Algorithmica (1992) 8: 233–249

Algorithmica
© 1992 Springer-Verlag New York Inc.

Maximum Concurrent Flows and Minimum Cuts[1]

C. K. Cheng[2] and T. C. Hu[2]

Abstract. In many applications, we need to find a minimum cost partition of a network separating a given pair of nodes. A classical example is the Max-Flow Min-Cut Theorem, where the cost of the partition is defined to be the sum of capacities of arcs connecting the two parts. Other similar concepts such as minimum weighted sparsest cut and flux cut have also been introduced. There is always a cost associated with a cut, and we always seek the min-cost cut separating a given pair of nodes. A natural generalization from the separation of a given pair is to find all minimum cost cuts separating all $\binom{n}{2}$ pairs of nodes, with arbitrary costs associated with all $2^{n-1} - 1$ cuts. In the present paper, we show that $n - 1$ minimum cost cuts are always sufficient to separate all $\binom{n}{2}$ pairs of nodes.

A further generalization is to consider k-way partitions rather than two-way partitions. An interesting relationship exists between k-way partitions, the multicommodity flow problem, and the minimum weighted sparsest cut. Namely, if the staturated arcs in a multicommodity flow problem form a k-way partition ($k \leq 4$), then the k-way partition contains a two-way partition. This two-way partition is the minimum weight sparsest cut.

Key Words. Network partition, Multicommodity flow, Cut tree.

1. Introduction. Since the discovery of the Max-Flow Min-Cut Theorem by Ford and Fulkerson [2] in 1956, the theorem has been generalized to two commodity flows by Hu [8] in 1963, and to multicommodity flows by Onaga [17] and Iri [11] in 1971. The concept of a minimum cut has been generalized in different ways such as minimum circular cut [9], bisection [12], minimum edge expansion or flux cut [16], weighted sparsest cut [5], [15], etc. Recently, Leighton and Rao [13] discovered an approximate algorithm which relates a maximum concurrent flow to the sparsest cut. They also proved that the bound is tight. In all of these cases, a function is defined on the partition of the vertices into a subset and its complement, where a given pair of vertices are separated by the partition.

In a VLSI circuit layout [10] and other applications [6], we may need to consider minimum partitions separating certain pairs of vertices. For the $\binom{n}{2}$ pairs of vertices, we shall show that there are only $n - 1$ essential minimum partitions in all cases, whether it is max-flow min-cut, minimum weighted sparsest cut, or flux cut. Hassin [7] proposed an algorithm to find the essential cut set with

[1] This work is supported in part by the NSF under Grant MIP-8700767 and micro program under Grants 506205 and 506215, Intergraph, and Data General.

[2] Computer Science and Engineering Department, University of California, San Diego, La Jolla, CA 92093, USA.

Received August 8, 1989; revised February 10, 1990, Communicated by C. L. Liu.

$O(n \log n)$ oracle calls or cut routines, under the assumption that distinct cuts have distinct costs. We shall present an algorithm to find the set of essential cuts with $n - 1$ oracle calls.

Although the problem of finding the minimum weighted sparsest cut is NP-hard [15], the concurrent flow problem, which maximizes the uniform flow demand between every pair of vertices, can be formulated as a linear programming problem. We can use column-generating techniques [3] to solve the maximum concurrent flow problem by approximation methods and other fast algorithms.

In using linear programming techniques, if we can find a set of arcs with nonzero shadow prices which also forms a two-way partition, then we have found the sparsest cut. In general, these arcs will form a K-way partition. We show that if $K \leq 4$, then there exists a two-way partition of the partitioned K subsets, which is also the minimum weighted sparsest cut. Independently, Shahrokhi and Matula [20] have shown the case $K \leq 4$ with a combinatorial approach. We shall give a proof that utilizes the techniques of linear programming.

2. Minimum Cut and Essential Cut Set.

Given a network $N = (V, E)$ where V is the set of vertices and E is the set of undirected arcs, $|V| = n$ and $|E| = m$. There is a capacity c_{ij} associated with each arc connecting vertices i and j.

Let $f(X, \bar{X})$ be a symmetric function on a vertex set X and its complement $\bar{X}$. We define the minimum value of a cut separating vertices i and j as

$$F_{ij} = \min_{X} f(X, \bar{X}) \qquad (i \in X, j \in \bar{X}). \tag{1}$$

For example, let $X\bar{X} = \sum_{i \in X} \sum_{j \in \bar{X}} c_{ij}$ be the sum of all arc capacities of arcs connecting X and $\bar{X}$. When $f(X, \bar{X}) = X\bar{X}$, we have the usual definition of the capacity of a minimum cut separating vertices i and j, as used in the Max-Flow Min-Cut Theorem. The minimum quotient separator or flux cut uses $f(X, \bar{X}) = X\bar{X}/\min(|X|, |\bar{X}|)$. If we define $f(X, \bar{X}) = X\bar{X}/|X| \times |\bar{X}|$, then

$$F_{ij} = \min_{X} \frac{X\bar{X}}{|X| \times |\bar{X}|} \qquad (i \in X, j \in \bar{X})$$

is the minimum sparsest cut. We shall use the term *ratio cut* of i and j, denoted by R_{ij}, to mean $\min_X(X\bar{X}/|X| \times |\bar{X}|)$ with $i \in X, j \in \bar{X}$. Note that all of the above functions are symmetric, i.e., $f(X, \bar{X}) = f(\bar{X}, X)$. The derivation in this section can be applied to all these symmetric cost functions. We shall use F_{ij} also to denote the cut $(X, \bar{X})$ associated with F_{ij} in (1) if no confusion should arise.

2.1. *Triangular Inequality*

THEOREM 1. *For any three vertices i, j, and $k \in V$, we have the following inequality*:

$$F_{ik} \geq \min(F_{ij}, F_{jk}). \tag{2}$$

PROOF. Let $(X, \bar{X})$ be the cut separating i and k with a minimum cost F_{ik} where $i \in X$ and $k \in \bar{X}$. The vertex j then belongs either to X or to $\bar{X}$.

(i) If $j \in X$, then the partition $(X, \bar{X})$ serves as a parition separating j and k. Since F_{jk} is the minimum cost of the cut separating j and k, then

$$F_{jk} = \min_{j \in X, k \in \bar{X}} f(X, \bar{X}) \leq F_{ik}. \tag{3}$$

(ii) If $j \in \bar{X}$, then we can also derive

$$F_{ij} = \min_{i \in X, j \in \bar{X}} f(X, \bar{X}) \leq F_{ik}. \tag{4}$$

We conclude that (2) is true, because either (i) or (ii) must be true. □

Since (2) is true for any three arbitrary vertices, we can see that the three values F_{ij}, F_{jk}, and F_{ik} cannot all be distinct. If they are three distinct values, then we can put the smallest value on the left-hand side of (2) and obtain a contradiction. Thus, among the three values, F_{ij}, F_{jk}, and F_{ik}, two values must be equal and both are not greater than the third value. Given a graph of n vertices, there are $\binom{n}{2}$ pairs of vertices. We can by induction from (2) show that there are at most $n - 1$ distinct values of min-cost partitions.

2.2. The Cuts for All Pairs of Nodes. The proof used in Theorem 1 is identical to the proof used by Gomory and Hu [4]. However, the construction of the Gomory–Hu cut tree makes use of the fact that there always exists a set of $n - 1$ noncrossing minimum cuts. For general cost functions associated with cuts, minimum cost cuts do cross each other (Figure 1). In this paper, we show that $n - 1$ minimum cost cuts are sufficient to separate all pairs of vertices for arbitrary cost functions defined by (1).

THEOREM 2. *Given network $N = (V, E)$, with $|V| = n$, we need at most $n - 1$ distinct cuts, such that for all $i, j \in V$, one of the $n - 1$ cuts is the minimum cut separating i and j.*

PROOF. Assuming that we find a minimum cut for each pair of vertices, then there are $\binom{n}{2}$ cuts which separate all pairs of vertices. We sort the cuts according to the cut value from small to large. We then select among the $\binom{n}{2}$ minimum cuts, one by one, into a cut set S. The smallest value is selected first.

Initially, cut set S contains only the first cut. Let $(X, \bar{X})$ be the partition made by the first cut. The second cut either: (a) coincides with the first cut, or (b) separates at least one of X and $\bar{X}$ into two subsets. If case (a) occurs, then the cut

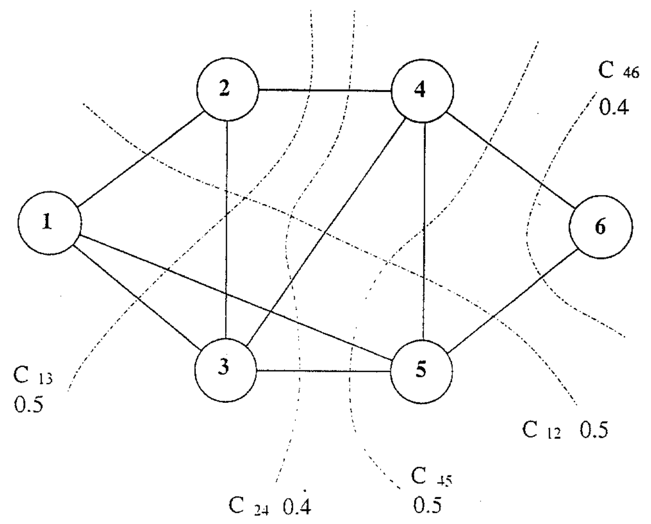

Fig. 1. A six-vertex network example. The capacity of each arc is one unit. Each dashed line labeled with C_{ij} and a number is a ratio cut separating vertices i and j with cut cost F_{ij}. Note that cut C_{12} crosses cuts C_{13}, C_{24}, and C_{45}.

is not selected. If case (b) occurs, we have more disjoint subsets $X_1, X_2, \dots, X_k$, where $k > 2$ and $\bigcup_{i=1}^{k} X_i = V$. We then select the cut into the cut set S. In other words, a cut is selected only if it further separates certain subsets into smaller subsets. The process continues and the number of disjoint subsets increases, until the cuts in S partition V into subsets, with each subset containing only a single vertex.

(i) Since each cut divides the subsets into smaller subsets and the smallest subset contains only a single vertex, it takes at most $n - 1$ cuts. Thus, $|S| \leq n - 1$.

(ii) For each pair of vertices i and j, let cut γ be a lowest cost cut in S such that cut γ separates the pair. Then, from the selection process for the set S, cut γ is also a minimum cut among $\binom{n}{2}$ cuts, such that the cut separates i and j. Therefore, cut γ is a minimum cut of i and j. Otherwise, if there is another cut that separates i and j with a smaller cut value, that cut should have been selected in the process. □

Let us call S, which is the set of distinct cuts separating all pairs of vertices, an *essential cut set*. We can build a tree of the essential cut set to represent the partition of all pairs of vertices in V.

2.3. Cut Tree Algorithm for Finding an Essential Cut Set. With respect to a given cost function f, let us assume we have a routine which can generate the minimum cut. Given a pair of vertices (p, q), the cut routine CUT(p, q) returns (F_{pq}, P, Q) where (P, Q) is a minimum cut that separates p and q ($p \in P$, $q \in Q$) with a cost of F_{pq}. We construct an algorithm to find the essential cut set in $n - 1$ CUT oracle calls.

We shall state the structure of the cut tree, then describe the algorithm, and finally we shall prove the correctness of the algorithm.

2.3.1. The Binary Cut Tree Structure. The essential cut set is represented by a binary tree with $n - 1$ internal nodes and n leaves, with the root at the top. Each internal node represents a cut, and each leaf contains a vertex in V. For each pair of leaves, their lowest common ancestor is the minimum cut of the corresponding pair of vertices.

The construction of the tree representation starts from a tree of a single leaf. Then we successively increase the number of internal nodes. We use a tree structure to store the information resulting from each iteration. The following is a detailed description of the tree structure.

I. A Directed Binary Tree Representation. The binary tree contains cut nodes (internal nodes) and leaves. Given a minimum cut (F_{pq}, P, Q), we use a cut node to store the cut cost F_{pq}. The two branches of the cut node are labeled with the subsets P and Q. From the root to each leaf, there is a unique path consisting of the branches. Let $(B_1, B_2, \ldots, B_k)$ be the path from the root to leaf L_i. We then use leaf L_i to represent the vertex subset $\bigcap_{i=1}^{k} B_i$, which is the intersection of the subsets B_i along the path. (However, if the tree contains a single leaf but no cut node, the leaf represents the vertex set V.)

II. The Set of Seeds. We define some vertices as seeds in the tree construction process. Given a tree, for each cut node C_{pq}, we set vertices p and q as the seeds. Thus, the set of seeds is a vertex set $\{p, q \mid C_{pq}$ *is a cut node of the tree*$\}$. If the tree contains only a single leaf but no cut node, we arbitrarily choose one vertex p as a seed. This seed p, together with another vertex $q \neq p$, will be used to create a cut node C_{pq}.

III. The Properties of the Tree Structure. We shall manipulate the tree so that the tree structure maintains three properties throughout the iterations of the algorithm. A tree satisfying the three properties is called *admissible*. The three properties are:

(i) The cut value of a cut node is always larger than or equal to the value of its father.
(ii) Each leaf L_i contains exactly one seed i.
(iii) For any pair of leaves L_i and L_j with seeds i and j, let the cut node C_{pq} be their lowest common ancestor. Then the cut node C_{pq}, together with its two branches, defines a minimum cut separating vertices i and j.

2.3.2. The Algorithm of the Tree Construction. The algorithm successively builds trees $T_0, T_1, \ldots, T_{n-1} = T$, where the tree T_i contains i cut nodes. Each T_{k+1} contains one more cut node together with one more leaf than its predecessor T_k.

Algorithm

Input: Given network $N = (V, E)$, a vertex in V is arbitrarily chosen as the seed of V.

1. Set $k = 0$. Initialize tree T_k with a single leaf which corresponds to vertex set V.
2. Given tree T_k, find a leaf L_p containing a seed p and $|L_p| \geq 2$. If such a leaf does not exist, then output tree T_k and exit.

3. Pick vertex $q \neq p$ in L_p. Call CUT(p, q) and create cut node C_{pq} with P and Q as labels of its two branches, where $p \in P$ and $q \in Q$.
4. Trace the ancestors of leaf L_p. If the cut cost F_{pq} of a cut node C_{pq} is not smaller than the cost of the father of leaf L_p, perform operation (i), or else perform operation (ii).
   (i) Put cut node C_{pq} at the position of leaf L_p. Append leaf L_p to the branch P of cut node C_{pq}.
   (ii) Let cut node C_{uv} be the highest ancestor such that its cut cost F_{uv} is greater than the cost F_{pq} (Figure 2). Place cut node C_{pq} at the position of cut node C_{uv} and append cut node C_{uv} to the branch P of cut node C_{pq}. The subtree of cut node C_{uv} follows cut node C_{uv} (Figure 3).
5. Create a leaf L_q on the branch Q of cut node C_{pq}. Update the vertex set of all the leaves by intersecting the labels in the branches.
6. Set the updated tree to be T_{k+1}. Let $k = k + 1$. Repeat step 2.

EXAMPLE. Figures 4–8 demonstrate the cut tree algorithm on the six-vertex graph (Figure 1). We use ratio cut as the minimum cut. Let vertices 1 and 2 be the first pair to be separated. The cut cost $F_{12} = 5/(3 \times 3) = 0.\dot{5}$. Figure 4 shows the cut node C_{12} with two branches, each labeled with a partitioned subset. The two leaves contain vertex sets $\{1^*, 3, 5\}$ and $\{2^*, 4, 6\}$ where vertices 1 and 2 are postfixed with *, indicating that these two vertices are the seeds of the two leaves.

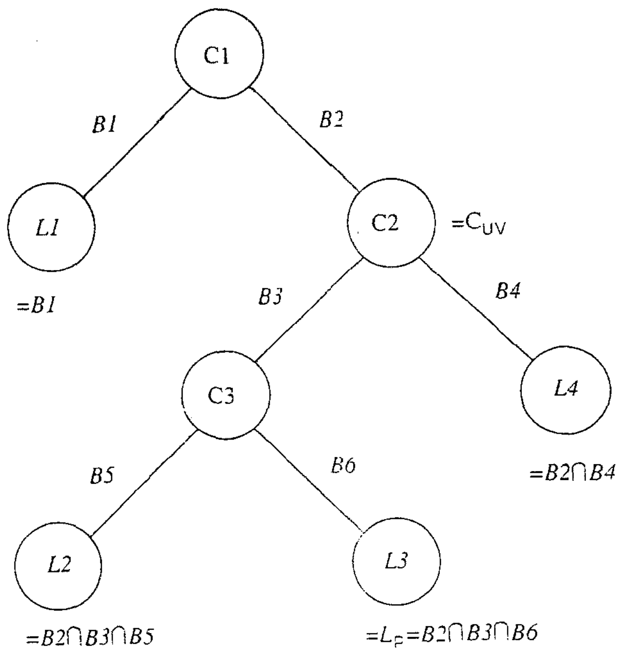

Fig. 2. A binary tree of $\{C1, C2, C3\}$ three cut nodes, $\{L1, L2, L3, L4\}$ four leaves, and $\{B1, B2, B3, B4, B5, B6\}$ six branches. Each leaf contains a vertex set which is the intersection of the branches along the path from the root to the leaf. The graph illustrates step 4(ii) of the tree construction algorithm with leaf $L3 = L_p$ and cut node $C2 = C_{uv}$.

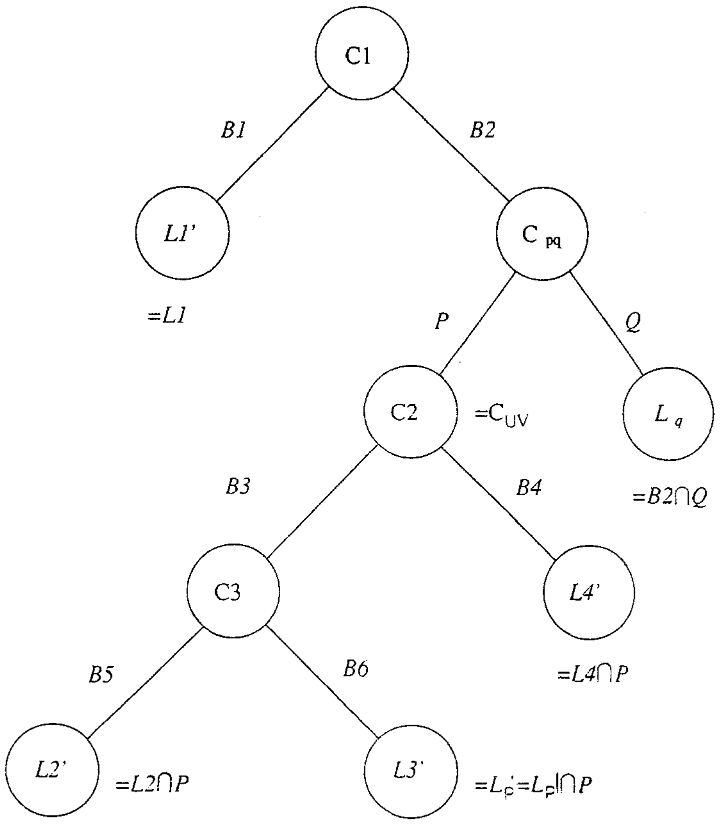

Fig. 3. Cut node C_{pq} is inserted at the position of C_{uv}. Cut node C_{uv} appends to branch P, and the subtree of C_{uv} follows the cut node C_{uv}. Note that each leaf L_i' in the subtree of C_{uv} is updated to $L_i' = L_i \cap P$.

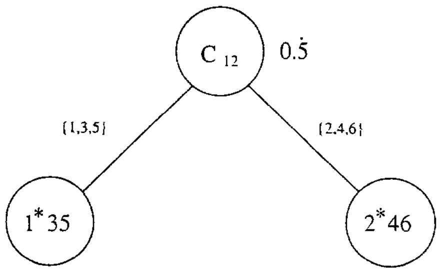

Fig. 4. The cut tree after the first iteration of the tree construction algorithm on the six-vertex example. Cut node C_{12} with two branches separates the vertex set into $\{1^*, 3, 5\}$ and $\{2^*, 4, 6\}$ two sets. The cut cost is $0.\dot{5}$. The vertices 1 and 2 are postfixed with $*$ indicating that the two vertices are the seeds of these two leaves.

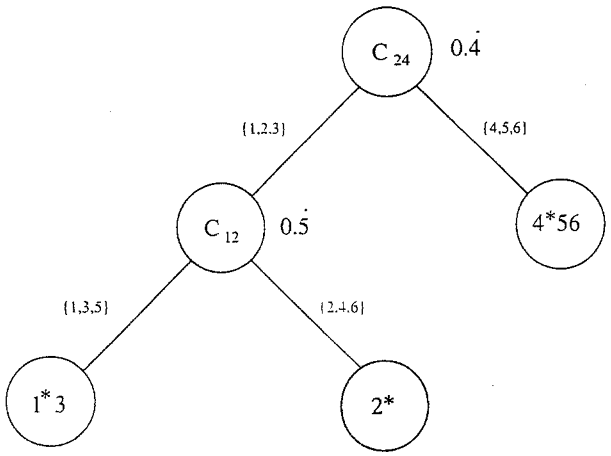

Fig. 5. Vertices 2 and 4 are chosen from leaf $\{2^*, 4, 6\}$. The cut node C_{24} contains a cut cost $F_{24} = 0.\dot{4}$, which is smaller than the cut cost of C_{12}. Thus, C_{24} is inserted above C_{12}. The leaves are updated according to the insertion.

If we choose seed 2 and vertex 4 in the next iteration, then the second cut separates vertices 2 and 4 with a cut cost $F_{24} = 4/(3 \times 3) = 0.\dot{4}$. Because $F_{24} < F_{12}$, step 4(ii) inserts cut node C_{24} to the top and appends cut node C_{12} to its left branch. The vertex sets of the leaves are updated as shown in Figure 5. Figures 6, 7, and 8 show the tree construction of the next three iterations. Cut

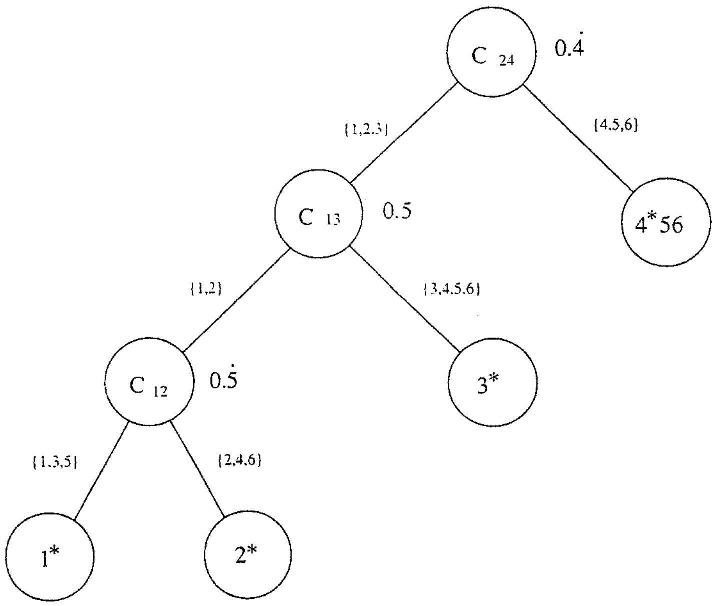

Fig. 6. The cut tree after the third iteration. Cut node C_{13} is inserted into the tree.

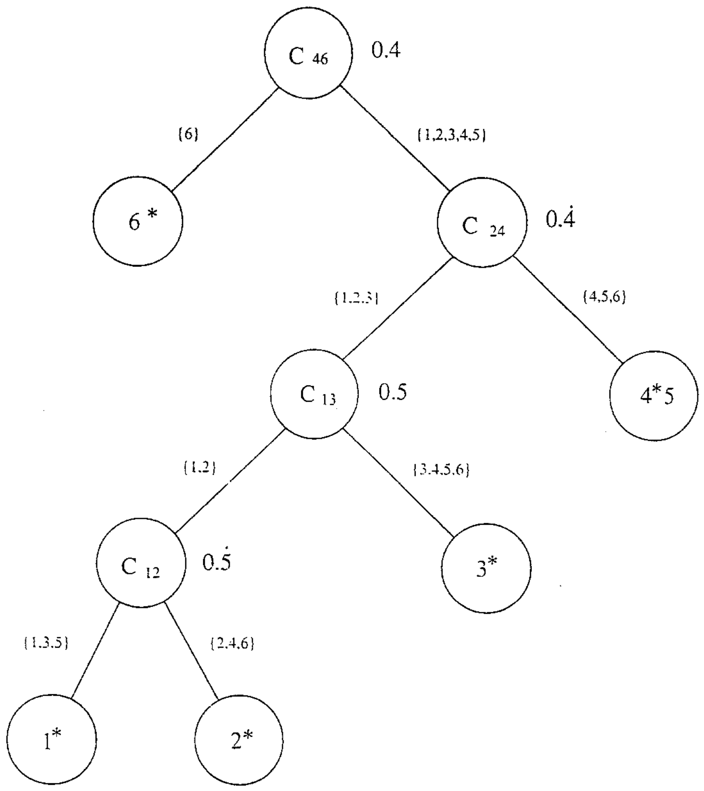

Fig. 7. The cut tree after the fourth iteration. Cut node C_{46} is inserted into the tree.

nodes C_{13}, C_{46}, and C_{45} are inserted according to the heap condition. All the cut nodes in the final tree structure (Figure 8) constitute an essential cut set.

2.3.3. The Proof of the Algorithm. It is important that tree T_k is admissible in each iteration k, so that the process can proceed properly. We show the tree is admissible after each iteration, by induction on the number of cut nodes in the tree. The tree is certainly admissible when there is only one cut node and two seeds in the two leaves. Given an admissible tree T_k, Lemmas 1–4 show that the tree T_{k+1} satisfies the three properties. Thus, Theorem 3 concludes that the tree in each iteration is admissible. Using the properties that the tree is admissible, we show in Theorem 4 that the algorithm finds an essential cut set with $n - 1$ CUT oracle calls.

LEMMA 1. *Given an admissible tree T_k, tree T_{k+1} satisfies property* (i).

PROOF. Given tree T_k, step 3 of the algorithm generates a new cut node. Step 4 inserts the new cut node in keeping with property (i). Therefore, tree T_{k+1} also satisfies property (i). □

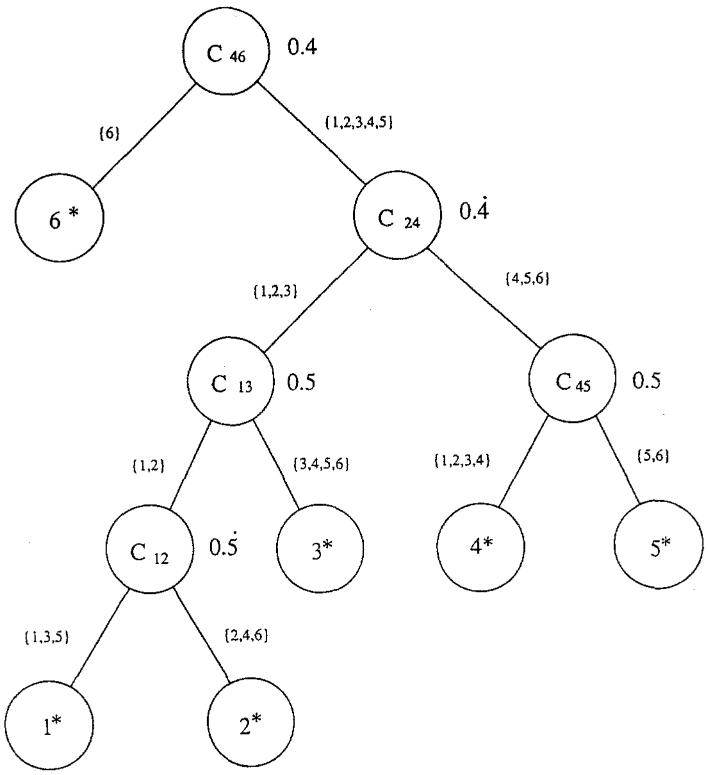

Fig. 8. The final tree structure. There are $n - 1$ cut nodes and n leaves. Each leaf L_i contains a single vertex i. The cut tree contains an essential cut set.

LEMMA 2. *Given an admissible tree T_k, for each leaf L_i with seed i, vertex i remains in the updated leaf L_i', and the new seed q belongs to the new leaf L_q.*

PROOF. We prove the lemma for the case of step 4(ii) in the algorithm. The proof for the case of step 4(i) is similar.

For the process of case 4(ii), cut node C_{pq} with its two branches P and Q is inserted as shown in Figure 3. Note that each leaf L_i in the subtree of C_{uv} is updated to $L_i' = L_i \cap P$, because branch P is inserted into the path from the root to the leaf L_i. With reference to Figure 3, we prove the lemma by the following three arguments:

(i) The updated leaf $L_i' = L_i$, for each leaf L_i $(i \neq q)$ not in the subtree with C_{uv} as the ancestor.
As shown in Figure 3, the path from the root to the leaf L_i is not changed by step 4, therefore leaf L_i' remains the same.

(ii) Seed i continues to be an element of the updated leaf L_i', for each leaf L_i under the subtree of C_{uv}.
As shown in Figure 3, leaf $L_i' = L_i \cap P$. Thus, if we can prove that i is an element of P, then seed $i \in L_i'$; the proof is by contradiction. If $i \in Q$, then the

cut (P, Q) separates i and p. The minimum cut separating i and p should have a value $F_{ip} \leq F_{pq}$. However, from property (iii), there is a cut node C_{ab} in the subtree of C_{uv}, which defines a minimum cut separating seed i and seed p. From property (i) and step 4, the cut value $F_{ip} = F_{ab} \geq F_{uv} > F_{pq}$, which contradicts the previous inequality expression that $F_{ip} \leq F_{pq}$. Therefore, we have $i \in P$.

(iii) The new seed q is an element of the new leaf L_q.
In tree T_k, assume $(B_1, B_2, \dots, B_w)$ is the path from the root to cut node C_{uv}. Then we have $q \in \bigcap_i^w B_i$, where B_i are the subsets along the path. Since $q \in Q$ and $L_q = \bigcap_{i=1}^w B_i \cap Q$, we have seed $q \in L_q$. When C_{uv} is the root, the proof is similar. □

LEMMA 3. *Given an admissible tree T_k, tree T_{k+1} satisfies property* (ii).

PROOF. From Lemma 2, we know every leaf in tree T_{k+1} contains exactly one seed. Thus, the tree T_{k+1} satisfies the property (ii). □

LEMMA 4. *Given an admissible tree T_k, tree T_{k+1} satisfies property* (iii).

PROOF. As shown in Figure 3, from Lemma 2 we know that for each pair of leaves L_i and L_j in tree T_k their lowest common ancestor remains the same in tree T_{k+1}. Thus, for every pair of leaves L_i and L_j with seeds $i \neq q$ and $j \neq q$, their lowest common ancestor is the minimum cut separating vertices i and j. To prove the lemma, it is sufficient to show that for every leaf L_i with seed $i \neq q$, the pair of leaves L_i and L_q has the minimum cut separating vertices i and q as their lowest common ancestor. Let C_{ab} be the ancestor. There are two cases:

(i) The minimum cut of vertices i and q has a cut value $F_{iq} \leq F_{ab}$.
Since cut node C_{ab} defines a cut that partitions vertices i and q, then $F_{iq} \leq F_{ab}$.

(ii) $F_{iq} \geq F_{ab}$.
As shown in Figure 3, cut node C_{pq} is the father of leaf L_q and an ancestor of leaf L'_p in tree T_{k+1}. For the case that leaf L_i is in the subtree of C_{uv}, we have $F_{ab} = F_{pq}$ and $F_{ip} \geq F_{pq}$ from property (i). Using Theorem 1, we have $F_{iq} \geq \min(F_{ip}, F_{pq}) = F_{pq} = F_{ab}$. When leaf L_i is located outside the subtree of C_{uv}, we have $F_{ab} = F_{ip}$ and $F_{pq} \geq F_{ip}$ from property (i). Using Theorem 1, we have $F_{iq} \geq \min(F_{ip}, F_{pq}) = F_{ip} = F_{ab}$. Therefore, we conclude that $F_{iq} \geq F_{ab}$.

From (i) and (ii), and since seeds i and q are partitioned by the two branches of C_{ab}, then the node C_{ab} with its two branches defines a minimum cut separating vertices i and q. □

From Lemmas 1, 3, and 4, we conclude, by induction, that the tree structure T_k is admissible in each iteration k.

THEOREM 3. *The tree construction algorithm generates an admissible tree T_k for each iteration k.*

Since the tree is admissible, we can continue the algorithm until every leaf contains one single vertex. The following theorem proves the correctness of the algorithm.

THEOREM 4. *The cut tree algorithm derives an essential cut set with $n-1$ CUT oracle calls.*

PROOF. Since each tree T_r is admissible, then from property (ii), the operation can be continued until every leaf contains one single vertex. Therefore, it takes $n-1$ iterations to create a tree of $n-1$ cut nodes and n leaves.

From property (iii), the tree contains a minimum cut for every pair of seeds. Therefore, the $n-1$ cut nodes constitute an essential cut set. Consequently, we conclude that the algorithm takes $n-1$ CUT oracle calls to generate an essential cut set. □

3. The Global Ratio Cut and the Maximum Concurrent Flow. Among the distinct cuts in the essential cut set, we can find a cut with the minimum cut cost. Let us denote this cut as the *global minimum cut*. In this section, we focus on the global ratio cut

$$R_1 = \min_{i,j \in V} R_{ij}. \tag{5}$$

This global ratio cut, defined by (5), is closely related to the uniform multicommodity flow problem, where f units of flow is transmitted between all pairs of vertices. Leighton and Rao [13] have shown that

$$R_1 \geq f \geq \Omega\left(\frac{R_1}{\log n}\right), \tag{6}$$

where the bound is tight.

The uniform multicommodity flow problem can be formulated as follows: Given a network $N = (V, E)$, the demand of flow between each pair of vertices is equal to an identical value f. The object is to maximize f under the constraint that the sum of all arc flows is less than the arc capacity. Let x_{ij}^p be the flow for commodity p on arc (i, j); let n be the total number of vertices and commodities; and let c_{ij} be the capacity of arc (i, j), we then have

$$Obj\colon\quad \max f \tag{7}$$

subject to the flow demand of commodity p from every vertex p to the other vertices i,

$$\sum_{j=1}^{n} x_{ij}^p - \sum_{j=1}^{n} x_{ji}^p = \begin{cases} -f & \text{if } i \neq p, 1 \leq i, p \leq n, \\ (n-1)f & \text{if } i = p, 1 \leq i, p \leq n, \end{cases} \tag{8}$$

and the constraints of the arc capacities,

$$(9) \qquad \sum_{p=1}^{n} x_{ij}^p + \sum_{p=1}^{n} x_{ji}^p \leq c_{ij}, \qquad 1 \leq i, j \leq n.$$

If we assign dual variables λ_i^p for the vertex i ($i = 1, \dots, n$), with respect to commodity p (eq. (8)), and d_{ij} to arc (i, j) (eq. (9)), then we have the following dual problem [1]:

$$(10) \qquad \mathit{Obj}: \ \min \sum_{(i,j)} c_{ij} d_{ij}$$

subject to

$$(11) \qquad d_{ij} \geq |\lambda_i^p - \lambda_j^p|, \qquad 1 \leq i, j, p \leq n,$$

$$(12) \qquad \sum_{p=1}^{n} \sum_{i \neq p, i=1}^{n} (\lambda_i^p - \lambda_p^p) \geq 1.$$

Let d_{ij} be an undirected distance function on the arc connecting vertex i and vertex j. From (11), d_{ij} can be interpreted as distance in a metric space with the triangular inequality [11], [18]:

$$(13) \qquad d_{ij} + d_{jk} \geq d_{ik}, \qquad 1 \leq i, j, k \leq n.$$

In the following two lemmas, we derive that the variables λ_i^p in the constraints can be replaced by the variables d_{ij}.

LEMMA 5. *There exists an optimal solution such that the equality of constraint* (12) *holds, i.e.*, $\sum_{p=1}^{n} \sum_{i \neq p, i=1}^{n} (\lambda_i^p - \lambda_p^p) = 1$.

PROOF. We prove the lemma by contradiction. For an optimal solution, if the inequality in (12) holds, i.e.,

$$\sum_{p=1}^{n} \sum_{i \neq p, i=1}^{n} (\lambda_i^p - \lambda_p^p) = h > 1,$$

then we can scale down all the variables by the ratio $1/h$. The new assignment $\hat{d}_{ij} = (1/h) d_{ij}$ and $\hat{\lambda}_i^p = (1/h) \lambda_i^p$ still satisfies the constraints (11) and (12). Since c_{ij} and d_{ij} are positive, then

$$\sum_{(i,j)} c_{ij} \hat{d}_{ij} = \sum_{(i,j)} c_{ij} \frac{d_{ij}}{h} = \frac{1}{h} \sum_{(i,j)} c_{ij} d_{ij} < \sum_{(i,j)} c_{ij} d_{ij}.$$

Thus, the updated value of the objective function becomes smaller; which contradicts the assumption that the original solution is optimal. □

Therefore, we can replace the constraint (12) with an equality expression:

$$\sum_{p=1}^{n} \sum_{i \neq p, i=1}^{n} (\lambda_i^p - \lambda_p^p) = 1. \tag{14}$$

LEMMA 6. *There exists an optimal solution such that* $\lambda_i^p - \lambda_p^p = d_{ip}$ *for all* $1 \leq i, p \leq n$.

PROOF. We prove the lemma by contradiction. For an optimal solution, let us assume that there exists a vertex p and a vertex k such that $d_{kp} > \lambda_k^p - \lambda_p^p$. From constraint (11), we can write

$$\sum_{p=1}^{n} \sum_{i \neq p, i=1}^{n} d_{ip} = \sum_{p=1}^{n} \sum_{i \neq p, i=1}^{n} (\lambda_i^p - \lambda_p^p) + \varepsilon \qquad \text{where} \quad \varepsilon > 0.$$

We update the variables by assigning $\hat{d}_{ij} = d_{ij}$, $\hat{\lambda}_p^p = \lambda_p^p$ and $\hat{\lambda}_i^p = d_{ip} + \lambda_p^p$ for all $1 \leq i, j, p \leq n$, $i \neq p$. The following two arguments prove the lemma:

(i) The updated values of the variables satisfy the constraint (11).
Since the distances d_{ij} satisfy the triangular inequality (13), we have

$$\hat{d}_{ij} = d_{ij} \geq d_{ip} - d_{jp} = (\hat{\lambda}_i^p - \hat{\lambda}_p^p) - (\hat{\lambda}_j^p - \hat{\lambda}_p^p) = \hat{\lambda}_i^p - \hat{\lambda}_j^p$$

for all $1 \leq i, j \leq n$.

(ii) For the new assignment, we have $\sum_{p=1}^{n} \sum_{i \neq p, i=1}^{n} (\hat{\lambda}_i^p - \hat{\lambda}_p^p) = 1 + \varepsilon$.
Since

$$\sum_{p=1}^{n} \sum_{i \neq p, i=1}^{n} (\hat{\lambda}_i^p - \hat{\lambda}_p^p) = \sum_{p=1}^{n} \sum_{i \neq p, i=1}^{n} d_{ip} = \sum_{p=1}^{n} \sum_{i \neq p, i=1}^{n} (\lambda_i^p - \lambda_p^p) + \varepsilon.$$

From (14), we have $\sum_{p=1}^{n} \sum_{i \neq p, i=1}^{n} (\hat{\lambda}_i^p - \hat{\lambda}_p^p) = 1 + \varepsilon$.

From (i) and (ii), the updated values constitute a feasible solution. From (ii) and the proof of Lemma 5, the value of the objective function can be scaled down by the ratio $1/(1 + \varepsilon)$; which contradicts the assumption that the original solution is optimal. □

Therefore, constraint (14) can be rewritten as a function of d_{ij}:

$$\sum_{p=1}^{n} \sum_{i \neq p, i=1}^{n} d_{ip} = 1. \tag{15}$$

The uniform concurrent flow problem can be stated as an objective function of (10) with the constraints (13) and (15), i.e.,

$$\mathit{Obj}: \quad \min \sum_{(i,j)} c_{ij} d_{ij}$$

subject to

$$d_{ij} + d_{jk} \geq d_{ik}, \qquad 1 \leq i, j, k \leq n,$$

$$\sum_{p=1}^{n} \sum_{i \neq p,\, i=1}^{n} d_{ip} = 1.$$

The distance is the *shadow price* [1] of the arc in the original problem. Lomonosov [14] observed that the arc with positive distance generates the partition of the vertex set V. When the arcs with positive distance d_{ij} form a two-way partition, we can show that the partition defines the global ratio cut. When the arcs with positive distance form a K-way partition with $K \leq 4$, we also find that there exists a two-way partition that again defines the global ratio cut.

THEOREM 5. *Let* $\mathbf{D} = \{(i,j) | d_{ij} > 0\}$ *define a partition that separates the network into K disconnected subsets. If* $K \leq 4$, *then there exists a global ratio cut that is a subset of* $\mathbf{D}$.

PROOF.

I. The case when $K = 2$

(1) First, we want to prove that for all $d_{ij} \in \mathbf{D}$, $d_{ij} = e$, where e is a constant:
Let $(X, \bar{X})$ be the partition made by $\mathbf{D}$. Let i and j be vertices in X, and let k be a vertex in $\bar{X}$. From (13), we have $d_{ij} + d_{jk} \geq d_{ik}$ and $d_{ij} + d_{ik} \geq d_{jk}$. Since $d_{ij} = 0$ by assumption, we have $d_{ik} = d_{jk}$. Similarly, if l is a vertex in $\bar{X}$, then we have $d_{jk} = d_{jl}$. Hence, $d_{ik} = d_{jk} = d_{jl}$ for all i and j in X, and k and l in $\bar{X}$.
(2) Second, we prove that $e = \frac{1}{2}(1/|X| \times |\bar{X}|)$:
Since $d_{ij} = e$ for all $d_{ij} \in \mathbf{D}$, from (15), we have $e = 1/2|X| \times |\bar{X}|$.
(3) From (1) and (2) above we have the object function

$$\sum_{(i,j)} c_{ij} d_{ij} = \frac{X\bar{X}}{2|X| \times |\bar{X}|}.$$

Since this is a minimum solution, $\mathbf{D}$ is a global ratio cut.

II. The case when $K = 3$

(1) Let X_1, X_2, X_3 be the disjoint vertex sets partitioned by $\mathbf{D}$. As in the proof of case I.1, we have $d_{kl} = e_{ij}$ for all $k \in X_i$, $l \in X_j$ and $i, j \in \{1, 2, 3\}$, where e_{ij} is a constant.
(2) From (1) above, (10), (13), and (15) can be expressed as functions of e_{12}, e_{13}, and e_{23}. The constraint (13) is expressed with the following three equations:

$$\begin{aligned} e_{12} + e_{13} + s_1 &= e_{23}, \\ e_{12} + e_{23} + s_2 &= e_{13}, \qquad (16) \\ e_{13} + e_{23} + s_3 &= e_{12}, \end{aligned}$$

where s_1, s_2, and s_3 are nonnegative slack variables. Thus, we have four constraint equations and six variables. From the theorem of linear programming, we can have two variables equal to zero in the optimal solution. If $e_{ij} = 0$ for $i, j \in \{1, 2, 3\}$, then by definition $\mathbf{D}$ becomes a two-way partition. Let us set two slack variables s_i, s_j equal to zero. From (16), we then can derive $e_{ij} = 0$. Consequently, we reduce $\mathbf{D}$ to a two-way partition as in case I.

III. The case when $K = 4$. The proof is similar to the proof of case II. For the case of $K = 4$, we have thirteen constraint equations and eighteen variables. Five slack variables can be equal to zero in the optimal solutions. Likewise, we can derive that a cost e_{ij} should be zero. Consequently, the problem is reduced to a three-way partition case II. $\square$

Acknowledgments. The encouragement of Professor Christo Papadimitriou is highly appreciated. We would also like to thank the referees for their suggestions.

References

[1] V. Chvatal, *Linear Programming*, Freeman, San Francisco, 1983, p. 67.
[2] L. R. Ford and D. R. Fulkerson, Maximal flow through a network, *Canad. J. Math.* **8** (3) (1956), 399–404.
[3] D. R. Fulkerson, Suggested computation for maximal multi-commodity network flow, *Management Sci.* **5** (1) (1958), 97–101.
[4] R. E. Gomory and T. C. Hu, Multi-terminal network flows, *J. SIAM* **9** (4) (1961), 551–570.
[5] F. Granot and R. Hassin, Multi-terminal maximum flows in node-capacitated networks, *Discrete Appl. Math.* **12** (1986), 157–163.
[6] D. Gusfield, Faster detection of compromised data In 2-D tables, Technical Report, CSE-89-30, Computer Science Division, University of California, Davis, November 1989.
[7] R. Hassin, Solution bases of multi-terminal cut problems, *Math. Oper. Res.* **13** (4) (1988), 535–542.
[8] T. C. Hu, Multi-commodity network flows, *J. ORSA* **11** (3) (1963), 344–360.
[9] T. C. Hu and F. Ruskey, Circular cut in a network, *Math. Oper. Res.* **5** (3) (1980), 422–434.
[10] T. C. Hu and E. S. Kuh, *VLSI Circuit Layout Theory and Design*, IEEE Press, IEEE Circuits and Systems Society, 1985, pp. 105–114.
[11] M. Iri, On an extension of the maximum flow minimum cut theorem to multicommodity flows, *J. Oper. Res. Soc. Japan*, **5** (4) (1967), 697–703.
[12] B. W. Kernighan and S. Lin, An efficient heuristic procedure for partitioning graphs, *Bell System Tech. J.* (1970), 291–307.
[13] T. Leighton and S. Rao, An approximate Max-Flow Min-Cut Theorem for uniform multicommodity flow problems with applications to approximation algorithm, *IEEE Annual Symposium on Foundations of Computer Science*, 1988, pp. 422–431.
[14] M. V. Lomonosov, Combinatorial approaches to multiflow problems, *Discrete Appl. Math.* **11** (1) (1985), 1–94.
[15] D. W. Matula and F. Shahrokhi, The maximum concurrent flow problem and sparsest cuts, Technical Report, Southern Methodist University March 1986.
[16] D. W. Matual, Determining edge connectivity in *O*(*nm*), *IEEE Annual Symposium on Foundations of Computer Science* October 1987, pp. 249–251.
[17] K. Onaga, A multi-commodity theorem, *Trans. Int. Electron. Commun. Engrs. Japan* **53-A** (7) (1970), 350–356.

[18] K. Onaga and O. Kakusho, On feasibility conditions of multicommodity flows in networks, *IEEE Trans. Circuit Theory* **18** (4) (1971), 425–429.

[19] S. Rao, Finding near optimal separators in planar graphs, *IEEE Symposium on Foundations of Computer Science*, 1987, pp. 225–237.

[20] F. Shahrokhi and D. W. Matula, The maximum concurrent flow problem, Technical Report, New Mexico Tech., March 1986.