AN IMPROVED ALGORITHM FOR MULTI-TERMINAL NETWORK FLOW SYNTHESIS

by

Dan Gusfield

Memorandum No. UCB/ERL M79/76

27 November 1979

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

An Improved Algorithm for Multi-Terminal Network Flow Synthesis

Dan Gusfield
University of California, Berkeley

1. INTRODUCTION

The multi-terminal network flow synthesis problem is one of the few nicely solved problems in the area of network design. It is used widely in courses and texts [1,2,3,4] on network flows and combinatorial optimization, as an example of an elegantly solved combinatorial optimization problem. The solution used in these texts is due to Gomory and Hu [5], and is also cited as an example of a non-direct application of maximum spanning trees. For examples where this problem arises, see also Chien [6]. For NP - hard network design problems see Wong [7], or Garey and Johnson [8].

We present a simpler algorithm, improving the Gomory-Hu method in speed, simplicity of needed data structures, and most important, in the clarity of the underlying combinatorics. We show that the use of the maximum spanning tree in the Gomory-Hu algorithm is both unnecessary and undesirable. We then discuss a secondary objective: minimizing the number of edges in the solution, and the distribution of node degree.

2. PROBLEM SET UP AND MAIN RESULT

Let R be a weighted undirected graph on n nodes and e edges, and for each edge (i,j) in R, let r(i,j) denote its weight. R is called the *flow requirements graph* and r(i,j) is the i,j *flow requirement*. Let r(i,j) = 0 for any i,j pair not an edge in R.

For G an undirected network with n nodes and a flow capacity on each edge, let f(i,j) denote the maximum achievable flow in G between nodes i and j. G is called *feasible* for R if $f(i,j) \geq r(i,j)$ for all node pairs i,j.

Research supported by NSF grants MCS77 - 09906, MCS78 - 07291

Given R, we seek a network G with edge capacities, which is feasible for R, and whose sum of edge capacities is minimum among all networks feasible for R. Any such network is called *optimal* for R. We may further seek an optimal network G* with flow function f*, such that for any other optimal network G and its flow function f, $f^*(i,j) \geqslant f(i,j)$ for all i,j pairs. Such a network is called *uniformly optimal*, and always exists [5].

Main Result

We give a simple algorithm which avoids spanning trees, runs in time Max[e, nlogn], and produces a network G* with the following desirable properties: G* is uniformly optimal; it is planar; no node has degree greater than four; and G* has as few edges as any uniformly optimal network produced by the Gomory - Hu method. This constitutes a strong arguement against the use of the maximum spanning tree.

In the next section, we sketch the Gomory - Hu method (algorithm A) to find an optimal network, and present a simpler algorithm (algorithm B) that produces a planar uniformly optimal network with one node of high degree. In section 4. we compare the two algorithms, and in section 5. we discuss the number of edges in the networks produced, and modifications to algorithm B so that no node has degree greater than four.

3. ALGORITHMS FOR AN OPTIMAL NETWORK

Algorithm A

1) Given the requirements graph R, compute a maximum weight spanning tree T of R.
2) Decompose T into a sum of subtrees, each having edges of equal weight. To do this, define the decomposition of a tree T_i recursively. If w_i is the smallest edge weight in T_i then T_i is decomposed into one copy of T_i with weight

w_i on each edge, plus the decomposition of each subtree of T_i resulting from deleting all edges of weight w_i from T_i, and subtracting w_i from the weights of all the remaining edges.

3) For every tree T_i in the decomposition, create a cycle C_i containing all the nodes of T_i. Set the capacity of every edge in C_i to $w_i/2$, where w_i is the weight of each edge in T_i. Superimpose all of the cycles, merging common edges and summing the capacities. The resulting network is optimal for R.

Figure 1 shows the workings of the algorithm.

Algorithm A produces an optimal network G for R, but in general G will not be uniformly optimal. To find a uniformly optimal network, Gomory and Hu suggest adding the following step at the beginning of algorithm A:

0) For each node i in R, compute $u(i) = \underset{k}{\mathrm{Max}}[r(i,k)]$. For every pair i,j change r(i,j) to Min[u(i),u(j)], making R a complete graph on n nodes.

We now present an alternative algorithm.

Algorithm B

```
1) For each node i, compute $u(i) = \underset{k}{\mathrm{Max}}[r(i,k)]$, and define $u(n+1) = 0$.
2) Sort the u(i) values. Assume $u(i) \geqslant u(i+1)$ for $i = 1,n$.
3) For i = 2 through n repeat the following:
   a. Create edge i,i-1 with capacity $u(i)/2$.
   b. Create edge i,1 with capacity $[u(i) - u(i+1)]/2$,
      provided that the capacity is non-zero.
```

Figure 2 shows the working of algorithm B on the same requirements graph R as in figure 1. Note that the network created by the algorithm is always planar.

We now show the correctness of algorithm B. Given R, let G^* be the network produced by algorithm B, and let f^* be the flow function of G^*.

Lemma 3.1: $f^*(i,j) = \mathrm{Min}[u(i),u(j)]$ for all node pairs i,j.

Proof: Let i,j be two arbitrary nodes, and $u(i) > u(j)$. Consider the path $P_{i,j}$ from i to j along the edges $(k,k+1)$ for $k = i$ through $j-1$. The edge with least capacity on $P_{i,j}$ is $(j-1,j)$, with capacity $u(j)/2$. Therefore, a flow of $u(j)/2$ is possible along the $P_{i,j}$ path.

Now consider $P_{1,i}$, the path with edges $(k,k+1)$ for $k = 1$ through $i - 1$. The edge with least capacity on $P_{1,i}$ is $(i-1,i)$, with capacity $u(i)/2 \geq u(j)/2$. G* is undirected, and so a flow of $u(j)/2$ from node i to node 1 is possible along the reverse of path $P_{1,i}$.

To complete the proof, we claim that a flow of $u(j)/2$ between nodes 1 and j is achievable without using any edge of $P_{1,j}$, the union of $P_{1,i}$ and $P_{i,j}$. The proof is by backward induction on the index j. For $j = n$, the claim is true, since the edge $(1,n)$ has capacity $u(n)/2$. Suppose the claim is true for $j = k+1 < n$, and consider node k. Let $F(k+1)$ be the flow of $u(k+1)/2$ from 1 to k+1 which avoids edges of $P_{1,k+1}$. By definition, $F(k+1)$ doesn't use edges $(k+1,k)$ or $(k,k-1)$, and so $F(k+1)$ also doesn't use edge $(1,k)$. Edge $(1,k)$ has capacity $u(k)/2 - u(k+1)/2$, and edge $(k+1,k)$ has capacity $u(k+1)/2$, for a total capacity of $u(k)/2$. Then to send $u(k)/2$ from 1 to k avoiding $P_{1,k}$, send $u(k+1)/2$ from k+1 to k along edge $(k+1,k)$, and send the rest along edge $(1,k)$. The flow of $u(k+1)/2$ to node k+1 is sent via $F(k+1)$, and the proof is complete.

Theorem 3.1: G* is uniformly optimal for R.

Proof: By the lemma, G* is clearly feasible for R. To show optimality, note that the total capacity of the edges incident to any node i is $u(i)$, which is the minimum capacity possible in any feasible network. Now suppose there is an optimal network G with flow function f, such that $f(i,j) > f^*(i,j)$, for some node pair i,j. Then $f(i,j) > \mathrm{Min}[u(i),u(j)]$, and if $u(i) > u(j)$, node j must be incident in G to edges with total capacity exceeding $u(j)$. Therefore, G can't be optimal, and G* is uniformly optimal.

4. COMPARING THE ALGORITHMS

Algorithm A takes time of order Min[e loglogn, n^2] in step 1) to compute a maximum spanning tree of R. Decomposition and synthesis in steps 2) and 3) can take $O(n^2)$ time.

Algorithm B requires time O(e) to find the u(i) in step 1), O(nlogn) to sort in step 2), and O(n) time to construct G* in step 3).

For a dense graph R, the running times of the two algorithms are comparable, but algorithm B still improves algorithm A in the constants of proportionality, and in the simplicity of the data structures needed. Finding the maximum weight edge incident with each node is simpler than finding a maximum weight spanning tree, which must also include all such edges; sorting is simpler than decomposition, which must also find successive minimums; and the synthesis step of algorithm B is clearly simpler to implement than the synthesis step of A.

Algorithm B is faster and simpler than algorithm A primarily because it avoids constructing or decomposing a maximum spanning tree, and because it produces a uniformly optimal network without revising the original requirements. Thus the use of the maximum spanning tree is undesirable, as well as unnecessary. In fact, it is the maximum spanning tree T which causes algorithm A to produce networks which are generally non - uniformly optimal: If the path between nodes i and j in T contains an edge of weight less than Min[u(i),u(j)], then f(i,j) will be less than f*(i,j) in the resulting network. To guarantee uniform optimality, step 0) is added to force out such low weight edges. Algorithm B shows that the spanning tree can be avoided, and with it the need to revise the requirements.

5. AVERAGE NODE DEGREE AND DISTRIBUTION

We now consider algorithm A modified to find a uniformly optimal network, i.e. with step 0) included. We consider the average node degree, or number of edges, and show that algorithm B produces networks which have as few edges as the networks produced by algorithm A. We then show that algorithm B can be modified so that no node in the network created has degree greater than four.

Given R, let G' and G* be the uniformly optimal networks produced by algorithms A and B. We examine first the number of edges in G*.

For each node i, let u(i) be defined as before, and call u(i) the *weight of node* i. Step 3b) of algorithm B produces the edge (1,i) if and only if u(i) > u(i+1), and so produces t edges in total, where t is the number of distinct node weights. Step 3a) produces n - 1 edges, and so G* contains no more than n - 1 + t edges. However, u(1) = u(2), so the edge (1,2) is counted twice if u(2) > u(3), hence:

Lemma 5.1: G* contains n - 1 + t edges if u(2) = u(3), and n - 2 + t edges if u(2) > u(3), where t is the number of distinct node weights.

Now let R' be the requirements graph modified by step 0) of algorithm A, and let T' be any maximum spanning tree of R'. To establish the number of edges in G' we first examine the structure of T'.

Lemma 5.2: Let x be any edge in T'. If x has weight w_x, then the removal of x from T' creates two connected components, at most one of which contains an edge of weight greater than w_x.

Proof: The lemma is trivially true if one of the endpoints of x is a leaf of T', so suppose this is not the case. Let G_y and G_z be the two components of T' - x, with edge y of weight $w_y > w_x$ in G_y, and edge z of weight $w_z > w_x$ in G_x. By the definition of R', w_y = u(i) for some node i in G_y, and w_z = u(j) for some node j in G_z. Then R' contains the edge (i,j) across the G_y, G_z cut, and (i,j) has weight greater than w_x, hence T' can't be a maximum weight spanning tree of R'.

Theorem 5.1: G' contains at least as many edges as G*.

Proof: Let $w_1 > w_2 > ... > w_t$ be the t distinct node weights of R, i.e. the t distinct values of u(i) for i = 1 through n. From the definition of R', the only edge weights in T' are w_1 through w_t, and all edges incident with any node i in T' have weight less than or equal to u(i). Further, since T' is a maximum spanning tree, every node i is incident in T' with at least one edge of weight u(i). It follows then from Lemma 5.2, that for any $k \leqslant t$, the deletion from T' of all edges of weight w_k or less leaves one connected subtree containing all nodes with node weights greater than w_k, and no nodes with weight w_k or less.

We now examine the edges generated by the synthesis step 3), ignoring the capacities assigned. We claim that G' is the superposition of t cycles, C_1 through C_t. C_t connects all the n nodes of T', and for $k < t$, C_k contains all nodes of weight w_k or more, and no nodes of weight w_{k+1} or less. To see this, recall that the decomposition step 2) of algorithm A generates a sequence of subtrees of T', by beginning with T' itself, and successively deleting all edges of weight w_t down to w_1. Step 3) creates a cycle through the nodes of every new subtree generated in this way, and the claim follows from the structure of these subtrees, which was established above.

We can now count the number of edges of G'. For k from 1 through k, let N_k be the number of nodes of weight w_k. For $k \geqslant 2$, cycle C_k contains all nodes of weight w_k, and at least one node of greater weight. Therefore, at least $N_k + 1$ edges of C_k are incident with some node of weight w_k. None of these $N_k + 1$ edges can appear in any other cycle C_j, for $j < k$, and so the cycles C_2 through C_t must contain at least $(n - N_1) + (t - 1)$ distinct edges. Cycle C_1 contains N_1 edges, and so G' contains at least $n - 1 + t$ edges if $N_1 > 2$, and $n - 2 + t$ edges if $N_1 = 2$, and the theorem is proven.

Corollary: G' contains the same number of edges as G* if and only if the nodes of weight w_k form a single subpath in C_k, for all k from 1 to t.

Note that without step 0), lemma 5.2 does not hold, and algorithm A may produce a non-uniformly optimal network G with fewer edges than G*. For example see G and G* of figures 1 and 2. Note also that there are uniformly optimal networks with fewer edges than G*. See figure 3. Such networks are, of course, not produced by either algorithm A or B, and it is an open question whether there exist fast algorithms to minimize the number of edges in a uniformly optimal network.

Distribution of node degrees

Algorithm B creates a network G* in which node 1 has high degree: t or t - 1. For many purposes, this is very undesirable. However, algorithm B can be modified to produce, with equal speed, a planar, uniformly optimal network which has as few edges as G*, and in which no node has degree greater than four. In general, the number of nodes of degree four equals the number of indices k such that $N_k = 1$, so if $N_k > 1$ for all k, no node will have degree greater than three.

Instead of presenting the modifications formally, we show in figure 4 such a network with eight nodes, four distinct node weights, and two nodes of each node weight. The nodes are labeled with their weights, $D > C > B > A$.

This example is easily extended to the general case. If there are more than two nodes of a given weight other than D, say B, insert the new nodes into either of the B to C edges, creating new edges of capacity B/2. For more than two D nodes, split the edge between the two D nodes into two parallel edges, one with capacity D/2, and the other with capacity (D - C)/2. Insert the new D nodes into the edge of capacity D/2. If there is only one node of a given weight, say B again, then merge either one of the B nodes with the unique C node that it is incident with, creating a C node of degree four. Figure 5 shows the network of figure 4 with unique node weights for all nodes except D. Extending the example for more distinct node weights is immediate.

Acknowledgment

Many thanks to David Lichtenstein for his helpful comments.

References

[1] E.L Lawler, Combinatorial Optimization: Networks and Matroids, Holt, Rinehart and Winston, New York (1976)

[2] L.R. Ford and D.R. Fulkerson, Flows in Networks, Princeton University Press, Princeton, New Jersey, (1962)

[3] H. Frank and I. Frisch, Communication, Transportation and Flow Networks, Addison - Wesley, Reading Mass. (1972)

[4] T.C. Hu, Integer Programming and Network Flows, Addison - Wesley, Reading Mass. (1969)

[5] R.E. Gomory and T.C. Hu, "Multi - Terminal Network Flows", SIAM Journal on Applied Mathematics, 9 (1961) 551-570.

[6] R.T. Chien, "Synthesis of a Communication Net", I.B.M. Journal, July 1960.

[7] R.T. Wong,"A Survey of Network Design Problems", Operations Research Center working paper OR 080-78, August 1978, Massachusetts Institute of Technology.

[8] M. Garey and D. Johnson, Computers and Intractability: A Guide to the Theory of NP - Completeness, W.H. Freeman and Co. (1979)

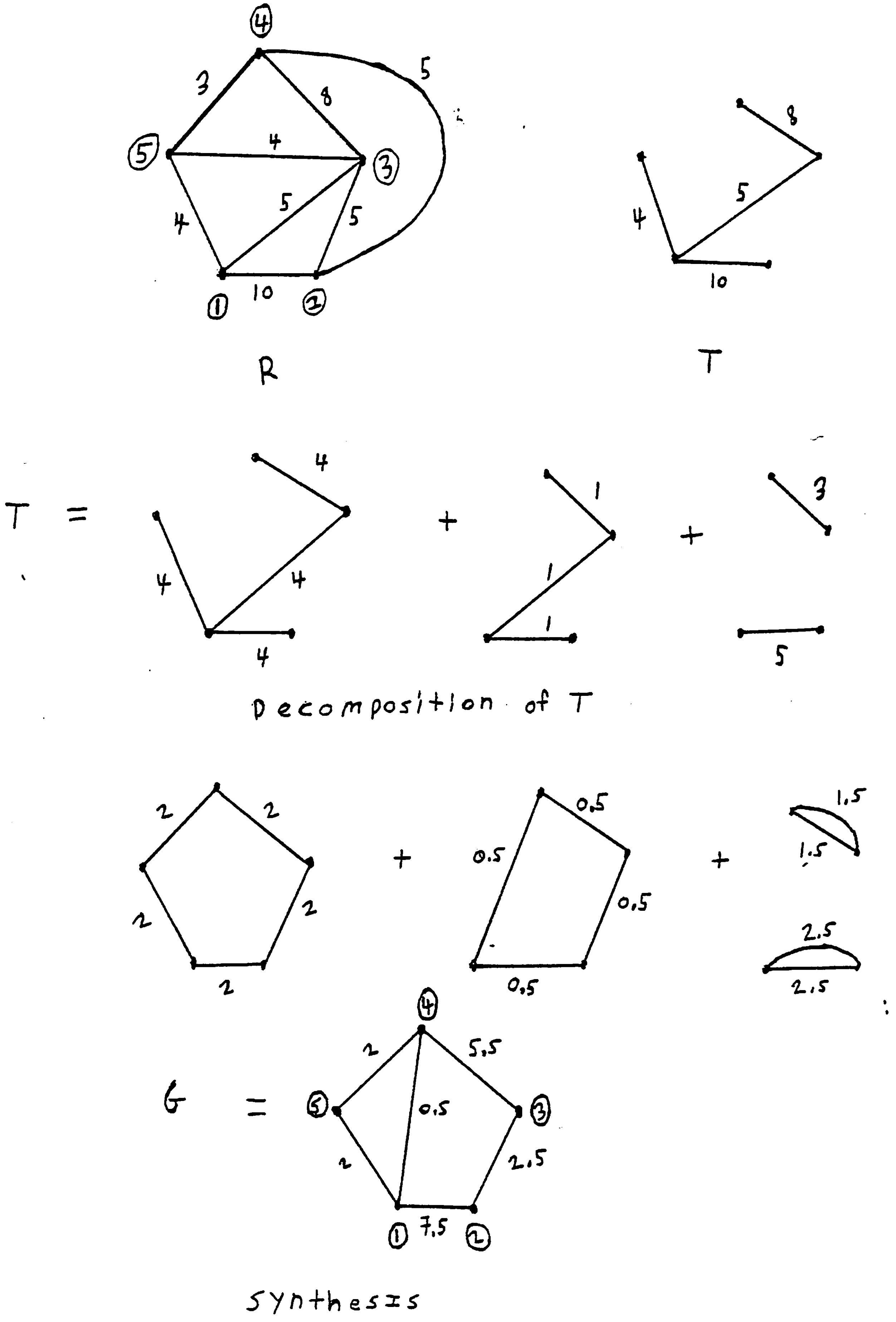
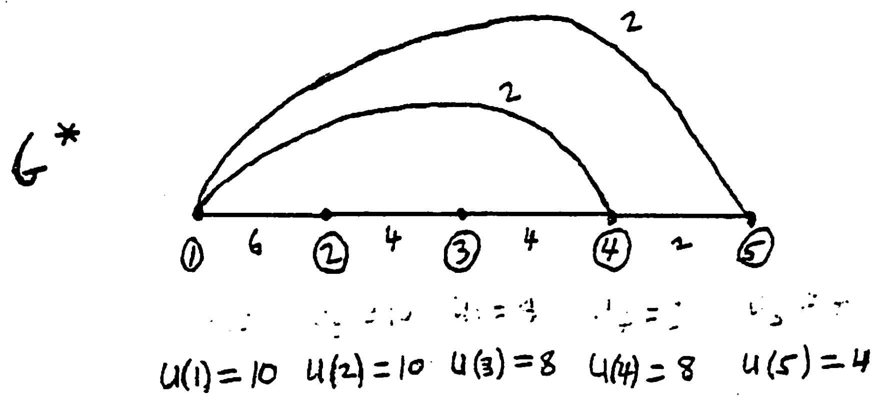
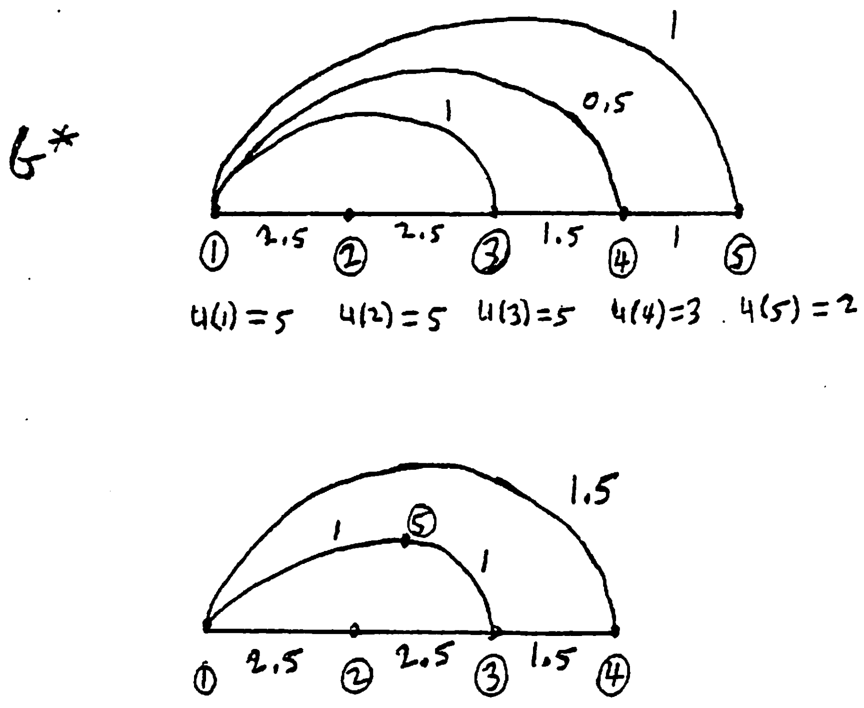


Figure 1.



Note that G in figure 1 is not uniformly optimal.
For example, f(1,4) = 5, but f*(1,4) = 8

Figure 2.



Equivalent network with one less edge.
Figure 3.

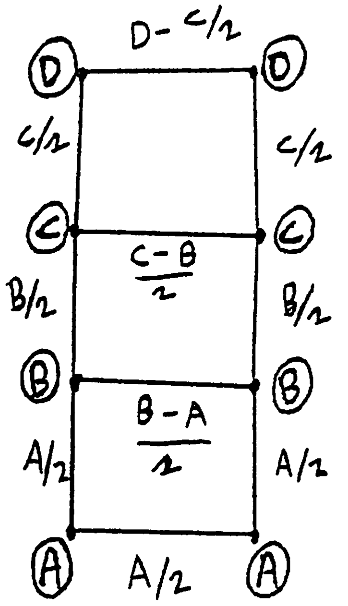


Figure 4

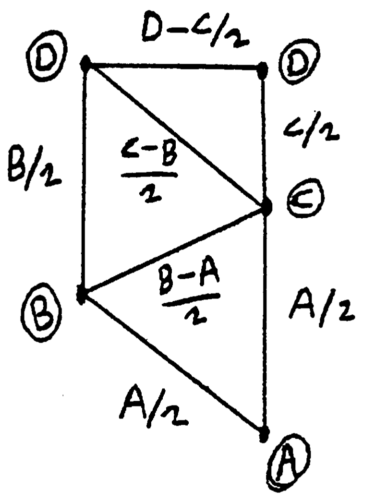


Figure 5.