ANCESTOR TREE FOR ARBITRARY MULTI-TERMINAL CUT FUNCTIONS

C.K. CHENG and T.C. HU
Department of Computer Science and Engineering, University of California, San Diego, La Jolla, CA 92093, USA

Abstract

In many applications, a function is defined on the cuts of a network. In the max-flow min-cut theorem, the function on a cut is simply the sum of all capacities of edges across the cut, and we want the minimum value of a cut separating a given pair of nodes. To find the minimum cuts separating $\binom{n}{2}$ pairs of nodes, we only need $n-1$ computations to construct the cut-tree. In general, we can define arbitrary values associated with all cuts in a network, and assume that there is a routine which gives the minimum cut separating a pair of nodes. To find the minimum cuts separating $\binom{n}{2}$ pairs of nodes, we also only need $n-1$ routine calls to construct a binary tree which gives all $\binom{n}{2}$ minimum partitions. The binary tree is analogous to the cut-tree of Gomory and Hu.

1. Introduction

Given an undirected network $G=(V,E)$, where V is the set of nodes $\{v_1, v_2, \ldots, v_n\}$ and E is the set of arcs in the network: arc e_{ij} has capacity c_{ij} and connects nodes i and j. A partition of the node set into a subset X and its complement $\overline{X}$ is called a *cut* and is denoted by $(X,\overline{X})$. If $s \in X$ and $t \in \overline{X}$, then the cut $(X,\overline{X})$ *separates* nodes s and t.

We can assign a value to an arbitrary cut $(X,\overline{X})$. In the max-flow min-cut theorem, the value of a cut $(X,\overline{X})$ is simply the sum

$$\sum_{i\in X,\, j\in \overline{X}} c_{ij}\,,$$

denoted by $C(X,\overline{X})$, and we wish to find the cut $(X,\overline{X})$ separating s and t which has the minimum value, i.e.

$$F_{st} = \min_X C(X,\overline{X}) \quad \text{with } s\in X \text{ and } t\in \overline{X}. \tag{1}$$

We can view this as minimization of a function.

In other applications [1,6,9–11], we may want to define this function F_{st} as, for example,

$$\min_X \frac{C(X,\overline{X})}{|X|\cdot|\overline{X}|} \quad \text{with } s\in X \text{ and } t\in \overline{X}, \tag{2}$$

where $|X|$ is the cardinality or the size of the set X. The minimum partition based on (2) will divide the network more evenly than the minimum partition based on (1). In a VLSI circuit layout, we prefer to have a minimum partition with the number of modules on both sides approximately the same. In other applications, we may need entirely different criteria. Thus, we allow arbitrary values to be associated with any partition and consider the computation for finding the minimum value partition separating a given pair of nodes as a routine call.

In general, we denote the minimum function by $F_{ij}(X,\overline{X})$ with the understanding that $i\in X$ and $j\in\overline{X}$, or we simply use F_{ij} or $F(X,\overline{X})$.

Assume that we wish to determine the $F_{ij}(X,\overline{X})$ values for all pairs of nodes i and j. Our main result is that to find the $\binom{n}{2}$ F_{ij}, we need only $n-1$ computations of the function F.

To see the difference between arbitrary cut functions and the original minimum cut capacity functions, we first introduce the definition of crossing of cuts.

Two cuts $(X,\overline{X})$ and $(Y,\overline{Y})$ are said to cross each other if each of the following four sets contains at least one node:

$$X\cap Y, \quad X\cap\overline{Y},$$
$$\overline{X}\cap Y, \quad \overline{X}\cap\overline{Y}.$$

To find the maximum flows between $\binom{n}{2}$ pairs of nodes in a network, we need $n-1$ computations where each computation gives a minimum cut. Also, there exists a set of $n-1$ minimum cuts which forms a cut-tree [4]. In other words, there exists a set of minimum cuts which do not cross each other.

Here, for arbitrary functions $F_{ij}(X,\overline{X})$, the cuts $(X,\overline{X})$ which yield the values for F_{ij} may cross each other (for example, the function defined in (2)). However, we still need only $n-1$ computations by constructing a binary tree called the ancestor tree.

In section 2, we prove the existence of the ancestor tree. In section 3, we give a numerical example. In section 4, we give the detailed algorithm and its proof. In section 5, we give some additional remarks, including a very simple proof for the Gusfield construction of the Gomory–Hu cut-tree.

2. Ancestor tree

Let us assume that we have done $\binom{n}{2}$ computations and have found values of F_{ij} between all pairs of nodes in the network. Assume $F_{st}(X,\overline{X})$ is the smallest value among the $\binom{n}{2}$ values. Then, for any pair of nodes i and j with $i\in X$ and $j\in\overline{X}$, the cut $(X,\overline{X})$ serves to separate i and j as well; hence, we have

$$F_{ij}(X,\overline{X}) = F_{st}(X,\overline{X}).$$

Let $F_{ab}(Y, \overline{Y})$ be the smallest value of F where both a and b are in X. Then, for any two nodes i and j, where $i \in Y \cap X$ and $j \in \overline{Y} \cap X$, we must have

$$F_{ij} = F_{ab}(Y, \overline{Y}).$$

Similarly, let $F_{cd}(Z, \overline{Z})$ have the smallest value with both c and d in $\overline{X}$. Then, for any two nodes i and j, where $i \in Z \cap \overline{X}$ and $j \in \overline{Z} \cap \overline{X}$, we must have

$$F_{ij} = F_{cd}(Z, \overline{Z}).$$

Note that the nodes of the network have been partitioned into four subsets, namely:

$$Y \cap X,\ \overline{Y} \cap X,\ Z \cap \overline{X},\ \overline{Z} \cap \overline{X};$$

then we know F_{ij} between any two nodes as long as i, j do *not* belong to the same subset.

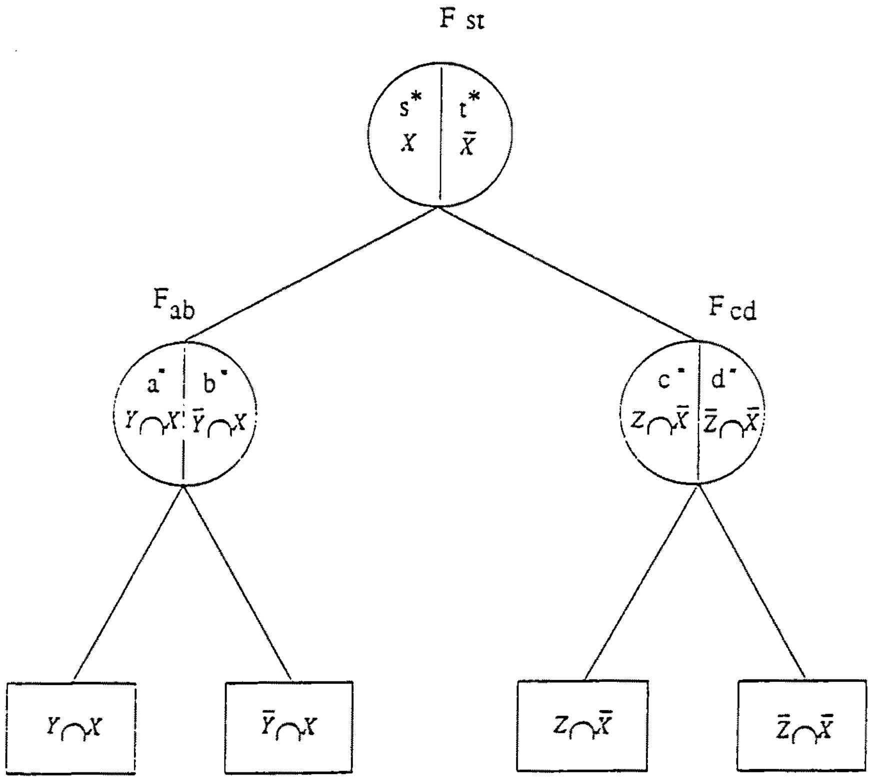

Fig. 1.

We can record the results of the three computations as a binary tree with three internal vertices, as shown in fig. 1, with $F_{st}(X, \overline{X})$ as the root, $F_{ab}(Y, \overline{Y})$ and

$F_{cd}(Z,\overline{Z})$ as the two sons, and the four subsets $Y \cap X$, $\overline{Y} \cap X$, $Z \cap \overline{X}$, $\overline{Z} \cap \overline{X}$ as the four leaves.

The process of partitioning subsets can be continued until each leaf of the tree contains only one node. Then, the binary tree has n leaves and $n-1$ internal vertices; each vertex is a computation of the defined function for a given pair of starred nodes. For any two nodes i and j, the lowest common ancestor of their respective leaves gives the partition as well as the value for F_{ij}.

This binary tree is then the ancestor tree. Of course, we have proved the existence of the ancestor tree by selecting $n-1$ values from the results of the $\binom{n}{2}$ computations.

In section 4, we will prove that we can construct the ancestor tree by performing only $n-1$ computations.

3. Numerical example

Given an n-node network, there are $2^{n-1}-1$ possible cuts. For each of these cuts, we can arbitrarily assign a value. For a given pair of nodes i and j, there are 2^{n-2} cuts separating i and j, and we use $F_{ij}(X,\overline{X})$ to denote the minimum value among the 2^{n-2} cuts. The algorithm for finding F_{ij} for a given pair i and j could be very easy or tedious. We simply consider the algorithm as a routine call. Our main result is to show that $n-1$ routine calls are sufficient for finding $\binom{n}{2}$ F_{ij}.

Consider the network in fig. 2 with arc capacities as shown. To fix the idea, let us assume that we want to find for all pairs of nodes i and j

$$F_{ij} = \min \frac{C(X,\overline{X})}{|X| \cdot |\overline{X}|} \quad \text{with } i \in X \text{ and } j \in \overline{X}.$$

Assume that we first choose a and b, and find $F_{ab}(X,\overline{X}) = 15/9$. We show the result in fig. 3. Since the computation is performed for a and b, we call a and b the *seeded* nodes or simply *seeds*, and c, d, e, f are *unseeded* nodes. (*Seeded* nodes are denoted by stars in the figures.)

We then select a leaf containing one or more unseeded nodes to do a computation between the seeded node and an unseeded node, say between b^* and c. (c then becomes seeded.) The result is shown in fig. 4. Note that since $F_{bc} < F_{ab}$, the internal vertex $F_{bc}(Y,\overline{Y})$ is put as the root and F_{ab} as its left son. The names of nodes in the leaves have been updated to reflect the three subsets

$$X \cap Y, \quad \overline{X} \cap Y, \quad \overline{Y}.$$

In general, let (P,Q) be the new internal vertex just created, with $p^* \in P$ and $q^* \in Q$, and $F_{ab}(A,B)$ be the highest ancestor of (P,Q) which satisfies $F_{ab}(A,B) > F_{pq}(P,Q)$. Then we put (P,Q) in the position of (A,B) and attach (A,B) together with its subtree as a son of (P,Q). In the subtree rooted at (A,B), there is a leaf

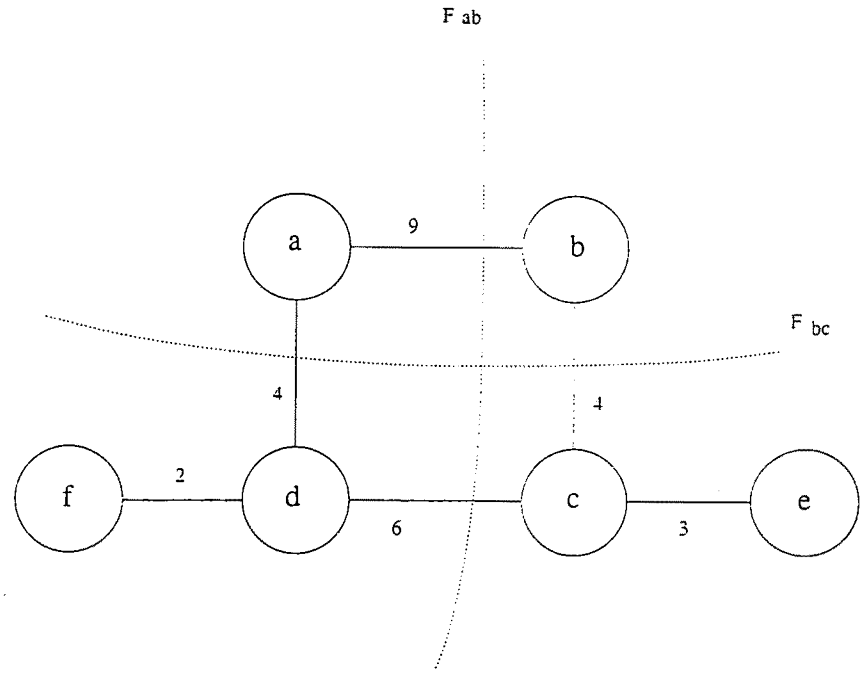

Fig. 2.

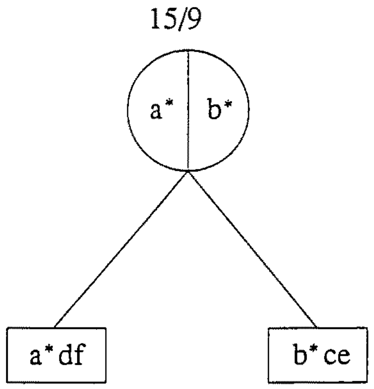

Fig. 3.

which contains the node p or q, say p. Then we attach the subtree on the P side of the vertex (P, Q). All unseeded nodes, in the leaves (of the subtree rooted at (A, B)) which belong to Q, are now attached as a leaf on the Q side of (P, Q).

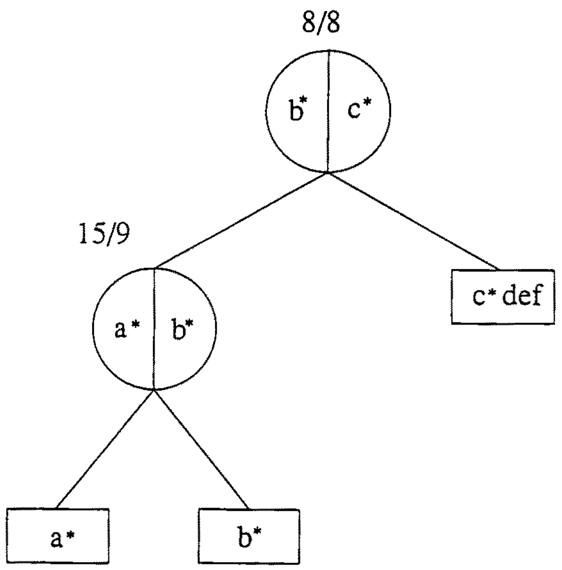

Fig. 4.

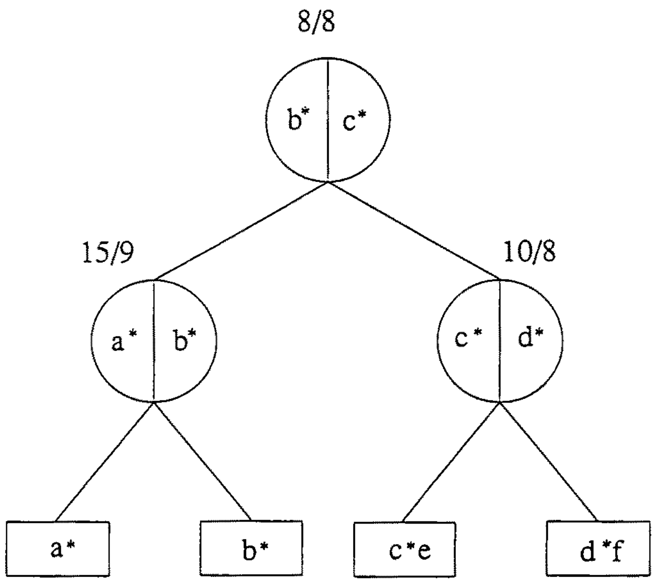

Fig. 5.

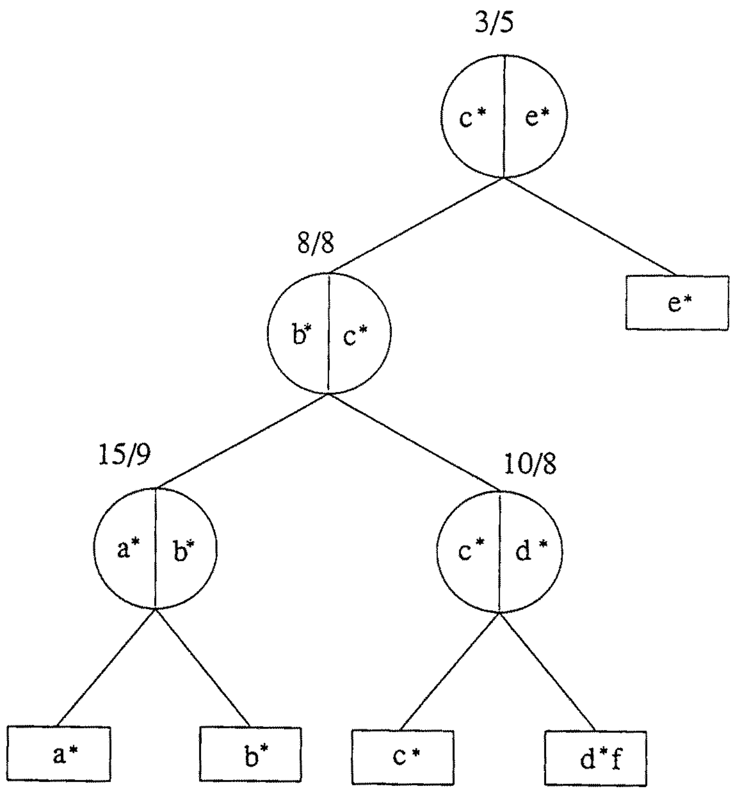
3/5
c* | e*
8/8
b* | c*
e*
15/9
a* | b*
10/8
c* | d*
a*
b*
c*
d*f

Fig. 6.

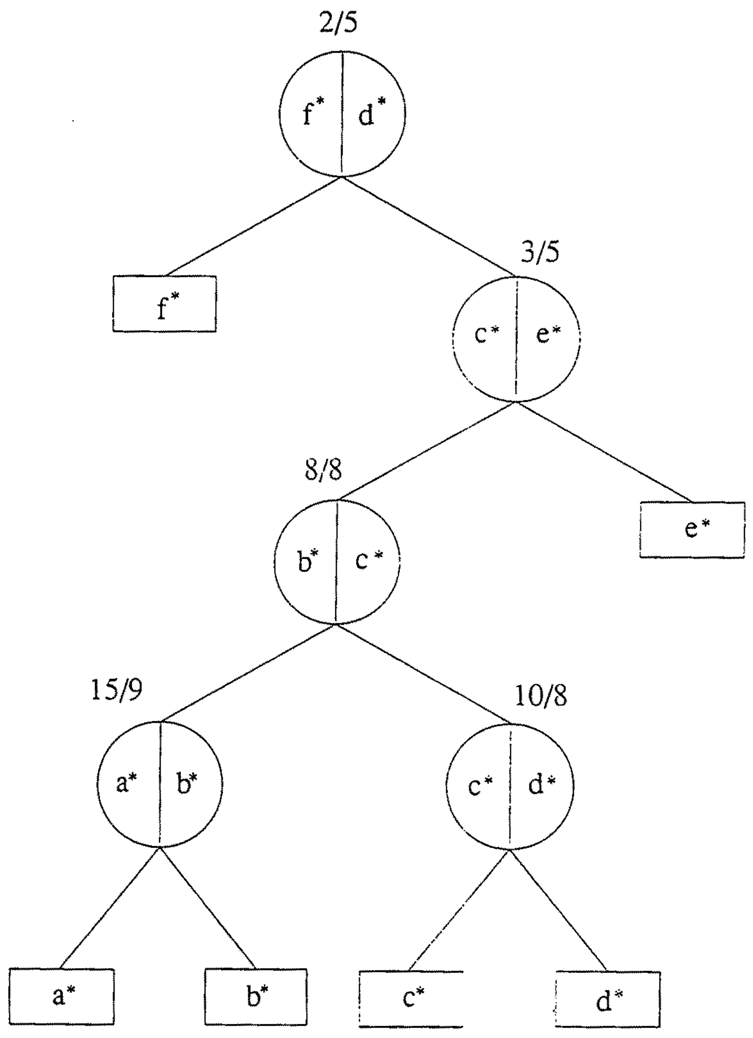
2/5
f* | d*
f*
3/5
c* | e*
8/8
b* | c*
e*
15/9
a* | b*
10/8
c* | d*
a*
b*
c*
d*

Fig. 7.

The successive computations are shown in figs. 5, 6, and 7. The process stops when every leaf contains only one seeded node.

4. The algorithm and its proof

Let T_i denote the ancestor tree with i leaves. We successively build T_i $(i = 1, 2, \dots, n)$.

Initially, the tree T_i consists of one leaf with the names of all nodes in the leaf. Arbitrarily set one node to be the seed.

ANCESTOR TREE ALGORITHM

Select a leaf of T_i which contains more than one node and do a computation between the seeded node and an unseeded node in the leaf. This creates a new internal vertex (P, Q) with $p^* \in P$ and $q^* \in Q$ (the unseeded node is now seeded).

(a) If the newly created vertex F_{pq} has value larger than its father, then F_{pq} remains in its position in T_i and we attach two leaves to F_{pq} to reflect its partitions. One leaf contains the seed p^* and the other leaf contains the seed q^*.

(b) If F_{pq} has its value less than its father, then we find the lowest ancestor F_{rs} and the highest ancestor F_{ab} which satisfy $F_{rs} \le F_{pq} < F_{ab}$. We put F_{pq} in the previous position of F_{ab} and attach F_{ab} (with its subtree) as a son of F_{pq}. In the subtree with root F_{ab}, there is a leaf which contains the name p or q, say p. Then we attach the subtree on the P side of (P, Q). All unseeded nodes in the subtree which belong to the Q side of the partition are now attached as the leaf on the Q side of (P, Q).

Repeat until there is only one seeded node per leaf in the ancestor tree.

Before we prove the ancestor tree, we shall prove some theorems and lemmas about the arbitrary functions F in a network.

THEOREM 1

For any three nodes i, j, and $k \in V$, we have

$$F_{ik} \ge \min(F_{ij}, F_{jk}). \tag{3}$$

Proof

Let F_{ik} have the cut $(X,\bar{X})$, where $i \in X$, $k \in \bar{X}$. Then, j either belongs to X or to $\bar{X}$. If $j \in X$, then the cut $(X,\bar{X})$ serves as a partition separating j and k and we have $F_{ik} \ge F_{jk}$. If $j \in \bar{X}$, then the cut $(X,\bar{X})$ serves as a partition separating i and j, and we have $F_{ik} \ge F_{ij}$.

In either case, we have $F_{ik} \ge \min(F_{ij}, F_{jk})$. □

THEOREM 2

Let $a, b, \ldots, y, z$ be a sequence of nodes in a network. Then,

$$F_{az} \geq \min(F_{ab}, F_{bc}, \ldots, F_{yz}). \tag{4}$$

Proof

By induction on theorem 1. □

LEMMA 1

For any three values F_{ij}, F_{jk}, and F_{ik} between three nodes, two values must be equal and the third value is either greater or the same.

Proof

Putting the smallest value among the three values on the left-hand side of (3) will contradict (3). □

Since we know that there are only *two* distinct values among three values, F_{ij}, F_{jk}, and F_{ik}, we may be able to find the two distinct values by *two* computations. If we arbitrarily compute F_{ij} and F_{jk} first and F_{ik} happens to be the largest value, then we need three computations. The following lemma shows how to avoid this situation.

LEMMA 2

If we compute $F_{ij}(X, \overline{X})$ first and find that $j, k \in \overline{X}$, then

$$F_{ik} = \min(F_{ij}, F_{jk}).$$

Proof

We know that

$$F_{ik} \geq \min(F_{ij}, F_{jk}),$$

so if $F_{ik} \neq \min(F_{ij}, F_{jk})$, then the only possibility is that

$$F_{ik} > \min(F_{ij}, F_{jk}).$$

By assumption $j, k \in \overline{X}$, so $(X, \overline{X})$ serves to separate i and k and $F_{ik} \leq F_{ij}$. This is a contradiction. □

Lemma 2 is for three nodes. The generalization of lemma 2 to four or more nodes is not straightforward, as seen in the following example.

Example 1

Let a, b, c, d be four nodes and suppose that we first compute

$F_{ab}(X, \bar{X})$, and find that b, c belong to $\bar{X}$;

then we compute

$F_{bc}(Y, \bar{Y})$, and find that c, d belong to $\bar{Y}$,

and then compute F_{cd}. Then it seems that the generalization of lemma 2 would suggest

$$F_{ad} = \min(F_{ab}, F_{bc}, F_{cd}). \tag{5}$$

Unfortunately, (5) is not true, as seen from the following example.

Example 2

Using definition (2), we find the cut function in fig. 8.

$F_{ab}(X, \bar{X}) < F_{bc}(Y, \bar{Y})$ and $F_{cd} = F(\bar{X}, X)$,

with $X = \{a, d\}$, $Y = \{a, b\}$, where $F_{ad}(Y, \bar{Y}) > \min(F_{ab}, F_{bc}, F_{cd})$.

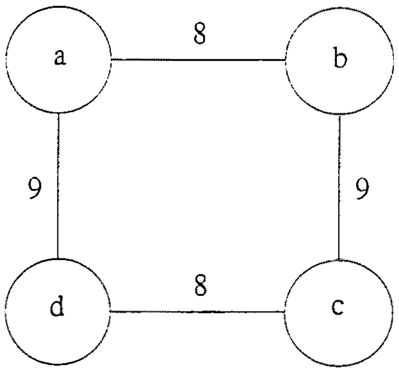

Fig. 8.

The reason for the counterexample to (5) is that $(X, \bar{X})$ and $(Y, \bar{Y})$ cross each other, and the three nodes a, c, d do not satisfy the conditions of lemma 2.

THEOREM 3

Let $a, b, \ldots, y, z$ be any sequence of nodes in a network with

$$\min(F_{ab}, F_{bc}, \ldots, F_{yz}) = F(X, \overline{X}), \qquad a \in X, \; z \in \overline{X};$$

then,

$$F_{az} = F(X, \overline{X}) = \min(F_{ab}, F_{bc}, \ldots, F_{yz}). \tag{6}$$

Proof

By assumption $(X, \overline{X})$ separates a and z, so we have

$$F_{az} \leq F(X, \overline{X}). \tag{7}$$

From theorem 2 and the assumption, we have

$$F_{az} \geq \min(F_{ab}, F_{bc}, \ldots, F_{yz}) = F(X, \overline{X}). \tag{8}$$

The inequalities (7) and (8) imply

$$F_{az} = F(X, \overline{X}).$$ □

Note that theorem 3 is, in a sense, a generalization of lemma 2.

Proof

Now we prove the algorithm of the ancestor tree by induction on the number of internal vertices in the tree.

In the first stage of construction, we have the internal vertex $F_{ab}\,(A, B)$ with seed $a \in A$ and seed $b \in B$. We have the following two properties:

(i) For any pairs of nodes i and j *not* in the same leaf, F_{ij} is less than or equal to the value of the lowest common ancestor of i and j. (Note that i and j could be seeds or unseeded nodes, and there is always one seed per leaf.)

(ii) Let i and j be two seeds and F_{rs} be their lowest common ancestor. Then there exists a sequence of seeds $i, a, b, \ldots, d, j$ with every adjacent pair of seeds constituting an internal vertex in the subtree rooted at F_{rs}, for which

$$F_{ij} \geq \min(F_{ia}, F_{ab}, \ldots, F_{dj}) = F_{rs}.$$

We shall prove that properties (i) and (ii) are maintained during the successive stages of constructing the ancestor tree. Assume that properties (i) and (ii) are true in the kth stage of construction of the ancestor tree, with $k + 1$ leaves. Now, we pick a leaf containing a seed p and an unseeded node q and create $F_{pq}\,(P, Q)$.

Let F_{rs} be the lowest ancestor and F_{ab} the highest ancestor of the leaf satisfying

$F_{rs}(R, S) \le F_{pq}(P, Q) < F_{ab}(A, B)$.

Without loss of generality, we shall assume that the tree rooted at $F_{rs}(R, S)$ has its left subtree containing nodes in R and $F_{ab}(A, B)$ is the root of the right subtree before F_{pq} is created. Also, F_{ab} becomes the left subtree of F_{pq}, as shown in fig. 9.

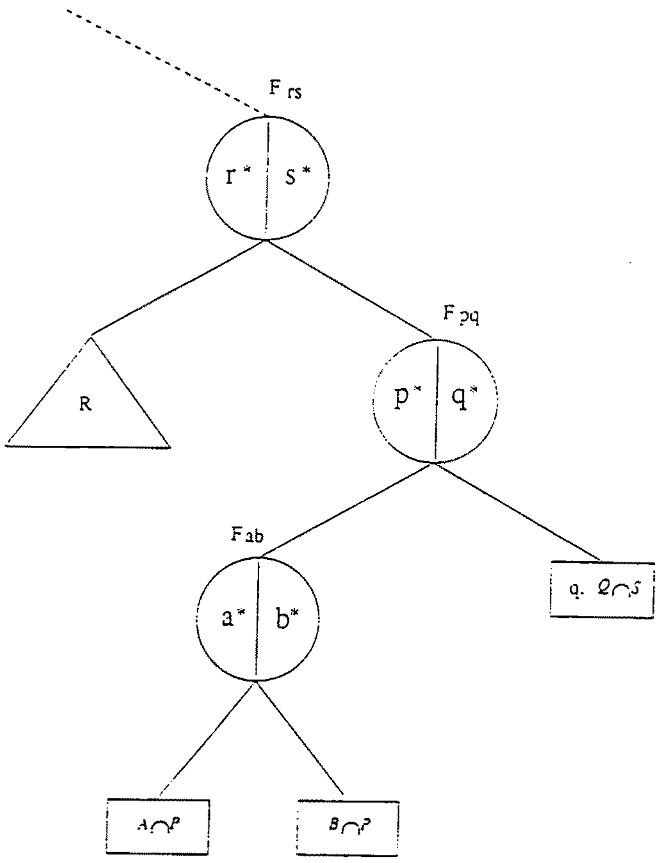

Fig. 9.

For any pair of nodes i and j with $i \in P \cap S$ and $j \in Q \cap S$, we have $F_{ij} \le F_{pq}$ since (P, Q) provides the cut of the subset S. For any pair of nodes i and j with $i \in R$ and $j \in S$, $F_{ij} \le F_{rs}$ since the new partition (P, Q) does not replace (R, S) as the lowest common ancestor. So, the induction hypothesis (i) still holds.

For the two seeds p and q, $F_{pq} = F_{pq}(P, Q)$ by the routine call and every pair of seeds in an internal vertex is correct by the same reason. Since the algorithm always picks a seed and an unseeded node in the same leaf, a spanning tree connecting seeds is created if every computation is represented by a link connecting two seeds. This means that for any path connecting i and j in the spanning tree, property (ii) holds.

When the algorithm stops, there is only one seed per leaf. For any two seeds i and j, we have

$$F_{ij} \leq \min(F_{i1}, F_{12}, \ldots, F_{kj}),$$

by hypothesis (i), and

$$F_{ij} \geq \min(F_{i1}, F_{12}, \ldots, F_{kj}),$$

by hypothesis (ii).

Thus, we have $F_{ij} = \min(F_{i1}, F_{12}, \ldots, F_{kj})$, where F_{ij} is the lowest common ancestor of i and j.

For every internal vertex, we need to keep the value, the associated partition and the names of the two seeds. Thus, we need $O(n^2)$ space. We have not shown the nodes in the associated partitions in any of the figures.

5. General remarks

Note that the proofs of theorems 1 and 2 and lemma 1 are exactly the same as that of Gomory and Hu [4]. However, in the original construction [4] of the cut-tree, the algorithm is to choose any two nodes in a supernode and do a maximum flow computation. Here, we restrict our selection to a seed and an unseeded node. The construction by Gusfield [5] also has this restriction. Having proved the ancestor tree, which does not need the property of non-crossing of cuts, we can more easily see why we obtain stronger results in the cut-tree for MAX flows. In the construction by Gusfield [5], he first constructed a star tree of n nodes, with node 1 at the center of the star tree, as shown in fig. 10. This is the initial configuration T_1 of cut-tree T. This tree T_1 will be successively modified into $T_2, T_3, \ldots, T_n = T$, where each modification requires one maximum flow computation. We call two nodes connected by a link of the tree T_j neighbors. Thus, in T_1, every node j has 1 as the neighbor. In T_1, no value is associated with any link, and we say every node j has an unlabeled neighbor 1. Later, links will have associated values, and we shall call the connected nodes labeled neighbors. In fig. 10, we declare 1 as a seeded node.

In the tree T_{j-1}, do a maximum flow computation between node j and its unlabeled neighbor i (j becomes a seeded node). The min cut value $C(X,\bar{X})$ is now associated with the link l_{ij} ($i \in X$, $j \in \bar{X}$). All neighbors of i that are in $\bar{X}$ become neighbors of j. (Note that all unlabeled neighbors of a seeded node have degree 1 in the tree.) The construction stops after all nodes become seeds.

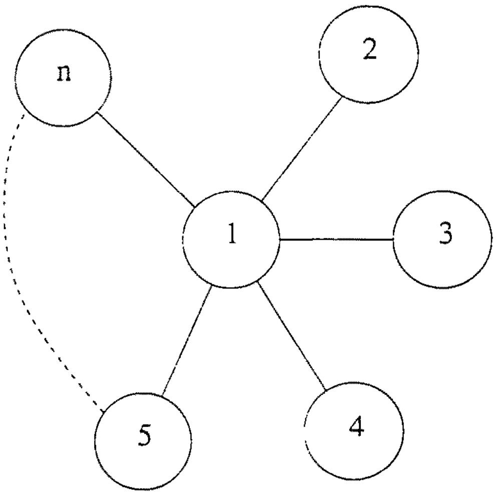

Fig. 10.

Note that the Gusfield construction requires $n - 1$ routine calls to a maximum flow minimum cut routine. It is extremely easy to implement (see [5]). The construction is correct even if the routine call provides minimum cuts which cross each other. To see why his construction works, we note the following facts.

FACT 1

At any stage of computation, let $a, b, \ldots, z$ be a sequence of seeded nodes connected by labeled links in the tree T_j; then

$$F_{az} = \min(F_{ab}, F_{bc}, \ldots, F_{yz}).$$

Proof

Since each labeled link connecting i and j is a maximum flow computation between i and j, we have the same theorem 2.

Since each labeled link represents a cut separating a and z, we have

$$F_{az} \leq \min(F_{ab}, F_{bc}, \ldots, F_{yz}). \tag{9}$$

Equations (4) and (9) imply $F_{az} = \min(F_{ab}, F_{bc}, \ldots, F_{yz})$ for any sequence of seeded nodes. □

FACT 2

Let i and j be unlabeled neighbors in the tree T and j is of degree one, and k is a labeled neighbor of i. (We denote all nodes on the k side of the tree by K, i.e. if the $i-k$ link is removed, all nodes in K are connected to k by a subtree.) Then there exists a minimum cut $(X,\bar{X})$ separating i and j, where the set K is either totally contained in X or totally contained in $\bar{X}$.

Proof

Since every labeled link represents a minimum cut, let $(Y,\bar{Y})$ be the minimum cut corresponding to the $i-k$ link, where $i, j \in Y$ and $k \in \bar{Y}$. Since we know there exists a minimum cut $(X,\bar{X})$ separating i and j which does not cross $(Y,\bar{Y})$, then in that cut K is totally in X or totally in $\bar{X}$.

Comment 1

If we condense all unlabeled neighbors of a seed with the seed, then we have a supernode as in [4].

Comment 2

The $n-1$ minimum cuts induced by the cut-tree by Gusfield construction may not coincide with the $n-1$ minimum cuts provided by the routine calls. However, the $n-1$ cuts represented by the cut-tree are sufficient for finding maximum flows and minimum cuts between all pairs of nodes. The reason is that the routine calls may provide two minimum cuts $(X,\bar{X})$ and $(Y,\bar{Y})$ which cross each other, and the cut-tree will provide two non-crossing minimum cuts $(X,\bar{X})$ and $(Z,\bar{Z})$ which serve the same function as the two crossing cuts (see (4,7,8]).

For minimum partitions separating k nodes ($k \geq 3$), we need $\binom{n-1}{k-1}$ computations. See Hassin [6].

References

[1] C.K. Cheng and T.C. Hu, Maximum concurrent flow and minimum ratio cut, Technical Report CS88-141, UCSD, La Jolla, CA (1988).

[2] L.R. Ford and D.R. Fulkerson, Maximal flow through a network, Can. J. Math. 8 (3)(1956) 399–404.

[3] L.R. Ford and D.R. Fulkerson, *Flows in Networks* (Princeton University Press, Princeton, NJ, 1962).

[4] R.E. Gomory and T.C. Hu, Multi-terminal network flows, J. SIAM 9 (4) (1961)551–570.

[5] D. Gusfield, Very simple methods for all pairs network flow analysis, SIAM J. Comput., to appear.

[6] R. Hassin, Solution bases of multi-terminal cut problems, Math. Oper. Res. 13 (4) (1988)535–542.

[7] T.C. Hu, *Integer Programming and Network Flows* (Addison–Wesley, 1969).

[8] T.C. Hu, *Combinatorial Algorithms* (Addison–Wesley, 1982).

[9] T. Leighton and S. Rao, An approximate max flow min cut theorem for uniform multi-commodity flow problem with applications to approximation algorithm, *IEEE Annual Symp. on Foundations of Computer Science* (1988), pp. 422–431.

[10] D.W. Matula, Determining edge connectivity in $O(nm)$, *Proc. 28th Symp. on Foundations of Computer Science* (1987), pp. 249–251.

[11] F. Shahrokhi and D.W. Matula, The maximum concurrent flow problem, Technical Report, New Mexico (1986).