The Max-Flow–Min-Cut Theorem and Its Theory

Math 581, Spring 1999

Notes by Thomas Zaslavsky

Final Version, 11 May 1999

I. Networks, flows, and cuts.

A *digraph*, denoted by D, is a graph in which each edge is directed from one endpoint, its *tail*, to another, its *head*. We may write $e:\overrightarrow{vw}$ to mean that e is an edge with tail v and head w. The underlying undirected graph is denoted by $G(D)$. We write $E(D)$ for the set of directed edges; $E(G(D))$ denotes the set of underlying undirected edges. (Some authors like to call a directed edge an *arc*.)

A *network* $N = (D, c, s, t)$ consists of four things: an underlying digraph D, a function $c : E(D) \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ called the *capacity function*, and two special vertices, s, the *source* of the network, and t, its *sink*. A *terminus* is either a source or a sink. (One can generalize to multiple sources or sinks; this is a problem in Section V.) We may write $D(N)$ for D, E or $E(N)$ for $E(D)$, c_N for c, and so on.

The things we're interested in are (s,t)-flows in N. These are a certain kind of function $f : E \to \mathbb{R}$. We call f *conservative at vertex* v if it satisfies the *conservation law*

$$\sum_{e:\overrightarrow{uv}\in E} f(e) = \sum_{e:\overrightarrow{vw}\in E} f(e).$$

The first sum in this expression is the *inflow* to v, sometimes written $f_{\text{in}}(v)$, and the second sum is the *outflow* from v, written $f_{\text{out}}(v)$. Thus the equation expresses the fact that the inflow at v equals the outflow. An (s,t)*-flow* is a function $f : E \to \mathbb{R}$ that is conservative at every vertex $v \neq s, t$. The set of all (s,t)-flows forms a vector space with ordinary functional addition and scalar multiplication. This fact can be useful.

A flow such that

$$0 \leq f(e) \leq c(e)$$

for all edges e is called *feasible*. The flows we are interested in are the feasible (s,t)-flows. The *value* of an (s,t)-flow f is

$$\operatorname{val}(f) := \sum_{e:\overrightarrow{sw}} f(e) - \sum_{e:\overrightarrow{vs}} f(e).$$

The object is to find the maximum value of a feasible (s,t)-flow in the network. We call this the *value of the network*.

For $B \subseteq E$ we define the *capacity of* B,

$$c(B) := \sum_{e\in B} c(e),$$

and similarly the *amount of flow through* B,

$$f(B) := \sum_{e\in B} f(e).$$

(Sometimes we abbreviate this expression to just "the flow through B".) In particular this applies to the set

$$A(X,Y) := \{e : \overrightarrow{vw} : v \in X \text{ and } w \in Y\},$$

defined when $X, Y \subseteq V$. (N.B. This should have been $E(X,Y)$ since I use E for the edge set. The A comes from the fact that many people call directed edges "arcs" and use A for the edge set of a digraph.)

If $X \subseteq V$ is such that $s \in X$ and $t \in Y = V \setminus X$, then $A(X,Y)$ is called an (s,t)*-cut* (briefly, a *cut*, since that's the only kind we normally use). The *flow from* X *to* Y is the number, dependent on f,

$$f(X,Y) := f(A(X,Y)) - f(A(Y,X)) = \sum_{e\in A(X,Y)} f(e) - \sum_{e\in A(Y,X)} f(e).$$

This is the net amount of flow from the s side to the t side of the cut, since it is the forward flow $f(A(X,Y))$ less the reverse flow $f(A(Y,X))$. In particular, using this notation we can say that

$$\mathrm{val}(f) := f(\{s\}, V \setminus \{s\}).$$

Now, the main theorem.

The Max-Flow–Min-Cut (MFMC) Theorem. *In a network N, the maximum value of a feasible (s,t)-flow equals the minimum capacity of an (s,t)-cut.*

A *maximum* (s,t)*-flow* in N is a feasible (s,t)-flow whose value is the maximum value of any (s,t)-flow in N. (We understand the maximum to be ∞ in case there is no (s,t)-cut with finite capacity, although of course this should technically be a supremum rather than a maximum.)

II. Proof of the Max-Flow–Min-Cut Theorem.

A. The easy part of the proof.

Lemma 1. *For any feasible (s,t)-flow f and any (s,t)-cut $A(X,Y)$ in a network N,*

$$\mathrm{val}(f) \le c(A(X,Y)).$$

I leave the proof to you.

B. Trails, paths, and cycles.

A *trail*, more precisely known as an *undirected trail*, in a digraph D (or in a network N) is a trail in $G(D)$. That is, it is a sequence $T = v_0 e_1 v_1 e_2 \cdots e_{l-1} v_l$ in $G(D)$ in which no edge is repeated. In T, edge e_i is called *forward* if it is directed $\overrightarrow{v_{i-1}v_i}$ and *backward* (or *reverse*) if it is directed $\overrightarrow{v_i v_{i-1}}$. Note that this definition depends not just on the edge but on the trail. A *directed trail* is a trail in which every edge is forward. We will usually be concerned with paths rather than general trails, but the definitions apply in general.

A *path*, or *undirected path*, in D is a trail which has no repeated vertices. A *circuit* is a trail which has no repeated vertices except that $v_l = v_0$. A *directed path*, also called a *dipath*, is a path in which every edge is a forward edge. A *cycle* is a circuit in which every edge is a forward edge.

We define δ_T, the *unit flow along a trail* T (which may be a directed or undirected trail), by

$$\delta_T(e) := 1 \qquad \text{if } e \text{ is a forward edge,}$$

$$\delta_T(e) := -1 \qquad \text{if } e \text{ is a backward edge, and}$$

$$\delta_T(e) := 0 \qquad \text{if } e \text{ is not in } T.$$

(By the way, δ_T is an (s,t)-flow if and only if the endpoints of T are s and t. It need not be feasible since it can be negative on an edge.) This definition applies to paths and cycles, in particular. Note that we can multiply δ_T by a scalar (that is, a real number) and get a new flow along T.

An (s,t)*-trail* in a network N is a trail in the underlying digraph $D(N)$ whose first vertex is s and whose last vertex is t. An (s,t)*-path* is a path whose endpoints are $v_0 = s$ and $v_l = t$. Thus we have, for instance, both directed and undirected (s,t)-paths.

C. Augmenting paths.

Suppose a feasible (s,t)-flow f is given. An edge e is called *saturated* if $f(e) = c(e)$. We say e is *positive* if $f(e) > 0$. Now let's look at any trail T. (For the moment, T may start and end anywhere; later we'll want it to start at s and sometimes to end at t.) We define the *slack of* e *with respect to* T to be

$$\epsilon(e) := c(e) - f(e) \text{ if } e \text{ is a forward edge and}$$

$$\epsilon(e) := f(e) \text{ if } e \text{ is a backward edge.}$$

(Thus the edges of T with positive slack are the unsaturated forward and positive reverse edges.) The *slack of* T is

$$\epsilon(T) := \min_{e \in T} \epsilon(e).$$

An f*-augmenting trail* T is an undirected (s,t)-trail of N which has positive slack; that is, in which each forward edge is unsaturated and each backward edge has positive flow quantity.

That defines the necessary terminology for an f-augmenting trail. (I defined it for trails because it's easy to do so, although what we really need is only f-augmenting paths.) Now here is what we do with it:

Sublemma 2.1. *$\epsilon(T)\delta_T$ is an (s,t)-flow (not necessarily feasible) with value $\epsilon(T)$.*

Sublemma 2.2. *$f+\epsilon(T)\delta_T$ is a feasible (s,t)-flow with value* $\mathrm{val}(f)+\epsilon(T)$.

The process of increasing the value of f by adding $\epsilon(T)\delta_T$ to f is called *augmenting f along T*; that's why we call T an augmenting trail.

Lemma 2. *If f has an augmenting path, then its value is not maximal.*

This is obvious. Not so obvious is the converse; indeed, its proof is the main part of the proof of the Max-Flow–Min-Cut Theorem and it requires quite a bit of machinery.

D. The Labelling Algorithm.

This is a means for finding an augmenting path or a saturated (s,t)-cut in a network with a given feasible (s,t)-flow f.

A *saturated (s,t)-cut* is an (s,t)-cut (X,Y) in which every forward edge is saturated (that is, $f(e)=c(e)$ for all $e\in A(X,Y)$) and every backward edge is zero (that is, $f(e)=0$ for all $e\in A(Y,X)$).

The algorithm maintains the following data:

- The current vertex. Initially it is s.
- A list of scanned vertices. Initially this is empty.
- A list of labelled vertices. Initially this is $\{s\}$.
- A label for each labelled vertex v, of the form $(p(v),e(v),\varphi(v))$. (The purpose of these labels is to encode a path from s to v that has positive slack. Shortly, you'll see how that works.) The first label is the vertex *preceding* v, i.e., from which v was labelled; the second is the edge *entering* v, i.e., along which v was labelled, and the third is the maximum possible amount of flow from s to v along its labelling path. ($p(v)$ is redundant since it is implied by $e(v)$, but it's convenient to keep track of it.) Initially, only s is labelled and its label is $(-,-,\infty)$ (intuitively, the reason is that s wasn't labelled from anywhere and the amount of flow from s to s is unbounded).

  The *labelling path to v* is the (s,v)-path $s=v_l,e_l,\ldots,v_1,e_1,v_0=v$ composed of entering edges. It is obtained by backtracking from v employing the rules: $v_i=p(v_{i-1})$ and $e_i=e(v_{i-1})$ for each $i=1,\ldots,l$. Note that there is only one such path once v is labelled, and none before v is labelled.

Outline of the algorithm: Initially, every vertex is unlabelled (except for s) and unscanned. At each step a labelled but unscanned vertex v is scanned, which means that each unlabelled vertex adjacent to v is examined and labelled if possible. Then v is marked "scanned". The process is repeated until either t is labelled—this outcome leads to an augmenting path—or every labelled vertex has been scanned without labelling t—this outcome leads to a saturated (s,t)-cut.

Here is the exact algorithm:

Step 0. (Initialization.) Mark all vertices unscanned. Label s with the label $(-,-,\infty)$.

Step 1. Choose a labelled but unscanned vertex v. If none exists, stop.

Step 2. (Scanning v.) For every unlabelled neighbor w of v, do one of the following:

(a) Find an unsaturated edge $e: \overrightarrow{vw}$, set $\varphi = \min(\varphi(v), c(e) - f(e))$, and label w with (v, e, φ). If $w = t$, stop.

(b) Find a positive edge $e : \overrightarrow{wv}$, set $\varphi = \min(\varphi(v), f(e))$, and label w with (v, e, φ). If $w = t$, stop.

(c) If neither (a) nor (b) applies, don't label w.

Note that in the labelling path to w, e is a forward edge if (a) was used and a backward edge if (b) was used. Sometimes both (a) and (b) are applicable; then it doesn't matter which you choose.

Step 3. Mark v scanned. Go to Step 1.

The algorithm terminates with t either labelled or not. If t is unlabelled, the cut (X, Y) is saturated, where X is the set of labelled vertices and Y the set of unlabelled vertices. (Prove this!)

Lemma 3. *If t is labelled, the labelling path to t is an augmenting path with slack $\varphi(t)$. If t is unlabelled, $c(X, Y) = \mathrm{val}(f)$.*

To prove the second part of Lemma 3, it's helpful to have a general fact, whose proof (which is not trivial and not too hard) I leave to you.

Lemma 4. *Given a feasible (s,t)-flow f in a network, then $f(X, Y) = \mathrm{val}(f)$ for every (s,t)-cut (X, Y).*

E. The Max-Flow–Min-Cut (MFMC) Algorithm.

Step I. Start with any feasible (s,t)-flow, e.g., the zero flow, $f \equiv 0$.

Step II. Apply the labelling algorithm to find an augmenting path P. Augment f along P by the slack. Repeat until no augmenting path is found.

Step III. When labelling finds no augmenting path, it finds an (s,t)-cut (X, Y) for which $c(X, Y) = \mathrm{val}(f)$ (by Lemma 3). Therefore f is a maximum (s,t)-flow and (X, Y) is a minimum (s,t)-cut (by Lemma 1).

Thus we have proved the Max-Flow Min-Cut Theorem. The theorem needs to be supplemented by a termination lemma.

Lemma 5. *If all the capacities are rational or infinite, the process as described will terminate after a finite number of steps. If there is no (s,t)-cut with finite capacity, then there is an augmenting path with infinite slack, and such a path will eventually turn up through the labelling algorithm.*

(With irrational capacities the algorithm need not be finite—see Ford and Fulkerson, p. 21, for an example. It's obviously important to know when an algorithm is finite. There are algorithms that are always finite but slower than the MFMC algorithm. For instance, the max-flow problem is a special kind of linear programming problem, and there are finitely terminating algorithms for linear programming; but applying a general LP algorithm to network flows is almost like getting a backhoe to dig holes for planting in your flower garden.)

III. Decomposition Into Paths and Cycles.

For applications it is often valuable to have a structural description of an (s,t)-flow. We especially want to apply this to maximum (s,t)-flows in special networks in order to deduce the four 'Menger's Theorems'.

A. Path and cycle flows.

Given a cycle C in a network N (remember, this is a directed circuit), the *unit C-flow* δ_C is the flow having value 1 on edges of C and 0 elsewhere. Clearly, this is an (s,t)-flow with value 0. Given a directed (s,t)-path P in N, the *unit P-flow* δ_P is the flow having value 1 on edges of P and 0 elsewhere. Clearly, this is an (s,t)-flow with value 1. A *path* or *cycle flow* in N is any nonzero scalar multiple of a unit path or cycle flow, respectively. If we want to specify that the path is directed we may speak of a *directed path flow* or *dipath flow*.

B. Decomposition.

Two flows (feasible or not; they needn't even be (s,t)-flows) are *disjoint* if their nonzero edge sets are disjoint.

Given a feasible (s,t)-flow f, the *slack* of a cycle or dipath is the slack $\epsilon(T)$, defined previously, of a directed walk T that goes once along the cycle or dipath. If it happens that there is no cycle with positive slack, we call f *acyclic*.

Decomposition Theorem. *Any feasible (s,t)-flow f is a sum of feasible dipath and cycle (s,t)-flows.*

Notice that, if f is such a sum, then $\mathrm{val}(f)$ is the sum of the values of the dipath and cycle flows that sum to f.

Proof. We use induction on the number of positive edges in f. The base case, where there are no positive edges, is trivial (it's the sum of no dipath and cycle flows).

*Case*1. Suppose f has a positive edge departing from s. We start tracing a directed trail of positive edges from s. If we repeat a vertex, say having traced the trail $T : s = v_0, e_1, v_1, \ldots, v_l = v_i$ where $0 \leq i < l$, then the segment of T from v_i to v_l is a cycle C with positive flow. Let $f' = f - \epsilon(C)\delta_C$. It is easy to see that, because of the various definitions, f' has fewer positive edges than f; hence it is a sum of dipath and cycle flows; therefore, so is f. On the other hand, if we get stuck at a vertex v without having repeated a vertex, which means we found a directed positive trail to v but there is no positive edge departing from v, then the conservation of f at every vertex other than s and t (and the fact that v can't be s, since that would mean we repeated a vertex) shows that $v = t$. Thus T is a directed (s,t)-path with positive flow. We repeat the induction argument with T instead of C to conclude that f decomposes.

*Case*2. If f has no positive edge from s, it must have a positive edge from some vertex $v_0 \neq s$. Also, $\mathrm{val}(f) = 0$. (Prove this.) If we trace a trail of positive edges from v_0 until we repeat a vertex, we get a cycle and can proceed as in the repeated-vertex case of Case 1. We can never be forced to stop tracing at t because, since $\mathrm{val}(f) = 0$, if there is a positive edge entering t there must also be a positive edge leaving it.

That completes the proof.

Note that this proof is algorithmic: it shows you how to find a decomposition into dipath and cycle flows.

Now, take f to be any (s,t)-flow that is not all zero. Decomposing it into dipath and cycle flows, we can discard the cycle flows, producing a new flow F that is a sum of dipath flows. Also, F has the same value as does f. However, F is not necessarily acyclic.

Problem III.M1. Find a feasible (s,t)-flow F that is a sum of feasible (s,t)-dipath flows but is not acyclic.

Nevertheless, we can adjust F until no cycle remains. We do this by repeating the same procedure as in the proof of the Decomposition Theorem but ignoring every (s,t)-path the procedure produces. Instead, we try every possible way of tracing a positive directed trail from every possible vertex until we either find a positive cycle, which we subtract from f, or we've seen that there is no such cycle. (This is a finite process but extremely long and inefficient. Fortunately, it's good enough for our proof.) Since every time we find a positive cycle and subtract it we reduce the number of positive edges, we must come to an end. Then we have an acyclic (s,t)-flow whose value equals that of our original flow. Thus:

Lemma 6. *For any feasible (s,t)-flow, there is an acyclic feasible (s,t)-flow with the same value.*

Acyclic Decomposition Theorem. *Any network whose value is finite has a maximum (s,t)-flow f that is acylic. Furthermore, f is a sum of feasible dipath (s,t)-flows.*

The dipath flows need not be pairwise disjoint. However, we will be able to show that in the networks we're interested in for Menger's theorems, it's possible to make them disjoint.

IV. Integral Flows.

An *integral flow* (or *integer flow*) is a flow f such that $f(e)$ is an integer for every edge.

Integrality Theorem. *(a) If N is a network, with finite value, in which all capacities are integral or infinite, then it has a maximum (s,t)-flow that is integral and acyclic. (b) Any integral, feasible (s,t)-flow is a sum of integral, feasible dipath and cycle (s,t)-flows.*

Proof. This depends on analyzing the proofs: the process by which we obtained a maximum (s,t)-flow, that by which we modified the maximum (s,t)-flow to be acyclic, and that by which we decomposed a feasible flow. The key point is that, at each step, all the slacks are integers (or infinite). From this observation the theorem is immediate.

This theorem is almost as important as the MFMC itself!

V. Menger's Theorems and Some MFMC Generalizations.

We are finally ready to prove Menger's theorems. The trick is to choose a network, derived from the graph or digraph, in which a maximum (s,t)-flow corresponds to a maximum-sized set of pairwise disjoint paths. (That's why we need the integrality and acyclic-decomposition theorems.) This is a challenging but fun process and I'm tempted to leave it to you. In fact, I will, for now, and present the solutions later.

By the way, it's interesting that, if I remember correctly something I think I once read, three of the "Menger's" theorems were not found by Menger but by the discoverers of the MFMC Theorem, long after Menger.

First, Menger's theorems.

Problem V.M1. Given a digraph D, find a network for which the maximum (s,t)-flow gives you a proof of Menger's edge theorem for D. (This is the easiest problem.) The theorem I mean is that, if s,t are distinct vertices in D, then the maximum number of edge-disjoint directed (s,t)-paths in D equals the minimum number of edges whose deletion eliminates all directed (s,t)-paths.

Problem V.M2. Given a digraph D, find a network for which the maximum (s,t)-flow gives you a proof of Menger's vertex theorem for D. (This requires a clever trick.)

Problem V.M3. Given a graph G, find a network for which the maximum (s,t)-flow gives you a proof of Menger's edge theorem for G. (This requires a different trick. Maybe it isn't hard.)

Problem V.M4. Given a graph G, find a network for which the maximum (s,t)-flow gives you a proof of Menger's vertex theorem for G.

Next, two obviously important generalizations of network flows that can be reduced to the ordinary network flow theory discussed previously.

Problem V.M5. One can generalize Menger's vertex theorems by allowing vertex capacities as well as edge capacities in a network. Show how to take a network with vertex capacities, $N = (D, s, t, c_0, c_1)$ (where c_0 and c_1 are the vertex and edge capacity functions), and construct a network N' without vertex capacities whose maximum flows correspond to maximum flows in the original network.

Problem V.M6. Generalize the Max-Flow–Min-Cut Theorem (and the related theorems) to networks with multiple sources and sinks. (Hint: Try to reduce this problem to the single-source, single-sink problem by a clever construction.)

And, circling back to pure graph theory:

Problem V.M7. Use the multiple MFMC (see Problem V.M6) to derive versions of Menger's Theorems in which s and t are replaced by whole sets S and T of vertices. (This is the beginning of the theory of "linkings" in graphs.)

VI. Example.

Example 1. The network N has $V = \{w, t, u, v\}$ and edges and capacities

$\overrightarrow{sv}$	$\overrightarrow{st}$	$\overrightarrow{us}$	$\overrightarrow{uv}$	$\overrightarrow{vu}$	$\overrightarrow{vt}$	$\overrightarrow{tu}$
3	4	5	3	7	4	3

Some (s,t)-paths in N are:

$$P_1 = (\overrightarrow{st}),$$

$$P_2 = (\overrightarrow{us}, \overrightarrow{uv}, \overrightarrow{vt}),$$

$$P_3 = (\overrightarrow{sv}, \overrightarrow{uv}, \overrightarrow{tu}).$$

Of these only P_1 is a dipath.

The unit path flow δ_{P_2} assigns edge flows

$\overrightarrow{sv}$	$\overrightarrow{st}$	$\overrightarrow{us}$	$\overrightarrow{uv}$	$\overrightarrow{vu}$	$\overrightarrow{vt}$	$\overrightarrow{tu}$
0	0	−1	1	0	1	0

This is an (s,t)-flow but not a dipath flow because P_2 is not a directed path.

The flow $f = \delta_{P_1} + 4\delta_{P_2} + \delta_{P_3}$ assigns edge flows

$\overrightarrow{sv}$	$\overrightarrow{st}$	$\overrightarrow{us}$	$\overrightarrow{uv}$	$\overrightarrow{vu}$	$\overrightarrow{vt}$	$\overrightarrow{tu}$
1	1	−4	5	0	1	−1

This is an (s,t)-flow but it's infeasible since it has negative edge flows. It's also infeasible because $f(\overrightarrow{uv}) > c(\overrightarrow{uv})$.

Problem VI.M1. Apply the Max-Flow–Min-Cut algorithm to this network, starting with the flow f whose amounts are

$\overrightarrow{sv}$	$\overrightarrow{st}$	$\overrightarrow{us}$	$\overrightarrow{uv}$	$\overrightarrow{vu}$	$\overrightarrow{vt}$	$\overrightarrow{tu}$
3	0	3	0	3	0	0

When finished, display the last flow f and the saturated cut, with $\mathrm{val}(f)$ and the capacity of the cut calculated. They should be equal (thus proving f is a maximum flow)—if not, you made a mistake somewhere, according to the MFMC Theorem.

VII. Variations.

The first variation is that with multiple sources and sinks, as in Problem V.M6.

The second variation involves one of the important aspects of network flows: cost. Here each edge has a cost (per unit of flow) and the objective is to minimize the total cost. I believe there are at least two versions: there may be a capacity function and one wants, among all maximum flows, one whose cost is minimal; or there might be no capacity function, but the value of the flow is prescribed and one wants to minimize the cost.

This leads me to the third variation: there is a given "supply" at the source and a given "demand" at the sink. A flow must satisfy the constraints that the net outflow at the source is no greater than the supply, and the net inflow to the sink is not less than the demand. This variation would seem to be particularly significant when there are multiple sources and sinks.

That leads to the fourth variation: the transportation problem, or Hitchcock problem. Here there are only sources and sinks, but many of them, and no intermediate nodes. There may be capacities, costs, supplies, and demands. I think this was the original network problem. It was developed and studied in order to help win World War II (really!).

VIII. Solution Techniques for Some Problems.

These are the ideas of the solutions. I leave the actual proofs to you.

Problem V.M1. The network consists of the digraph D, s and t as source and sink, and capacity 1 on every edge. Then the value of the network equals $\lambda(s,t)$, the edge connectivity from s to t. The hard part is to prove that there is a maximum flow that is the sum of exactly $\lambda(s,t)$ unit path flows.

Problem V.M2. Split each vertex v of D into two vertices, v_- and v_+. Each edge $e : \overrightarrow{vw}$ becomes an edges $e : \overrightarrow{v_+ w_-}$; that is, v_- becomes the head of all incoming edges at v and v_+ becomes the tail of all outgoing arcs at v. Also, add an edge $\overrightarrow{v_- v_+}$ for every vertex—call this a *vertex edge*. Then make a network as in Problem V.M1, but put a capacity of 1 on each vertex edge ∞ on each original edge. (The capacities of the original edges may be anything ≥ 1, but capacity ∞ guarantees that a minimum cut in the network consists only of vertex edges and so corresponds to an (s,t)-separating set of vertices. That makes it easier to deduce Menger's theorem.)

Problem V.M3. Most simply, replace each undirected edge by two directed edges in opposite directions. This enables you to deduce the undirected Menger theorem from the directed one, or from the MFMC by going through the previously described construction of a network.

Problem V.M4. Combine the solutions to V.M2 and V.M3.

Problem V.M5. Similar to the method of Problem V.M2. You put the original capacity on each original edge and the vertex capacity on the vertex edge. However, here you don't have the convenience of infinite capacities to allow you to prove (as in Menger's vertex theorems) that the minimum cuts correspond to sets of vertices—and it isn't true anyway, in this generality.

Problem V.M6. Create a new vertex s_0 to be the source, connected to every former source s by an edge $\overrightarrow{s_0 s}$, and a new sink t_0 similarly connected from every former sink t

by $\overrightarrow{tt_0}$. Give the new edges infinite capacity so they can't belong to any minimum cut.

Problem V.M7. This is routine, but a very useful version of Menger's theorem(s).

IX. References.

The original book on network flows is

L.R. Ford, Jr., and D.R. Fulkerson, *Flows in Networks*, Princeton University Press, Princeton, N.J., 1962.

It's a short book with a lot in it. It should be in our library but it seems that somehow we didn't acquire it. We'll get it soon.

A simple and relatively short article that discusses network flows and applications (to real life) is

Giuseppe Avondo-Bodino, "Graph Theory and Operations Research", Chapter 8, pp. 223–253, in Robin J. Wilson and Lowell W. Beineke, eds., *Applications of Graph Theory*, Academic Press, London, 1979. Our library has it under call number QA166 .W49.