Chapter 4
Networks

Section 4.1 Flows

You are designing an oil pipeline. The pipeline is to have a pumping station p and a receiving station r. Further, to minimize the possibility that pipe repairs will completely halt the flow of oil, there will be several routes. There will be a northern route and a southern route, and each will have two intermediate pumping stations, n_1 and n_2 on the northern route and s_1 and s_2 on the southern route. Finally, there will also be a pipeline from s_2 to n_1, from s_2 to n_2 and from n_2 to s_1. Our pipeline will also cross several county borders, and each county has a different restriction on the size of underground pipes. Thus, the amount of oil we can pump on any section of the pipeline differs. For example, from p to n_1 the maximum capacity is 4000 barrels of oil per hour (b/h). Figure 4.1.1 shows the restrictions on pipe capacity for each section of pipe in our system. Under all these conditions, what is the maximum rate at which oil can reach the receiving station r?

capacity (b/h) -->	n_1	n_2	s_1	s_2	r
p	4000		3000		
n_1		3000			
n_2			1700		4500
s_1				5400	
s_2	1000	2000			3200

Figure 4.1.1. Pumping capacity restrictions along pipeline sections.

Once again we wish to form a graph model for our problem. Clearly, the pipes and pumping stations can be modeled by a digraph. Our problem is to devise a method for determining how the pipe capacity restrictions and the pipeline interconnections (the graph structure) restrict the rate at which oil reaches r.

Before we attempt to solve this problem, we should note several fairly natural restrictions. Clearly, no pipe can carry more oil then its capacity allows. Also, all intermediate stations can only pump out as much oil as they receive. We now formulate our problem in more mathematical terms.

A *network* N is a 5-tuple $N = (V, E, s, t, c)$ where (V, E) is a digraph with distinguished vertices s and t called the *source* and *sink*, respectively. Each arc e of N is

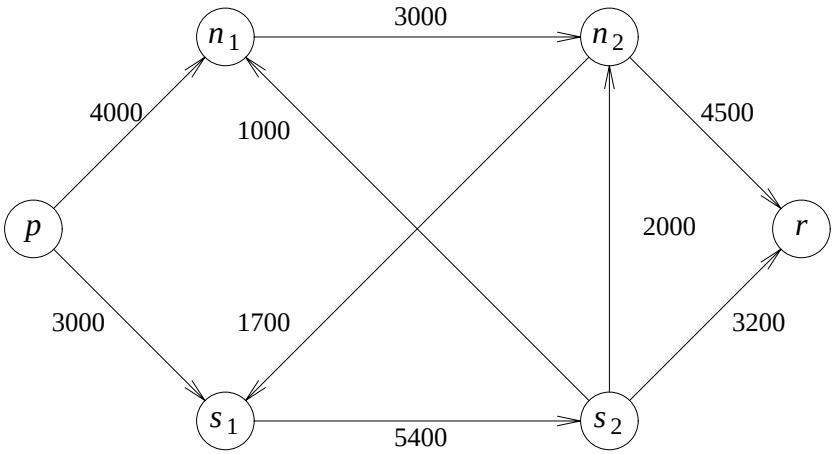


Figure 4.1.2. The pipeline network model with capacities shown.

assigned a nonnegative real number $c(e)$, called the *capacity* of the arc; that is, c is a function which assigns nonnegative real numbers to the arcs of N. Here, $in(v)$ $(out(v))$ denotes the set of all arcs entering (leaving) the vertex v.

A (legal) *flow* f in N is a mapping from the arc set E to the real numbers such that:

1. **(Capacity constraint)** $f(e) \leq c(e)$ for every $e \in E$

2. **(Flow conservation)** for each vertex v other than s and t,

$$0 = \sum_{e \in in(v)} f(e) - \sum_{e \in out(v)} f(e). \tag{4.1}$$

Clearly, a flow is a mapping that describes the movement (or flow) of material along the arcs of the network, while the capacity is a mapping that describes the maximum amount that can move along the arcs. Further, legal flows conform to the natural restrictions we determined earlier. That is, the flow on an arc never exceeds the capacity and flow conservation states that all that flows into a vertex (other than s or t) also flows out of that vertex.

Note that we allow s and t to be arbitrarily chosen within the digraph, and, thus, $in(s)$ and $out(t)$ need not be empty. We further note that loops never add anything to a network, because what flows out along the loop immediately flows back into the same vertex. Also, parallel arcs really add nothing to our study, since we can view the flows (and capacities) on these arcs as actually occurring on only one arc whose flow (and capacity) is the sum of the flows (and capacities) on all the parallel arcs. These restrictions do, however, allow us to be sure that $|E| \leq |V|$ $(|V| - 1)$.

Given a network N with flow f, the *total flow* F (sometimes called the value of the flow) is defined as:

$$F = \sum_{e \in in(t)} f(e) - \sum_{e \in out(t)} f(e). \tag{4.2}$$

That is, F is the net flow into the sink t. Our problem is to maximize F.

Ford and Fulkerson [6] were the first to study the maximum flow problem computationally. They established a number of results and developed an algorithm for finding the maximum flow. Strictly speaking, their original method was not an algorithm in that it was not always guaranteed to terminate. Edmonds and Karp [5] modified this method to ensure that the process would terminate. In fact, they showed that the modified algorithm had complexity $O(|V|\ |E|^2)$.

Many others have studied the maximum flow problem. Dinic [4] developed a $O(|V|^2|E|)$ algorithm and later Karzanov [7] developed a $O(|V|^3)$ improvement on Dinic's technique. In 1978, Malhorta, Kumar and Maheshwari [8] simplified this approach while maintaining the time complexity of $O(|V|^3)$. The fastest known algorithm is that of Sleator [9]. This algorithm has complexity $O(|V||E|\log(|V|))$; however, it requires some sophisticated data structure techniques and is beyond the scope of this text. In the remaining sections of this chapter, we will investigate flows in networks and the algorithms of Ford and Fulkerson, Dinic, and Malhorta, Kumar and Maheshwari.

Section 4.2 The Ford and Fulkerson Approach

The Ford and Fulkerson approach to finding the maximum total flow in a network N is based on the idea of improving the present flow along some "path" between the source s and sink t. Once this is done, we repeat the process on N with its modified flow. We continue to repeat the process until we cannot find a path whose flow can be improved. This technique has come to be known as the *augmenting path* technique.

Before we develop an algorithm for the augmenting path technique, it will be helpful to observe an interesting relationship between the maximum flow and the capacity of a set of arcs. Let S be a subset of V such that $s \in S$ and $t \in \overline{S} = V - S$. Recall that we denote by $out(S)$ the set of all arcs from vertices of S to vertices in $\overline{S}$ and, similarly, by $in(S)$ the set of all arcs from vertices in $\overline{S}$ to vertices in S. Simply put, $out(S)$ is the set of arcs that start anywhere in the set S, but end outside S, while $in(S)$ is the set of arcs that originate outside of S but end in S. Together these two sets constitute all arcs

between the vertex sets S and $\overline{S}$, and $out(S) \cup in(S)$ is called the *cut* determined by S. An interesting and important fact is that the total flow F can actually be measured at any cut in the network N.

Lemma 4.2.1 Given a network $N = (V, E, s, t, c)$ with flow f, then for every $S \subseteq V$ such that $s \in S$ and $t \in \overline{S}$,

$$F = \sum_{e \in out(S)} f(e) - \sum_{e \in in(S)} f(e). \tag{4.3}$$

Proof. In order to verify this equation, consider the sum of equation (4.2) and all equations on flow conservation (4.1) for vertices $v \in \overline{S} - \{ t \}$. The result of this sum has F on the left-hand side. To see what happens on the right-hand side, we must consider an arc from x to y. If x and y are both in S, then $f(e)$ never appears on the right-hand side of any equation in the sum. If x and y are both in $\overline{S}$, then $f(e)$ appears positively once and negatively once, and, hence, has no effect on the right-hand side. If $x \in S$ and $y \in \overline{S}$, then $f(e)$ appears positively on the right-hand side of the equation for y, and it appears in no other equation. If $x \in \overline{S}$ and $y \in S$, then $f(e)$ appears negatively on the right-hand side of the equation for x and in no other equation. Each of these cases agrees with equation (4.3) and, thus, the result is verified. □

For a set S of vertices, we call $c(S) = \sum_{e \in out(S)} c(e)$ the *capacity* of the cut determined by S. From what we have already seen about cuts and the total flow, one should expect that the capacity $c(S)$ is related to the total flow F. That this is indeed the case is our next result.

Proposition 4.2.1 Given a network N, for every flow f with total flow F and for every $S \subseteq V$,

$$F \leq c(S).$$

Proof. By Lemma 4.2.1 we know that

$$F = \sum_{e \in out(S)} f(e) - \sum_{e \in in(S)} f(e).$$

But by the capacity constraint, we know that $0 \leq f(e) \leq c(e)$ for each $e \in E$. Thus, by maximizing the positive term and minimizing the negative term we get that

$$F \leq \sum_{e \in out(S)} c(e) - 0 = c(S). \quad \square$$

A very important corollary of Proposition 4.2.1 is now easy to obtain.

Corollary 4.2.1 Given a network N with flow f and $S \subseteq V$ such that S contains s but not t, if $F = c(S)$, then F is a maximum and the cut determined by S is of minimum capacity.

Proof. Let F^* be a maximum flow and let K determine a cut of minimum capacity. Then, applying Proposition 4.2.1 and the natural relationships of these quantities, we see that

$$F \leq F^* \leq c(K) \leq c(S).$$

But since $F = c(S)$, we see that equality must hold throughout the above expression, and so F must be a maximum and S must have minimum capacity. □

We now wish to examine the Ford and Fulkerson process for finding the maximum flow. Their idea, again, was to use paths (not necessarily directed paths) from the source to the sink that were not being "optimally" used. The direction of the arcs on such a path would be used to help determine how to modify the present flow along the path in order to "push" more flow to the sink.

If the arc $e = x \rightarrow y$ is on such an $s - t$ path (we follow the direction on e from x to y) and we wish to use e to push more flow to t, then e presently must not be up to capacity. That is, $f(e) < c(e)$ must hold so that there is room for improvement on e. The amount of improvement is, of course, limited to $\Delta(e) = c(e) - f(e)$. The value $\Delta(e)$ is called the *slack* of e. If $f(e) = c(e)$, we say the arc e is *saturated.* If, instead, $e = y \rightarrow x$, then in order to increase the flow from s to t, we must cancel some of the flow into x on this arc (as the flow on e is away from t). Thus, there must already be some flow ($f(e) > 0$) on e if we are to accomplish our goal.

An *augmenting path* is a (not necessarily directed) path from s to t that can be used to increase the flow from s to t. As we attempt to find an augmenting path, we will label the vertices of N as we examine them. Initially, we label s, and then we label every vertex v for which we can find an augmenting $s - v$ path. When we finally label t, we have found an augmenting $s - t$ path. This path is then used to increase the total flow in N, and the process is begun again. If at some point, we cannot find any vertex to label, then no augmenting path exists and the present value of the flow is maximum.

In performing the labeling process, a *forward labeling* of vertex v using arc e (where e is directed from u to v) is done when u is labeled and v is not labeled and $c(e) > f(e)$. Here the label assigned to v is e^+. If the arc e is used for forward labeling, let

$\Delta(e) = c(e) - f(e)$. A *backward labeling* of vertex v using arc e (where $e = v \rightarrow u$) is done when u is labeled and v is not labeled and $f(e) > 0$. The label assigned to v is e^-, and in this case $\Delta(e) = f(e)$.

We now state the fundamental result about augmenting paths. The proof of Theorem 4.2.1 is left as an exercise.

Theorem 4.2.1 In a network N with flow f, the total flow F is maximum if, and only if, no augmenting path from s to t exists.

We now state the Ford and Fulkerson algorithm [6].

Algorithm 4.2.1 The Ford and Fulkerson Algorithm.

Input: A network $N = (V, E, s, t, c)$ and a flow f. (Initially, we usually choose $f(e) = 0$ for every arc e.)

Output: A modified flow f^* or the answer that the present total flow is maximum.

Method: Augmenting paths.

1. Label s with * and leave all other vertices unlabeled.
2. Find an augmenting path P from s to t.
3. If none exists,
   then halt, noting that the present flow is maximum;
   else compute and record the slack of each arc of P and compute the minimum such slack λ. Now, refine the flow f by adding λ to f for all forward arcs of P and subtracting λ from f for all backward arcs of P.
4. Set $f^* = f$ and repeat this process for N and the new flow f^*.

We are now capable of answering the question raised at the beginning of the chapter. We will use the Ford and Fulkerson algorithm for finding augmenting paths to determine a maximum flow in the oil pipeline model. We find the first augmenting path, say p, s_1, s_2, n_2, r, has minimum slack 2000. All arcs are forward, so we push 2000 units of additional flow along each arc. We picture this in Figure 4.2.1.

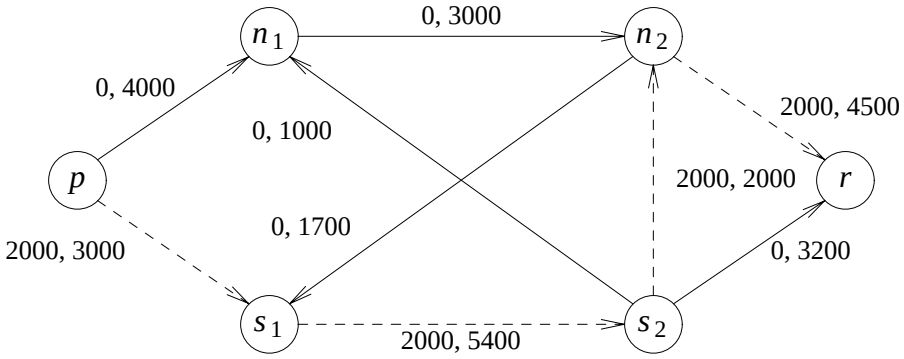


Figure 4.2.1. After the first augmenting path is found.

A second augmenting path is p, n_1, n_2, s_2, r, which includes a backward labeling of s_2. The minimum slack is again 2000. The resulting network and flow are shown in Figure 4.2.2.

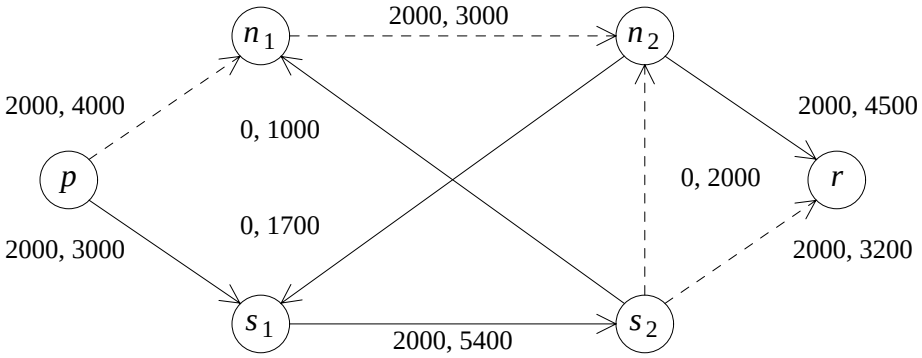


Figure 4.2.2. After the second augmenting path is found.

We continue with a third augmenting path p, n_1, n_2, r. This path has a minimum slack of 1000. We update the flows accordingly to obtain the network in Figure 4.2.3.

Finally, the augmenting path p, s_1, s_2, r, with minimum slack 1000, is found (Figure 4.2.4). At this stage the arc $p \rightarrow s_1$ is up to capacity, while $p \rightarrow n_1$ is not. However, the remaining arcs at n_1 are either up to capacity or directed into n_1 and without positive flow. Thus, no further augmenting paths can exist.

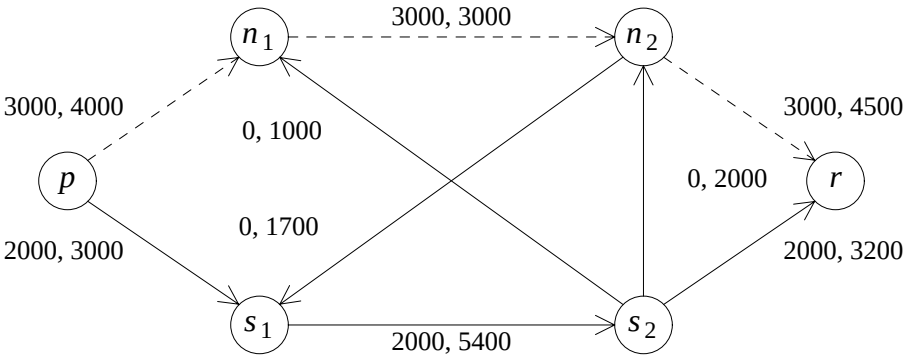


Figure 4.2.3. After the third augmentation.

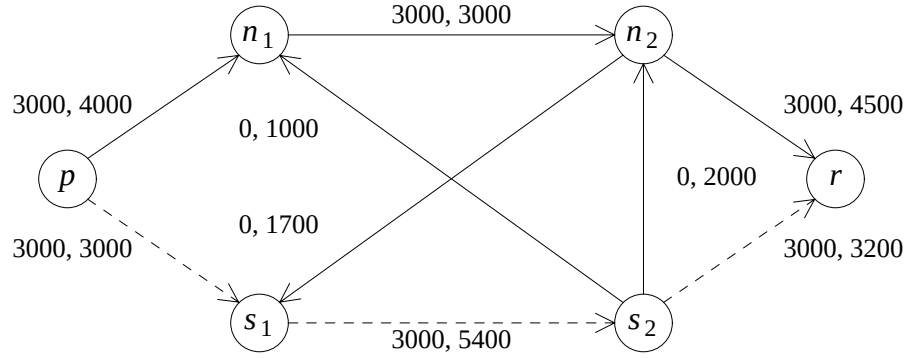


Figure 4.2.4. After the final augmentation, $F = 6000$.

We should make several points here. We found the maximum flow $F = 6000$, and the Ford and Fulkerson algorithm terminated. If both $f(e)$ and $c(e)$ are integers for each e, then since the algorithm only adds or subtracts the slack, it continues to produce only integer flows. Thus, once t is labeled and an augmenting path has been found, we will increase the value of the flow by at least 1. Since F is bounded above by $c(S)$ for any S, the process must terminate.

At first glance, you might argue that since all computer storage is limited, capacities and flows are effectively rational numbers anyway. Thus, the algorithm should terminate by an argument similar to that for integer flows and capacities. However, there is a fundamental problem with this kind of thinking. This problem is most easily seen in Figure 4.2.5, where the arc $x \to y$ has capacity 1 and all other arcs have capacity C. By alternating the augmenting paths from s, x, y, t to s, y, x, t, we need $2C$ augmentations

to find the maximum flow. Since we can make C as large as we like, the algorithm can still fail in practice.

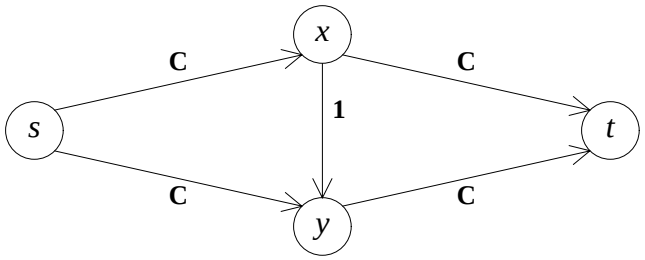


Figure 4.2.5. A network taking $2C$ augmentations.

Recall that we said that the Ford and Fulkerson algorithm might not terminate. Ford and Fulkerson showed that their procedure might take an infinite number of steps if all the capacities were irrational. Their example is somewhat complex, and we shall not examine it here. The important point in their example, as well as in the oil pipeline example and the last example, was that we did not do a very effective job of finding the augmenting paths. In fact, in each example we did more work than we should have. This is essentially what Ford and Fulkerson did in their example. They delayed as long as possible actually saturating arcs. Edmonds and Karp [5] were able to show that if one uses a breadth-first search in the labeling algorithm and always selects a shortest augmenting path, then the algorithm will terminate in at most $O(|V|^2|E|)$ steps (even if irrational capacities are allowed).

We now present an algorithm for finding the augmenting path. In this algorithm, consider the term *scan* to imply that a breadth-first search is being done.

Algorithm 4.2.2 Finding an Augmenting Path.
Input: A network N and a flow f.
Output: An augmenting path P or a message that none exists.
Method: A modified breadth-first search.

1. Label s with an *.
2. If all labeled vertices have been scanned,
   then halt, noting that no augmenting path exists; hence, the present flow is maximum,
   else find a labeled but unscanned vertex v and scan as follows: For each

$e = vu \in out(v)$, if $c(e) - f(e) > 0$ and u is unlabeled, label u with e^+. For each $e = uv \in in(v)$, if $f(e) > 0$ and u is unlabeled, then label u with e^-.

3. If t has been labeled,
   then starting with t, use the labels to backtrack to s along an augmenting path. The label at each vertex indicates its predecessor in the path. When you reach s output the path and halt; else repeat step 2.

Note that the labeling process also indicates the type of labeling that was done. Thus, the appropriate computation for the slack can be done as we trace back from t while finding the augmenting path.

We conclude this section with the most famous result concerning networks (Ford and Fulkerson [6]). As we shall see, it is closely related to many other results in graph theory, as well as results in such other areas as linear programming and combinatorics.

Theorem 4.2.2 (The Max Flow–Min Cut Theorem) In a network N, the maximum value of a flow equals the minimum capacity of a cut.

Proof. Let F be a maximum total flow and S be a cut of minimum capacity. It follows from Proposition 4.2.1 that $F \leq c(S)$. To show that the reverse inequality also holds, let flow f have maximum flow F. Then by Theorem 4.2.1, there can be no augmenting $s - t$ paths in N. Let C be the set of all vertices v such that there is an augmenting path from s to v. Clearly, t is not in C. By the definition of an augmenting path and the definition of C, we have that for all vertices $v \in C$ and for all vertices $w \in V - C$, the flow from v to w equals the capacity of the cut determined by C that separates v and w. Further, the flow from w to v is zero. If either the flow from v to w is less than the capacity of a cut that separates them or the flow from w to v is positive, we could find an augmenting path from s to w. Thus, we have found a cut, namely C, with the same capacity as the maximum flow, and by Corollary 4.2.1, this cut must be of minimum capacity. □

Section 4.3 The Dinic Algorithm and Layered Networks

Dinic's approach [4] to finding a maximum flow starts with some legal flow (possibly zero everywhere) and repeatedly attempts to improve it using augmenting paths. When no further improvement can be found, the total flow is maximum. The difference in Dinic's approach is the way in which the network is viewed and the restrictions this view

places on the way augmenting paths are formed.

The main idea behind this approach is the construction of a *layered network.* Layers are special subsets of the vertices. What is special about the layers is their structure and that they are determined by the present flow. Recall that in the Ford and Fulkerson approach, we concentrated on forward arcs $e = u \rightarrow v$ such that $f(e) < c(e)$ or on backward arcs $e = v \rightarrow u$ such that $f(e) > 0$. We again want to address such arcs, which we now designate as *useful arcs.* Denote the useful arcs between layers L_i and L_{i+1} by U_{i+1}. We build layers based on a breadth-first search using only useful arcs. Thus, as arcs become saturated, we have fewer and fewer useful arcs to use in relayering the network. Dinic's algorithm for construction of a layered network is as follows:

Algorithm 4.3.1 The Layering Algorithm.
Input: A network N and flow f.
Output: A sequence of layers of vertices $L_0, L_1, \ldots, L_d$ or the message that the present flow is maximum.
Method: A modified BFS using only useful arcs.

1. $L_0 \leftarrow \{s\}$ and $i \leftarrow 0$.
2. Set $T \leftarrow \{ v \mid v \notin \bigcup_{j=0}^{i} L_j$ and there is $e = u \rightarrow v$ or $v \rightarrow u$ such that e is useful $\}$.
3. If $T = \emptyset$, say that the present flow is maximum and halt.
4. If $t \in T$,
   then $k \leftarrow i + 1$ and $L_k \leftarrow \{t\}$ and halt with the layers $L_0, \ldots, L_k$;
   else $L_{i+1} \leftarrow T$, set $i \leftarrow i + 1$ and go to step 2.

In constructing the layered network, we examine each arc two times at most (once in each of two consecutive layers), hence, we note that the time complexity of the layering algorithm is $O(|E|)$.

In the sequence of layers $L_0, \ldots, L_d$, consecutive layers are joined only by useful arcs. We seek a *maximal flow* f^* in this layered network. By this we mean a flow f^* such that for every path $s = v_0, e_1, v_1, \ldots, e_d, v_d = t$, where $v_i \in L_i$ and $e_i \in U_{i+1}$, there is at least one saturated arc e. That is, every feasible augmenting path with vertices in consecutive layers has an arc whose flow is at capacity.

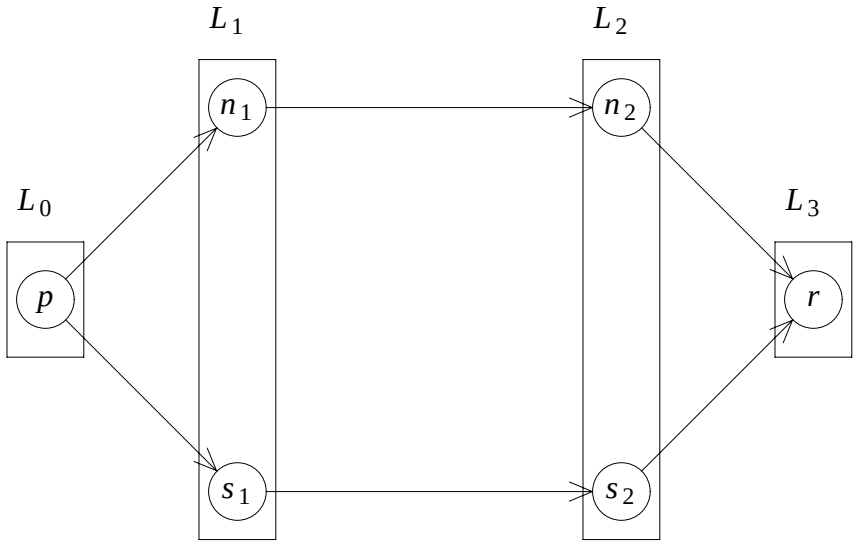


Figure 4.3.1. First layering of the network of Figure 4.1.2.

Typically, the process of finding a layered network, then finding a maximal flow on the layered network and improving the original flow, is called a *phase*. The important point about phases is that we can determine a bound on the number of phases we need to perform in order to find a maximum flow. This bound will allow us to compute a bound on the time complexity of Dinic's algorithm.

To do this, let the *length* of a layered network be the index of the final layer. Clearly, this is also a measure of the length of an augmenting path in the layered network. Denote the length of the layered network obtained in the jth phase by $len(j)$. Also, denote by $L_a(b)$, the ath layer in the bth phase.

Theorem 4.3.1 If phase $m + 1$ is not the final phase, then $len(m + 1) > len(m)$, and, hence, the number of phases is at most $|V| - 1$.

Proof. Suppose phase $m + 1$ is not the last phase of the algorithm. Then, in the $(m + 1)$-st layered network, there must be a path

$$P\colon\ s = v_0,\ e_1,\ v_1,\ e_2\ ,\ \ldots,\ e_{len(m+1)},\ v_{len(m+1)} = t.$$

We first assume that all the vertices of P also appear in the m-th layered network. We claim that if $v_j \in L_k(m)$, then $j \geq k$. That is, a vertex in layer k of phase m must come from layer j of phase $m + 1$ where $j \geq k$.

To prove this claim, we use induction on j. For $j = 0$ (recall $v_0 = s$), the claim is clearly true. Thus, assume the result is true for all vertices through j and assume $v_{j+1} \in L_i(m)$. If $i \leq j + 1$, the induction is trivial. But if $i > j + 1$, then the arc e_{j+1} has not been used in the *mth* phase since it is not even in the $m-th$ layered network, in which only arcs between adjacent layers appear. If e_{j+1} has not been used but is useful from v_j to v_{j+1} at the start of phase $m + 1$, then it was useful from v_j to v_{j+1} in the beginning of phase m. Thus, v_{j+1} cannot belong to $L_i\ (m)$ (by the algorithm). Now, in particular, $t = v_{len(m+1)}$ and $t \in L_{len(m)}(m)$. Hence, from our induction we conclude that $len(m + 1) \geq len(m)$. However, equality cannot hold, because in this case the entire path is in the *mth* layered network, and if all its arcs are still useful at the start of phase $m + 1$, then the flow in phase m was not maximal, and we have a contradiction.

We now assume that not all the vertices of P appear in the mth layered network. Then let $e_{j+1} = v_j v_{j+1}$ be the first arc such that for some b, $v_j \in L_b(m)$; but v_{j+1} is not in the *mth* layered network. Thus, e_{j+1} was not used in phase m. Since it is useful in the beginning of phase $m + 1$, it was also useful in the beginning of phase m. The only possible reason for v_{j+1} not to belong to $L_{b+1}(m)$ is that $b + 1 = len(m)$. By the previous paragraph, $j \geq b$. Thus, $j + 1 \geq len(m)$, and therefore,

$$len(m + 1) > len(m).$$

It is now clear that the number of phases is at most $|V| - 1$. □

We now present Dinic's algorithm. To do this, we make use of a simple data structure known as a stack. Recall that a stack is a last in-first out device. That is, whatever item is last placed on the stack is the first one to be removed from the stack (just like a stack of trays in a cafeteria). The act of placing X on the stack ST will be denoted ST <== X, while removing X from the top of the stack will be denoted X <== ST. These are the only two operations we allow on the stack.

Algorithm 4.3.2 Dinic's Maximal Flow Algorithm.

Input: A layered network N with $f(e) = 0$ and e marked unblocked for each arc e.

Output: A maximal flow on N.

1. Let $v \leftarrow s$ and empty the stack ST.
2. If all arcs to the next layer are blocked, then
   if $s = v$, then halt and note that the present flow is maximal;

else e <== ST (say $e = uv$), mark e as "blocked," $v \leftarrow u$, and repeat step 2.

3. Choose an unblocked arc $e = vu$ with u in the next layer, ST <== e and let $v \leftarrow u$. If v does not equal t, then go to step 2.

4. The edges on ST form an augmenting path P. Find the minimum slack Δ of an arc on P. For every arc e on P, set $f(e) \leftarrow f(e) + \Delta$ and if $f(e) = c(e)$, mark e as "blocked." Go to step 1.

It is clear that in Algorithm 4.3.2, an arc is marked as blocked only if no later augmenting path can use it. Thus, when this algorithm halts, the flow is maximal. The number of arcs examined between two declarations of arc blocking is at most $|V| - 1$. Since no arc is ever unblocked, the number of arcs examined is bounded by $|V|\,|E|$ Thus, the algorithm for finding a maximal flow in the layered network is bounded by $O(|V|\,|E|)$. Further, since the number of layerings is bounded (Theorem 4.3.1), the entire process is bounded by $O(|V|^2\,|E|)$.

Section 4.4 Layered Networks and Potential

In the last section, we developed an algorithm to block a layered network of length d of all augmenting paths of length d. We accomplished this by finding a flow which saturated at least one arc on every $s - t$ path. We call such a flow a *blocking flow* for the layered network. (Since Dinic developed the idea of a layered network, all new flow algorithms have concentrated on improved methods for finding blocking flows.)

Malhorta, Kumar, and Maheshwari [8] developed a simple algorithm to accomplish this in $O(|V|^3)$ time. We can use their algorithm in place of Dinic's Maximal Flow Algorithm to create a modified phase.

The Malhorta, Kumar and Maheshwari improvement centers on the idea of saturating arcs more rapidly. Given a layered network N with layers $L_0, L_1, \ldots, L_d$, define the *potential* of a vertex to be

$$p(v) = \min \{ \sum_{e \in out(v)} c(e), \sum_{e \in in(v)} c(e) \}.$$

Intuitively, the potential is the amount of flow that we can force through a vertex. We are then interested in a vertex of minimum potential so that we can saturate an arc (or hopefully more) at this vertex. Thus, if $p(w) = \min_{v \in V} p(v)$, we call w a *minimum vertex.*

Theorem 4.4.1 If a layered network N has minimal potential p^*, then N admits a flow with value p^*.

Proof. Let w be a vertex with minimum potential p^*. We begin by distributing a flow of p^* across the arcs from s to L_1, saturating them one by one. Since the flow into any vertex $v \in L_1$, is no greater than p^*, the flow into v can be distributed in the same manner among arcs from v to vertices in L_2. Repeating this process, we can push the flow all the way to $L_d = \{ t \}$. Clearly, at each layer, p^* enters and p^* leaves; thus the flow into t is p^*. □

The proof of Theorem 4.4.1 describes an algorithm for pushing a flow of $p(w) = p^*$ from s to t. Call this algorithm PUSH($p(w)$). Using this algorithm, we can now present the Malhorta, Kumar and Maheshwari algorithm for creating a blocking flow in the layered network.

Algorithm 4.4.1 The Potential Method Blocking Flow Algorithm.
Input: A layered network N with layers $L_0, L_1, \ldots, L_d$.
Output: A blocking flow or the message that the present flow is maximum.

1. Set all potentials to zero.
   While $\{s, t\} \cup V \neq \emptyset$, do the following:
2. Find a vertex w of minimum potential.
3. PUSH($p(w)$).
4. Update the arc capacities and potentials.
5. Delete saturated arcs and vertices without paths to both s and t.
6. endwhile

Section 4.5 Variations on Networks

In this section we consider several variations on the standard flow maximization problem. In the first variation, we wish to restrict the capacities that are possible. One goal is to see if this restriction does anything to the complexities of the algorithms we have studied. Our restriction will be fairly strong; for every arc e, let $c(e) = 1$. We call such networks $0 - 1$ *networks.* (Note that in applications of networks to other types of

graph problems or to other areas of discrete mathematics, this restriction is a useful and natural one.)

Suppose that N is a $0-1$ network with flow f. Recall from our earlier discussion of integer flows on networks with integer capacities that the flows remain integer in value. Thus, when we apply any algorithm to modify the present flow, arcs with flow zero will have their flows increase to 1 and, hence, become saturated. For example, in Dinic's algorithm, each time we find an augmenting path, all arcs on this path become blocked. In case the last arc leads to a dead end, we backtrack on this arc and it becomes blocked (that is, if the last arc does not lead to t, backtrack and consider this arc blocked). Thus, the total number of arc traversals is bounded by $|E|$, and, hence, an entire phase has time complexity $O(|E|)$. Since we know that the number of phases is at most $|V|-1$, the time complexity of Dinic's algorithm for $0-1$ networks can be reduced to $O(|V|\,|E|)$. Our goal now is to show that an even stronger improvement is possible.

Let N be a $0-1$ network with an integer flow function f. We define a new network NZ from N and f as follows: Let $V(NZ)=V(N)$ and if $e=uv\in E(N)$ and $f(e)=0$, then $e\in E(NZ)$, while if $e=vu\in E(N)$ and $f(e)=1$, then $e'=uv\in E(NZ)$. Thus, it is clear that $|E(N)|\geq|E(NZ)|$ and the useful arcs of N (in their direction of usefulness) are all arcs of NZ. Let M_N denote the maximal flow in the network N, F_N denote the present flow in N and $c_N(S)$ denote the capacity of S in the network N.

Lemma 4.5.1 Given a $0-1$ network N with flow f and the corresponding network NZ, then $M_{NZ}=M_N-F_N$.

Proof. Let $S\subseteq V$ with $s\in S$ and $t\in\bar{S}$. From the definition of NZ we have that

$$c_{NZ}(S)=|out_{NZ}(S)|=\sum_{e\in out_N(S)}(1-f(e))+\sum_{e\in out_N(\bar{S})}f(e).$$

However,

$$F_N=\sum_{e\in out_N(S)}f(e)-\sum_{e\in out_N(\bar{S})}f(e).$$

Thus,

$$c_{NZ}(S)=|out_N(S)|-F_N=c_N(S)-F_N.$$

This implies that a minimum cut of N corresponds to a minimum cut of NZ, that is, it is defined by the same vertex set S. By the max flow–min cut theorem, the capacity of a minimum cut of N is M_N and the capacity of a minimum cut of NZ is M_{NZ}. Thus, the lemma is proved. □

Lemma 4.5.2 The length of the layered network for the $0-1$ network defined by N and zero flow everywhere is at most $\dfrac{|E|}{M_N}$.

Proof. Since $f(e) = 0$ for every $e \in E$, the useful arcs are all forward. Thus, every U_i is equal to $out_N(S)$ where $S = \bigcup_{j=0}^{i-1} L_j$. Thus, by Lemma 4.5.1, we see that $M_N \leq |U_i|$. Summing this last inequality for every $i = 1, 2, \ldots, len(N)$ we get that $(len(N))\ M_N \leq |E|$, and the result follows. □

Theorem 4.5.1 For a $0 - 1$ network N, Dinic's algorithm is of time complexity $O(|E|^{3/2})$.

Proof. If $M_N \leq |E|^{1/2}$, then the number of phases is bounded by $|E|^{1/2}$, and the result follows easily. Otherwise, consider the phase during which the total flow reaches $M_N - |E|^{1/2}$. The total flow F in N, when the layered network for this phase is constructed, satisfies $F < M - |E|^{1/2}$. This layered network is identical with one constructed for NZ with zero flow everywhere. Thus, by Lemma 4.5.1, $M_{NZ} = M_N - F > |E|^{1/2}$. By Lemma 4.5.2, the length (*len*) of this layered network satisfies

$$len \leq \frac{|E|}{M_{NZ}} < \frac{|E|}{|E|^{1/2}} = |E|^{1/2}.$$

Thus, the number of phases up to this point is less than $|E|^{1/2}$, and the number of additional phases is at most $|E|^{1/2}$; thus the total number of phases is at most $2|E|^{1/2}$, and the result follows. □

Another variation of our original question is a simple generalization of the idea of a network. In all our previous problems, the flow on any arc e was at least zero. Now we wish to vary this lower bound and assume another function b assigns lower bounds on the flow that is legal on any arc. That is, suppose that $b(e) \leq f(e) \leq c(e)$ is our restriction on legal flows, where the function b assigns a real number to any arc e. We define the *generalized network GN* to be the 6-tuple $GN = (V, E, s, t, b, c)$.

In altering networks in this manner, we are faced with a new concern. Previously, it was easy to find an initial legal flow by simply taking the zero flow on each arc. Now, it is not even clear that a legal flow exists. In fact, it is easy to produce networks that have no legal flows. For example, the generalized network below with arcs labeled with the bounding pairs b, c has no legal flows.

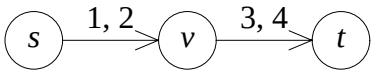


Figure 4.5.1. A generalized network with no legal flow.

Thus, the maximum flow problem for generalized networks calls for a solution in two phases. First, we must determine if the given network admits any legal flows, and if so, we must find one. Secondly, given a legal flow, we want to increase this flow until a maximum total flow is found.

Ford and Fulkerson developed a method for determining if a generalized network has a legal flow and for finding a legal flow. This method begins with a very natural idea: try to modify the network in some way that allows us to apply the techniques we have already developed. This modified network is called an *auxiliary network.* We now present Ford and Fulkerson's algorithm for constructing an auxiliary network.

Algorithm 4.5.1 Auxiliary Network Construction.
Input: A network $N = (V, E, s, t, b, c)$.
Output: An auxiliary network $\overline{N} = (\overline{V}, \overline{E}, \overline{s}, \overline{t}, \overline{b}, \overline{c})$.

1. Modify N as follows: Let $\overline{V} = \{ \overline{s}, \overline{t} \} \cup V$. The vertices $\overline{s}$ and $\overline{t}$ are called the auxiliary source and sink, respectively.
2. For every $v \in V$, construct an arc $\overline{e} = v\overline{t}$ with capacity $\overline{c}(\overline{e}) = \sum_{e \in out(v)} b(e)$. Let $\overline{b}(\overline{e}) = 0$.
3. For every $v \in V$, construct the arc $\overline{e} = \overline{s}v$ with $\overline{c}(\overline{e}) = \sum_{e \in in(v)} b(e)$. Now let $\overline{b}(\overline{e}) = 0$.
4. Modify the bounds on all arcs $e \in E$ by letting $\overline{b}(e) = 0$ and $\overline{c}(e) = c(e) - b(e)$.
5. Construct new arcs $e_1 = st$ and $e_2 = ts$ and let $\overline{c}(e_1) = \overline{c}(e_2) = \infty$ and $\overline{b}(e_1) = \overline{b}(e_2) = 0$.

Note that in the resulting auxiliary network, we regard s and t as normal vertices; hence, they must also satisfy the flow conservation constraint.

We have modified N to obtain the auxiliary network $\overline{N}$, but what have we gained?

Since each arc e in the original generalized network contributes $b(e)$ to an arc of $out(\overline{s})$ and an arc of $in(\overline{t})$, if all of $out(\overline{s})$ is saturated, then all of $in(\overline{t})$ must also be saturated. The answer to what we have gained is now provided by the following result from Ford and Fulkerson [6].

Theorem 4.5.2 The generalized network GN has a legal flow if, and only if, the maximum flow of $\overline{N}$ saturates all arcs of $out(\overline{s})$.

Proof. Suppose that a maximum flow function $\overline{f}$ of $\overline{N}$ saturates all arcs of $out(\overline{s})$. Define the following flow function f for N: For every $e \in E$, let $f(e) = \overline{f}(e) + b(e)$. Since

$$0 \le \overline{f}(e) \le \overline{c}(e) = c(e) - b(e),$$

it is clear that $b(e) \le f(e) \le c(e)$.

Now, consider $v \in V - \{ s, t \}$. Let $e_1 = \overline{s} \to v$ and $e_2 = v \to \overline{t}$ be arcs of N. Then

$$\sum_{e \in in(v)} \overline{f}(e) + \overline{f}(e_1) = \sum_{e \in out(v)} \overline{f}(e) + \overline{f}(e_2).$$

By hypothesis,

$$\overline{f}(e_1) = \overline{c}(e_1) = \sum_{e \in out(v)} b(e) \text{ and } \overline{f}(e_2) = \overline{c}(e_2) = \sum_{e \in in(v)} b(e).$$

Thus,

$$\sum_{e \in in(v)} f(e) = \sum_{e \in out(v)} f(e);$$

hence, f is a legal flow on N.

To prove the reverse implication, we assume N has a legal flow f and we define the flow $\overline{f}$ as $\overline{f}(e) = f(e) - b(e)$ for each $e \in E$. Since $b(e) \le f(e) \le c(e)$, it is clear that

$$0 \le \overline{f}(e) \le c(e) - b(e) = \overline{c}(e).$$

Since f satisfies the conservation constraint, if we let $e_1 = \overline{s} \to v$ and $e_2 = v \to \overline{t}$ and let $\overline{f}(e_1) = \overline{c}(e_1)$ and $\overline{f}(e_2) = \overline{c}(e_2)$, then

$$\sum \overline{f}(e) + \overline{f}(e_1) = \sum \overline{f}(e) + \overline{f}(e_2),$$

and so $\overline{f}$ also satisfies flow conservation when all arcs of $out(\overline{s})$ are saturated.

Finally, the net flow out of *s* equals the net flow into *t*; hence, they will satisfy the conservation constraint if we push this flow on the arcs *st* and *ts* constructed in step 5 of the algorithm. □

We now have an approach to solving our two problems. Given the network $N = (V, E, s, t, b, c)$, we construct the auxiliary network $\overline{N}$. Then we apply one of our algorithms for finding a maximum flow in $\overline{N}$. If $out(\overline{s})$ is saturated, we use the techniques of the proof of Theorem 4.5.2 to construct an initial legal flow in *N*.

Next, we must maximize the flow in *N*. With a few minor modifications, the Ford and Fulkerson algorithm can be adjusted to work on *N*. The necessary adjustments center on the backward labelings. A backward labeling is done for the arc $e = v \rightarrow u$ if *u* is labeled and *v* is not labeled and $f(e) > b(e)$. Otherwise, the algorithm remains intact.

All the results about the Ford and Fulkerson algorithm can be shown to hold under these conditions, provided we also modify the definition of cut capacity to be: $c(S) = \sum c(e) - \sum b(e)$.

Section 4.6 Connectivity and Networks

In this section we examine the use of network techniques in obtaining results about connectivity of graphs and digraphs. In particular, we begin with the relationship between Menger's theorem and the max flow–min cut theorem. One should immediately notice a very similar style in the statements of these two results. Menger's theorem relates the maximum number of vertex disjoint paths and the minimum number of vertices in a separating set, while the Ford and Fulkerson result relates the maximum flow and the minimum capacity of a cut. Thus, both results involve the equality of two quantities, one of which is a maximum and the other a minimum. Dantzig and Fulkerson [2] showed that the max flow-min cut theorem can be used to prove Menger's theorem. We now present this proof.

Theorem 4.6.1 (Menger's Theorem) For distinct nonadjacent vertices *u* and *w* in a graph *G*, the maximum number of pairwise internally disjoint $u - w$ paths equals the minimum number of vertices in a $u - w$ separating set.

Proof. It is clear that the minimum number of vertices that separate *u* and *w* is at least as large as the maximum number of vertex disjoint $u - w$ paths, since every path from *u* to

w uses at least one vertex of the separating set.

To prove the converse, we construct a digraph D from the graph G as follows. For every $v \in V(G)$, we create two vertices v_1 and v_2 along with the arc $e^v = v_1v_2$, called an *internal arc.* For every edge $e = xy$ in G, insert two directed arcs $e' = x_2y_1$ and $e'' = y_2x_1$ in D, called *external arcs* (see Figure 4.6.1). We now define a network on D with source u_2 and sink w_1. Assign all arcs of the form $v_1 \to v_2$ a capacity of 1. Assign all other arcs an infinite capacity.

We now claim that the maximum number of disjoint $u - w$ paths in G equals the maximum flow F in the network. To see that this is indeed true, assume that there are n vertex disjoint $u - w$ paths in G. Note that each such path $u, e_1, a, e_2, b, \ldots, e_k, w$ determines a directed path in the network, namely

$$u_2, e'_1, a_1, e^a, a_2, e'_2, b_1, \ldots, w_1.$$

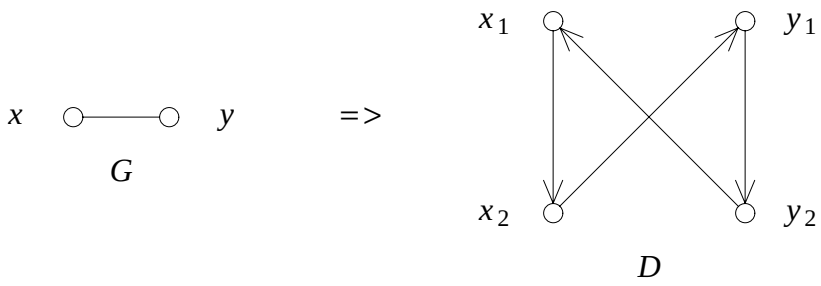


Figure 4.6.1. Construction of D.

Now let f be a flow that achieves the maximum total flow F. We can assume the flow on any arc is either 0 or 1 as the arcs of infinite capacity either enter a vertex with only one outgoing arc whose capacity is 1 or leave a vertex with only one incoming arc whose capacity is 1. Further, since the flow will stay integral, the claim is verified.

The last observation really tells us that the flow along any path can be at most 1. The flow into the sink can be decomposed into paths that each supply the sink with 1 unit of flow. Further, these paths are vertex disjoint, since the flow through any vertex is at most 1. Thus, there are F vertex disjoint paths from u_2 to w_1, and each determines a $u - w$ path in G. Hence, F is less than or equal to the maximum number of disjoint $u - w$ paths, but then we have that F equals the number of such paths.

By the max flow–min cut theorem, F equals $c(S)$ for some cut S of the network, where $u_2 \in S$ and $w_1 \in \overline{S}$. Since $c(S) = \sum_{e \in out(S)} c(e)$, the cut at S consists only of internal arcs. Every directed path from u_2 to w_1 in the network uses at least one arc of

the cut determined by S; hence, every $u - w$ path in G uses at least one vertex v such that e^v goes to $\overline{S}$. Therefore, the set

$$R = \{ v \mid v \in V \text{ and } e^v \in cut(S) \}$$

is a $u - w$ separating set. Clearly, $|R| = c(S)$. Thus, we have a $u - w$ separating set whose cardinality is F, and, thus, the minimum number of vertices in a $u - w$ separating set is at most F. This completes the proof. □

It should be clear from the proof that we can also use Dinic's algorithm for finding the connectivity of G.

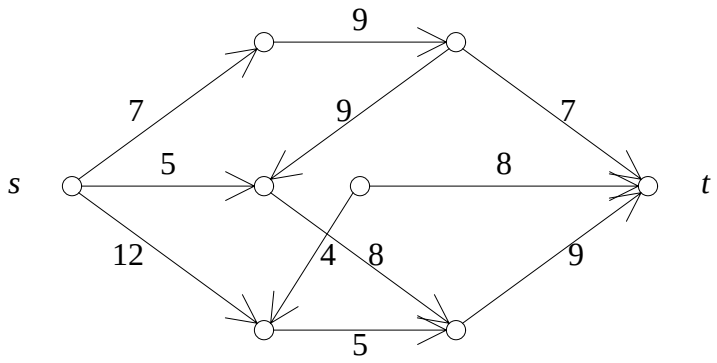
We conclude this section with a brief sketch of the proof of the edge version of Menger's theorem. The details are left to the exercises.

Theorem 4.6.2 In a graph G, the maximum number of edge disjoint $u - v$ paths equals the minimum number of edges in a $u - v$ separating set.

Proof (Sketch). Again, the idea is to form a digraph, say D from G. However, this time there is no need to split vertices, so let $V(D) = V(G)$ and let $E(D)$ contain both $e_1 = u \rightarrow v$ and $e_2 = v \rightarrow u$ whenever $e = uv \in E(G)$. Now, proceed in a fashion similar to that in the proof of Menger's theorem. □

Exercises

1. Find the maximum flow in the network below, using each of the algorithms presented in this chapter.

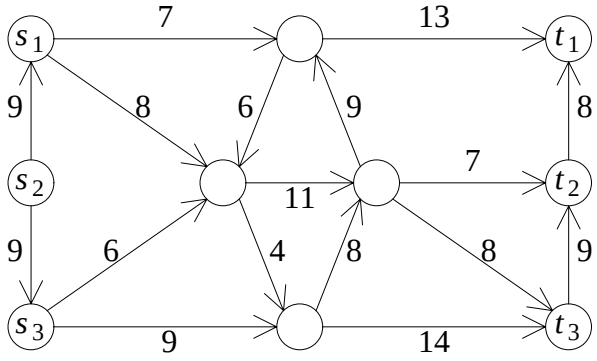


2. Use Algorithm 4.3.2 to find the maximum flow in the network of Figure 4.1.2.

3. Use Algorithm 4.4.1 to find the maximum flow in the network of Figure 4.1.2.

4. Let u and v be two vertices of a digraph $D = (V, E)$ and let $E_1 \subseteq E$ such that every $u - v$ path in D contains at least one arc of E_1. Prove that there exists a set of arcs of the form $(W, \overline{W})$ (where $W \subseteq V$) such that $u \in W, v \in \overline{W}$ and $(W, \overline{W}) \subseteq E_1$.

5. Suppose the following network has multiple sources s_1, s_2 and s_3 and they have available supplies 6, 12, and 7, respectively. Further suppose that t_1, t_2 and t_3 are all sinks with the demands 7, 12 and 6, respectively. Determine whether all these demands can be met simultaneously.

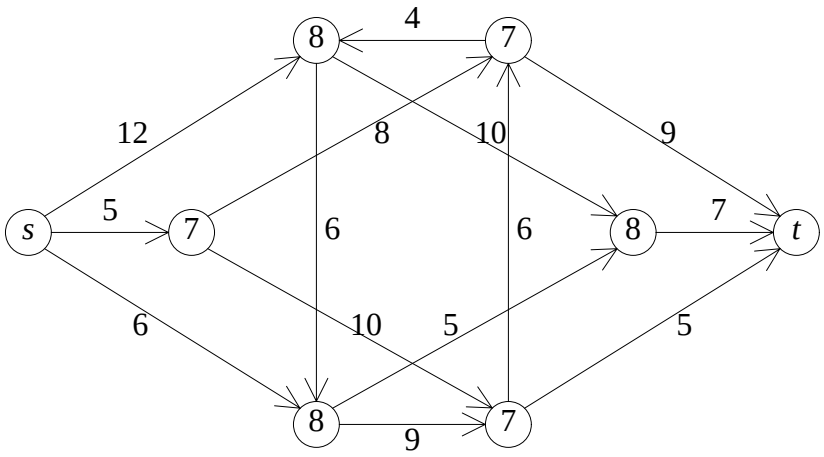


6. Let N be a network with underlying digraph D. Prove that if D contains no $s - t$ path, then the value of a maximum flow in N and the value of a minimum cut in N are both equal to zero.

7. Prove or disprove (each direction): The flows f_1 and f_2 in the network N agree on $(W, \overline{W})$ and $(\overline{W}, W)$ if, and only if, f_1 and f_2 are maximum flows in N.

8. Use networks to prove that $k(\ K_{m,n}\ ) \ = \ \min\{\ m,\ n\ \}$.

9. In the following network, in addition to the usual constraints, we have the additional constraint that each vertex (other than *s* and *t*) has an upper bound on the capacity that may flow through it. This capacity is indicated by the value inside the vertex. Find the maximum flow for this network.



10. Prove that in a network with a nonnegative lower bound on each arc, but no upper bound, there exists a legal flow if, and only if, for every arc *e*, either *e* is in a directed circuit or *e* is in a directed path from *s* to *t* (or *t* to *s*).

11. Prove that a network with both lower and upper bounds on the flow in the arcs has no legal flow if, and only if, there exists a set of vertices which includes neither the source nor the sink and is required to produce flow or absorb it.

12. A *path cover* of the vertices of a digraph $D = (V,\ E)$ (or graph) is a set of paths with the property that each vertex of D lies on exactly one path (note that a path may be trivial). Use the following network to develop an algorithm for determining the minimum number of paths necessary to cover an acyclic digraph $D = (V,\ E)$: Let $N = (V_1,\ E_1)$ where

$$V_1 = \{\ s,\ t\ \} \cup \{\ u_1, \ldots,\ u_{|V|}\ \} \cup \{\ v_1\ , \ldots,\ v_{|V|}\ \}$$

$$E_1 = \{\, s \to u_i \mid 1 \le i \le |V| \,\} \cup \{\, v_i \to t \mid 1 \le i \le |V| \,\} \cup \{\, u_i \to v_j \mid x_i \to x_j \in E(D) \,\}.$$

Also, set the capacities of each arc to 1. (Hint: Show that the minimum number of paths which cover V equals $|V|$, the max flow in the network.)

13. In the previous problem, was the assumption that D was acyclic necessary?

14. Suppose we ease the path cover restriction in the sense that the paths are no longer required to be vertex disjoint. Further, into the network constructed in exercise 11, insert the additional edges $\{\, u_i \to v_i \mid 1 \le i \le |V| \,\}$. Now, let the lower bound on each of these new arcs be 1 and on all other arcs be 0 and let all upper bounds be ∞. Describe an algorithm for finding the minimum number of paths covering the vertices of the original digraph D.

15. (Dilworth's theorem [3]) Two vertices are called *concurrent* if no directed path exists between them. A set of vertices is *concurrent* if its members are pairwise concurrent. Prove that the minimum number of paths which cover the vertices of D equals the maximum number of concurrent vertices. (Hint: See the previous exercise.)

16. Verify that the modified Ford and Fulkerson algorithm for networks with upper and lower bounds actually gives the desired results.

17. Complete the proof of the edge version of Menger's theorem (Theorem 4.6.2).

18. If $paths(u, v)$ is the maximum number of pairwise disjoint $u - v$ paths in a graph G, show that if G is not complete, then

$$\min_{u,v \in V(G)} paths(u, v) = \min_{uv \notin E(G)} paths(u, v);$$

that is, the minimum value of $paths(u, v)$ occurs for some nonadjacent pair of vertices u, v.

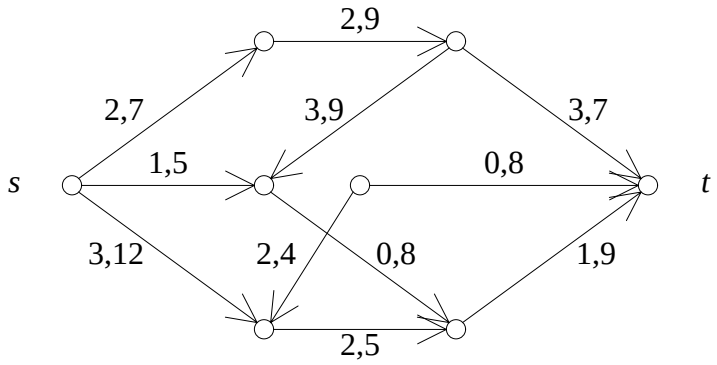
19. Show that if the connectivity of a graph G is k, then

$$k = \min_{u,v \in V(G)} paths(u, v).$$

20. Let N be a $0 - 1$ network with no multiple arcs and let M be the maximum total flow from s to t in N. Show that the length of the first layered network (with zero flow everywhere) is at most $\dfrac{2|V|}{M^{1/2}} + 1$.

21. Show that for the network N of the previous problem, Dinic's algorithm has time complexity $O(|V^{2/3}|\,|E|)$.

22. Determine if the network below has a legal flow. If it does not, modify as few capacities as possible to obtain a network that does have a legal flow.



References

1. Chvátal, V., *Linear Programming.* W. H. Freeman Company, New York (1980).
2. Dantzig, G. B. and Fulkerson, D. R., On the Max-Flow Min-Cut Theorem of Networks, Linear Inequalities and Related Systems, *Annals of Math.*, Study 38, Princeton University Press, (1956), 215 – 221.
3. Dilworth, R. P., A Decomposition Theorem for Partially Ordered Sets. *Ann. Math.*, Vol. 51 (1950), 161 – 166.
4. Dinic, E. A., Algorithm for Solution of a Problem of Maximum Flow in a Network with Power Estimation, *Soviet Math. Dokl.*, 11(1970), 1277 – 1280.
5. Edmonds, J., and Karp, R. M., Theoretical Improvements in Algorithm Efficiency for Network Flow Problems, *J. ACM,* 19(1972), 248 – 264.
6. Ford, L. R., Jr., and Fulkerson, D. R., *Flows in Networks*. Princeton University Press, (1962).
7. Karzanov, A. V., Determining the Maximal Flow in a Network by the Method of Preflows, *Soviet Math. Dokl.*, 15(1974), 434 – 437.
8. Malhorta, V. M., Kumar, M. P., and Maheshwari, S. N., An $O(|V|^3)$ Algorithm for Finding Maximum Flows in Networks, Computer Science Program, Indian Institute of Technology, Kanpur 208016, India (1978).

9. Sleator, D. D., An *O*(nm log *n*) Algorithm for Maximum Network Flow, Stanford University, Stanford, CA, Computer Science Department report STAN-CS-80-831 (1980).