A Gaussian Elimination Algorithm for the Enumeration of Cut Sets in a Graph

ALBERTO MARTELLI

Istituto di Elaborazione della Informazione, Pisa, Italy

ABSTRACT. By defining a suitable algebra for cut sets, it is possible to reduce the problem of enumerating the cut sets between all pairs of nodes in a graph to the problem of solving a system of linear equations An algorithm for solving this system using Gaussian elimination is presented in this paper The efficiency of the algorithm depends on the implementation of sum and multiplication Therefore, some properties of cut sets are investigated, which greatly simplify the implementation of these operations for the case of undirected graphs. The time required by the algorithm is shown to be linear with the number of cut sets for complete graphs Some experimental results are given, proving that the efficiency of the algorithm increases by increasing the number of pairs of nodes for which the cut sets are computed.

KEY WORDS AND PHRASES: graph theory, cut set enumeration, regular algebra, Gaussian elimination

CR CATEGORIES. 5 20, 5 32

1. *Introduction*

The problem of enumerating all cut sets between all pairs of nodes in a graph is an important combinatorial problem. For example, it can be the first step for the computation of the terminal reliability in a communication network [1, 2, 3].

Several methods have been presented for enumerating all cut sets between two given nodes i and j. For instance, the method described in [2] first computes all simple paths and then finds all combinations of arcs cutting all paths. Another method [3] searches the given graph starting from node i and constructs a tree whose terminal nodes are the cut sets. Furthermore, the cut sets could be obtained by taking all linear combinations of the cut sets belonging to a suitably constructed set [4].

A regular algebra method was presented by the author in [5]. By giving a suitable algebra for cut sets, it is possible to reduce the problem of enumerating all cut sets to the problem of solving a system of linear equations in this algebra. Using this method, it is possible to compute simultaneously the cut sets between all pairs of nodes. A similar approach has been used for enumerating all simple paths in a graph [6] or for finding shortest paths [7].

In Section 2 we present our method by giving the algebra for cut sets and by showing how our problem becomes one of solving a system of linear equations. Moreover, we give a Gaussian elimination algorithm for solving this system.

A simple implementation of sum and multiplication in our algebra would make the Gaussian elimination algorithm very inefficient. Therefore, in Sections 3 and 4 we show how these operations can be implemented in an efficient way for undirected graphs by taking into account some properties of cut sets. Some aspects regarding the computa-

Author's address · Consiglio Nazionale delle Ricerche, Istituto di Elaborazione della Informazione Pisa, Italy.

tional complexity of the algorithm are analyzed in Section 5. Eventually, some experimental results are given, showing that satisfactory computing times can be obtained with our implementation of sum and multiplication.

2. *An Algebra for Cut Sets*

Let $G = (N, A)$ be a directed graph, where the set of nodes N is $N = \{1, 2, \cdots, n\}$. An *i-j cut set* is a set of arcs such that, by removing these arcs, there is no path in G from node i to node j. A *minimal i-j cut set* is an i-j cut set such that no subset of it is an i-j cut set.

In this section we give an algebra for minimal cut sets and show that the problem of enumerating all minimal i-j cut sets can be reduced to the problem of solving a system of linear equations in this algebra.

The algebra $C = (S, +, \cdot)$ consists of a set S with two binary operations, sum and multiplication. An element α of S is defined as a set of sets of arcs of G such that if a set of arcs s is an element of α, there is no other element of α which is a subset of s. For example, $\{\{(2\ 3)\}, \{(1\ 3), (2\ 4)\}\}$ is an element of S and $\{\{(2\ 3)\}, \{(1\ 3), (2\ 3)\}\}$ is not. The last requirement has been introduced because we want to consider only minimal cut sets.

If α and β are two elements of S, sum and multiplication are defined as follows:

$\alpha + \beta$ is a set obtained by making the union of each element of α with each element of β, and then deleting all elements which are a superset[1] of some other element,

$\alpha\beta$ is a set obtained by making the union of α and β, and then deleting all elements which are a superset of some other element.

For example, if $\alpha = \{\{(2\ 3)\}, \{(1\ 3), (2\ 4)\}\}$ and $\beta = \{\{(1\ 3), (2\ 3)\}, \{(1\ 3), (2\ 4)\}\}$ we have $\alpha + \beta = \{\{(1\ 3), (2\ 3)\}, \{(1\ 3), (2\ 4)\}\}$ and $\alpha\beta = \{\{(2\ 3)\}, \{(1\ 3), (2\ 4)\}\}$.

The zero element $\varnothing$ is the set whose only element is the empty set, and the unit element e is the empty set.

It is possible to show [5] that this algebra is a regular algebra, i.e. that it satisfies the set of axioms which were originally defined for regular expressions [8].

Given any regular algebra R, we can form a new regular algebra consisting of all $n \times n$ matrices whose elements belong to R. If $A = [a_{ij}]$ and $B = [b_{ij}]$ are two $n \times n$ matrices, sum and multiplication are

$$A + B = [a_{ij} + b_{ij}] \quad \text{and} \quad AB = \left[\sum_{k=1}^{n} a_{ik}b_{kj}\right].$$

The unit matrix $E = [e_{ij}]$ is the $n \times n$ matrix with $e_{ij} = e$ if $i = j$ and $e_{ij} = \varnothing$ if $i \neq j$. The powers of a matrix are $A^0 = E$, $A^k = A^{k-1}A$ $(k = 1, 2, \cdots)$. The *closure* of A is $A^* = \sum_{k=0}^{\infty} A^k$.

By the axioms of regular algebra, it is easy to see that the closure A^* can be obtained as a solution of the equation $Y = AY + E$. This equation suggests an analogy with linear algebra and, in fact, it has been shown [7] that it is possible to define algorithms for solving it, similar to the algorithms of linear algebra. These results can be applied to our problem because the problem of enumerating all minimal i-j cut sets can be reduced to the problem of obtaining the closure of a given matrix.

Let G be a directed graph with n nodes. An $n \times n$ matrix A, whose elements belong to the previously defined algebra C, is constructed by setting $a_{ij} = \varnothing$ (the zero element of the algebra) if there is no arc $(i\ j)$ and $a_{ij} = \{\{(i\ j)\}\}$ otherwise. For example, we can build the following matrix for the graph in Figure 1:

$$A = \begin{vmatrix} \varnothing & \{\{(1\ 2)\}\} & \varnothing & \{\{(1\ 4)\}\} \\ \varnothing & \varnothing & \varnothing & \{\{(2\ 4)\}\} \\ \{\{(3\ 1)\}\} & \{\{(3\ 2)\}\} & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing \end{vmatrix}$$

[1] We say that a is a superset of b whenever $a \supseteq b$.

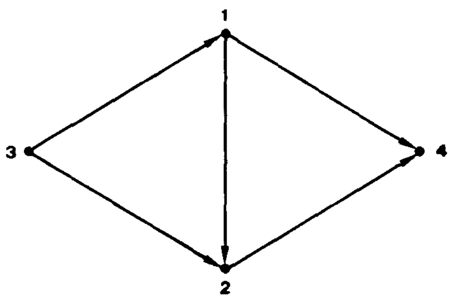

FIG. 1. A directed graph

We can prove that the elements a^*_{ij} of the closure A^* of matrix A give the set of all minimal i-j cut sets for the graph G [5]. For instance, for the graph in Figure 1 we have

$$A^2 = \left|\begin{array}{cccc} \varnothing & \varnothing & \varnothing & \{\{(1\ 2)\},\ \{(2\ 4)\}\} \\ \varnothing & \varnothing & \varnothing & \varnothing \\ \varnothing & \{\{(3\ 1)\},\ \{(1\ 2)\}\} & \varnothing & \begin{array}{c}\{\{(3\ 1),\ (3\ 2)\},\ \{(3\ 1),\ (2\ 4)\}, \\ \{(1\ 4),\ (3\ 2)\},\ \{(1\ 4),\ (2\ 4)\}\}\end{array} \\ \varnothing & \varnothing & \varnothing & \varnothing \end{array}\right|,$$

$$A^3 = \left|\begin{array}{cccc} \varnothing & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \{\{(3\ 1)\},\ \{(1\ 2)\},\ \{(2\ 4)\}\} \\ \varnothing & \varnothing & \varnothing & \varnothing \end{array}\right|$$

A^k is the zero matrix for $k > 3$.

$$A^* = E + A + A^2 + A^3 = \left|\begin{array}{cccc} e & \{\{(1\ 2)\}\} & \varnothing & \{\{(1\ 2),\ (1\ 4)\},\ \{(2\ 4),\ (1\ 4)\}\} \\ \varnothing & e & \varnothing & \{\{(2\ 4)\}\} \\ \{\{(3\ 1)\}\} & \begin{array}{c}\{\{(3\ 1),\ (3\ 2)\}, \\ \{(1\ 2),\ (3\ 2)\}\}\end{array} & e & \begin{array}{c}\{\{(3\ 1),\ (3\ 2)\},\ \{(3\ 1),\ (2\ 4)\}, \\ \{(3\ 2),\ (1\ 2),\ (1\ 4)\},\ \{(1\ 4),\ (2\ 4)\}\}\end{array} \\ \varnothing & \varnothing & \varnothing & e \end{array}\right|.$$

In our algebra C the additional axiom $e + \alpha = e$ holds. As a consequence of this axiom, the closure of a matrix is given by

$$A^* = E + A + A^2 + \cdots + A^r = E + A + A^2 + \cdots + A^{n-1} \quad \text{for all} \quad r \geq n - 1.$$

As pointed out before, the closure A^* can also be obtained by solving a system of linear equations in our algebra. A simple method of solution is Gaussian elimination, for which the following algorithm can be given. This algorithm computes, given r and c, the submatrix $[a^*_{ij}]$ of A^* with $r \leq i \leq n$ and $c \leq j \leq n$.

Algorithm G

```
for $i$ := 1 until $n$ do
for $j$ := 1 until $n$ do if $i = j$ then $b^0_{ij} := e$ else $b^0_{ij} := a_{ij}$;
for $h$ := 1 until $n$ do
for $i$ := $min(h + 1, r)$ until $n$ do
for $j$ := $min(h + 1, c)$ until $n$ do
if $i \neq h$ and $j \neq h$ then
  $b^h_{ij} := b^{h-1}_{ij} + b^{h-1}_{ih} b^{h-1}_{hj}$
else
  $b^h_{ij} := b^{h-1}_{ij}$.
```

Similar algorithms have been used for solving path finding problems in suitable regular algebras. For example, both Floyd's algorithm for finding the shortest path [9] and Warshall's algorithm for finding the transitive closure of a graph [10] can be considered as Gaussian elimination algorithms.

If we apply Algorithm G with $r = 1$ and $c = 1$ to our example, we get

$$B^1 = \begin{vmatrix} e & \{\{(1\ 2)\}\} & \varnothing & \{\{(1\ 4)\}\} \\ \varnothing & e & \varnothing & \{\{(2\ 4)\}\} \\ \{\{(3\ 1)\}\} & \begin{matrix}\{\{(3\ 1),\ (3\ 2)\}, \\ \{(1\ 2),\ (3\ 2)\}\}\end{matrix} & e & \{\{(3\ 1)\},\ \{(1\ 4)\}\} \\ \varnothing & \varnothing & \varnothing & e \end{vmatrix},$$

$$A^* = B^4 = B^3 = B^2 = \begin{vmatrix} e & \{\{(1\ 2)\}\} & \varnothing & \{\{(1\ 2),\ (1\ 4)\},\ \{(2\ 4),\ (1\ 4)\}\} \\ \varnothing & e & \varnothing & \{\{(2\ 4)\}\} \\ \{\{(3\ 1)\}\} & \begin{matrix}\{\{(3\ 1),\ (3\ 2)\}, \\ \{(1\ 2),\ (3\ 2)\}\}\end{matrix} & e & \begin{matrix}\{\{(3\ 1),\ (3\ 2)\},\ \{(3\ 1),\ (2\ 4)\}, \\ \{(3\ 2),\ (1\ 2),\ (1\ 4)\},\ \{(1\ 4),\ (2\ 4)\}\}\end{matrix} \\ \varnothing & \varnothing & \varnothing & e \end{vmatrix}.$$

3. *Some Implementations of Sum*

By a simple analysis of Algorithm G, with $r = c = 1$, we see that the following operation s repeated $O(n^3)$ times:

$$b_{ij}^h := b_{ij}^{h-1} + b_{ih}^{h-1}\, b_{hj}^{h-1} \qquad (h, i, j = 1, \cdots, n), \tag{3.1}$$

where the elements b_{ij}^h are sets of cut sets. Since the cardinality of b_{ij}^h can be exponential with n, the complexity of the computation will depend, in general, on the implementation of (3.1).

The following Algorithm SUM1 gives a simple implementation of the sum of two sets of cut sets $s1$ and $s2$. The function *choose* (s) returns an element of the set s.

Algorithm SUM1

```
sum := ∅;                                [see footnote 2]
x1 := s1;
while x1 ≠ ∅ do
begin
  α := choose (x1),
  x1 := x1 − {α};
  x2 := s2;
  while x2 ≠ ∅ do
  begin
    β := choose (x2);
    x2 := x2 − {β};
    γ := α ∪ β;
    x3 := sum;
    while x3 ≠ ∅ do
    begin
      δ := choose (x3);
      x3 := x3 − {δ};
      if δ ⊆ γ then goto l;
      if δ ⊃ γ then sum := sum − {δ}
    end;
    sum := sum ∪ {γ}
l. end
end.
```

Therefore, (3.1) can be implemented by first computing the set $(b_{ih}^{h-1} \cup b_{hj}^{h-1})$ and then computing the sum of this set with b_{ij}^{h-1} using Algorithm SUM1.

A simple analysis of Algorithm SUM1 shows that this implementation is very inefficient. In fact, let p and q be the cardinalities of the sets $s1$ and $s2$ and let r be the cardinality of the set $(s1 + s2)$, and let us compute the number of comparisons performed by this algorithm. Since every element of the set $(s1 + s2)$ will at least be compared with all other elements of $(s1 + s2)$ obtained before itself, we need at least $r(r - 1)/2$ compari-

[2] Whereas in Section 2 the symbol $\varnothing$ denotes the zero element of the algebra, in the sequel it will denote the empty set The zero element will be denoted by $\{\varnothing\}$.

sons for the elements of $(s1 + s2)$. The minimal number of comparisons $c_{\min}$ for Algorithm SUM1 is obtained by assuming that the remaining $(pq - r)$ elements are eliminated with only one comparison: $c_{\min} = (pq - r) + r(r - 1)/2$.

On the other hand, we can assume that, in the worst case, the set *sum* will reach the cardinality $(pq - 1)$, and only the last element will delete all nonminimal cut sets, thus reducing the cardinality of *sum* to r. Therefore, the maximal number of comparisons is $c_{\max} = pq(pq - 1)$.

A more realistic analysis of this algorithm can be performed by assuming that the cardinality of the set *sum* grows in a uniform way from 0 to r, and that one comparison will be required for the first pq/r elements, two comparisons for the next pq/r elements, and so on. Thus we can conclude that Algorithm SUM1 performs, on the average, $O(pqr)$ comparisons.

Because such an implementation is too inefficient, we show in this and the following section how to implement (3.1) in more efficient ways *for undirected graphs*, by taking into account some properties of Gaussian elimination.

Let $G = (N, A)$ be an n-node undirected graph where $N = \{1, 2, \cdots, n\}$. Let $G^h = (N^h, A^h)$ $(h = 1, \cdots, n)$ be the subgraph[3] of G corresponding to the set of nodes $N^h = \{1, 2, \cdots, h\}$. Of course, we have $G^n = G$. Let G_{ij}^h $(h = 1, \cdots, n)$ be the subgraph of G corresponding to the set of nodes $N^h \cup \{i\} \cup \{j\}$. Furthermore, let G_{ij}^0 be the subgraph of G corresponding to the nodes i and j.

A useful property of the Gaussian elimination algorithm is stated in the following theorem.

THEOREM 3.1. *Every b_{ij}^h $(h = 0, 1, \cdots, n)$ computed by Algorithm G gives the set of all minimal i-j cut sets of graph G_{ij}^h.*

PROOF. We will prove the theorem by induction on h. It is certainly true for $h = 0$. Let us assume that it holds for $h - 1$; we show that it holds for h.

First we show that every element of b_{ij}^h is an i-j cut set of G_{ij}^h. Let $\alpha \in b_{ij}^{h-1}$ and $\beta \in b_{ih}^{h-1}$. We show that $\alpha \cup \beta$ is an i-j cut set of G_{ij}^h. Assume it is not. Then there must be a path in G_{ij}^h from i to j not containing any arc of $\alpha \cup \beta$. But, if this path does not pass through node h, it is also a path of G_{ij}^{h-1} and thus α is not an i-j cut set of G_{ij}^{h-1}, contradicting the hypothesis. If the path passes through node h, then β is not an i-h cut set in G_{ih}^{h-1}, still contradicting the hypothesis. Therefore, $\alpha \cup \beta$ must be an i-j cut set of G_{ij}^h. Analogously if β belongs to b_{hj}^{h-1}.

Now we show that every minimal i-j cut set of G_{ij}^h must be an element of b_{ij}^h. In fact, let α be a minimal i-j cut set of G_{ij}^h. There must be a subset β of α which is a minimal i-j cut set of G_{ij}^{h-1}, since G_{ij}^{h-1} is a subgraph of G_{ij}^h. Let us assume now that α divides G_{ij}^h in such a way that node h is disconnected from i. Then there is a subset γ which is a minimal i-h cut set in G_{ih}^{h-1}. By the induction hypothesis we have $\beta \in b_{ij}^{h-1}$ and $\gamma \in b_{ih}^{h-1}$ and, thus, $\alpha = \beta \cup \gamma$ must belong to b_{ij}^h. Similarly, if α disconnects h from j, there must be a subset δ of α such that $\delta \in b_{hj}^{h-1}$, and $\alpha = \beta \cup \delta$ must belong to b_{ij}^h. Q.E.D.

Let us now consider the computation of (3.1). We can have three cases:

(i) $b_{ih}^{h-1} = \{\varnothing\}$ or $b_{hj}^{h-1} = \{\varnothing\}$. In this case we have $b_{ij}^h = b_{ij}^{h-1}$.

(ii) $b_{ij}^{h-1} = \{\varnothing\}$. This means that i and j are not connected in G_{ij}^{h-1} and thus each path from i to j in G_{ij}^h must pass through node h. It is easy to see that each element of b_{ih}^{h-1} and each element of b_{hj}^{h-1} are disjoint and therefore b_{ij}^h is simply given by $b_{ih}^{h-1} \cup b_{hj}^{h-1}$.

(iii) $b_{ij}^{h-1} \neq \{\varnothing\}$ and $b_{ih}^{h-1} \neq \{\varnothing\}$ and $b_{hj}^{h-1} \neq \{\varnothing\}$. This general case will be considered in the remainder of this section.

Let us rewrite (3.1) as

$$b_{ij}^h := (b_{ij}^{h-1} + b_{ih}^{h-1})(b_{ij}^{h-1} + b_{hj}^{h-1}). \tag{3.2}$$

[3] A graph $G' = (N', A')$ is called a subgraph of $G = (N, A)$ iff $N' \subseteq N$ and $A' = A \cap (N' \times N')$, i.e. if A' contains exactly all arcs of A whose both extrema are in N'.

Therefore, to get b^h_{ij}, we make the union of each element $\alpha \in b^{h-1}_{ij}$ with each element $\beta \in b^{h-1}_{ih}$ and of each element $\alpha \in b^{h-1}_{ij}$ with each element $\gamma \in b^{h-1}_{hj}$ and, finally, we keep only the elements which are minimal i-j cut sets for graph G^h_{ij}.

If we use Algorithm SUM1, in order to decide whether a cut set is minimal or not, we have to compare it with all other cut sets. The following theorems show that it is possible to decide whether a cut set $\alpha \cup \beta$ ($\alpha \in b^{h-1}_{ij}$, $\beta \in b^{h-1}_{ih}$, or $\beta \in b^{h-1}_{hj}$) is minimal without comparing it with any other cut set, thus obtaining a more efficient implementation of sum. This can be achieved by keeping some information about how each cut set separates the nodes of the graph.

Since we are considering case (iii), the nodes i and j are connected in G^h_{ij}. Therefore we can define the graph $F^h_{ij} = (N^h_{ij}, A^h_{ij})$ as the maximal connected subgraph of G^h_{ij} containing nodes i and j. Of course, the elements of b^h_{ij} are the minimal i-j cut sets of F^h_{ij}.

An example is given in Figure 2. The subgraphs G^4_{79} and F^4_{79} are given in Figures 3(a) and 3(b), respectively.

Let $\alpha \in b^h_{ij}$ be a minimal i-j cut set of F^h_{ij}. It is easy to show that it divides F^h_{ij} into exactly two connected subgraphs Let $N_\alpha{}^i$ and $N_\alpha{}^j$ be the two subsets of nodes into which N^h_{ij} is partitioned by α, such that $i \in N_\alpha{}^i$ and $j \in N_\alpha{}^j$. We have $N_\alpha{}^i \cup N_\alpha{}^j = N^h_{ij}$ and $N_\alpha{}^i \cap N_\alpha{}^j = \emptyset$, where $\emptyset$ denotes the empty set.

Let us now consider the extrema of the arcs of α. For every arc, one extremum must belong to $N_\alpha{}^i$ and the other one to $N_\alpha{}^j$; thus, let us denote with $M_\alpha{}^i$ and $M_\alpha{}^j$ the two sets of extrema of α such that $M_\alpha{}^i \subseteq N_\alpha{}^i$ and $M_\alpha{}^j \subseteq N_\alpha{}^j$.

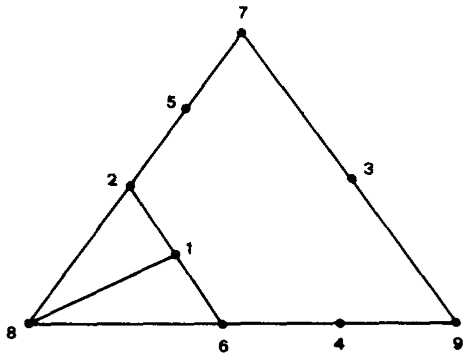

FIG. 2. An undirected graph

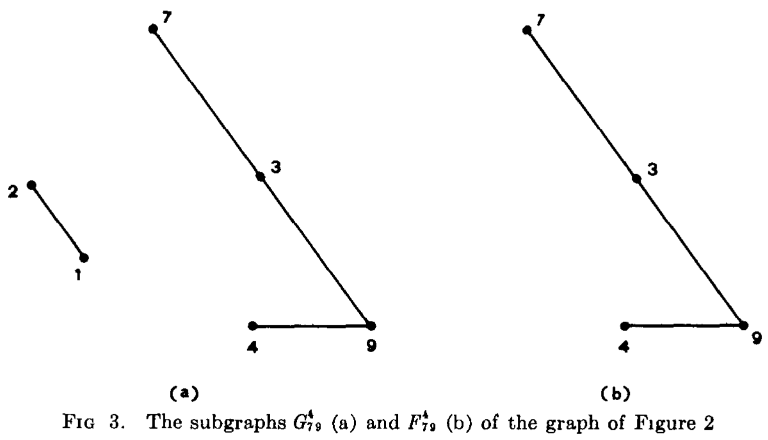

FIG 3. The subgraphs G^4_{79} (a) and F^4_{79} (b) of the graph of Figure 2

For our example in Figure 2, if we take $\alpha = \{(3\ 7)\}$ as a minimal 7-9 cut set of F_{79}^4, then we have $N_\alpha^7 = \{7\}$, $N_\alpha^9 = \{3, 4, 9\}$, $M_\alpha^7 = \{7\}$, and $M_\alpha^9 = \{3\}$.

Let us now assume that for each minimal i-j cut set $\alpha \in b_{ij}^h$, the sets N_α^i, N_α^j, M_α^i, M_α^j are known. The information contained in these sets can be used for deciding whether a cut set $\alpha \cup \beta$ is minimal or not. A necessary condition is given by Theorem 3.2. Let us first prove the following lemma.

LEMMA 3.1. *Let $\alpha \in b_{ij}^{h-1}$ and $\beta \in b_{ih}^{h-1}$. Then any arc $(n_1\, n_2)$ of F_{ij}^h such that $n_1 \in N_\alpha^i \cap N_\beta^i$ and $n_2 \notin N_\alpha^i \cap N_\beta^i$ must belong to $\alpha \cup \beta$.*

PROOF. Let us assume that $(n_1\, n_2)$ belongs neither to α nor to β, and that $n_2 \neq h$ and $n_2 \neq j$. Then, the node n_2 must belong to N_α^i, because it is connected to $n_1 \in N_\alpha^i$ through arc $(n_1\, n_2)$ in F_{ij}^{h-1}. Analogously, n_2 must belong to N_β^i, because it is connected to $n_1 \in N_\beta^i$ in F_{ih}^{h-1}. Hence, n_2 must belong to $N_\alpha^i \cap N_\beta^i$, contradicting our hypothesis.

On the other hand, if $n_2 = h$ and if $(n_1\, n_2)$ does not belong to β, then node i will be connected to node h through node n_1, thus contradicting the hypothesis that β is a minimal i-h cut set in F_{ih}^{h-1}. Similarly, if $n_2 = j$, then $(n_1\, n_2)$ must belong to α. Q.E.D.

THEOREM 3.2. *Let $\alpha \in b_{ij}^{h-1}$ and $\beta \in b_{ih}^{h-1}$. If $\alpha \cup \beta$ is a minimal i-j cut set of F_{ij}^h, then we have*

$$M_\alpha^i \subseteq N_\beta^i \quad \text{and} \quad M_\beta^i \subseteq N_\alpha^i. \tag{3.3}$$

PROOF. According to Lemma 3.1, the i-j cut set $\alpha \cup \beta$ disconnects the set of nodes $N_\alpha^i \cap N_\beta^i$ from j in F_{ij}^h. Assume now that $M_\beta^i \nsubseteq N_\alpha^i$. Then there is an arc $(n_1\, n_2)$ of β, with $n_1 \in M_\beta^i$, such that $n_1 \notin N_\alpha^i$ and, therefore, $n_1 \notin N_\alpha^i \cap N_\beta^i$. Since n_2 belongs to M_β^h it cannot belong to N_β^i and, therefore, $n_2 \notin N_\alpha^i \cap N_\beta^i$. If we now remove arc $(n_1\, n_2)$ from the set $\alpha \cup \beta$, we still have an i-j cut set of F_{ij}^h, because the set of nodes $N_\alpha^i \cap N_\beta^i$ is still disconnected from j. Therefore, $\alpha \cup \beta$ is not a minimal i-j cut set. Analogously if $M_\alpha^i \nsubseteq N_\beta^i$. Q.E.D.

This theorem gives a necessary condition for an i-j cut set $\alpha \cup \beta$ to be minimal. For example, let us consider the graph in Figure 2 and let $\alpha = \{(1\ 2), (1\ 8), (6\ 8)\}$ be a minimal 8-9 cut set of F_{89}^6 and $\beta = \{(1\ 2), (2\ 8)\}$ be a minimal 8-7 cut set of F_{87}^6. We see that $\alpha \cup \beta = \{(1\ 2), (1\ 8), (2\ 8), (6\ 8)\}$ is not a minimal 8-9 cut set of F_{89}^7, since $M_\beta^8 = \{1, 8\}$ is not a subset of $N_\alpha^8 = \{2, 5, 8\}$. Instead, if we have $\alpha = \{(1\ 6), (6\ 8)\}$, then $\alpha \cup \beta = \{(1\ 6), (6\ 8), (1\ 2), (2\ 8)\}$ could be a minimal 8-9 cut set of F_{89}^7, since $M_\beta^8 = \{1, 8\} \subseteq N_\alpha^8 = \{1, 2, 5, 8\}$ and $M_\alpha^8 = \{1, 8\} \subseteq N_\beta^8 = \{1, 4, 6, 8\}$.

The next lemma and theorem give sufficient conditions for minimal cut sets

LEMMA 3.2. *If $\alpha \in b_{ij}^{h-1}$ and $\beta \in b_{ih}^{h-1}$ satisfy conditions (3.3), then F_{ij}^h is divided by $\alpha \cup \beta$ into two subgraphs corresponding to the two subsets of nodes $N_{\alpha\cup\beta}^i = N_\alpha^i \cap N_\beta^i$, $N_{\alpha\cup\beta}^j = N_\alpha^j \cup N_\beta^h$.*

PROOF. Since, according to Theorem 3.2, every arc of $\alpha \cup \beta$ has one and only one extremum belonging to $N_\alpha^i \cap N_\beta^i$, F_{ij}^h is divided by $\alpha \cup \beta$ into two subgraphs corresponding to $N_{\alpha\cup\beta}^i = N_\alpha^i \cap N_\beta^i$ and $N_{\alpha\cup\beta}^j = N_{ij}^h - N_{\alpha\cup\beta}^i$ It is easy to see that $N_{ij}^h = N_{ij}^{h-1} \cup N_{ih}^{h-1}$, i.e. $N_{ij}^h = N_\alpha^i \cup N_\alpha^j \cup N_\beta^i \cup N_\beta^h$.

Hence we have $N_{\alpha\cup\beta}^j = (N_\alpha^i \cup N_\alpha^j \cup N_\beta^i \cup N_\beta^h) - (N_\alpha^i \cap N_\beta^i)$. Assume now that a node k belongs to N_α^i but not to N_β^i. Since α is a minimal i-j cut set of F_{ij}^{h-1}, node k must be connected to node i in F_{ij}^{h-1}. This means that node k must also belong to N_{ih}^{h-1} and thus $k \in N_\beta^h$, since $k \notin N_\beta^i$. Analogously, every node belonging to N_β^i which does not belong to N_α^i must belong to N_α^j. Therefore, every node of $N_{\alpha\cup\beta}^j$ must belong either to N_α^j or to N_β^h, i.e. $N_{\alpha\cup\beta}^j = N_\alpha^j \cup N_\beta^h$. Q.E.D.

THEOREM 3.3. *If $\alpha \in b_{ij}^{h-1}$ and $\beta \in b_{ih}^{h-1}$ satisfy conditions (3.3), then $\alpha \cup \beta$ is a minimal i-j cut set of F_{ij}^h if and only if the two subgraphs of F_{ij}^h corresponding to $(N_\alpha^i \cap N_\beta^i)$ and $(N_\alpha^j \cup N_\beta^h)$ are connected.*

PROOF. Assume that the two subgraphs are connected. We know that every arc of $\alpha \cup \beta$ has one extremum in $(N_\alpha^i \cap N_\beta^i)$ and the other one in $(N_\alpha^j \cup N_\beta^h)$. Therefore, by removing any arc from $\alpha \cup \beta$, we obtain a path connecting i with j.

Assume now that one of the two subgraphs is not connected. Then F^h_{ij} is divided by $\alpha \cup \beta$ into more than two connected subgraphs and thus $\alpha \cup \beta$ is not minimal. Q.E.D.

For instance, if $\alpha = \{(1\ 6), (6\ 8)\}$ is a minimal 8-9 cut set of F^6_{89} for the graph in Figure 2, and if $\beta = \{(1\ 2), (2\ 8)\}$ is a minimal 8-7 cut set of F^6_{87}, then conditions (3.3) are satisfied. Furthermore, the subgraph corresponding to the nodes $N^8_{\alpha\cup\beta} = N_\alpha{}^8 \cap N_\beta{}^8 = \{1, 8\}$ is connected, and the subgraph corresponding to the nodes $N^9_{\alpha\cup\beta} = N_\alpha{}^9 \cup N_\beta{}^7 = \{2, 3, 4, 5, 6, 7, 9\}$ is connected. Hence, $\alpha \cup \beta$ is a minimal 8-9 cut set of F^7_{89}.

According to this theorem, in order to decide whether a cut set $\alpha \cup \beta$ is minimal, we have to check that two subgraphs are connected. Theorem 3.4 shows that, in most cases, this operation can be performed very simply.

THEOREM 3.4. *If $\alpha \in b^{h-1}_{ij}$ and $\beta \in b^{h-1}_{ih}$ satisfy conditions* (3.3) *and if the following two conditions hold:*

$$N_\alpha{}^i \subseteq N_\beta{}^i \quad \text{or} \quad N_\beta{}^i \subseteq N_\alpha{}^i, \quad N_\alpha{}^j \cap N_\beta{}^h \neq \emptyset \quad \text{or there is an arc } (h\ j) \text{ in } G, \tag{3.4}$$

then $\alpha \cup \beta$ is a minimal i-j cut set of F^h_{ij}.

PROOF. If the first condition holds, then $N^i_{\alpha\cup\beta}$ is either $N_\alpha{}^i$ or $N_\beta{}^i$. The subgraphs of F^h_{ij} corresponding to $N_\alpha{}^i$ and $N_\beta{}^i$ are connected because α and β are minimal cut sets; hence the subgraph corresponding to $N^i_{\alpha\cup\beta}$ is connected.

If the second condition holds, then the subgraph of F^h_{ij} corresponding to $N^j_{\alpha\cup\beta} = N_\alpha{}^j \cup N_\beta{}^h$ is connected, since the subgraphs of F^h_{ij} corresponding to $N_\alpha{}^j$ and $N_\beta{}^h$ are connected and either they have a common node or they are connected through arc $(h\ j)$.

The theorem follows from Theorem 3.3. Q.E.D.

For instance, for the graph in Figure 2, if $\alpha = \{(1\ 8), (2\ 8), (6\ 8)\}$ is a minimal 8-9 cut set of F^6_{89} and $\beta = \{(1\ 8), (2\ 8), (6\ 8)\}$ is a minimal 8-7 cut set of F^6_{87}, then conditions (3.3) are satisfied. Moreover we have $N_\alpha{}^8 = \{8\}$, $N_\alpha{}^9 = \{1, 2, 3, 4, 5, 6, 9\}$, $N_\beta{}^8 = \{8\}$, $N_\beta{}^7 = \{1, 2, 3, 4, 5, 6, 7\}$. Conditions (3.4) are satisfied and $\alpha \cup \beta = \{(1\ 8), (2\ 8), (6\ 8)\}$ is a minimal 8-9 cut set of F^7_{89}.

The lemmas and theorems stated so far refer to the computation of the first terms in (3.2). Similar lemmas and theorems can be stated for $\alpha \in b^{h-1}_{ij}$ and $\beta \in b^{h-1}_{hj}$ by simply substituting $N_\alpha{}^i$ with $N_\alpha{}^j$, $M_\alpha{}^i$ with $M_\alpha{}^j$, $N_\alpha{}^j$ with $N_\alpha{}^i$, $N_\beta{}^i$ with $N_\beta{}^j$, $M_\beta{}^i$ with $M_\beta{}^j$.

The results of the previous theorems are used by Algorithm SUM2 for computing $(b^{h-1}_{ij} + b^{h-1}_{ih})$. (More precisely, Algorithm SUM2 computes only those elements of $(b^{h-1}_{ij} + b^{h-1}_{ih})$ which are minimal *i-j* cut sets of F^h_{ij}.)

```
Algorithm SUM2
sum := $\emptyset$;
x1 := $b^{h-1}_{ij}$;
while x1 $\neq \emptyset$ do
begin
  $\alpha$ := choose (x1);
  x1 := x1 − $\{\alpha\}$;
  x2 := $b^{h-1}_{ih}$;
  while x2 $\neq \emptyset$ do
  begin
    $\beta$ := choose (x2);
    x2 := x2 − $\{\beta\}$,
    if $M_\alpha{}^i \subseteq N_\beta{}^i$ and $M_\beta{}^i \subseteq N_\alpha{}^i$ and
      ($N_\alpha{}^i \subseteq N_\beta{}^i$ or $N_\beta{}^i \subseteq N_\alpha{}^i$ or connected ($N_\alpha{}^i \cap N_\beta{}^i$)) and ($N_\alpha{}^j \cap N_\beta{}^h \neq \emptyset$ or there-is-an-arc (h, j))
        then
    begin
      $N^i_{\alpha\cup\beta}$ := $N_\alpha{}^i \cap N_\beta{}^i$;
      $N^j_{\alpha\cup\beta}$ := $N_\alpha{}^j \cup N_\beta{}^h$;
      $M^i_{\alpha\cup\beta}$ := $M_\alpha{}^i \cup M_\beta{}^i$,
      $M^j_{\alpha\cup\beta}$ := $M_\alpha{}^j \cup M_\beta{}^h$;
      sum := sum $\cup$ $\{\alpha \cup \beta\}$
    end
  end
end.
```

Of course, an analogous algorithm can be given for computing the second term of (3 2), and b_{ij}^h can be obtained by making the union of the two terms.

If we have $|b_{ij}^{h-1}| = p$ and $|b_{ih}^{h-1}| = q$, then the body of the inner loop will be executed pq times. The most difficult operation which can be executed in the inner loop is the call to the procedure *connected*, which detects whether a subgraph corresponding to a given set of nodes is connected. This operation can take $O(n^2)$ steps. Therefore, the computation of b_{ij}^h using Algorithm SUM2 will require, in the worst case, $O(pq\,n^2)$ steps.

The computation of b_{ij}^h using Algorithm SUM1 requires, on the average, $O(pqr)$ comparison. Since cut sets will be implemented as bit patterns of a computer word, we can assume that a comparison takes a constant time, independent of the size of the graph. However, the cardinality r of b_{ij}^h can be $O(2^n)$, and thus, Algorithm SUM2 gives an improvement on Algorithm SUM1.

4. *An Efficient Implementation of Sum Using Topological Sorting*

In order to compute $(b_{ij}^{h-1} + b_{ih}^{h-1})$ using Algorithm SUM2, we have to compare each element $\alpha \in b_{ij}^{h-1}$ with each element $\beta \in b_{ih}^{h-1}$. In this section we show how to improve Algorithm SUM2 by reducing the number of the comparisons. This can be achieved by assuming that the elements b_{ij}^h, instead of being unordered sets of cut sets, are suitably ordered *lists* of cut sets. Precisely, the elements of b_{ij}^h will be *topologically sorted with respect to the sets* N_α^i; i.e. the elements α_k of b_{ij}^h are arranged into a linear sequence $\alpha_1, \alpha_2, \cdots$ such that whenever $N_{\alpha_k}^i \subset N_{\alpha_l}^i$ we have $k < l$.

Let us consider, for instance, the graph in Figure 4. A possible arrangement of the elements of b_{45}^2 is the following:

$$b_{45}^2 = (\{(1\ 4), (2\ 4)\}, \{(1\ 2), (1\ 5), (2\ 4)\}, \{(1\ 2), (1\ 4), (2\ 5)\}, \{(1\ 5), (2\ 5)\}).$$

In fact, the sets N_α^4 of these cut sets are, respectively $\{4\}, \{1, 4\}, \{2, 4\}, \{1, 2, 4\}$.

The next theorem shows how Algorithm SUM2 can be improved by assuming that b_{ij}^{h-1} and b_{ih}^{h-1} are topologically sorted lists. Let us prove first the following lemma.

LEMMA 4.1. *Let α and β be two distinct minimal i-j cut sets of F_{ij}^h. If $N_\beta^i \not\subset N_\alpha^i$, then we have $M_\beta^i \not\subset N_\alpha^i$.*

PROOF. If $N_\beta^i = N_\alpha^i$, then α and β must be equal because they divide the same subgraphs in F_{ij}^h. Let us assume therefore that $N_\beta^i \not\subseteq N_\alpha^i$ and let us suppose that $M_\beta^i \subset N_\alpha^i$. Let us consider the set $\bar{N}_\beta^i$ of nodes of N_β^i which do not belong to N_α^i: $\bar{N}_\beta^i = N_\beta^i - (N_\beta^i \cap N_\alpha^i)$. We can easily see that there is no arc of F_{ij}^h connecting a node of $\bar{N}_\beta^i$ with a node of N_β^j. In fact, if there were such an arc, it would belong to β and therefore we would have $M_\beta^i \not\subset N_\alpha^i$.

Let us now consider the set N_α^j. We have $N_\alpha^j = (N_\alpha^j \cap N_\beta^j) \cup \bar{N}_\beta^i$, because $\bar{N}_\beta^i$ is not contained in N_α^i. But we have just shown that there is no arc between any node of $\bar{N}_\beta^i$ and any node of N_β^j. Therefore, the subgraph of F_{ij}^h corresponding to N_α^j is not connected and α is not minimal, thus contradicting the hypothesis. Q.E.D.

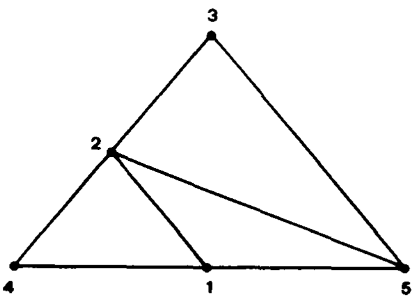

FIG 4 The computation of $(b_{45}^2 + b_{43}^2)$ for this graph using Algorithm SUM3 is summarized in Table I

We can now prove the following theorem.

THEOREM 4.1. *Let $N_\alpha^i \subseteq N_\beta^i$ with $\alpha \in b_{ij}^{h-1}$ and $\beta \in b_{ih}^{h-1}$. Then, for each $\gamma \in b_{ih}^{h-1}$ such that $N_\gamma^i \not\subseteq N_\beta^i$, we have that $\alpha \cup \gamma$ is not a minimal i-j cut set of F_{ij}^h.*

PROOF. By Lemma 4.1 we have that $M_\gamma^i \not\subset N_\beta^i$ and, thus, $M_\gamma^i \not\subset N_\alpha^i$ by the hypothesis of the theorem. Therefore, condition (3.3) does not hold for $\alpha \cup \gamma$ and this cut set is not minimal. Q.E.D.

According to this theorem, Algorithm SUM2 can be modified as follows when b_{ij}^{h-1} and b_{ih}^{h-1} are topologically sorted lists. If we are comparing an element $\alpha \in b_{ij}^{h-1}$ with an element $\beta \in b_{ih}^{h-1}$ and if we have $N_\alpha^i \subseteq N_\beta^i$, then we know that there is no element $\gamma \in b_{ih}^{h-1}$ following β which gives a minimal cut set with α, and thus we can pass immediately to the element of b_{ij}^{h-1} following α. Similarly, if we encounter an element β such that $N_\beta^i \subseteq N_\alpha^i$, then we can immediately delete β from b_{ih}^{h-1}, since we know that β cannot give a minimal cut set with any element following α in b_{ij}^{h-1}.

We can give now our Algorithm SUM3 for computing $(b_{ij}^{h-1} + b_{ih}^{h-1})$ when b_{ij}^{h-1} and b_{ih}^{h-1} are topologically sorted with respect to the sets N_α^i.

```
Algorithm SUM3
sum := nil;
b := $b_{ih}^{h-1}$;
xl := $b_{ij}^{h-1}$;
while xl $\neq$ nil do
begin
  $\alpha$ := head (xl),
  xl := tail (xl);
  xh := nil;
  xt := b;
  while xt $\neq$ nil do
  begin
    $\beta$ := head (xt);
    xt := tail (xt);
    if $M_\alpha^i \subseteq N_\beta^i$ and $M_\beta^i \subseteq N_\alpha^i$ and ($N_\alpha^i \subseteq N_\beta^i$ or $N_\beta^i \subseteq N_\alpha^i$ or connected ($N_\alpha^i \cap N_\beta^i$)) and ($N_\alpha^j \cap$
      $N_\beta^h \neq \emptyset$ or there-is-an-arc (h, j)) then
    begin
      $N_{\alpha\cup\beta}^i := N_\alpha^i \cap N_\beta^i$,
      $N_{\alpha\cup\beta}^j = N_\alpha^j \cup N_\beta^h$,
      $M_{\alpha\cup\beta}^i := M_\alpha^i \cup M_\beta^i$;
      $M_{\alpha\cup\beta}^j := M_\alpha^j \cup M_\beta^h$;
      sum := append (sum, ($\alpha \cup \beta$))
    end;
    if $N_\beta^i \not\subseteq N_\alpha^i$ then xh := append (xh, ($\beta$));
    if $N_\alpha^i \subseteq N_\beta^i$ then goto l
  end;
l b : = append (xh, xt)
end.
```

In this algorithm, **nil** denotes the empty list, whereas *head* (x) returns the first element of the list x and *tail* (x) returns the list x without the first element. *Append* (x, y) returns a list obtained by appending the list y at the end of the list x.

Let us compute, for instance, the sum $(b_{45}^2 + b_{43}^2)$ for the graph in Figure 4, using Algorithm SUM3. A possible arrangement of the elements of b_{45}^2 is, as we said before, the following:

$$b_{45}^2 = (\{(1\ 4),\ (2\ 4)\},\ \{(1\ 2),\ (1\ 5),\ (2\ 4)\},\ \{(1\ 2),\ (1\ 4),\ (2\ 5)\},\ \{(1\ 5),\ (2\ 5)\}),$$

whereas the only possible arrangement of the elements of b_{43}^2 is

$$b_{43}^2 = (\{(1\ 4),\ (2\ 4)\},\ \{(1\ 2),\ (2\ 4)\},\ \{(2\ 3)\}).$$

The execution of Algorithm SUM3 is summarized in Table I, where each row gives

TABLE I

α	β	N_α^4	N_β^4	b
$\{(1\ 4), (2\ 4)\}$	$\{(1\ 4), (2\ 4)\}$	$\{4\}$	$\{4\}$	$(\{(1\ 4), (2\ 4)\}, \{(1\ 2), (2\ 4)\}, \{(2\ 3)\})$
$\{(1\ 2), (1\ 5), (2\ 4)\}$	$\{(1\ 2), (2\ 4)\}$	$\{1, 4\}$	$\{1, 4\}$	$(\{(1\ 2), (2\ 4)\}, \{(2\ 3)\})$
$\{(1\ 2), (1\ 4), (2\ 5)\}$	$\{(2\ 3)\}$	$\{2, 4\}$	$\{1, 2, 4\}$	$(\{(2\ 3)\})$
$\{(1\ 5), (2\ 5)\}$	$\{(2\ 3)\}$	$\{1, 2, 4\}$	$\{1, 2, 4\}$	$(\{(2\ 3)\})$

the value of α, β, N_α^4, N_β^4, b when the inner loop is entered. The result is

$$sum = (\{(1\ 4), (2\ 4)\}, \{(1\ 2), (1\ 5), (2\ 4)\}, \{(1\ 2), (1\ 4), (2\ 3), (2\ 5)\}, \{(1\ 5), (2\ 3), (2\ 5)\}),$$

Notice that this list *sum* is topologically sorted with respect to the sets N_α^4. The next theorem shows that this is always true, i.e. that Algorithm SUM3 always returns a topologically sorted list.

THEOREM 4.2. *Let b_{ij}^{h-1} and b_{ih}^{h-1} be two lists of cut sets, topologically sorted with respect to the sets N_α^i. Then, by computing $(b_{ij}^{h-1} + b_{ih}^{h-1})$ with Algorithm SUM3, we get a topologically sorted list with respect to the sets N_α^i.*

PROOF. Let $\alpha \cup \beta$ ($\alpha \in b_{ij}^{h-1}$, $\beta \in b_{ih}^{h-1}$) and $\gamma \cup \delta$ ($\gamma \in b_{ij}^{h-1}$, $\delta \in b_{ih}^{h-1}$) be two elements of *sum* such that $N_{\alpha\cup\beta}^i \subset N_{\gamma\cup\delta}^i$. We shall prove that $\gamma \cup \delta$ must follow $\alpha \cup \beta$ in *sum*. In fact, let us assume that $\gamma \cup \delta$ is obtained before $\alpha \cup \beta$. We can have two cases:

(i) α follows γ in b_{ij}^{h-1}. Since b_{ij}^{h-1} is topologically sorted, we have $N_\alpha^i \not\subset N_\gamma^i$, and, from Lemma 4.1, $M_\alpha^i \not\subset N_\gamma^i$. Hence we have $M_\alpha^i \not\subset N_{\gamma\cup\delta}^i = N_\gamma^i \cap N_\delta^i$. Furthermore, conditions (3.3) must be satisfied by α and β, and thus $M_\alpha^i \subset N_\beta^i$. Since, of course, we have $M_\alpha^i \subset N_\alpha^i$, we have also $M_\alpha^i \subset N_{\alpha\cup\beta}^i = N_\alpha^i \cap N_\beta^i$. Therefore, there is at least one element of M_α^i which is contained in $N_{\alpha\cup\beta}^i$ and not in $N_{\gamma\cup\delta}^i$, and thus we cannot have $N_{\alpha\cup\beta}^i \subset N_{\gamma\cup\delta}^i$.

(ii) $\alpha = \gamma$ and β follows δ in b_{ih}^{h-1}. A proof analogous to the one of case (i) can be given. Q.E.D.

When either i or j is smaller than h, a substantial improvement can be achieved in the computation of (3.2). In fact we can prove the following theorem.

THEOREM 4.3. *If $j < h$, then $(b_{ij}^{h-1} + b_{ih}^{h-1})$ can be obtained by taking all cut sets $\alpha \in b_{ih}^{h-1}$ such that $j \in N_\alpha^h$.*

PROOF. If $j < h$ we have $G_{ij}^h = G_{ih}^{h-1}$ and thus $F_{ij}^h = F_{ih}^{h-1}$ since j and h are connected. Therefore, every minimal i-h cut set $\alpha \in b_{ih}^{h-1}$ with $j \in N_\alpha^h$ is also a minimal i-j cut set of F_{ij}^h. On the other hand, every minimal i-j cut set β of F_{ij}^h belonging to $(b_{ij}^{h-1} + b_{ih}^{h-1})$ must have $h \in N_\beta^j$ and thus it must be a minimal i-h cut set of $F_{ih}^{h-1} = F_{ij}^h$. Q.E.D.

According to this theorem, when $j < h$, $(b_{ij}^{h-1} + b_{ih}^{h-1})$ can simply be obtained by scanning b_{ih}^{h-1} and deleting all cut sets α for which $j \notin N_\alpha^h$. Therefore, this computation requires a time linear with the cardinality of b_{ih}^{h-1}. Similarly if $i < h$.

The following simple algorithm can be given for computing $(b_{ij}^{h-1} + b_{ih}^{h-1})$ when $j < h$.

Algorithm SIMPLESUM

```
sum := nil;
x := $b_{ih}^{h-1}$;
while x ≠ nil do
begin
  α := head (x);
  x := tail (x);
  if $j \in N_\alpha^h$ then sum := append (sum, (α))
end.
```

Of course, if the list b_{ih}^{h-1} is topologically sorted with respect to the sets N_α^i, this algorithm gives a result which is still topologically sorted.

Finally we can give an algorithm for computing (3.2) using Algorithms SUM3 and SIMPLESUM.

```
Algorithm SUMMULT
if $j < h$ then
  $sum1 := SIMPLESUM\ (b_{ih}^{h-1})$
else
  $sum1 := SUM3\ (b_{ij}^{h-1}, b_{ih}^{h-1})$;
if $i < h$ then
  $sum2 := SIMPLESUM\ (b_{jh}^{h-1})$
else
  $sum2 := SUM3\ (b_{ji}^{h-1}, b_{jh}^{h-1})$;
$b_{ij}^{h} := append\ (sum1, reverse\ (sum2))$
```

It is easy to show that, if b_{ij}^{h-1} and b_{ih}^{h-1} are topologically sorted with respect to the sets $N_\alpha{}^i$ and b_{ji}^{h-1} and b_{jh}^{h-1} are topologically sorted with respect to the sets $N_\alpha{}^j$, then the result b_{ij}^{h} will be topologically sorted with respect to the sets $N_\alpha{}^i$. Note that, since the graph is undirected, both b_{ij}^{h-1} and b_{ji}^{h-1} contain the same cut sets, but they are different lists because the former is sorted with respect to $N_\alpha{}^i$ and the latter is sorted with respect to $N_\alpha{}^j$.

According to the previous theorems, *sum*2 will be topologically sorted with respect to the sets $N_\alpha{}^j$, i.e. whenever $N_\alpha{}^j \subset N_\beta{}^j$, β must follow α. But whenever $N_\alpha{}^j \subset N_\beta{}^j$, we have $N_\alpha{}^i \supset N_\beta{}^i$ and therefore, by reversing *sum*2, we get a list topologically sorted with respect to the sets $N_\alpha{}^i$. Furthermore, for every cut set $\alpha \in sum2$, $N_\alpha{}^i$ contains the node h, and for every cut set $\beta \in sum1$ $N_\beta{}^i$ does not contain h. Hence we cannot have $N_\alpha{}^i \subset N_\beta{}^i$. Therefore, b_{ij}^{h} obtained with Algorithm SUMMULT is topologically sorted with respect to the sets $N_\alpha{}^i$.

For instance, let us compute b_{45}^{3} for the graph in Figure 4. The computation of $sum1 = b_{45}^{2} + b_{43}^{2}$ has been shown before. Let us assume now that b_{54}^{2} is the reverse of b_{45}^{2}, and

$$b_{53}^{2} = (\{(1\ 5), (2\ 5), (3\ 5)\}, \{(1\ 2), (2\ 5), (3\ 5)\}, \{(2\ 3), (3\ 5)\}).$$

By applying Algorithm SUM3 we get

$$sum2 = b_{54}^{2} + b_{53}^{2} = (\{(1\ 5), (2\ 5), (3\ 5)\}, \{(1\ 2), (1\ 4), (2\ 5), (3\ 5)\}).$$

Finally, we get

$$\begin{aligned} b_{45}^{3} = (&\{(1\ 4), (2\ 4)\}, \{(1\ 2), (1\ 5), (2\ 4)\}, \{(1\ 2), (1\ 4), (2\ 3), (2\ 5)\}, \\ &\{(1\ 5), (2\ 3), (2\ 5)\}, \{(1\ 2), (1\ 4), (2\ 5), (3\ 5)\}, \{(1\ 5), (2\ 5), (3\ 5)\}). \end{aligned}$$

The sets $N_\alpha{}^4$ of these cut sets are, respectively, $\{4\}$, $\{1,4\}$, $\{2,4\}$, $\{1,2,4\}$, $\{2,3,4\}$, and $\{1,2,3,4\}$.

5. *Some Considerations on Computational Complexity*

Let us now analyze the complexity of Algorithm SUMMULT. Using a suitable representation of lists, the operation of appending the reverse of *sum*2 to *sum*1 can be performed in a constant number of steps. Therefore the complexity of SUMMULT is given by the complexity of SUM3 and SIMPLESUM.

As pointed out before, Algorithm SIMPLESUM requires a number of steps which is linear with the cardinality of b_{ih}^{h-1}. Algorithm SUM3 differs from Algorithm SUM2 only because it can jump out of the body of the inner loop. Therefore, Algorithm SUM3 cannot take more steps than Algorithm SUM2, and thus, in the worst case, it will execute pq times the inner loop, where $p = |b_{ij}^{h-1}|$ and $q = |b_{ih}^{h-1}|$. On the other hand, in the best case, Algorithm SUM3 will enter the inner loop only when $\alpha \cup \beta$ is a minimal cut set, thus taking $O(r)$ steps, where r is the cardinality of $(b_{ij}^{h-1} + b_{ih}^{h-1})$.

For example, the lower bound is actually achieved in the computation of $(b_{45}^{2} + b_{43}^{2})$ for the graph in Figure 4. In fact, the cardinality of $(b_{45}^{2} + b_{43}^{2})$ is 4, and the computation, summarized in Table I, requires four steps, without any call to the procedure *connected*.

Instead, the computation of $(b_{45}^2 + b_{43}^2)$ with Algorithm SUM2 would require twelve steps.

As a further example of graphs for which the lower bound is achieved by Algorithm SUM3, we will now consider the complete graphs. The efficiency of our algorithm is proved by showing that it requires a time proportional to the number of cut sets if we have topologically sorted lists; whereas, using other implementations described in Section 3, it would take times of the order of the square or of the cube of the number of cut sets.

Given a complete graph with n nodes, we want to determine the number of steps necessary to compute all minimal i-j cut sets for all pairs of nodes using Algorithm SUMMULT. We recall that the minimal i-j sets are $C = 2^{n-2}$ for each pair.

Let us consider the hth step of Algorithm G. We can distinguish three cases:

(i) $i > h$ and $j > h$. The cardinality of b_{ij}^{h-1} is $|b_{ij}^{h-1}| = 2^{h-1}$, since the graph $G_{ij}^{h-1} = F_{ij}^{h-1}$ has $h + 1$ nodes. Analogously, we have $|b_{ih}^{h-1}| = 2^{h-1}$ and $|b_{hj}^{h-1}| = 2^{h-1}$. The list b_{ij}^{h-1} was obtained, according to Algorithm SUMMULT, by appending the two lists *sum*1 and *reverse* (*sum*2), such that for each $\alpha \in$ *sum*1 we have $h-1 \notin N_\alpha{}'$ and for each $\alpha \in$ *sum*2 we have $h-1 \in N_\alpha{}'$. But the same was true for every previous step of the construction of b_{ij}^{h-1} and, thus, the sets $N_\alpha{}'$ for the elements α of b_{ij}^{h-1} are arranged as follows:

$$\{i\}\ \{i, 1\}\ \{i, 2\}\ \{i, 1, 2\} \cdots \{i, h-1\}\ \{i, 1, h-1\}\ \{i, 2, h-1\}\ \{i, 1, 2, h-1\} \cdots .$$

The list b_{ih}^{h-1} was constructed in the same way and, therefore, the sequence of sets $N_\alpha{}'$ for its elements is exactly the same as the sequence given above for b_{ij}^{h-1}.

If we now apply Algorithm SUM3 to b_{ij}^{h-1} and b_{ih}^{h-1}, at the first step we have $N_\alpha{}' = N_\beta{}' = \{i\}$ and, thus, β is erased and we can jump immediately to the beginning of the external loop. At the next step, α is the second element of b_{ij}^{h-1} and β is the second element of b_{ih}^{h-1} and we have $N_\alpha{}' = N_\beta{}' = \{i, 1\}$. Since the same happens at every step of the execution of the algorithm, we see that every element of b_{ij}^{h-1} is compared with only one element of b_{ih}^{h-1} and, therefore, the execution of the algorithm requires 2^{h-1} steps Similarly for the computation of $(b_{ij}^{h-1} + b_{hj}^{h-1})$.

Since the procedure *connected* is never called by SUM3, we can assume that the execution of the body of the inner loop takes a constant time a. Therefore the time for computing b_{ij}^h is $t = 2\,2^{h-1}\cdot a = 2^h a$. Since the cardinality of b_{ij}^h is $l = |b_{ij}^h| = 2^h$, we have $t = al$.

(ii) $i > h$ and $j < h$ (or $i < h$ and $j > h$). We have $|b_{ij}^{h-1}| = 2^{h-2}$, since $G_{ij}^{h-1} = F_{ij}^{h-1}$ has h nodes, $|b_{ih}^{h-1}| = 2^{h-1}$, and $|b_{hj}^{h-1}| = 2^{h-2}$. The first half of b_{ij}^h is obtained by using Algorithm SIMPLESUM, since $j < h$. This algorithm performs a simple scan of b_{ih}^{h-1} and thus it requires 2^{h-1} steps. The second half of b_{ij}^h is obtained as in case (i) by using Algorithm SUM3 with b_{ij}^{h-1} and b_{hj}^{h-1}, and therefore its computation requires 2^{h-2} steps. By assuming that the execution of the body of the loop of Algorithm SIMPLESUM takes time b, then the time for computing b_{ij}^h is $t = 2^{h-2}a + 2^{h-1}b$.

Since $l = |b_{ij}^h| = 2^{h-1}$, we have $t = (\frac{1}{2}a + b)l$.

(iii) $i < h$ and $j < h$. We have $|b_{ij}^{h-1}| = 2^{h-3}$, $|b_{ih}^{h-1}| = 2^{h-2}$, and $|b_{hj}^{h-1}| = 2^{h-2}$. The solution is obtained by applying Algorithm SIMPLESUM twice to b_{ih}^{h-1} and b_{hj}^{h-1} and, therefore, the time for computing b_{ij}^h is $t = 2\,2^{h-2}b = 2^{h-1}b$.

Since $l = |b_{ij}^h| = 2^{h-2}$, we have $t = 2bl$.

The number of elements for which case (i) holds is $(n - h)(n - h - 1)$, since the elements on the main diagonal are not considered. The number of elements for which case (ii) holds is $2(h - 1)(n - h)$ and the number of elements for which case (iii) holds is $(h - 1)(h - 2)$. Therefore, the time for computing all minimal i-j cut sets for all pairs of nodes is

$$T = \sum_{h=1}^{n} (n - h)(n - h - 1)2^h a + 2(h - 1)(n - h)(2^{h-2}a + 2^{h-1}b) + (h - 1)(h - 2)2^{h-1}b.$$

We want to show that the time T is linear with the total number of cut sets. In order to do this, let us assume that both a and b are equal to a time unit: $a = b = 1$. Let us consider the time T_{ij} required to compute element b_{ij}^n. The value of this element is modified $n - 2$ times by Algorithm G (for $h = i$ and $h = j$ this element is unchanged) and, every time it is modified, its cardinality is doubled starting with cardinality one. Therefore we have $T_{ij} = t_{ij}^1 + t_{ij}^2 + \cdots + t_{ij}^{n-2}$, where t_{ij}^k $(k = 1, \cdots, n - 2)$ gives the time for computing a list of cut sets with 2^k elements.

It is easy to show that $t_{ij}^{k+1} \geq 2t_{ij}^k$ $(k = 1, \cdots, n - 3)$. According to the previous discussion we have $t_{ij}^k = pl'$ and $t_{ij}^{k+1} = rl''$, where $l' = 2^k$, $l'' = 2^{k+1} = 2l'$, p and r can be 1, $\frac{3}{2}$, or 2 depending on which one of the three cases is considered. By examining all possibilities we see that we must always have $r \geq p$ and therefore $t_{ij}^{k+1} = rl'' = 2rl' \geq 2pl' = 2t_{ij}^k$.

We can now compute a lower bound and an upper bound for T_{ij}. The minimal possible value for t_{ij}^1 is 2, achieved for $h = 1$ in case (i). Therefore, a lower bound for T_{ij} is obtained by taking $t_{ij}^1 = 2$ and $t_{ij}^{k+1} = 2t_{ij}^k$ $(k = 1, \cdots, n - 3)$, because we have just shown that we must have $t_{ij}^{k+1} \geq 2t_{ij}^k$.

$$T_{\min} = 2 + 2^2 + \cdots + 2^{n-2} = 2(2^{n-2} - 1) \cong 2C,$$

where $C = 2^{n-2}$ is the number of minimal i-j cut sets for each pair of nodes i and j.

This lower bound is actually achieved by the computation of $b_{n-1,n}^n$. In fact, this element is modified by Algorithm G for $h = 1, 2, \cdots, n - 2$ and we are always in case (i).

The maximal possible value for t_{ij}^{n-2} is 2^{n-1}, achieved for $h = n$ in case (iii). Therefore, an upper bound for T_{ij} is obtained by taking $t_{ij}^{n-2} = 2^{n-1}$ and $t_{ij}^k = t_{ij}^{k+1}/2$ $(k = 1, \cdots, n - 3)$.

$$T_{\max} = 2^2 + 2^3 + \cdots + 2^{n-1} = 4(2^{n-2} - 1) \cong 4C.$$

This upper bound is actually achieved by the computation of $b_{1,2}^n$. In fact, this element is modified for $h = 3, \cdots, n$ and we are always in case (iii).

We can conclude that the computation of each element b_{ij}^n requires time $T_{ij} = k_{ij}C$, where $2 \leq k_{ij} \leq 4$. Therefore, the time T required by Algorithm G for computing all cut sets between all pairs of nodes for a complete graph is proportional to the total number of cut sets $T = kC_{\text{tot}}$, where $C_{\text{tot}} = n(n - 1)C$ and $2 \leq k \leq 4$, for $a = b = 1$.

In Table II we give the total time T, the total number of cut sets C_{tot} and the coefficient k for $a = b = 1$, for complete graphs with n nodes $(3 \leq n \leq 10)$. For $n = 3$, k is smaller than 2 because the approximation $T_{\min} \cong 2C$ is not valid for small values of n.

Notice that the linearity of time T with respect to the total number of cut sets depends on the lists of cut sets b_{ij}^h being topologically sorted. In fact, by taking into account the properties of topologically sorted lists of cut sets described in Section 4, Algorithm SUM3 can take 2^h steps for adding two lists each of length 2^h. Instead, if the elements of the sets b_{ij}^h were unordered, the computation of the sum of two sets of cardinality 2^h using Algorithm SUM2 would require $2^h 2^h$ steps.

TABLE II

No. nodes	T	C_{tot}	T/C_{tot} $(a = b = 1)$
3	$6\,a + 12\,b$	12	1 5
4	$32\,a + 80\,b$	48	2 33
5	$110\,a + 340\,b$	160	2 81
6	$312\,a + 1176\,b$	480	3.1
7	$798\,a + 3612\,b$	1344	3 28
8	$1920\,a + 10272\,b$	3584	3 40
9	$4446\,a + 27684\,b$	9216	3 48
10	$10040\,a + 71720\,b$	23040	3 55

TABLE III

No. nodes	$r = c = n - 1$		$r = c = 1$	
	Time (sec)	No. cut sets	Time (sec)	No. cut sets
6	0 2	16	0.6	240
7	0 4	32	1.4	672
8	0.8	64	3 4	1792
9	1.6	128	8 0	4608
10	3 2	256	18 5	11520
11	6 3	512	—	28160

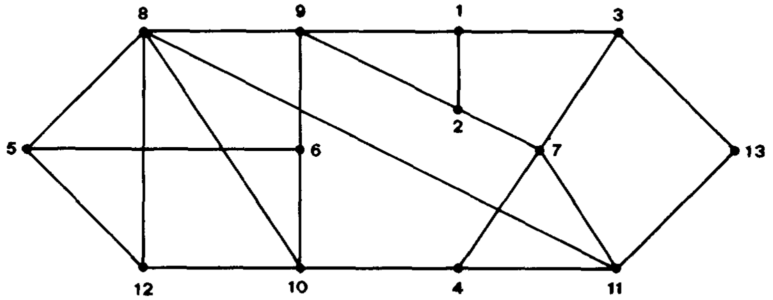

FIG 5. Reduced ARPA computer network

6. *Experimental Results*

The algorithm described in this paper for enumerating the minimal cut sets was implemented in Algol W on an IBM 360/67 computer. The data type *bits* of Algol W provided an efficient representation of cut sets and sets of nodes as bit patterns, where each bit denotes a different arc or node. The elements b_{ij}^h have been internally represented as lists, according to the requirements discussed in Section 4.

Since our algorithm is defined only for undirected graphs, it is possible to halve the computing time by considering only half matrix, i.e. by computing only the elements b_{ij}^h with $i < j$. In order to do this, the elements b_{ij}^h have been represented as doubly linked lists. In fact, it is easy to see that, if we have a suitable ordering of the cut sets of b_{ij}^h by following the list in one direction, then, by following the list in the other direction, we have a suitable ordering of the elements of b_{ji}^h.

The program was used for the computation of the minimal cut sets for complete graphs with n nodes, $6 \leq n \leq 11$. The cut sets were obtained for only a pair of nodes (i.e. Algorithm G was applied with $r = c = n - 1$) and for all pairs of nodes (i.e. $r = c = 1$). The times required by the program and the number of cut sets are given in Table III. The storage was not sufficient for the computation of all cut sets for $n = 11$.

A version of the program implementing Algorithm SUM2 was also applied to the complete graphs. This program took 1.5 sec for computing all minimal cut sets for all pairs of nodes of the complete graph with 6 nodes, 6 sec for the complete graph with 7 nodes, and 28 sec for the complete graph with 8 nodes.

An important feature of our algorithm is that it can compute simultaneously several entries of the matrix, i.e. the minimal cut sets between several pairs of nodes. Indeed, by increasing the number of computed entries, the efficiency of the algorithm increases, if we take the average time per cut set as an index of the efficiency. For example, Table III shows that the average time per cut set is 12.5 msec if only one entry is computed, whereas the time decreases to about 2 msec if all entries are computed.

This feature makes comparison with other algorithms difficult, since all algorithms which have been briefly presented in Section 1 work only for one pair of nodes, and, if the cut sets between several pairs are required, they have to be used separately for each pair.

As a further example let us take the reduced ARPA computer network in Figure 5. The program, applied to this graph, computes the 214 minimal 12-13 cut sets in 2.6 sec with an average time per cut set of 12 msec. Instead, the program implementing Algorithm SUM2 computes these cut sets in 4 8 sec. Another version using Algorithm SUM1 was still running after 2 min.

As pointed out before, the average time per cut set decreases if we increase the number of pairs of nodes. This happens because, when $i < h$ or $j < h$, Algorithm SUMMULT uses SIMPLESUM, which is simpler than SUM3. If we compute all minimal i-j cut sets with $4 \leq i \leq 13$ and $8 \leq j \leq 13$ for the graph in Figure 5, then the 8168 cut sets are obtained in 19.5 sec, thus lowering the average time to 2.4 msec The same is not true for the program which uses algorithm SUM2. For example, the 1148 i-j cut sets with $10 \leq i \leq 13$ and $11 \leq j \leq 13$ were obtained by this program in 40 sec.

As a final remark, we note that, when the graph is not complete, the computing time depends on the order of elimination of the nodes, as in the case of the solution of a sparse system of equations in linear algebra [11]. For example, by eliminating the nodes of the graph in Figure 5 in the order 3, 1, 9, 8, 7, 2, 6, 5, 11, 4, 10, 12, 13, the minimal cut sets between nodes 12 and 13 are obtained in 4.5 sec, instead of 2.6 sec required by the previous ordering.

7. *Conclusion*

An efficient algorithm for enumerating the cut sets between all pairs of nodes in an undirected graph has been presented. An algebra for cut sets has been given and we have shown that the cut sets can be obtained by solving a system of linear equations using Gaussian elimination. The efficiency of the algorithm depends heavily on the implementation of sum and multiplication Therefore, some properties of cut sets were investigated in order to simplify these operations The experimental results show that satisfactory computing times are achieved by an implementation of our algorithm

An advantage of this algorithm with respect to other algorithms is that it can simultaneously compute the cut sets between several pairs of nodes, thus increasing the efficiency. The experimental results show that the average time per cut set is reduced almost an order of magnitude if we compute the cut sets between several pairs of nodes instead of computing the cut sets between only one pair of nodes.

ACKNOWLEDGMENTS The author wishes to thank U Montanari for many helpful discussions and C. Lami for writing an earlier version of the program.

REFERENCES

1. Fratta, L , and Montanari, U A Boolean algebra method for computing the terminal reliability in a communication network *IEEE Trans. Circuit Theory CT-20* (May 1973), 203–211
2. Nelson, A.C., Batts, J.R., and Beadles, R L A computer program for approximating system reliability. *IEEE Trans Reliability R-19* (May 1970), 61–65
3. Jensen, P.A , and Bellmore, M An algorithm to determine the reliability of a complex system *IEEE Trans Reliability R-18* (Nov 1969), 169–174
4. Liu, C L. *Introduction to Combinatorial Mathematics* McGraw-Hill, New York, 1968.
5. Martelli, A. An application of regular algebra to the enumeration of cut sets in a graph. *Information Processing 74*, North-Holland Pub Co , Amsterdam, 1974, pp 511–515.
6. Fratta, L., and Montanari, U. A vertex elimination algorithm for enumerating all simple paths in a graph. *Networks* (to appear).
7. Backhouse, R.C., and Carré, B.A. Regular algebra applied to path-finding problems. *J. Inst Maths Applics* (to appear)
8. Salomaa, A. *Theory of Automata* Pergamon Press, New York, 1969.
9. Floyd, R W Algorithm 97· Shortest path *Comm ACM 5*, 6 (June 1962), 345.
10. Warshall, S. A theorem on Boolean matrices *J ACM 9*, 1 (Jan 1962), 11–12.
11. Martelli, A., and Montanari, U. An application of heuristically guided search to elimination processes Proc. Internat. Computing Symp , Venice, April 1972, pp 59–72

RECEIVED JULY 1974; REVISED MARCH 1975