J. Soc. Indust. Appl. Math.
Vol. 9, No. 4, December, 1961
Printed in U.S.A.

MULTI-TERMINAL NETWORK FLOWS*

R. E. GOMORY† AND T. C. HU†

The network flow problem was first considered by Ford and Fulkerson [1] who introduced the basic concepts of flow, cut, etc. used here and provided the main tool, the maximum-flow minimum-cut theorem. Ford and Fulkerson wrote about the flow between two special points, the source and the sink. Mayeda [2] then took up the multi-terminal problem, where flows are considered between all pairs of nodes in a network, and Chien [3] discussed the synthesis of such a network. In this paper we continue with the multi-terminal problem, giving results on realizability, analysis, and synthesis. Although this paper is self-contained we will use throughout many of the notions of [1, 2, 3].

We consider connected networks consisting of nodes N_i and branches B_{ij} connecting the ith and jth nodes. Each branch (or arc) has associated with it a nonnegative number b_{ij} called the branch capacity. The branch capacity is the maximum amount of flow that can pass through the branch, from N_i to N_j. We assume $b_{ij} = b_{ji}$ throughout. Given such a network there is for each pair of nodes a maximal flow possible between them. The value of this flow between the ith and jth nodes will be denoted by f_{ij}.

Consequently for each network there are two associated symmetric matrices: The matrix B of the b_{ij} and the matrix F of the resulting flows f_{ij}. Not any matrix can be an f_{ij} matrix so it is natural to ask when can a given set of flows f_{ij} be realized by some network. One answer in terms of the ability to repeatedly partition the matrix of f_{ij} in a particular way has already been given by Mayeda [2]. Here we give another necessary and sufficient condition—a sort of "triangle inequality."

This condition reduces the problem of deciding whether or not a given matrix F is realizable by some network to the well known problem of constructing a maximal tree of a network—a problem already solved in a very effective manner first by Kruskal [4] and even more efficiently by Prim [5].

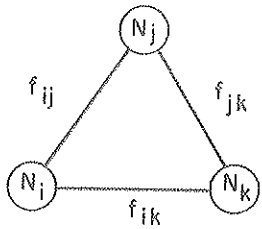
THEOREM. *A necessary and sufficient condition for a matrix F to be realizable is that*

$$f_{ik} \geqq \min (f_{ij}, f_{jk}) \tag{1}$$

for all i, j, k.

* Received by the editors November 18, 1960 and in revised form April 28, 1961.
† International Business Machines Corporation Research Center, Yorktown Heights, New York.

Geometrically this means simply that the flow from N_i to N_k must be at least as great as the smaller of the two flows round by way of node N_j.



We first show necessity. Suppose the theorem were false and for some $i, j, k, f_{ik} < \min(f_{ij}, f_{jk})$. Then there exists by the maximum-flow minimum-cut theorem [1] a cut or division of the nodes into two sets A and $\bar{A}$ with $N_i \in A$ and $N_k \in \bar{A}$ such that the total capacity of the links connecting nodes in A with nodes in $\bar{A}$ is f_{ik}. Now N_j belongs either to A or $\bar{A}$. If it is in A, then it is cut off from N_k by the cut; since the capacity of the cut is $< f_{jk}$, this is a contradiction. Similarly N_j can not be in $\bar{A}$, for then it is cut off from N_i. Therefore $f_{ik} \geqq \min(f_{ij}, f_{jk})$.

Once established, the relation $f_{ik} \geqq \min(f_{ij}, f_{jk})$ has, by induction, the immediate consequence

$$f_{ip} \geqq \min(f_{ij}, f_{jk}, f_{kl}, \cdots, f_{op}) \tag{2}$$

where the $N_i, N_j, \cdots, N_p$ form any path from N_i to N_p.

Now for the sufficiency. For this we need first the notion of spanning tree, then the notion of maximal spanning tree. The notion of tree we take as known, see for example [7]. A spanning tree is simply a tree that includes all nodes. If there are numbers n_{ij} attached to the arcs of a tree one may introduce the value of a tree as the sum of the numbers n_{ij} on the arcs of the tree. We can do this using as numbers either the b_{ij} or the f_{ij} for the arc connecting N_i and N_j.

Among spanning trees there is one or more whose value is maximal among spanning trees. This is a maximal spanning tree and can easily be constructed by Prim's method. Any maximal spanning tree has the following easily established property. Let N_i and N_p be two nodes whose direct connecting arc is not in the tree, then the number in that connecting arc satisfies $n_{ip} \leqq \min(n_{ij}, n_{jk}, \cdots, n_{op})$ where the $n_{ij}, \cdots, n_{op}$ are numbers on the arcs of the (unique) path connecting N_i to N_p within the tree. For if the inequality did not hold, the smallest arc in the tree path could be removed and the direct arc N_iN_p substituted to form a tree with value larger than the maximum.

Using the f_{ij} as attached numbers we consider any maximal spanning

tree. From the maximality we have as above for a direct arc being compared with a path through the tree,

$$f_{ip} \leqq \min (f_{ij}, \cdots, f_{op}),$$

while from (2) we have the opposite inequality. So for any arc not in the tree

$$f_{ip} = \min (f_{ij}, \cdots, f_{op}). \tag{3}$$

However this is precisely the flow which results if a network is constructed with branch capacity $b_{ij} = f_{ij}$ for arcs in the tree, and $b_{ij} = 0$ otherwise. Thus any F matrix satisfying the condition is realizable. This ends the proof.

To see if a given matrix is in fact realizable one could construct a maximal spanning tree using Prim's method, then check to see if condition (3) is satisfied. Actually it is far more economical (and extremely easy) to check (1) during the course of Prim's algorithm as described in the appendix.

We next turn to the problem of analysis, i.e., given a network of arcs B_{ij} with capacities b_{ij}, what are the resulting flows f_{ij}? We have seen from (3) that any flow is numerically equal to some flow in the maximal spanning tree. As there are only $n - 1$ arcs in a spanning tree (where n is the number of nodes) there are only $n - 1$ numerically different flows possible. This makes it reasonable to suspect that all $n(n - 1)/2$ flows f_{ij} can be obtained by something better than doing $n(n - 1)/2$ flow problems. We will in

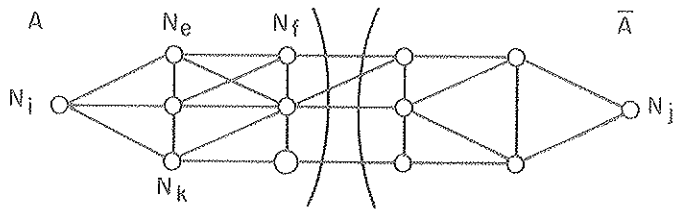


FIG. 1.

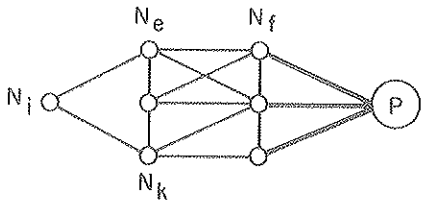


FIG. 2.

fact show that *all flows can be deduced after only* $n - 1$ *flow problems have been computed.*

Consider first a network such as the one shown in Fig. 1 with a minimal cut $(A, \bar{A})$ separating N_i and N_j, $N_i \in A$, $N_j \in \bar{A}$.

Then let us construct a slightly different network, one in which all nodes in $\bar{A}$ are replaced by a single special node P to which all the arcs of the cut are attached (we can replace several arcs connecting the same two nodes by one arc having the total capacity). In this condensed network (Fig. 2) let us consider the maximal flow between two ordinary nodes, N_e and N_k. We will show

LEMMA 1. *The flow between two ordinary nodes* N_e *and* N_k *in the condensed network is numerically equal to the flow* f_{ek} *in the original network.*

Proof. Let $(B, \bar{B})$ be a minimal cut separating N_e and N_k in the original network and define sets of nodes

$$X = A \cap B, \qquad \bar{X} = A \cap \bar{B},$$

$$Y = \bar{A} \cap B, \qquad \bar{Y} = \bar{A} \cap \bar{B}.$$

Here $\bar{X}$ is the complement of X in A, $\bar{Y}$ is the complement of Y in $\bar{A}$. We may assume that $N_e \in X$, $N_k \in \bar{X}$ and $N_i \in X$. Let $b_{XY} = \Sigma b_{ij}$ where $N_i \in X$ and $N_j \in Y$.

Case 1. $N_j \in Y$. Now

$$b_{A\bar{A}} = b_{XY} + b_{X\bar{Y}} + b_{\bar{X}Y} + b_{\bar{X}\bar{Y}},$$

$$b_{B\bar{B}} = b_{X\bar{X}} + b_{X\bar{Y}} + b_{\bar{X}Y} + b_{Y\bar{Y}}.$$

Since $(B, \bar{B})$ is a minimal cut separating N_e and N_k, and since $(X \cup Y \cup \bar{Y}, \bar{X})$ separates N_e and N_k, we have

$$b_{B\bar{B}} - b_{X\cup Y\cup\bar{Y},\bar{X}} = b_{X\bar{Y}} + b_{Y\bar{Y}} - b_{\bar{X}\bar{Y}} \geqq 0. \tag{3.1}$$

Since $(A, \bar{A})$ is a minimal cut separating N_i and N_j, and since $(X \cup \bar{X} \cup \bar{Y}, Y)$ separates N_i and N_j, then

$$b_{A\bar{A}} - b_{X\cup\bar{X}\cup\bar{Y},Y} = b_{X\bar{Y}} + b_{\bar{X}\bar{Y}} - b_{Y\bar{Y}} \geqq 0. \tag{3.2}$$

Adding (3.1) and (3.2) shows that $b_{X\bar{Y}} \geqq 0$ and hence $b_{X\bar{Y}} = 0$. It then follows from (3.1) and (3.2) that $b_{Y\bar{Y}} - b_{\bar{X}\bar{Y}} = 0$ also. Hence $(X \cup Y \cup \bar{Y}, \bar{X}) = (X \cup \bar{A}, \bar{X})$ is also a minimal cut separating N_e and N_k.

Case 2. $N_j \in \bar{Y}$. A similar proof shows that $(X, \bar{X} \cup \bar{A})$ is a minimal cut separating N_e and N_k in this case.

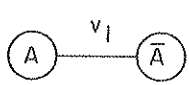


FIG. 3.

In other words, there is always a minimal cut separating N_a and N_b such that the set of nodes $\bar{A}$ is on one side of this cut. Since the flow value is determined by the value of this minimum cut which is unchanged by the condensing process, condensing $\bar{A}$ to a single node does not affect the value of a maximal flow from N_a to N_b.

Thus any flow between N_a and N_b in the condensed network gives rise to an equal flow in the original network. This gives the desired inequality.

Since a cut in the condensed network gives a cut in the original, and the maximal flow values are the same, a minimal cut in the condensed network gives, simply by replacing P by $\bar{A}$, a minimal cut in the original.

We now proceed to the analysis.

One procedure is simply this. We take two nodes and do a maximal flow computation [1] to find a minimal cut $(A, \bar{A})$. We represent this by two generalized nodes connected by an arc bearing the cut value (Fig. 3). In one node are listed the nodes of A, in the other those of $\bar{A}$. We now repeat this process. Choose two nodes in A (or two in $\bar{A}$), and solve the flow problem in the condensed network in which $\bar{A}$ (or A) is a single node. The resulting cut has a value v_2 and is represented by a link connecting the two parts into which A is divided by the cut, say A_1 and A_2. $\bar{A}$ is attached to A_1 if it is in the same part of the cut as A_1, to A_2 if it is in the same part as A_2 (Fig. 4).

The cutting up is then continued. At each stage we have certain generalized nodes (which may represent many nodes of the original network), and certain arcs connecting them. To proceed with the computation we select a generalized node A_i and two original nodes N_a and N_b in A_i. Upon removing all arcs which connect to A_i, the network of generalized nodes falls into a number of disconnected components. We condense each component, except A_i itself, into a single node and solve the network flow problem consisting of these condensed nodes and the original nodes within A_i, and using N_a and N_b as source and sink. The minimal cut obtained by this flow calculation splits A_i into two parts, A_{i_1}, A_{i_2}. This is represented in the diagram by replacing A_i by two generalized nodes A_{i_1} and A_{i_2} connected by an arc bearing the cut value. All other arcs and generalized nodes in the diagram are unchanged except those arcs which formerly

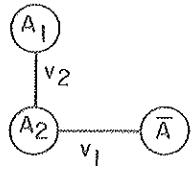


FIG. 4.

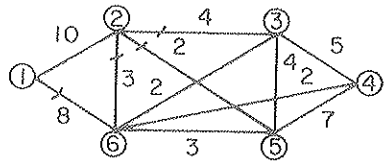


FIG. 5.

connected to A_i. Such an arc is now attached to A_{i_1} if its component was on the same side of the cut as the nodes in A_{i_1}, and attached to A_{i_2} if its component fell on the other side.

This process is repeated until the generalized nodes of the diagram consist of exactly one node each. This point is reached after exactly $n - 1$ cuts, for the array is a tree at all times so when the process stops it is an n-node tree and so has $n - 1$ branches each created by solving a flow problem in a network equal to or smaller in size than the original.

We then assert

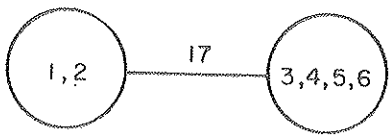
LEMMA 2. *The flow between any two points is simply*

$$\min (v_{i_1}, v_{i_2}, \cdots, v_{i_r}),$$

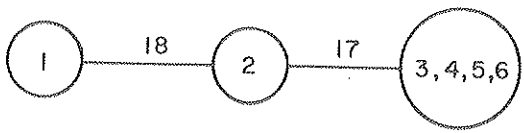
where the v_i are values of a series of arcs of the tree connecting the two points.

Before proceeding to prove this last assertion it is probably a good idea to illustrate the process by an example.

Taking as our B_{ij} network the net in Fig. 5, we arbitrarily choose nodes 2 and 6, and upon doing a flow problem we find the minimum cut to be (as indicated in Fig. 5) (1, 2, | 3, 4, 5, 6) with capacity 17. This is represented by



To get the flow 1-2 we consider 3, 4, 5, 6 as a single node, obtaining Fig. 6, in which the minimum cut 1-2 is (1 | 2, 3, 4, 5, 6) with capacity 18, so



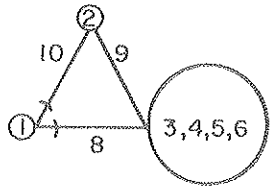


FIG. 6.

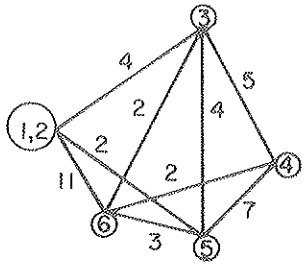


FIG. 7.

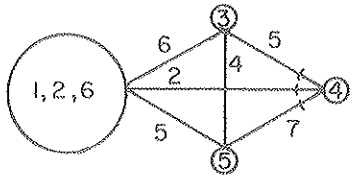
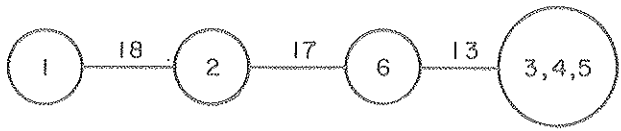
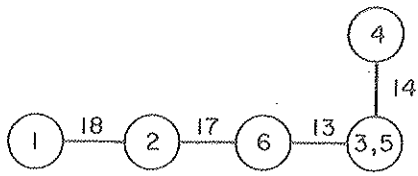


FIG. 8.

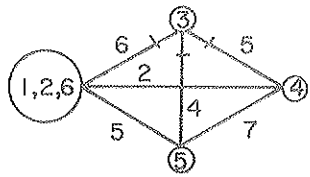
We next choose 3 and 6. Considering 1 and 2 as a single node, we find the minimum cut (1, 2, 6 | 3, 4, 5), capacity 13. So



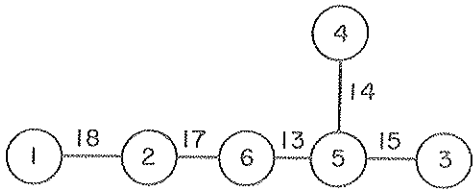
We next consider the flow 4-5, taking 1, 2, 6 as a node (Fig. 8), the resulting minimum cut being (4 | 1, 2, 3, 5, 6) with capacity 14. So



Finally we consider the flow 3-5, taking 1, 2, 6 as one node and 4 as the other to get the same network as above,



the minimum cut 3-5 being $(3 \mid 1, 2, 4, 5, 6)$ capacity 15, giving as the final tree,

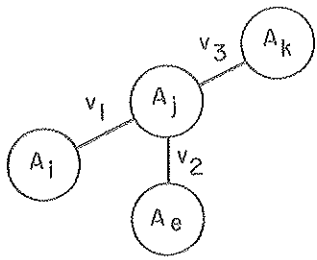


We would now assert that the maximum flow 1-3 is 13, the maximum flow 1-6, 17, etc.

We now proceed to prove Lemma 2. Consider two nodes N_i and N_j. We certainly have

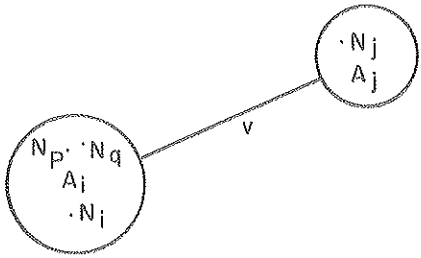
$$f_{ij} \leqq \min (v_1, \cdots, v_r),$$

for each v_i on the path connecting N_i and N_j corresponds to a cut separating N_i and N_j. To show the reverse inequality is a little more difficult. Consider any stage of the construction

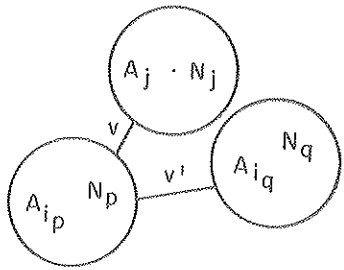


where we have ares representing cuts and nodes representing sets. We assert that if an arc of value v connects sets A_i and A_j then there is a node N_i in A_i and a node N_j in A_j such that $f_{ij} = v$.

This is certainly true after the first cut. We will show that the property is maintained. Consider an A_i about to be cut,



with A_j representing the set attached by v. By the induction hypothesis there is an N_i in A_i and N_j in A_j with $f_{ij} = v$. After cutting N_p from N_q, A_i divides into A_{i_p} and A_{i_q}. We can assume A_j is attached to A_{i_p}.



Clearly N_p and N_q provide the desired flow $f_{pq} = v'$ across the new link. As to the old link of value v there are two cases:

(i) $N_i \in A_{i_p}$.

Then the flow $f_{ij} = v$ is still applicable.

(ii) $N_i \in A_{i_q}$.

Then consider the nodes N_j, N_i, N_q, and N_p. From (2)

$$f_{jp} \geqq \min (f_{ji}, f_{iq}, f_{qp}).$$

Since N_j and N_p are on one side of the cut whose value is v' and N_i and N_q are on the other, we know that the flow f_{jp} is unaffected if A_{i_q} is replaced by a single node or, what is the same thing, if all arcs within A_{i_q} are given an arbitrarily large value M. Doing this makes f_{iq} large so we have

$$f_{jp} \geqq \min (f_{ji}, f_{qp}).$$

Since $f_{ji} = v$ and $f_{qp} = v'$, we have

$$f_{jp} \geqq \min (v, v').$$

Since the cut separating N_j and N_i is of value v', we must have $v' \geqq f_{ij} = v$. So finally $f_{jp} \geqq v$. As v is the value of a cut separating N_j and N_p, this implies $f_{jp} = v$. Thus N_j and N_p provide the two needed nodes.

Since we now know that in the final tree the values on the links actually represent flow values between the adjacent points, the reverse inequality

$$f_{ij} \geqq \min (v_1, \cdots, v_r)$$

is once again just an application of (2).

This establishes the desired result

$$f_{ij} = \min (v_1, \cdots, v_r).$$

Hence the flow matrix for our example is

	1	2	3	4	5	6
1	d	18	13	13	13	17
2	18	d	13	13	13	17
3	13	13	d	14	15	13
4	13	13	14	d	14	13
5	13	13	15	14	d	13
6	17	17	13	13	13	d

In any tree diagram an arc from N_i to N_j can be considered as representing a cut of the nodes since its removal divides the nodes of the tree into two sets, A and $\bar{A}$. If in addition each cut so obtained is a minimal cut between N_i and N_j in some network N, and the attached branch value in the tree is the capacity of the cut in N, then the tree is called a cut tree of N.

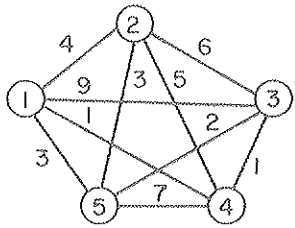
We have just shown a way of obtaining a cut tree of a network by solving $n - 1$ flow problems.

Given an $n \times n$ matrix of nonnegative numbers $r_{ij} (r_{ij} = r_{ji})$ we will call an n-node network N satisfactory if its flows satisfy

$$f_{ij} \geqq r_{ij} \qquad \text{(all } i, j; i \neq j). \tag{4}$$

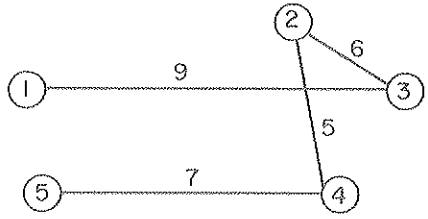
The *synthesis problem* we consider is the one of finding a satisfactory network having smallest cost. If we say that the cost of installing one unit of branch capacity between N_i and N_j is c_{ij}, then it seems necessary to use the apparatus of linear programming and this approach is developed in [6]. However if we try to find the satisfactory network of least total branch capacity, or what is the same thing, take the case when $c_{ij} = 1$ for all i, j, special methods can be devised, and it is this problem we take up now. (For other approaches applicable to a realizable requirement matrix in partitioned form see [3] and [8]).

We first introduce a tree T of dominant requirements. This is simply any maximal spanning tree constructed using the r_{ij} as arc values. For example given the requirements



	①	②	③	④	⑤
①	x	4	9	1	3
②	4	x	6	5	3
③	9	6	x	1	2
④	1	5	1	x	7
⑤	3	3	2	7	x

an easily constructed dominant requirement tree is



	①	②	③	④	⑤
①	x	0	9	0	0
②	0	x	6	5	0
③	9	6	x	0	0
④	0	5	0	x	7
⑤	0	0	0	7	x

Any satisfactory network must of course satisfy all requirements $f_{ij} \geqq r_{ij}$ where r_{ij} are attached to arcs in T. This is also sufficient since the missing r_{ip} satisfy the usual relation

$$r_{ip} \leqq \min (r_{ij}, r_{jk}, \cdots, r_{qp})$$

when the r's referred to form a path in T. In any network satisfying the dominant tree requirements (dominant requirements for short), the flow f_{ip} must automatically satisfy

$$f_{ip} \geqq \min (f_{ij}, f_{jk}, \cdots, f_{qp}) \geqq \min (r_{ij}, \cdots, r_{qp}) \geqq r_{ip},$$

and so satisfy all requirements.

Because of this we will henceforth consider only the dominant requirements. This is not necessary, only convenient: the methods we will develop can easily be modified to apply directly to the original requirement network, but the exposition is simplified and essentials brought out more easily by working with the dominant requirements.

We define the total branch capacity to be $\frac{1}{2}\sum_{i \neq j} b_{ij}$. We now follow Chien [3] in introducing a lower bound C_L for this quantity. Consider any node N_i of the network. Let $u_i = \max_j r_{ij}$, that is, u_i is the largest flow requirement out of N_i. Define $C_L = \frac{1}{2}\sum_i u_i$. Then as any satisfactory

network N must provide capacities b_{ij} capable of carrying this flow out of N_i, $\sum_j b_{ij} \geqq u_i$, and thus the total branch capacity $= \frac{1}{2}\sum b_{ij} \geqq \frac{1}{2}\sum_i u_i = C_L$.

The number C_L can also be computed directly from T without reverting to the original requirement, since the same u_i results if the $\max_j r_{ij}$ is taken only over branches of T adjacent to N_i.

Now consider a fixed tree T with attached numbers r_{ij} and resulting bound C_L. If the r_{ij} are replaced by a different set r'_{ij} a new bound C_L' results. If we use $r''_{ij} = r'_{ij} + r_{ij}$ on the arcs of the same fixed tree we get C_L'' always with

$$C_L'' \leqq C_L' + C_L ,$$

but if r_{ij} (or r'_{ij}) are "uniform" requirements, i.e., $r_{ij} = \beta$ for all r_{ij} in T, then obviously equality holds so

$$C_L'' = C_L' + C_L , \tag{5}$$

a result we will use in what follows.

One further remark is needed. If a branch capacity network B_{ij} with capacities b_{ij} and flows f_{ij} is superposed on another having the same nodes, arcs, and different capacities b'_{ij} and flows f'_{ij}, then the resulting network is taken as having arcs with capacities $b''_{ij} = b'_{ij} + b_{ij}$. The new flows f''_{ij} clearly satisfy

$$f''_{ij} \geqq f'_{ij} + f_{ij} . \tag{6}$$

Consider a requirement tree T with $n - 1$ requirements which we will now designate. Let the smallest be $r_{\min}$. Since all requirements can be

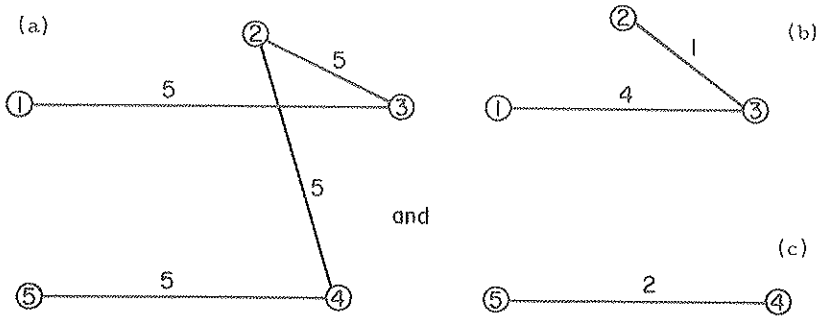


FIG. 9.

written as $r_{\min} + (r_{ij} - r_{\min})$ we can regard T as being obtained by superposing a uniform requirement tree T with uniform requirement $r_{\min}$ on two smaller trees with requirements $r_{ij} - r_{\min}$. For example the dominant requirement tree in the preceding example can be regarded as the superposition of a uniform 5-tree, i.e., dominating requirement tree with $r_{ij} = 5$ (Fig. 9).

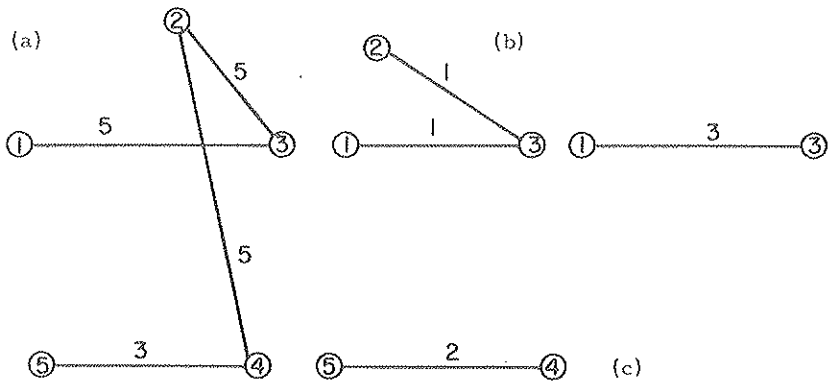
If the uniform tree and the residual parts can be synthesized by networks N_a, N_b, N_c so that their individual C_L's are actually attained, then by simply superposing the synthesized networks we get new flows which by (6) equal or exceed the requirements, while the total capacity used is by (5) equal to the lowest possible amount C_L. From now on we shall use the phrase "synthesizing trees" to mean "constructing a network with maximum flows greater or equal to the requirements in the tree."

The synthesis problem for T then has been reduced to the synthesis of smaller trees and a uniform requirement tree such that their lower bounds are actually obtained.

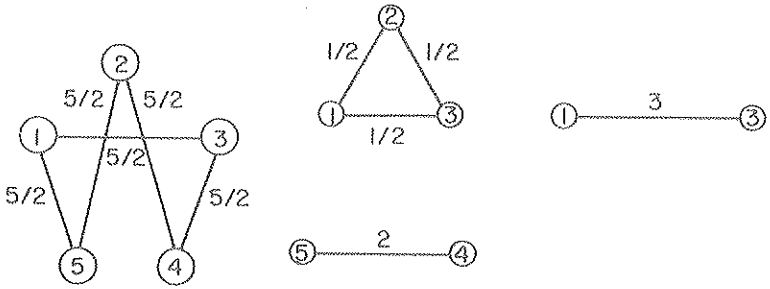
However we can repeat the decomposition process on the smaller trees until only uniform trees remain so that the problem is actually reduced to synthesizing uniform trees.

This however is extremely easy. Given any tree T' with uniform requirement β, the lower bound C_L is $n\beta/2$ and a suitable network is constructed by drawing any loop through the nodes and then assigning capacity $\beta/2$ to each arc of the loop. (In the smallest case, $n = 2$, both links of the loop coincide and a single arc of capacity β is used.)

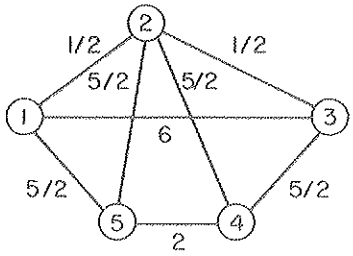
In the case of our example, to carry out the process we continue the decomposition begun in Fig. 9 by decomposing (b) further so that the tree T becomes the sum of



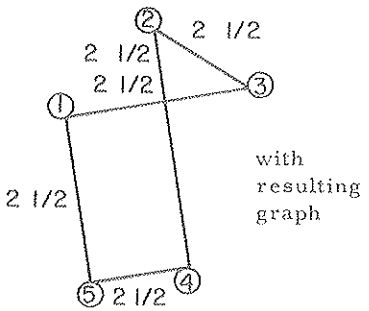
Each of the above is synthesized by a loop,

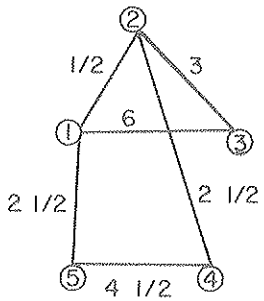


and these are superposed to give a minimal satisfactory network



Note that in synthesizing a uniform requirement tree we may use any loop passing through the nodes in any order. For example, (a) could have been synthesized by





Similarly any convex combination of loops can be used, i.e., if b_{ij}, b'_{ij}, b''_{ij}, $\cdots$ are the capacities of a link in various loops, then the graph with $\lambda_1 b_{ij} + \lambda_2 b'_{ij} + \lambda_3 b''_{ij}$, $\cdots$, $\Sigma\lambda_i = 1$ is also a minimal synthesis and can be used.

What we have shown is that this method of synthesis will produce

$f_{ij} \geqq r_{ij}$. On actually checking the two networks synthesized above, we find that the second meets all requirements in the dominating tree exactly, while the first gives some excess flows—but of course at no cost in capacity. We will take up first the problem of getting as much excess as possible and then the problem of exactly meeting requirements in the dominating requirement tree.

Note that the numbers u_i defined above are the ones that determine C_L, not the requirements. Consequently, once the u_i are determined all r_{ij} can be revised upward to $\bar{r}_{ij} = \min(u_i, u_j)$ without affecting C_L (and clearly no further increase is possible on any arc without affecting C_L). If the new requirements $\bar{r}_{ij}$ are now synthesized they will be met exactly, i.e., the resulting flows $\bar{f}_{ij}$ satisfying $\bar{f}_{ij} = \bar{r}_{ij}$ for $\bar{f}_{ij} > \bar{r}_{ij}$ would necessitate, at either N_i or N_j, a larger u and hence a larger C_L. Also, the synthesized network has for the same reason the following property: let f'_{ij} be the flows provided by any other minimal capacity network satisfactory with respect to the original r_{ij}, then

$$\bar{f}_{ij} \geqq f'_{ij} \qquad \text{(all } i, j.\text{)},$$

i.e., the network obtained by revising the requirements to $\bar{r}_{ij}$ and then synthesizing provides, at no cost in total capacity, more (or the same) flow between *every* pair of points as does any other satisfactory minimal network. More flow between *any* pair of points can be bought only by increasing total capacity.

We can summarize this property of uniform dominance in the following theorem which involves $u_i = \max_j r_{ij}$ and $\bar{C}_L = \frac{1}{2}\sum_{i \neq j} b_{ij}$.

THEOREM. *Given requirements r_{ij} there is a satisfactory network $\bar{N}$ having capacity $\bar{C}_L$ and giving flows*

$$\bar{f}_{ij} = \min(u_i, u_j) \qquad \text{(all } i \neq j\text{)},$$

while if f_{ij} are the flows from any other satisfactory network N then either N's total capacity C satisfies

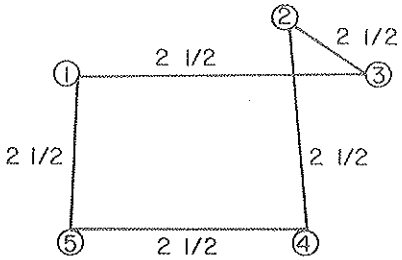
$$C > \bar{C}_L$$

or

$$f_{ij} \leqq \bar{f}_{ij} \qquad \text{(all } i \neq j\text{)}.$$

We now turn to the problem of exactly meeting the requirements in the general case and, of course, at minimal capacity. We already know how to secure flows $f_{ij} \geqq$ the requirements r_{ij}. To secure the opposite inequality it is only necessary, after decomposing the original requirements into a sum of uniform requirement trees, to synthesize each uniform tree so that the links of the tree represent not only requirements, but also minimal cuts of the synthesized network. For instance in our example the

cut tree of the loop used in synthesizing the uniform requirement of 5 (Fig. 9) does *not* have a cut of capacity 5 separating 1, 2 and 3 from 4 and 5, as it would if it were a cut tree. However the synthesis by the



loop 1, 3, 2, 4, 5, 1 does have the required cut tree. In superposing requirement trees that are also cut trees, minimal cuts in the synthesized networks are superposed on minimal cuts to form minimal cuts. (Clearly if $(A\bar{A})$ is the minimal cut between N_i and N_j in one network and also in a second then it is also a minimal cut in the superposed network.) Thus the original requirement tree is synthesized with a cut corresponding to each link. Hence the flow f_{ij} between two N_i and N_j satisfies

$$f_{ij} \leqq \min (r_{ik}, r_{kt}, \cdots, r_{pj}),$$

(where the r_{ik}, etc., are in T) for each represents a cut separating N_i from N_j, i.e., if one can synthesize a uniform tree with desired cut tree then the original requirement tree can be synthesized as a cut set tree, which results in exact synthesis.

To synthesize a given uniform tree as a cut tree is, however, quite easy. We give the following rule:

If T' is a uniform requirement tree it is synthesized as follows.

Preliminaries: Label all arcs in T' with a zero, and choose any node as starting point. Label this node with 1.

In what follows when we speak of labeling a node we mean to assign the first node labeled the number 1, the second node labeled the number 2, etc.

(1) Find the arc with smallest label v incident to the starting node (any one will do if there are several).

(1a) if $v \neq 0$ you must label the node.

(1b) if $v = 0$ you may or may not label the node.

(2) Proceed over the arc mentioned in (1) to the next node, increasing the arc label by 1.

(3) Continue this process until you return to your starting node and find all incident arcs labeled 2. (This will happen.) Then stop.

We assert that at this point you will have traversed all arcs of T' exactly twice (all will be labeled 2), all nodes will be labeled, and that the loop consisting of arcs of capacity $\beta/2$ with the nodes taken in order of their labeling (and then returning to the starting node) is a synthesis of the desired cut tree.

Applying this process to the uniform 5 tree in our example gives (among others) the loop used in Fig. 9 which resulted in exact synthesis.

Proving that this general procedure works is rather tedious so we give the following procedure which is a specialization of the one above and whose properties are more easily verified.[1]

Suppose T' is a uniform requirement tree whose edges all have value β. We may construct a loop λ, through all the nodes of T', such that λ has T' as a cut tree, as follows. Label any node of T' with the number 1. Then repeat the following step until T' is completely labeled: If the last label was n, then label with $n + 1$ any unlabeled node which is then adjacent to node n, if such an unlabeled node exists; if none exists, label with $n + 1$ an unlabeled node adjacent to a labeled node with the largest label possible. When T' is completely labeled, we define λ to go from node 1 to node 2 to node 3, etc., and finally to return to node 1. We let every edge of λ have capacity $\beta/2$.

Proof. To see that T' is indeed a cut tree of λ, consider any edge e of T'. Suppose the two nodes of e are labeled i and j, with $i < j$. Let k be the largest label which occurs in the component of $T' - e$ which contains j. Then the minimal cut whose edges are $j - 1, j$ and $k, k + 1$ corresponds to e. (If $j = 1$, then $j - 1$ means the largest label, and if $k =$ the largest label, then $k + 1$ means 1.)

APPENDIX

The following algorithm is to construct a maximum spanning tree and to check whether a given symmetrical matrix is realizable during the construction of the tree. If one wants only the maximum spanning tree, then the algorithm in [5] or [6] is more efficient. Geometrically, the algorithm is to choose the longest arc (the largest number), and then the next longest arc and so forth (arcs can be chosen only if they form a subtree). This will result in many disconnected subtrees. For each of the subtrees, we check that condition (2) is satisfied. Because the length of the arcs is monotonically decreasing, any arc δ which connects nodes belonging to a single subtree but did not do so until the arc ν was selected should have the same length as ν. At the end, the maximum spanning tree is formed when all the subtrees are connected.

[1] We are grateful to the referee for providing this second procedure together with a simple proof of its properties.

All this boils down to the following simple arithmetical steps in the requirement matrix.

The requirement matrix has a border row on the top and a border row in the leftmost column to indicate the ith row or jth column, as shown in Table A1.

In what follows r_{ij} means a matrix element in a row with border element i and column border element j. The algorithm is then the repetition of the following two steps:

Step 1. Select the largest number in the matrix proper that has not been selected or crossed out. (In the beginning, no number has been selected or crossed out.) Let this number be r_{ij}. Make a check mark in its box. If $p = \min(i, j)$ and $q = \max(i, j)$, change all q's in both borders to p's. For example, if r_{35} is chosen, then change ⑤ in the border row and column into ③.

Step 2. Consider all entries (not yet crossed out or selected) whose border entries are both p. If they are equal to the last entry selected, cross them out and return to step 1. If even one of them is not equal to the last entry selected, the matrix is not realizable.

Step 1 and Step 2 are repeated until $n - 1$ numbers are chosen. If this can be done the matrix is realizable.

Take the following table for example.

TABLE A1

	①	②	③	④	⑤	⑥
①	d	7	5	3	6	4
②		d	6	7	5	3
③			d	3	9 ✓	4
④				d	8	4
⑤					d	5
⑥						d

omitted due to symmetry

Step 1. Select $r_{35} = 9$ and change ⑤ into ③.

Step 2. The only number r_{33} is 9 itself, so no crossing out is required.

Step 1. Select $r_{43} = 8$ and change ④ into ③. The result is shown in the following table.

TABLE A2

	①	②	③	③	③	⑥
①	d	7	5	3	6	4
②		d	6	7	5	3
③			d	⊠	9 ✓	4
③				d	8 ✓	4
③					d	5
⑥						d

Step 2. Check if $3 = 8$?. As $3 \neq 8$, we know that the matrix is not realizable, but to illustrate the algorithm, we shall continue checking. Cross out 3.

Step 1. Select $r_{12} = 7$($r_{22} = 7$ can equally well be chosen). Change ② to ①.

Step 2. Check $r_{11} = 7$?.

Step 1. Select $r_{23} = 7$ and change all ③'s into ① as shown below.

TABLE A3

	①	①	①	①	①	⑥
①	d	7 ✓	5	3	6	4
①		d	6	7 ✓	5	3
①			d	⊠	9 ✓	4
①				d	8 ✓	4
①					d	5
⑥						d

Step 2. Check if $5 = 7$? $6 = 7$? $3 = 7$?, and cross out 5, 6, 3, etc., as shown in Table A4.

TABLE A4

	①	①	①	①	①	⑥
①	d	7 ✓	⊠	⊠	⊠	4
①		d	⊠	7 ✓	⊠	3
①			d	⊠	9 ✓	4
①				d	8 ✓	4
①					d	5
⑥						d

Step 1. Select 5 in the sixth column and change ⑥ into ①.

Step 2. Check if all elements in the sixth column equal 5.

It may be noted that if one rearranges rows and columns such that rows and columns with same labeling are next to each other, then the matrix is in Mayeda's partitioned form.

REFERENCES

1. L. R. FORD, JR. AND D. R. FULKERSON, *Maximal flow through a network*, Canad. J. Math., 8 (1956), pp. 399–404.
2. W. MAYEDA, *Terminal and branch capacity matrices of a communication net*, IRE Transactions on Circuit Theory, Vol. CT-7 (1960), pp. 261–269.
3. R. T. CHIEN, *Synthesis of a communication net*, IBM J. Res. Develop., 3 (1960), pp. 311–320.
4. J. B. KRUSKAL, JR., *On the shortest spanning subtree of a graph and the traveling salesman problem*, Proc. Amer. Math. Soc., 7 (1956), pp. 48–50.
5. R. C. PRIM, *Shortest connection networks and some generalizations*, Bell System Tech. J., 36 (1957), pp. 1389–1401.
6. R. E. GOMORY AND T. C. HU, *An application of generalized linear programming to network flows*, IBM Research Report RC-319, September, 1960.
7. C. BERGE, *Theorie des Graphes*, Dunod, Paris, 1958.
8. O. WING, *Minimal realization of a communication net under uniform cost*, IBM Research Report ES 0026, August, 1960.