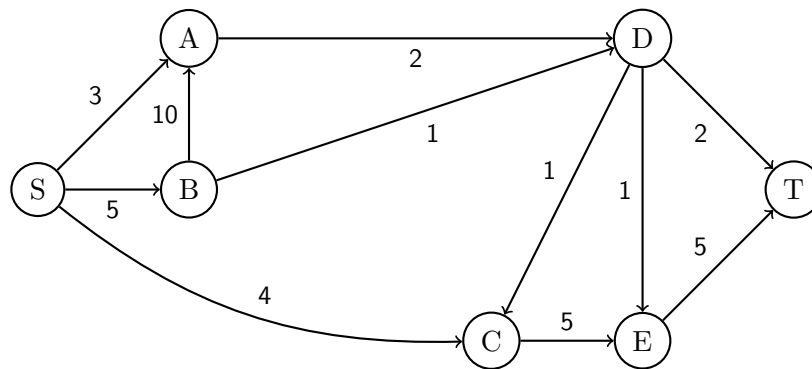


Figure 3: A large example flow network referred to throughout the lecture.

Instead we'd like to move the maximum flow through the network. Since this network is fairly small, it's not hard to see that you can move 3 barrels per hour through the network. It would be nice since realistically there are many more cities and many more pipes. For instance, consider the network above; the flow is 7 as we'll find later, but this is hard to find without an algorithm. Furthermore since this can be applied to many more applications, it would be nice to find an algorithm to work on general graphs.

2 Min-Cut

When we talk about Max-Flow, we often also talk about Min-Cut; we'll understand why later, but for now we'll introduce Min-Cut. An s-t cut of a flow network $G = (V, E)$ are sets of vertices $L \subset V$ and $R = V - S$, such that $s \in L$, $t \in R$. The capacity of an s-t cut (L, R) is

$$c(L, R) = \sum_{u \in L, v \in R} c(u, v)$$

¹

¹For all vertices in L , and all vertices in R , if there is a capacity (or an edge) between them, add the capacity. Note that we don't consider the edges that are solely in L , solely in R , or that go from R to L .

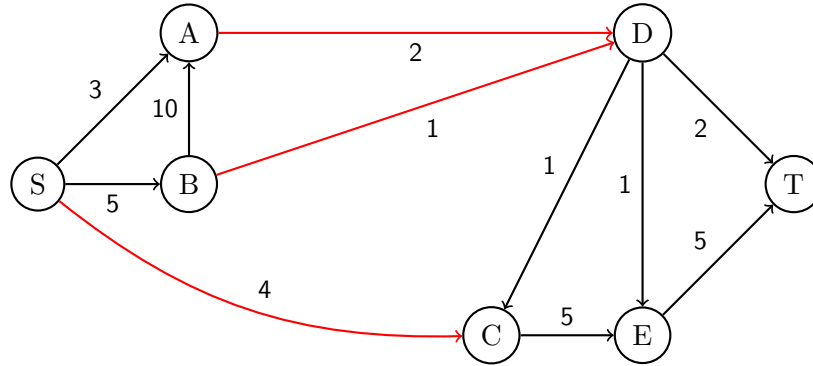


Figure 4: An s-t cut through the graph with $L = \{S, A, B\}$.

Consider the following s-t cuts on the large graph above where L is equal to $\{s\}$, $\{s, a, b, c, d, e\}$, $\{s, a, b\}$. In the first s-t cut, there are three edges from L to R , SA , SB , and SC . The capacity across this s-t cut is $3 + 5 + 4$ which is 12. The next s-t cut has two edges DT and ET which is $2 + 5$ or 7. The last s-t cut has edges AD , BD and SC , making the capacity 7.

Like Max-Flow, Min-Cut also has many real world applications. We won't get into these too much, but you can imagine you're a city planner and there a number of roads moving cars within the city. You want to make sure that construction on one of the roads will not cause a large outage. You can find the minimum s-t cut through the network and increase capacity of that s-t cut, then the smallest s-t cut will cause less of a problem.² We see though that this too is an important problem, and as it turns out, the two are related.

3 Max-Flow Min-Cut Theorem

It turns out individual s-t cuts can be really high because the capacities can be arbitrarily high, but flow is limited by the global value of all the edges.

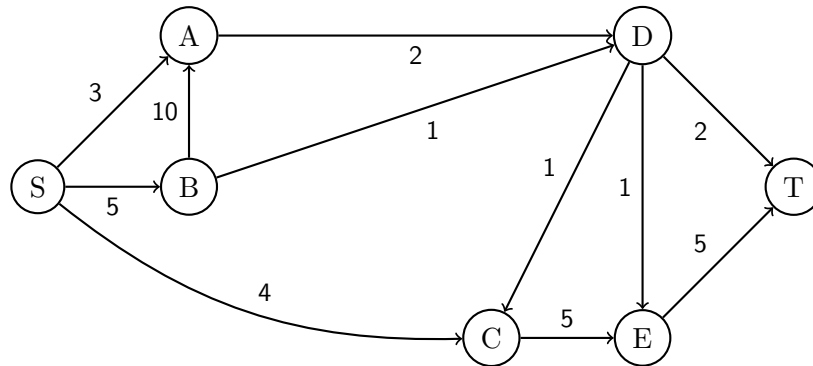


Figure 5: A large flow network with Max-Flow 7 and Min-Cut 7.

In the above graph, as we've mentioned the flow could zero to seven; the smallest s-t

²This of course assumes that each edge has equal rate or that no edge holds all the weight because then of course, you could cut that one edge and drastically decrease capacity.

cut capacity we've found is also seven, while the largest is around 20. You may come to the conclusion that the capacity of all s-t cuts is greater than or equal to the possible flows through the network, and this would be right. This might turn some gears, and actually the Max-Flow Min-Cut theorem states that the maximum flow is equal to the minimum cut.

4 Baseball Elimination

We've already talked about a number of applications, but another appreciable example is for baseball elimination. Say you have the following data about the top four teams.

i	Team	Wins	Losses	To Play	PHL	NYC	BOS	CHI	HOU
1	Phillies	75	59	28	-	6	8	7	7
2	Yankees	71	63	28	6	-	8	7	7
3	Red Sox	69	66	31	8	8	-	8	7
4	Cubs	63	72	28	7	7	8	-	6
5	Astros	49	86	27	7	7	7	6	-

Figure 6: The baseball statistics of several teams playing.

From this, it might seem that the Astros have a chance to make it to the finals if they win all of their remaining 27 games, and the Phillies lose all their games, putting the two of them at 76 wins and 75 wins respectively. The issue with this analysis is that there are knock on effects to this, and the Phillies losing all of their games implies that the Red Sox would have 77 wins with 23 games remaining, meaning that the Red Sox would be first and the Astros still cannot be 1st with this scenario. This goes on, but it turns out that the Astros were eliminated the whole time. We could keep going but it's not important.

Instead of going through this, we can actually formulate this as a Max-Flow problem, and then solve it. First we define the baseball elimination problems such that we have a set of teams X , a particular team $x \notin X$, there are remaining games between the teams in X . Each team has currently w_i wins for team i , r_i remaining games for team i , and $g_{i,j}$ remaining games between teams i, j .

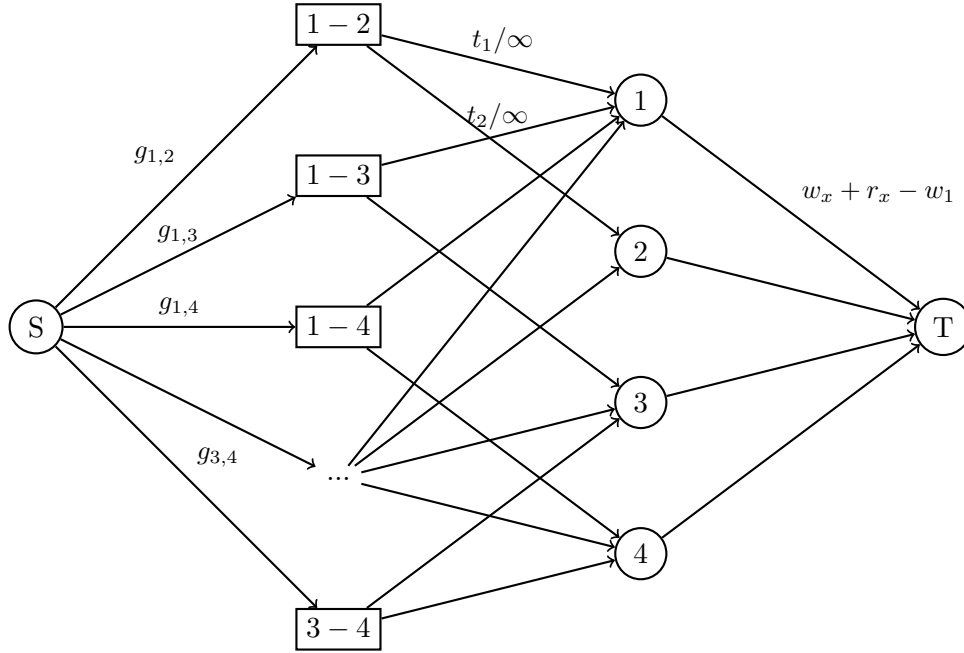


Figure 7: An example flow network for baseball.

Intuitively, it's easy to think of a single flow as a game being played, so let's set up the network like that. Our network will have a first layer which represents the games being played between each two teams $i, j \in X$ where $i < j$. The weight going from the source to those nodes will be the number of games left between the pair g_i, j . Then, each team either wins or loses a game, so we can draw two lines to another layer of nodes, and these lines will have infinite capacity. We have to call the flow through these lines something, so let's call them t_1, t_2 . Due to flow conservation, $t_1 + t_2 = g_{1,2}$ where t_1 would be the number of games team 1 won against team 2. Lastly, we'll take all of these team nodes and connect them to the sink. The capacity of these edges will be $w_x + r_x - w_i$. If $w_x + r_x < w_i$, then team x was eliminated to began with, so there is no need to find the flow of this graph. Therefore, we can assume that $w_x + r_x - w_i$ is non-negative and represents the amount of games that team i can win without exceeding the maximum number of wins that team x can achieve and eliminating team x all together.

To cap it off³, we'll define the following value $R = \sum_{i \in X} r_i$ which is the total amount of games remaining. Here's the claim: if a flow of R exists in the network, then team x is not eliminated. The logic here is that if a flow of R exists, it must have filled all edges leaving S since the sum of those edges would also be the total amount of games remaining and equal to R . As a result, all the games have been played, and still the teams all have at most as many wins as team x because of the capacities on the final layer. In other words, the last layer, which is restricting how much the other teams can win, is not restricting the number of games that can be played for team x to win.

Alternatively, if a flow of R does not exist in the network, then the team is eliminated. Then that means that all the teams have met the capacities put upon them by the last layer,

³Pun not intended

meaning that all the teams have at least tied team x . However, there are still R minus the flow of the games left to be played, making it impossible for x to have the maximum amount of wins.

5 Ford-Fulkerson

Ford-Fulkerson sort of isn't an algorithm; the pair wrote an article which outlined an approach you could take to solving Max-Flow Min-Cut, but they did not give a strict algorithm for solving it. This means, that the different implementations of the algorithm have different runtimes, so you should not worry about the runtime or in-depth implementation details.⁴

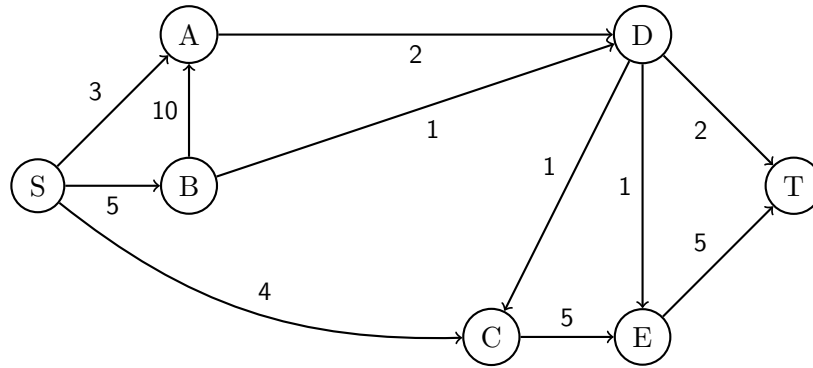


Figure 8: A large flow network for Ford-Fulkerson.

At a high level, suppose we modified depth first search to try to greedily solve this problem. Say we do something along the lines of keeping track of the maximum flow entering nodes and then use that to calculate what we think the flow is. The issue with this approach is that we'll need to correct for things we may have overcompensated for and we'll need to do a lot of backtracking.

Ford-Fulkerson is able to retain metadata so it doesn't have to do this backtracking. More specifically, it maintains a residual network $G_f = (V, E_f)$ for the original graph G .

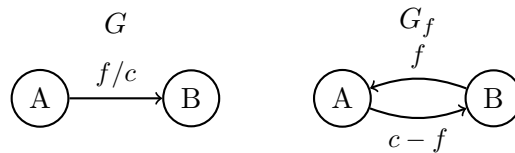


Figure 9: Comparison of edges in G vs G_f .

For each edge in G , G_f has forward edges representing the remaining capacity $c - f$ and backwards edges representing the current flow f for that edge. The Ford Fulkerson method then finds augmenting paths on G_f to continuously increase the flow of the graph.

⁴If you're curious what an implementation of Ford-Fulkerson looks like, refer to the website of Stephen Huan. The algorithm they implement is Edmonds-Karp.

```

def Ford-fulkerson(G):
    initialize the residual graph Gf as G
    initialize flow to zero

    while Gf has an (s-t) path
        find a path and compute its bottleneck b
        augment Gf by subtracting the path from Gf with b
        add reverse edges of the path from Gf with b

    return flow

```

Figure 10: Ford-Fulkerson

Again we'll go over this at a high level: given an (s-t) path, find the bottleneck, augment the residual graph with this information, and add to the flow. The ways we generate (s-t) paths are again not explicitly mentioned. A popular way to do this is with BFS.⁵

The bottleneck on the path is the edge with minimum capacity on that path. If you consider the path *SADCET*, the bottleneck is edge *DC* because that has capacity one.

The last and most important thing about Ford-Fulkerson is the trick it does on the residual graph so it doesn't have to do any big backtracking. When we have a path and a bottleneck, we augment the graph such that all the edges on that path are subtracted by the bottleneck, and add the reverse of the path with the bottleneck.

This may sound a bit complicated so we'll do an example. Suppose we have the following paths in sequence,⁶ *SADCET*, *SADET*, *SCET*, *SCEDT*, and *SBDT*. We'll go through the Ford-Fulkerson algorithm.

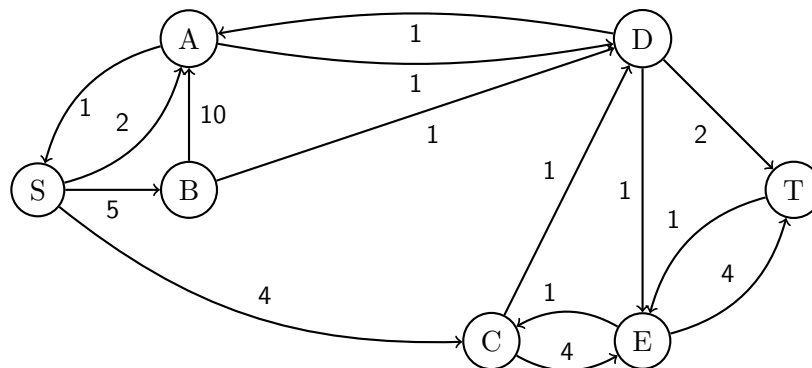


Figure 11: Ford-Fulkerson with modifications from path *SADCET*. Our total flow is 1.

⁵If you're interested think about why this might be.

⁶Again, the ways in which you generate (s-t) paths can cause different things to happen but you should get the same flow provided you're doing everything else correctly.

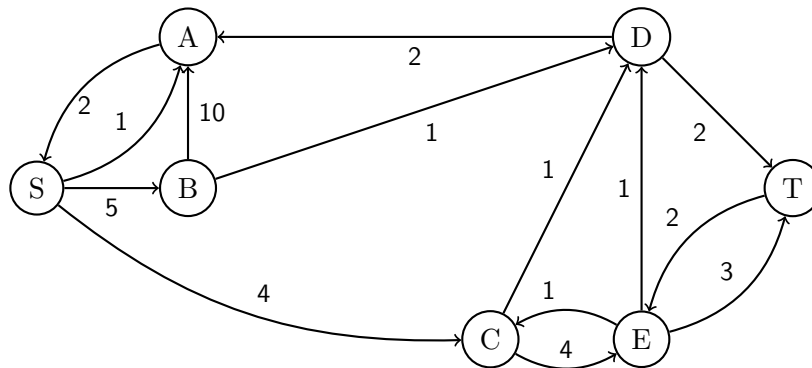


Figure 12: Ford-Fulkerson with modifications from path *SADET*. Our total flow is 2.

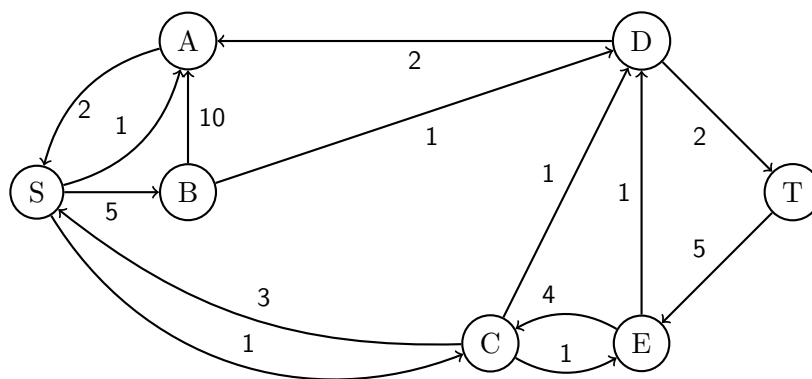


Figure 13: Ford-Fulkerson with modifications from path *SCET*. Our total flow is 5.

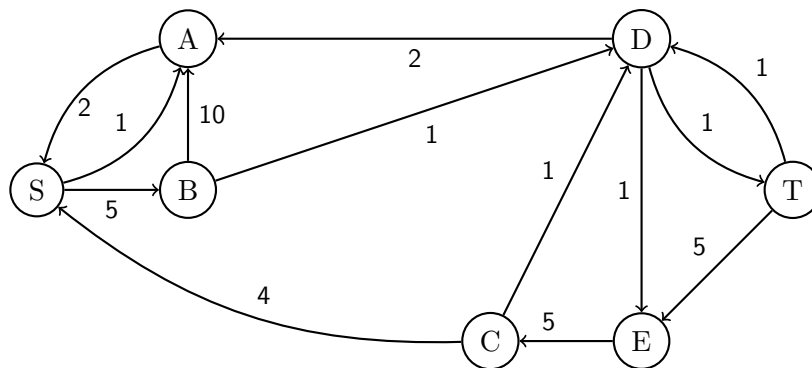


Figure 14: Ford-Fulkerson with modifications from path *SCEDT*. Our total flow is 6.

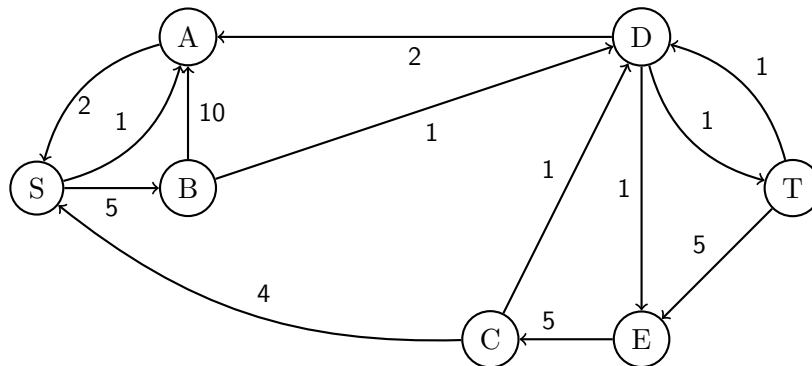


Figure 15: Ford-Fulkerson with modifications from path *SBDT*. Our total flow is 7.

Now there are no more (s-t) paths, meaning there can be no more iteration on the graph. We end with a final flow of 7.

We need to justify that the flow returned is maximum. Certainly since we found a cut equaling a flow in this example, then that would have to be the maximum flow and the minimum cut. In general, we can show that the max flow is achieved when there is no (s-t) path in G_f . Suppose for contradiction that flow f was the maximum but the corresponding G_f had an augmenting (s-t) path with bottleneck b . We then can add b to the flow for each edge in G (not G_f) corresponding to the (s-t) path and produce a greater flow of $f + b$, contradicting that f was the max flow.

Additionally, the algorithm also produces the minimum cut. Call `explore(S)` on G_f and take the set of nodes which are reachable from it as L and everything else as R . Then then in G , these edges had to have totally saturated capacity since there don't exist any $L - R$ forward edges in G .

6 Proof of the Theorem

We want to prove that the algorithm terminates when it finds a maximum flow, and along the way, show that the maximum flow is equal to the minimum cut.

If we know that every flow f is less than cut c_1 , and less than cut c_2 , then we know that $f \leq c_1$ and $f \leq c_2$. We then also know that $f \leq \min(c_1, c_2)$. We then know that every flow is less than every cut, so we know that the max flow is less than or equal to the minimum cut. $\max(\text{flows}) \leq \min(\text{cuts})$. It is then sufficient for us to show there is a maximum flow which is equivalent to some cut.

First we prove that if f is a max flow, then G_f has no $(s - t)$ path (the stopping condition of the ford-fulkerson algorithm.) Suppose to the contrary that f was maximum with G_f having an $(s - t)$ -path still. Compute the bottleneck b of the path and note that we may increase the flow by this much exactly along this path. This gives us a new flow of $f + b$, contradicting the fact that f was maximum.

Next we show that if f is a max flow, then it is equivalent to some cut. Since f is maximum, there is no $(s - t)$ path in G_f . Call `explore(G_f, s)` and note that since t is not reachable from s , this is a partition, a cut of G . We called `explore` on the residual graph but got a cut on the original graph. Consider the edges in G_f with one end in L and the

other in R . Let these edges be like (u, v) with $u \in L$ and $v \in R$. These edges have some capacity and flow f, c . Note that the forward edge $c - f$ must be zero in G_f , otherwise v would be reached by the explore call, so $c - f = 0$ implies that $c = f$. The edges of this cut are all fully saturated, so the capacity of the cut is equivalent to the flow on the cut. Also notice that any edges $R - L$, going backwards, must be zero. The flow is then equivalent to the flows of the outgoing edges on this cut, minus the flow of the incoming edges. The outgoing edges sum to the capacity of the cut, and the incoming edges sum to zero, so we see that this maximum flow is equivalent to this cut, meaning the cut was minimal. We may conclude then that the maximum flow is equal to the minimum cut.

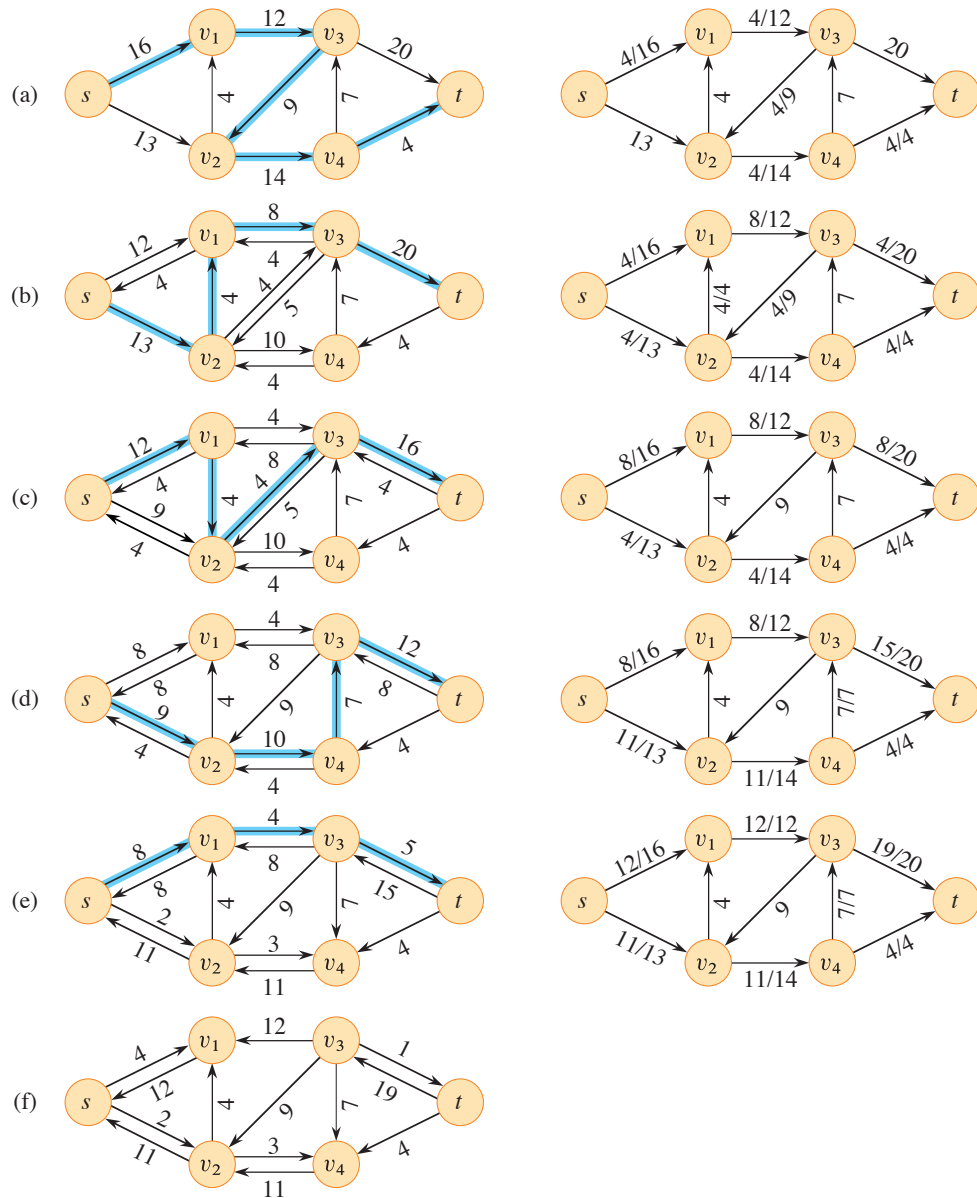


Figure 24.6 The execution of the basic Ford-Fulkerson algorithm. (a)–(e) Successive iterations of the **while** loop. The left side of each part shows the residual network G_f from line 3 with a blue augmenting path p . The right side of each part shows the new flow f that results from augmenting f by f_p . The residual network in (a) is the input flow network G . (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.

Figure 7.6 The max-flow algorithm applied to the network of Figure 7.4. At each iteration, the current flow is shown on the left and the residual graph on the right. The paths chosen are shown in bold.

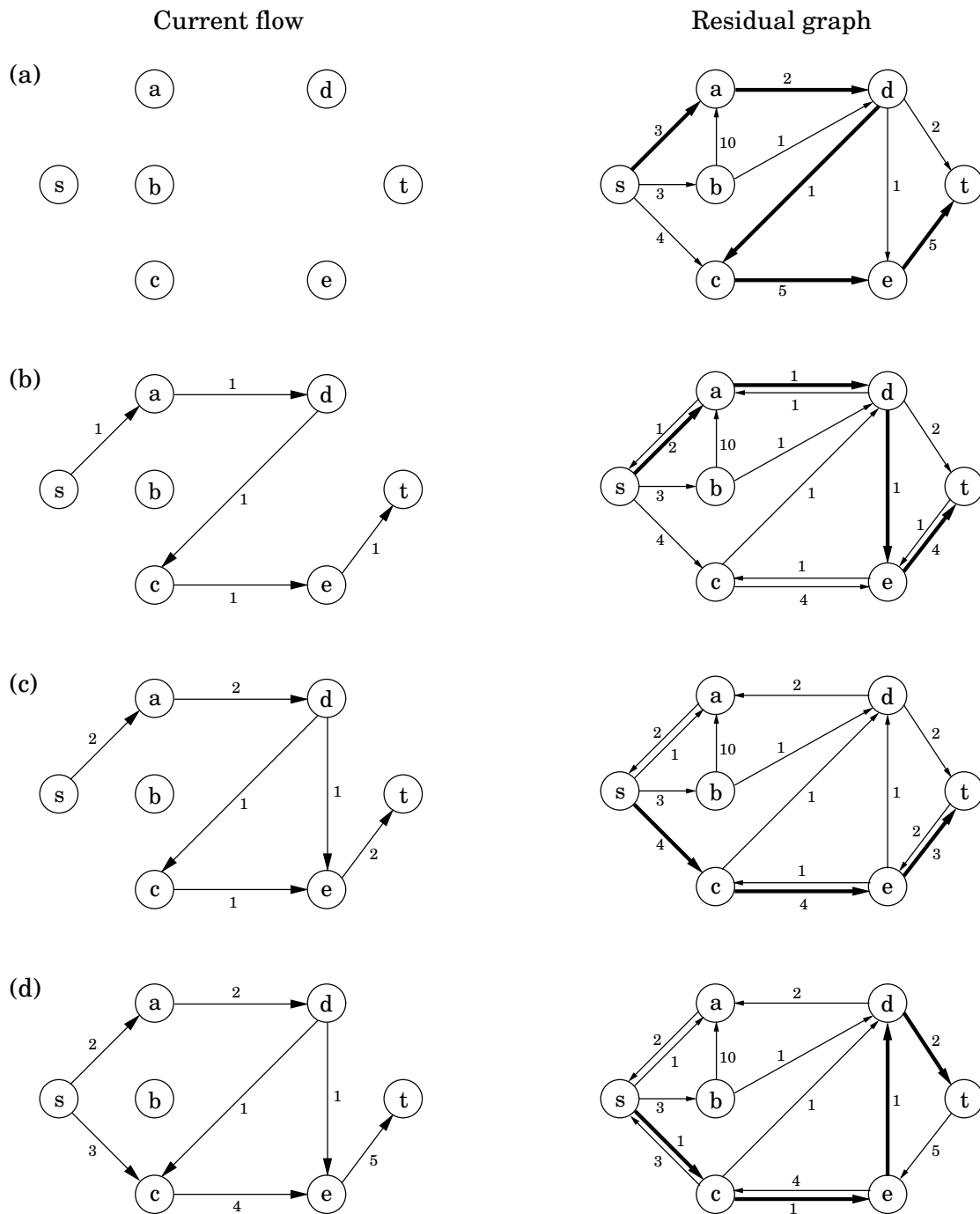


Figure 7.6 *Continued*

