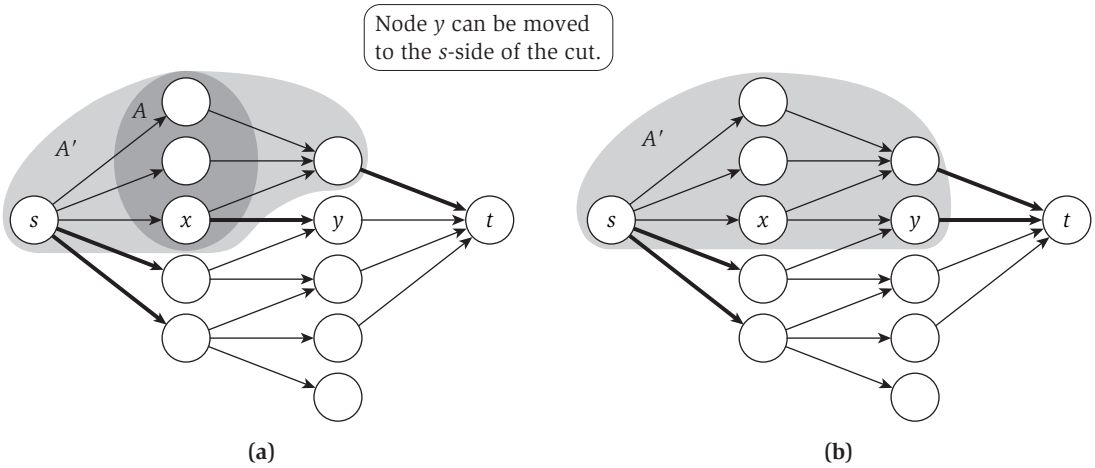

Figure 7.11 (a) A minimum cut in proof of (7.40). (b) The same cut after moving node y to the A' side. The edges crossing the cut are dark.

don't have to be concerned about nodes $x \in X$ that are not in A. The two ends of the edge (x, y) will be on different sides of the cut, but this edge does not add to the capacity of the cut, as it goes from B' to A'.)

Next consider the capacity of this minimum cut (A', B') that has $\Gamma(A) \subseteq A'$ as shown in Figure 7.11(b). Since all neighbors of A belong to A', we see that the only edges out of A' are either edges that leave the source s or that enter the sink t. Thus the capacity of the cut is exactly

$$c(A', B') = |X \cap B'| + |Y \cap A'|.$$

Notice that $|X \cap B'| = n - |A|$, and $|Y \cap A'| \geq |\Gamma(A)|$. Now the assumption that $c(A', B') < n$ implies that

$$n - |A| + |\Gamma(A)| \leq |X \cap B'| + |Y \cap A'| = c(A', B') < n.$$

Comparing the first and the last terms, we get the claimed inequality $|A| > |\Gamma(A)|$. ■

7.6 Disjoint Paths in Directed and Undirected Graphs

In Section 7.1, we described a flow f as a kind of "traffic" in the network. But our actual definition of a flow has a much more static feel to it: For each edge e, we simply specify a number $f(e)$ saying the amount of flow crossing e. Let's see if we can revive the more dynamic, traffic-oriented picture a bit, and try formalizing the sense in which units of flow "travel" from the source to

the sink. From this more dynamic view of flows, we will arrive at something called the *s-t Disjoint Paths Problem*.

The Problem

In defining this problem precisely, we will deal with two issues. First, we will make precise this intuitive correspondence between units of flow traveling along paths, and the notion of flow we've studied so far. Second, we will extend the Disjoint Paths Problem to *undirected* graphs. We'll see that, despite the fact that the Maximum-Flow Problem was defined for a directed graph, it can naturally be used also to handle related problems on undirected graphs.

We say that a set of paths is *edge-disjoint* if their edge sets are disjoint, that is, no two paths share an edge, though multiple paths may go through some of the same nodes. Given a directed graph $G = (V, E)$ with two distinguished nodes $s, t \in V$, the *Directed Edge-Disjoint Paths Problem* is to find the maximum number of edge-disjoint *s-t* paths in G. The *Undirected Edge-Disjoint Paths Problem* is to find the maximum number of edge-disjoint *s-t* paths in an undirected graph G. The related question of finding paths that are not only edge-disjoint, but also node-disjoint (of course, other than at nodes s and t) will be considered in the exercises to this chapter.

Designing the Algorithm

Both the directed and the undirected versions of the problem can be solved very naturally using flows. Let's start with the directed problem. Given the graph $G = (V, E)$, with its two distinguished nodes s and t, we define a flow network in which s and t are the source and sink, respectively, and with a capacity of 1 on each edge. Now suppose there are k edge-disjoint *s-t* paths. We can make each of these paths carry one unit of flow: We set the flow to be $f(e) = 1$ for each edge e on any of the paths, and $f(e') = 0$ on all other edges, and this defines a feasible flow of value k.

(7.41) *If there are k edge-disjoint paths in a directed graph G from s to t, then the value of the maximum s-t flow in G is at least k.*

Suppose we could show the converse to (7.41) as well: If there is a flow of value k, then there exist k edge-disjoint *s-t* paths. Then we could simply compute a maximum *s-t* flow in G and declare (correctly) this to be the maximum number of edge-disjoint *s-t* paths.

We now proceed to prove this converse statement, confirming that this approach using flow indeed gives us the correct answer. Our analysis will also provide a way to extract k edge-disjoint paths from an integer-valued flow sending k units from s to t. Thus computing a maximum flow in G will

not only give us the maximum *number* of edge-disjoint paths, but the paths as well.

Analyzing the Algorithm

Proving the converse direction of (7.41) is the heart of the analysis, since it will immediately establish the optimality of the flow-based algorithm to find disjoint paths.

To prove this, we will consider a flow of value at least k, and construct k edge-disjoint paths. By (7.14), we know that there is a maximum flow f with integer flow values. Since all edges have a capacity bound of 1, and the flow is integer-valued, each edge that carries flow under f has exactly one unit of flow on it. Thus we just need to show the following.

(7.42) *If f is a 0-1 valued flow of value v, then the set of edges with flow value $f(e) = 1$ contains a set of v edge-disjoint paths.*

Proof. We prove this by induction on the number of edges in f that carry flow. If $v = 0$, there is nothing to prove. Otherwise, there must be an edge (s, u) that carries one unit of flow. We now "trace out" a path of edges that must also carry flow: Since (s, u) carries a unit of flow, it follows by conservation that there is some edge (u, v) that carries one unit of flow, and then there must be an edge (v, w) that carries one unit of flow, and so forth. If we continue in this way, one of two things will eventually happen: Either we will reach t, or we will reach a node v for the second time.

If the first case happens—we find a path P from s to t—then we'll use this path as one of our v paths. Let f' be the flow obtained by decreasing the flow values on the edges along P to 0. This new flow f' has value $v - 1$, and it has fewer edges that carry flow. Applying the induction hypothesis for f', we get $v - 1$ edge-disjoint paths, which, along with path P, form the v paths claimed.

If P reaches a node v for the second time, then we have a situation like the one pictured in Figure 7.12. (The edges in the figure all carry one unit of flow, and the dashed edges indicate the path traversed so far, which has just reached a node v for the second time.) In this case, we can make progress in a different way.

Consider the cycle C of edges visited between the first and second appearances of v. We obtain a new flow f' from f by decreasing the flow values on the edges along C to 0. This new flow f' has value v, but it has fewer edges that carry flow. Applying the induction hypothesis for f', we get the v edge-disjoint paths as claimed. ■

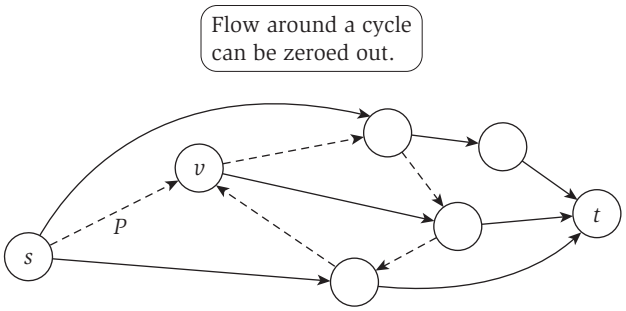

Figure 7.12 The edges in the figure all carry one unit of flow. The path P of dashed edges is one possible path in the proof of (7.42).

We can summarize (7.41) and (7.42) in the following result.

(7.43) *There are k edge-disjoint paths in a directed graph G from s to t if and only if the value of the maximum value of an s-t flow in G is at least k.*

Notice also how the proof of (7.42) provides an actual procedure for constructing the k paths, given an integer-valued maximum flow in G. This procedure is sometimes referred to as a *path decomposition* of the flow, since it "decomposes" the flow into a constituent set of paths. Hence we have shown that our flow-based algorithm finds the maximum number of edge-disjoint s-t paths and also gives us a way to construct the actual paths.

Bounding the Running Time For this flow problem, $C = \sum_{e \text{ out of } s} c_e \leq |V| = n$, as there are at most $|V|$ edges out of s, each of which has capacity 1. Thus, by using the $O(mC)$ bound in (7.5), we get an integer maximum flow in $O(mn)$ time.

The path decomposition procedure in the proof of (7.42), which produces the paths themselves, can also be made to run in $O(mn)$ time. To see this, note that this procedure, with a little care, can produce a single path from s to t using at most constant work per edge in the graph, and hence in $O(m)$ time. Since there can be at most $n - 1$ edge-disjoint paths from s to t (each must use a different edge out of s), it therefore takes time $O(mn)$ to produce all the paths.

In summary, we have shown

(7.44) *The Ford-Fulkerson Algorithm can be used to find a maximum set of edge-disjoint s-t paths in a directed graph G in $O(mn)$ time.*

A Version of the Max-Flow Min-Cut Theorem for Disjoint Paths The Max-Flow Min-Cut Theorem (7.13) can be used to give the following characteri-

zation of the maximum number of edge-disjoint s-t paths. We say that a set $F \subseteq E$ of edges *separates* s from t if, after removing the edges F from the graph G, no s-t paths remain in the graph.

(7.45) *In every directed graph with nodes s and t, the maximum number of edge-disjoint s-t paths is equal to the minimum number of edges whose removal separates s from t.*

Proof. If the removal of a set $F \subseteq E$ of edges separates s from t, then each s-t path must use at least one edge from F, and hence the number of edge-disjoint s-t paths is at most $|F|$.

To prove the other direction, we will use the Max-Flow Min-Cut Theorem (7.13). By (7.43) the maximum number of edge-disjoint paths is the value ν of the maximum s-t flow. Now (7.13) states that there is an s-t cut (A, B) with capacity ν. Let F be the set of edges that go from A to B. Each edge has capacity 1, so $|F| = \nu$ and, by the definition of an s-t cut, removing these ν edges from G separates s from t. ■

This result, then, can be viewed as the natural special case of the Max-Flow Min-Cut Theorem in which all edge capacities are equal to 1. In fact, this special case was proved by Menger in 1927, much before the full Max-Flow Min-Cut Theorem was formulated and proved; for this reason, (7.45) is often called *Menger's Theorem*. If we think about it, the proof of Hall's Theorem (7.40) for bipartite matchings involves a reduction to a graph with unit-capacity edges, and so it can be proved using Menger's Theorem rather than the general Max-Flow Min-Cut Theorem. In other words, Hall's Theorem is really a special case of Menger's Theorem, which in turn is a special case of the Max-Flow Min-Cut Theorem. And the history follows this progression, since they were discovered in this order, a few decades apart.[2]

Extensions: Disjoint Paths in Undirected Graphs

Finally, we consider the disjoint paths problem in an undirected graph G. Despite the fact that our graph G is now undirected, we can use the maximum-flow algorithm to obtain edge-disjoint paths in G. The idea is quite simple: We replace each undirected edge (u, v) in G by two directed edges (u, v) and

[2] In fact, in an interesting retrospective written in 1981, Menger relates his version of the story of how he first explained his theorem to König, one of the independent discoverers of Hall's Theorem. You might think that König, having thought a lot about these problems, would have immediately grasped why Menger's generalization of his theorem was true, and perhaps even considered it obvious. But, in fact, the opposite happened; König didn't believe it could be right and stayed up all night searching for a counterexample. The next day, exhausted, he sought out Menger and asked him for the proof.

(v, u), and in this way create a directed version G' of G. (We may delete the edges into s and out of t, since they are not useful.) Now we want to use the Ford-Fulkerson Algorithm in the resulting directed graph. However, there is an important issue we need to deal with first. Notice that two paths P_1 and P_2 may be edge-disjoint in the directed graph and yet share an edge in the undirected graph G: This happens if P_1 uses directed edge (u, v) while P_2 uses edge (v, u). However, it is not hard to see that there always exists a maximum flow in any network that uses at most *one* out of each pair of oppositely directed edges.

(7.46) *In any flow network, there is a maximum flow f where for all opposite directed edges $e = (u, v)$ and $e' = (v, u)$, either $f(e) = 0$ or $f(e') = 0$. If the capacities of the flow network are integral, then there also is such an integral maximum flow.*

Proof. We consider any maximum flow f, and we modify it to satisfy the claimed condition. Assume $e = (u, v)$ and $e' = (v, u)$ are opposite directed edges, and $f(e) \neq 0$, $f(e') \neq 0$. Let δ be the smaller of these values, and modify f by decreasing the flow value on both e and e' by δ. The resulting flow f' is feasible, has the same value as f, and its value on one of e and e' is 0. ■

Now we can use the Ford-Fulkerson Algorithm and the path decomposition procedure from (7.42) to obtain edge-disjoint paths in the undirected graph G.

(7.47) *There are k edge-disjoint paths in an undirected graph G from s to t if and only if the maximum value of an s-t flow in the directed version G' of G is at least k. Furthermore, the Ford-Fulkerson Algorithm can be used to find a maximum set of disjoint s-t paths in an undirected graph G in $O(mn)$ time.*

The undirected analogue of (7.45) is also true, as in any s-t cut, at most one of the two oppositely directed edges can cross from the s-side to the t-side of the cut (for if one crosses, then the other must go from the t-side to the s-side).

(7.48) *In every undirected graph with nodes s and t, the maximum number of edge-disjoint s-t paths is equal to the minimum number of edges whose removal separates s from t.*

7.7 Extensions to the Maximum-Flow Problem

Much of the power of the Maximum-Flow Problem has essentially nothing to do with the fact that it models traffic in a network. Rather, it lies in the fact that many problems with a nontrivial combinatorial search component can