An Efficient Heuristic Cluster Algorithm for Tearing Large-Scale Networks

ALBERTO SANGIOVANNI-VINCENTELLI, MEMBER, IEEE, LI-KUAN CHEN, MEMBER, IEEE, AND LEON O. CHUA, FELLOW, IEEE

***Abstract*—An efficient heuristic algorithm for solving a cluster problem associated with the tearing of an undirected graph is presented via the concept of a contour tableau. The required computation time is shown to be bounded by $\mathcal{O}(nb)$, where n and b are the number of nodes and branches of the input graph, respectively.**

Experimental results show that our algorithm is highly efficient and yields near optimal solutions.

I. Introduction

In DEALING WITH large-scale networks and systems, extensive decomposition algorithms have been proposed in various fields. In *operation research,* for example, we have the Dantzig–Wolfe decomposition principle for linear programming [1] and the Hu decomposition algorithm for shortest path calculations [2]. In *circuit theory*, we have the diakoptic analysis, the generalized hybrid analysis, and the node-tearing nodal analysis [3]–[6]. Finally, decomposition techniques also play an important role in the *computation* [7] and *stability analysis* [8] of *large-scale systems.*

All decomposition methods require that the large network or system be partitioned into subsystems (i.e., clusters) such that elements in the same subsystem are *strongly* interconnected, whereas elements in the different subsystems are *weakly* interconnected. In some cases where the system has a simple layout, a fairly good cluster partition can be determined by inspection. For arbitrary systems, however, an algorithm must be used to systematically partition the associated graph into an optimal, or suboptimal, arrangement of clusters.

Some attempts have been made at finding an optimal cluster partition in computer logic and page partitioning [9], [10], in power system bus clustering [11], in network decomposition [12], in IC placement problems [13], and in statistical data grouping [14]. In general, the cluster partition problem is formulated as a *graph optimization problem* in which an optimal partition of nodes is sought, such that a certain measure on the interconnection between different groups of nodes is minimized. Depending on the nature of the problem, the minimization objectives may be based upon the number of interconnection nodes [2], [5]–[8], the number of interconnection branches [3], [4], [9]–[12], the total cost of interconnection branches [13], or the distance between the "centroids" of clusters [14]. The various approaches for solving the cluster partition problems may be classified into four major categories:

i) growing clusters from scratch [9], [12];
ii) interchanging nodes until some local optimality condition is satisfied [13];
iii) transforming the problem into some associated mathematical equation [10], [14];
iv) finding the "contour" of an associated graph [11].

It has to be pointed out here that none of the above attempts [9]–[14] yields an efficient global algorithm. Actually, none of them even yields an efficient heuristic algorithm [15] and there are reasons to believe [5] that all cluster partition problems belong to a class of hard problems, the so-called *NP-complete class* [16], [17], where no polynomial-bounded global solutions are likely to exist.

In this paper, we will restrict ourselves to the cluster problem associated with the *node-tearing nodal analysis* of large-scale networks [5], [6]. Since it has been shown in [5] that this cluster problem is NP-complete, we will concentrate on finding an efficient but heuristic cluster algorithm.

II. The Cluster Problem and the Contour Approach

We shall briefly recall the cluster problem[1] as defined in [5]. Basically, given an $n\times n$ structurally symmetric matrix $\boldsymbol{Y}$, we want to permute $\boldsymbol{Y}$ into a bordered-block-diagonal form such that each diagonal block has dimension $\leqslant n_{\max}$ and the dimension of the border is minimized. This problem has a straightforward graph-theoretic interpretation [5]. Given a matrix $\boldsymbol{Y}$, let $\mathcal{G}_Y=(\mathcal{N}_Y,\mathcal{B}_Y)$ be the associated undirected graph constructed in accordance with the following properties:

i) $\mathcal{G}_Y$ contains n nodes (i.e., $|\mathcal{N}_Y|=n$);
ii) an undirected branch $\hat{b}\in\mathcal{B}_Y$ joins n_i and n_j if, and only if, $Y_{ij}\neq 0$.

$\mathcal{G}_Y$ is the so-called *sparsity graph* of Y [18], [19]. Let $\{\mathcal{N}_{Y_1},\mathcal{N}_{Y_2}\}$ denote a partition of nodes of $\mathcal{G}_Y$. Then, the cluster problem consists of minimizing $|\mathcal{N}_{Y_2}|$ over all

[1]It is referred to as GOP1 in [5].

Manuscript received November 22, 1976; revised June 23, 1977. This work was supported in part by the Joint Services Electronics Program under Contract F44620-76-C-0100, in part by the National Science Foundation under Grant ENG75-22282, and in part by the Naval Electronics System Command under Contract N00039-75-C-0034.

A. Sangiovanni-Vincentelli and L. O. Chua are with the Department of Electrical Engineering and Computer Sciences and the Electronics Research Laboratory, University of California, Berkeley, CA 94720.

L. K. Chen is with the American Electric Power Service Corporation, New York, NY.

IS	AS	CN
IS(1)	AS(1)	CN(1)
IS(2)	AS(2)	CN(2)
IS(3)	AS(3)	CN(3)
IS(4)	AS(4)	CN(4)
⋮	⋮	⋮

Fig. 1. A contour tableau.

distinct partitions $\{\mathfrak{N}_{Y_1}, \mathfrak{N}_{Y_2}\}$ such that

i) the removal of all nodes in $\mathfrak{N}_{Y_2}$ would *disconnect* the remaining graph into m components $\mathcal{G}^1_{Y_1}, \mathcal{G}^2_{Y_1}, \cdots, \mathcal{G}^m_{Y_1}$;

ii) $|\mathfrak{N}^k_{Y_1}| \leqslant n_{\max}$, where $\mathfrak{N}^k_{Y_1}$ denote the set of nodes contained in $\mathcal{G}^k_{Y_1}$, $k = 1, 2, \cdots, m$.

In [5], this cluster problem is shown to be NP-complete. Hence, any practical solution would call for a heuristic approach. The heuristic algorithm to be proposed in this paper follows a strategy similar to the one introduced in [11], and is based on the concept of a *contour tableau* which consists of an array of three columns, as shown in Fig. 1. The leftmost column is called the *iterating set* (IS), the middle column the *adjacent set* (AS), and the rightmost column the *contour number* (CN). The entries of the tableau are determined as follows.[2]

Contour Tableau Construction Algorithm:

Step 1) Choose an *initial iterating node* and store it in IS(1).

Step 2) Store in AS(1) all nodes that are adjacent to the node in IS(1).

Step 3) Place the cardinality of AS(1) in CN(1).

Step 4) Let $i = 1$.

Step 5) If CN $(i) = 0$, stop!

Step 6) Choose the *next iterating node*, denoted by n_{i+1}, from AS(i) and place it in IS($i+1$).

Step 7) Update AS($i+1$) from AS(i) by deleting the node n_{i+1} and adding the set V representing all node adjacent to n_{i+1} that are not already in AS(i) or $\{\cup^i_{j=1}\mathrm{IS}(j)\}$.

Step 8) CN($i+1$) = |AS($i+1$)|.

Step 9) Let $i = i + 1$, go to Step 5.

Let us first clarify Step 7 with the aid of Fig. 2. In AS(i) and AS($i+1$), we store the adjacent nodes of the sets of iterated nodes

$$\left\{\bigcup_{j=1}^{i} \mathrm{IS}(j)\right\} \quad \text{and} \quad \left\{\bigcup_{j=1}^{i+1} \mathrm{IS}(j)\right\},$$

respectively. Instead of finding AS($i+1$) from scratch at each iteration, we want to find an efficient way of updating AS($i+1$) from AS(i). Now, let us look at Fig. 2, where the solid lines denote adjacency relations and the dotted lines denote possible adjacency relations. Sets $\{\mathrm{IS}(i+1)\}$ and $\{\mathrm{AS}(i) - \mathrm{IS}(i+1)\}$ are adjacent to

$$\left\{\bigcup_{j=1}^{i} \mathrm{IS}(j)\right\}.$$

Since $\{\mathrm{AS}(i) - \mathrm{IS}(i+1)\}$ and V are adjacent to

$$\left\{\bigcup_{j=1}^{i+1} \mathrm{IS}(j)\right\}$$

we can therefore update AS($i+1$) from AS(i) by deleting IS($i+1$) and adding V, which is precisely Step 7.

Now, let us pause to look at an example. Fig. 3 shows a graph with nine nodes. It is clustered into two groups of nodes, $\{n_1, n_2, n_3, n_4\}$ and $\{n_6, n_7, n_8, n_9\}$ which are separated by the hinged node n_5. Let us start the construction of our contour tableau by selecting arbitrarily the initial node, say n_1, and store it in IS(1). Since $\{n_2, n_3, n_4, n_5\}$ are the nodes adjacent to n_1, they are stored in AS(1). Consequently, CN(1) = 4. Let us choose arbitrarily an iterating node from AS(1), say n_3, and put it in IS(2). Observe that the nodes that are adjacent to $\{n_1, n_3\}$ are $\{n_2, n_4, n_5\}$. So they are put in AS(2) and hence CN(2) = 3. Choose the next iterating node as IS(3) = n_5, then AS(3) = $\{n_2, n_4, n_6, n_7, n_8, n_9\}$ and hence CN(3) = 6. The complete tableau is shown in Fig. 4(a).

In order to understand how the preceding algorithm can be used to separate the graph into clusters, let us observe that if X denotes the set of nodes of a given graph, then the set of AS nodes always separates X into three subsets; namely,

$$\left[Z(i) \triangleq \bigcup_{j=1}^{i} \mathrm{IS}(j) \right] \mathrm{AS}(i)$$

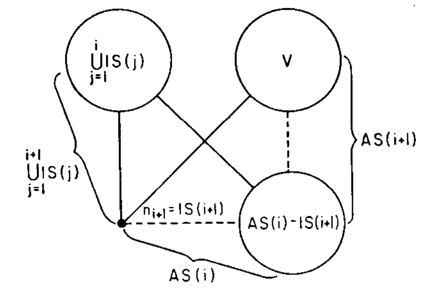


Fig. 2. A graphic interpretation of Step 7 for updating AS($i+1$) from AS(i).

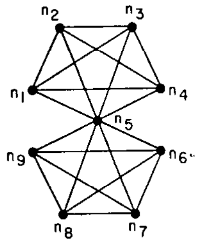


Fig. 3. An example for illustrating the contour tableau construction algorithms.

[2]The graph is assumed to be connected for simplicity.

IS	AS	CN
n_1	n_2, n_3, n_4, n_5	4
n_3	n_2, n_4, n_5	3
n_5	$n_2, n_4, n_6, n_7, n_8, n_9$	6
n_6	n_2, n_4, n_7, n_8, n_9	5
n_2	n_4, n_7, n_8, n_9	4
n_9	n_4, n_7, n_8	3
n_7	n_4, n_8	2
n_4	n_8	1
n_8	ϕ	0

(a)

IS	AS	CN
n_1	n_2, n_3, n_4, n_5	4
n_2	n_3, n_4, n_5	3
n_3	n_4, n_5	2
n_4	n_5	1
n_5	n_6, n_7, n_8, n_9	4
n_6	n_7, n_8, n_9	3
n_7	n_8, n_9	2
n_8	n_9	1
n_9	ϕ	0

(b)

	AS	CN
n_5	$n_1, n_2, n_3, n_4, n_6, n_7, n_8, n_9$	8
n_1	$n_2, n_3, n_4, n_6, n_7, n_8, n_9$	7
n_2	$n_3, n_4, n_6, n_7, n_8, n_9$	6
n_3	n_4, n_6, n_7, n_8, n_9	5
n_4	n_6, n_7, n_8, n_9	4
n_6	n_7, n_8, n_9	3
n_7	n_8, n_9	2
n_8	n_9	1
n_9	ϕ	0

(c)

Fig. 4. Three different contour tableaus associated with the graph in Fig. 17 by using three different strategies during the construction. (a) Arbitrary choice. (b) Greedy strategy in choosing the next iterating node. (c) Initial iterating node selection.

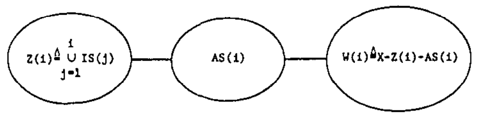

$Z(i) \triangleq \bigcup_{j=1}^{i} IS(j)$
AS(i)
$W(i) \triangleq X - Z(i) - AS(i)$

Fig. 5. The graphical interpretation of the role of AS(i) as a separating set.

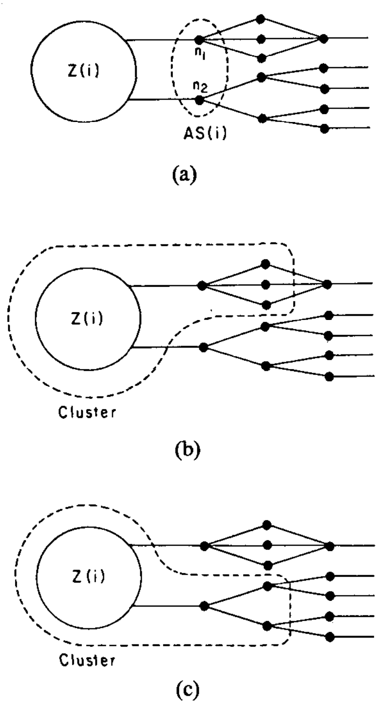


Fig. 6. An example showing that the greedy strategy may sometimes give undesirable results. (a) The example graph. (b) Cluster obtained by choosing n_1 as the next iterating node. (c) Cluster obtained by choosing n_2 as the next iterating node.

and

$$W(i) \triangleq X - Z(i) - \mathrm{AS}(i)$$

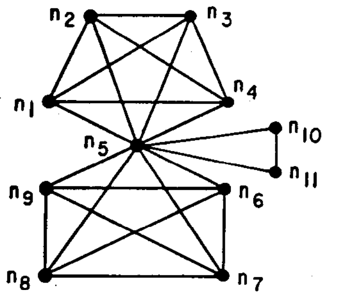
where $Z(i)$ nodes are *not* adjacent to $W(i)$ nodes (Fig. 5). As we construct the tableau, the size of AS(i) (i.e., CN(i)) in each step varies. It is when CN(i) is very small, henceforth called a *bottleneck*, that $Z(i)$ and $W(i)$ form clusters. Our aim then is to choose a particular contour tableau construction algorithm that would yield a good cluster whenever CN(i) encounters a bottleneck. By using arbitrary choices in Steps 1 and 6, as in the preceding example, the best AS(i) is $\{n_2, n_4, n_5\}$ (Fig. 4(a)). However, it is far from the optimal result; namely, AS(i) = $\{n_5\}$, which in this case can be obtained by inspection.

In the original contour construction algorithm, there are only two places where choices are made. They are in Step 1 when choosing the *initial iterating node*, and in Step 6 when choosing the *next iterating node*. Let us first examine Step 6. In [11], the strategy chosen is the *minimum-fill-in strategy* which is quite time-consuming and hence inefficient. In this paper, we propose a *greedy strategy*;[3] namely, at every iteration, we simply choose the node in AS(i) that yields minimum CN($i+1$) = |AS($i+1$)| or, equivalently, we choose the node that yields minimum $|V|$. If a tie is encountered, we choose arbitrarily among the ties. To illustrate this strategy, we start with n_1 and eventually construct the tableau shown in Fig. 4(b). Indeed, it yields our desired goal; namely, to separate the two clusters $\{n_1, n_2, n_3, n_4\}$ and $\{n_6, n_7, n_8, n_9\}$ through the bottleneck $\{n_5\}$.

[3]"greedy" is a very common term in the graph literature [2] and is attributed to Jack Edmonds [24]. It means that the algorithm determines the direction for iteration generally by checking for the "cheapest" local condition.

Our main reason for choosing the greedy strategy is that it can be easily implemented. To analyze the efficiency of this strategy, we will shortly derive its computational complexity. Before doing this, however, let us first identify its shortcomings by analyzing the example shown in Fig. 6(a). Suppose after the ith iteration, AS(i) = $\{n_1, n_2\}$. If we choose n_1 to iterate next, we will end up with the cluster shown by the dotted line in Fig. 6(b) which has two bottleneck nodes. On the other hand, since $|V(n_1)| = 3$ and $|V(n_2)| = 2$, application of our greedy strategy would require that n_2 be iterated next. The resulting cluster is shown by the dotted line in Fig. 6(c) which has five bottleneck nodes. This result, i.e., five bottleneck nodes versus two bottleneck nodes, of course is undesirable.

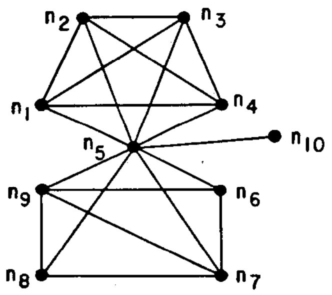
Let us examine next the choice of the *initial iterating node*. If we start the tableau construction from n_5 in Fig. 3 and use the greedy strategy, then the resulting tableau is shown in Fig. 4(c). Observe that the basic contour property for identifying the clusters is lost. In this case the choice of node n_5 as the starting node is highly undesirable because n_5 is a bottleneck node. In fact, the initial choice of a bottleneck node—separating, for example, two clusters—ends up in a contour which identifies the union of the two "real" clusters and of the bottleneck node as a "unique" cluster. Therefore a sound strategy for choosing the initial node is to avoid bottleneck nodes. Since a typical characteristic of a bottleneck node is revealed by its degree, which is in general large compared with the degree of the other nodes in the graph, a good rule is to start with a node with the minimum degree. In our example, all nodes except n_2 have degree 1. Observe that if we



(a)

IS	AS	CN
n_{11}	n_{10}, n_5	
n_{10}	n_5	1
n_5	$n_1, n_2, n_3, n_4, n_6, n_7, n_8, n_9$	8
n_1	$n_2, n_3, n_4, n_6, n_7, n_8, n_9$	7
n_2	$n_3, n_4, n_6, n_7, n_8, n_9$	6
n_3	n_4, n_6, n_7, n_8, n_9	5
n_4	n_6, n_7, n_8, n_9	4
n_6	n_7, n_8, n_9	3
n_7	n_8, n_9	2
n_8	n_9	1
n_9	ϕ	0

(b)



(c)

Fig. 7. Examples showing how the "minimum-degree initial-choice" strategy may sometimes give undesirable results. (a) An example graph. (b) Contour tableau obtained by choosing n_{10} as the initial node. (c) Another example graph.

choose any one of them as the starting node, they will all yield a tableau similar to Fig. 4(b). Besides, this minimum-degree strategy coincides with our greedy strategy since a node with the minimum degree will yield a minimum CN(1). We still may have problems in identifying correctly clusters in the graph as shown in Fig. 7(a) and (b).

In this case we identify $\{n_{11}, n_{10}\}$ and $\{n_1, \cdots, n_4, n_6, \cdots, n_9\}$ as the clusters with n_5 as the bottleneck node. This phenomenon can be classified as a special case of the so-called *redundancy phenomenon*, which we will now

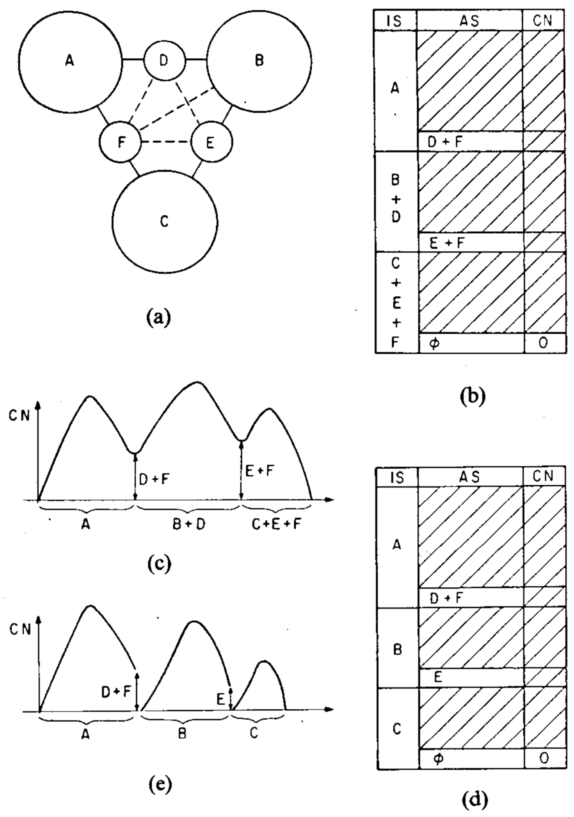


Fig. 8. A graphical illustration of the redundancy phenomenon and the dynamic contour cutting strategy to overcome it. (a) Example with three clusters. (b) Original contour tableau. (c) Original CN curve. (d) Contour tableau with dynamic contour cutting. (e) CN curve with dynamic contour cutting.

illustrate with the help of the example shown in Fig. 8(a).[4] This example shows three clusters A, B, and C separated by bottleneck nodes, D, E, and F. Let us start with A and use solid lines to denote adjacency relations and dotted lines to denote possible adjacency relations. Using the preceding cluster algorithm, we will end up with the tableau shown in Fig. 8(b) and the associated CN curve shown in Fig. 8(c). Observe that bottleneck node F is redundant in the sense that it appeared twice, in $\{D+F\}$ and $\{E+F\}$. Therefore, in selecting the best place to cut the CN curve into clusters, we have inaccurate information because $|\{D+F\} \cup \{E+F\}| \neq |\{D+F\}| + |\{E+F\}|$. The resulting cut may not be the best one that is possible. Moreover, it is unnecessary to iterate on D, E, and F in the tableau because, once they are determined to be bottleneck nodes, their adjacency is of no more concern to the remaining graph.

To overcome this redundancy phenomenon, we must resort to the concept of *dynamic contour cutting*; namely, after we have determined cluster A and its bottleneck $\{D+F\}$, we throw away $\{D+F\}$ from any future itera-

[4]Although CN is actually a *discrete* function of the iteration step, we will approximate it by drawing a continuous curve through these discrete points as shown in Figs. 8 and 9.

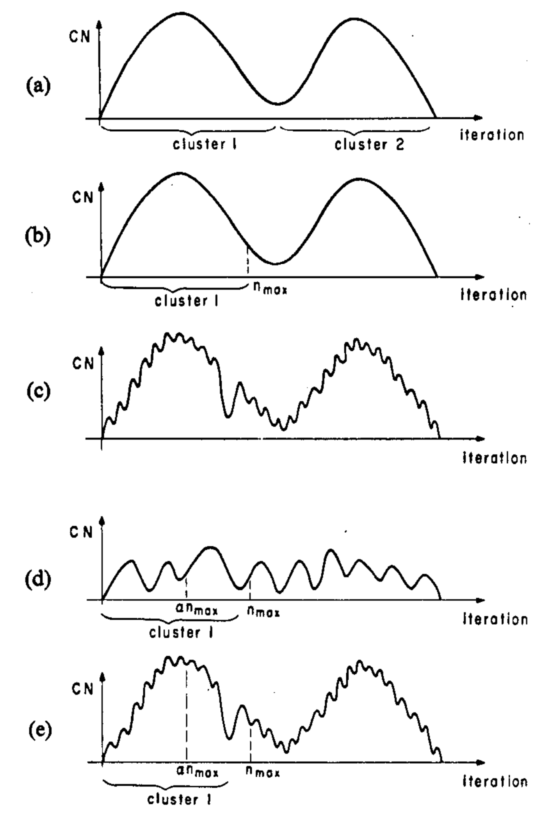


Fig. 9. An illustration of the various shapes of CN versus iteration step and some methods for grouping the nodes into clusters. (a) Smooth curve with well-defined clusters. (b) A cluster containing n_{max} nodes before a local minimum is reached. (c) A cluster containing many small wiggles. (d) A cluster containing many small clusters. (e) Least-local-minimum clustering strategy.

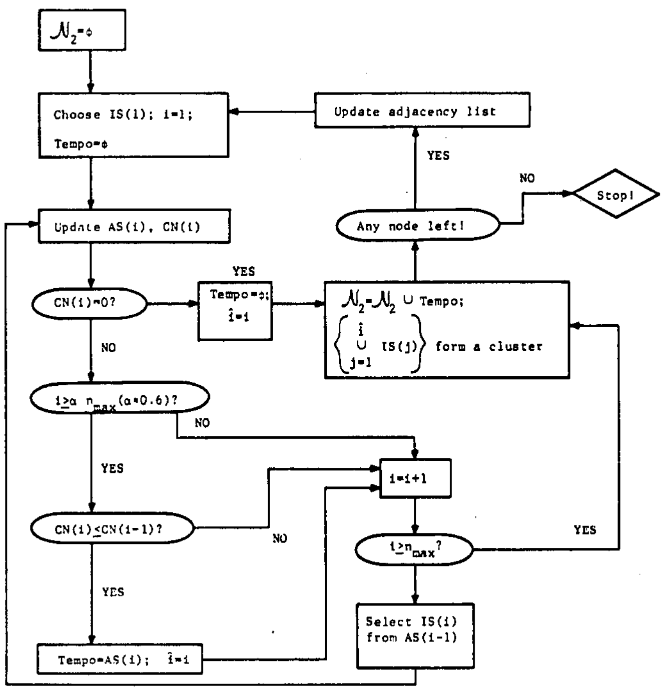

$\mathcal{N}_2=\phi$
Choose IS(1); i=1; Tempo=ϕ
Update adjacency list
YES
NO
Stop!
Any node left!
Update AS(i), CN(i)
CN(i)=0?
YES
Tempo=ϕ; i=i
$\mathcal{N}_2=\mathcal{N}_2$ ∪ Tempo;
$\left\{\bigcup_{j=1}^{i} IS(j)\right\}$ form a cluster
NO
$i\geq\alpha\, n_{max}(\alpha=0.6)$?
NO
YES
i=i+1
CN(i)≤CN(i-1)?
NO
$i\geq n_{max}$?
YES
YES
Select IS(i) from AS(i-1)
Tempo=AS(i); î=i

Fig. 10. Flow chart for the refined cluster algorithm.

tion. The dynamic contour cutting strategy will therefore yield a smaller and more efficient tableau as illustrated in Fig. 8(d) and (e). Let us now return to the example shown in Fig. 7(a). After we have determined the cluster formed by $\{n_{10}, n_{11}\}$ and bottleneck node n_5, we throw away n_5. The identification of $\{n_1,n_2,n_3,n_4\}$ and of $\{n_6,n_7,n_8,n_9\}$ as clusters is then immediate. This final result is the correct identification of the three clusters of the graph. Therefore the dynamic contour cutting enhances the efficiency of the "minimum-degree initial-choice" strategy. Another possible shortcoming of the minimum-degree selection, which cannot be avoided by the dynamic control cutting, is shown in Fig. 7(c). In this case, the graph has a cluster formed by one node only and this node is selected as the initial choice. Although it is unlikely that this situation will occur in practice, we can nevertheless avoid such poor initial choice by rejecting all initial nodes which are characterized by a large value of CN(2)-CN(1).

In our original cluster problem, the number of nodes in each cluster is constrained to be less than or equal to n_{max}. In the preceding cluster algorithm, this constraint has not yet been taken into consideration. However, we can easily incorporate it by cutting the contour whenever the number of nodes in the cluster reaches n_{max} before a local minimum is attained (Fig. 9(a) and (b)).

Another assumption that we have made in the preceding cluster algorithm is that the CN curves in Fig. 9(a) and (b) are very "smooth." In practice, the CN curve could be very erratic and may in fact contain many small wiggles as illustrated in Fig. 9(c). Moreover, it may also contain many small clusters as in Fig. 9(d). In such situations, our cluster algorithm would simply yield too many clusters, each with a very small dimension. Besides, the total number of bottleneck nodes would become too large.

To overcome the occurrence of small clusters, we can delay our searching for a local minimum until after αn_{max} nodes have been iterated, where $\alpha\approx 0.6$–0.8 (Fig. 9(d)). To overcome the occurrence of small wiggles, we can keep a record of all local minima and choose the *smallest local minimum* that occurs between αn_{max} and n_{max} as the cutoff point. This is illustrated in Fig. 9(e).

We are now ready to present a "refined" cluster algorithm which takes into consideration all of the problems identified in the preceding discussions; namely, the n_{max} constraint, the small wiggle and small cluster properties of CN curves, and the redundancy phenomenon. The flow chart for this refined cluster algorithm is presented in Fig. 10. It has to be noted that the block "update adjacency list" implements the dynamic contour cutting by removing the adjacency relations involving elements from $\mathcal{N}_2$ and from the cluster formed by

$$\left\{\bigcup_{j=1}^{i} \mathrm{IS}(j)\right\}.$$

TABLE I
TESTING RESULTS OF THE IMPLEMENTATION OF THE CLUSTER ALGORITHM

Example	n = No. of nodes	b = No. of branches	The product nb	n_{max}	No. of clusters	No. of bottleneck nodes	computer time spent*
1	25	44	1100	10	3	3	.132
2	46	69	3174	19	3	4	.208
3	14	33	462	8	2	2	.087
4	32	54	1728	12	4	4	.159
5	94	176	16544	27	5	6	.502
6	51	126	6426	15	4	9	.328
7	50	100	5000	20	3	7	.275
8	77	180	13860	30	3	10	.465
9	61	164	10004	17	4	9	.426
10	70	180	12600	25	3	12	.443

*The computer used is CDC 6400.

Let us now analyze the computational complexity[5] of the cluster algorithm.

Theorem: Let n and b denote the number of nodes and branches of the input graph, respectively; then the computational complexity of the cluster algorithm is bounded by $\mathcal{O}(nb)$.

Proof: The most time-consuming step in the cluster algorithm is the choice of the next iterating node from AS. Applying our greedy strategy, each adjacency list [20] of nodes in AS is scanned once. Let $l_0(n_k)$ denote the length of the original adjacency list of node n_k and let $l_i(n_k)$ denote the length of the adjacency list of node n_k in AS(i). The reason for distinguishing $l_0(n_k)$, $l_1(n_k), \cdots$, is that the adjacency lists actually become shorter after every iteration. Now, the computational bound can be expressed as

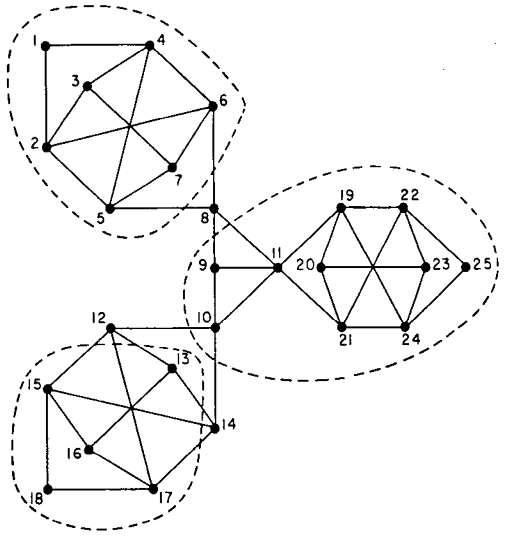
$$\sum_{i=1}^{n} \sum_{n_k \in \mathrm{AS}(i)} l_k(n_i) \leqslant \sum_{k=1}^{n} \sum_{n_k \in \mathrm{AS}(i)} l_0(n_i) = n \cdot 2b.$$

The last equality holds because each list appears throughout at most n times in the whole tableau. Hence the computational complexity of our cluster algorithm is bounded by $\mathcal{O}(nb)$. □

A computer program for implementing this cluster algorithm has been developed and the detailed results are given in [21]. We will just mention here that the program employs an efficient data structure—the *edge-oriented adjacency list* [20]— and a novel "flag" system in updating the list structures.

Part of the test results are shown in Table I, which includes a total of ten examples. For each example, we have listed the number of nodes n, the number of branches b, the product nb, the n_{max} constraint, the number of clusters yielded by the cluster algorithm, the total number of bottleneck nodes, and the computer time spent. In the sequel, we are going to discuss some of these examples in detail.

[5]The complexity used here is defined to be the number of comparisons involved.



(a)

	IS	AS	CN	
1	1	2,4	2	
2	2	4,3,5,6	4	
3	4	3,5,6	3	
4	3	5,6,7	3	
5	7	5,6	2	
6	5	6,8	2	
7	6	8	1	
8	8	9,11	2	
9	9	11,10	2	
10	11	10,19,21	3	← 1st cluster {1,2,4,3,7,5,6}; throw away cluster and bottlene bottleneck nodes (i.e., node 8); Start again.
1	11	9,10,19,21	4	
2	9	10,19,21	3	
3	10	19,21,12,14	4	
4	19	21,12,14,20,22,24	6	
5	21	12,14,20,22,24	5	
6	20	12,14,22,24,23	5	
7	23	12,14,22,24	4	
8	22	12,14,24,25	4	
9	24	12,14,25	3	
10	25	12.14	2	← 2nd cluster {11,9,10,19,21,20, 23,22,24,25}; throw away cluster and bottleneck nodes (i.e., nodes 12, 14); Start again.
1	13	16	1	
2	16	15,17	2	
3	15	17,18	2	
4	17	18	1	
5	18	φ	0	← 3rd cluster {13,16,15,17,18}; Stop!

(b)

Fig. 11. An example illustrating the cluster algorithm. (a) Example with three clusters and $n_{max} = 10$. (b) The resulting contour tableau.

Let us now examine Example 1 of Table I thoroughly, using the graph shown in Fig. 11(a) with $n_{max} = 10$. The tableau derived from our cluster algorithm is shown in Fig. 11(b). Observe that the resulting three clusters coincide with those enclosed by the three dotted lines shown in Fig. 11(a). The bottleneck is identified as $\{n_8, n_{12}, n_{14}\}$. This result is quite good since the optimal solution as obtained by inspection consists of one of the following three possibilities: $\{n_8, n_{10}\}$, $\{n_8, n_{11}\}$, or $\{n_{10}, n_{11}\}$.

Three more examples, i.e., Examples 2, 5, and 9 of

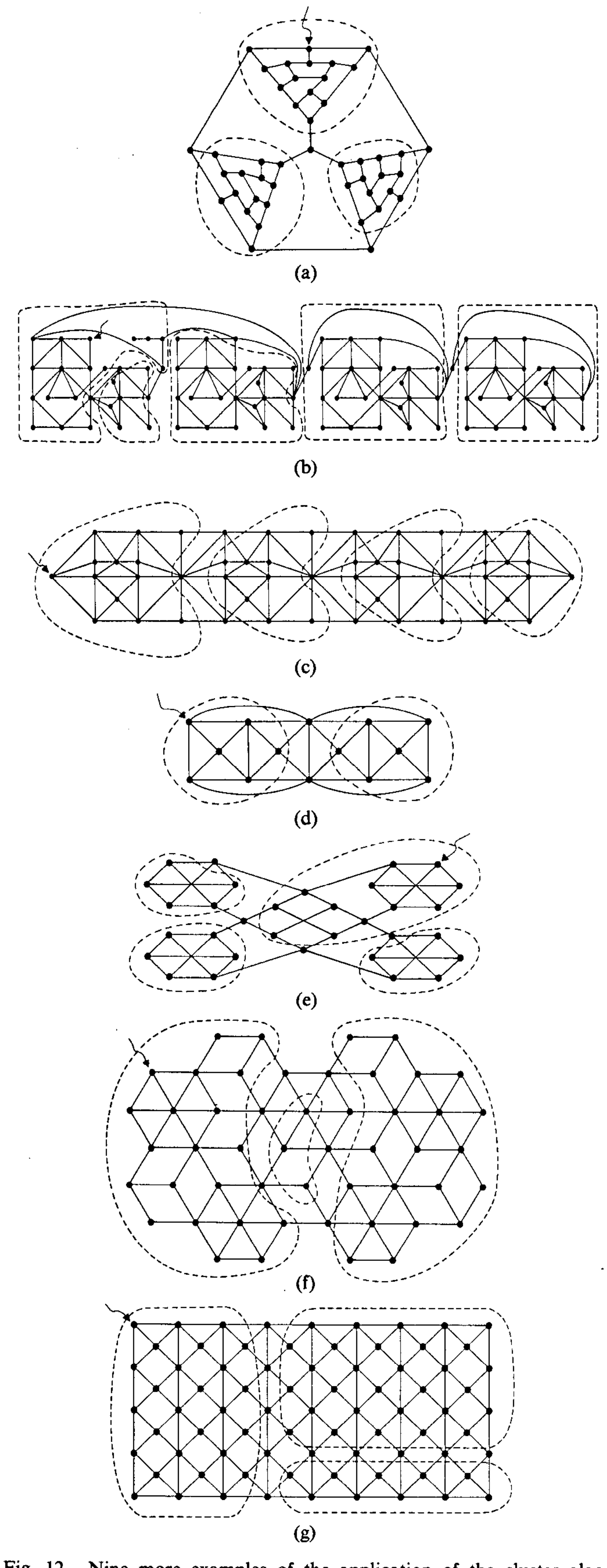


Fig. 12. Nine more examples of the application of the cluster algorithm. (a) Example 2 with three clusters and $n_{max} = 19$. (b) Example 5 with five clusters and $n_{max} = 27$. (c) Example 9 with four clusters and $n_{max} = 17$.

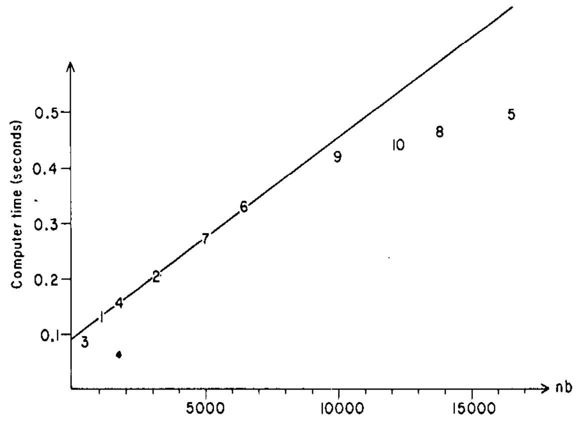


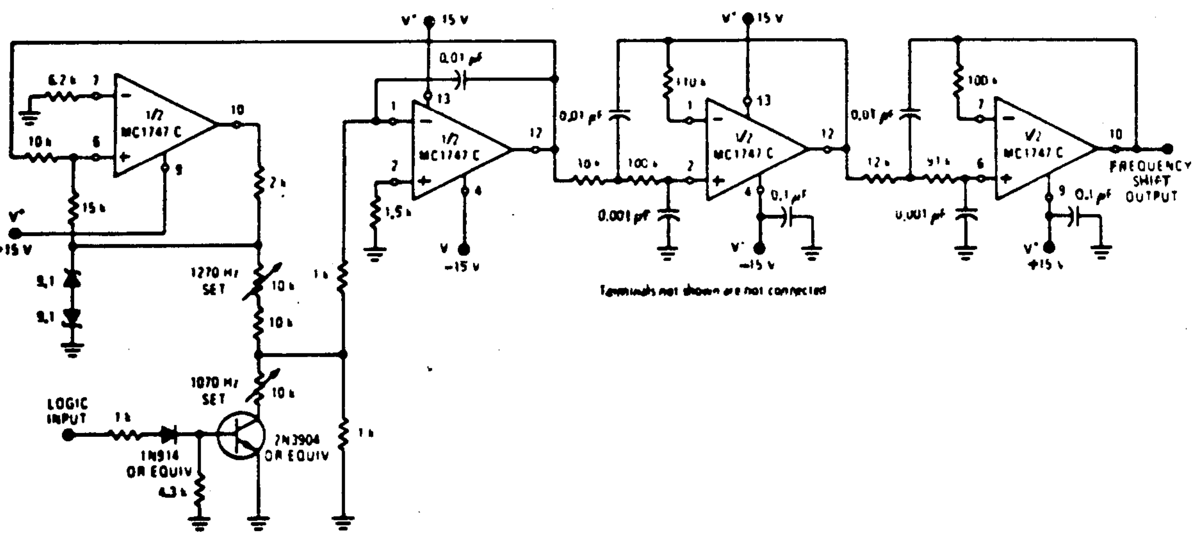
Fig. 13. The computer time spent versus *nb* plot illustrating the $\mathcal{O}(nb)$ bound. The number in this plot corresponds to the example number of Table I.

Table I, are shown in Fig. 12(a)–(c), respectively, where the initial nodes are identified by arrows and the clusters are encircled by dotted lines. The other examples used in Table I can be found in [5].

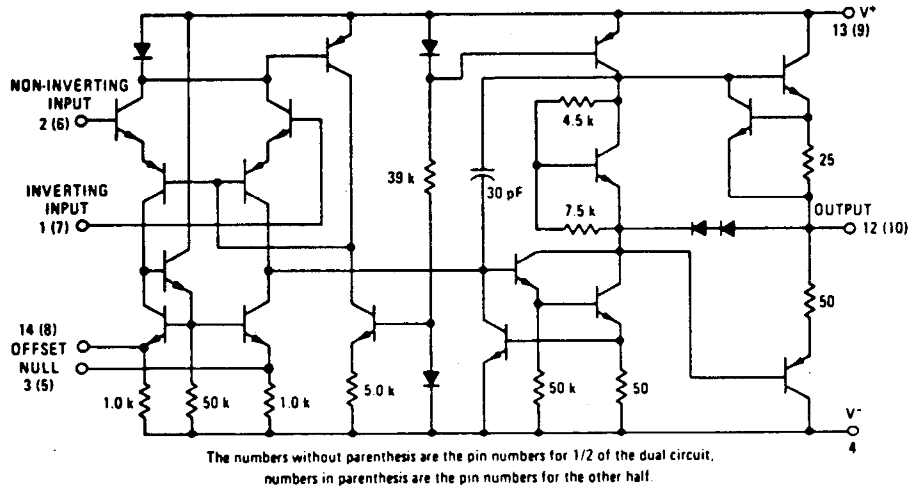
As a final remark about the computational complexity associated with the cluster algorithm, let us plot the computer times spent of Table I versus the product of *nb* in Fig. 13. It is clear that $\mathcal{O}(nb)$ is an upper bound for the complexity because all the data points are bounded by a straight line.

Before we finish this section, let us look at the practical circuit example shown in Fig. 14(a), where the schematic circuit diagram for each operational amplifier is shown in Fig. 14(b) [22]. Using the Ebers–Moll model (Fig. 14(c)) [23], each transistor is replaced by a triangular graph in the induced sparsity subgraph (Fig. 14(d)). Our associated graph optimization problem (i.e., Example 5 in Table I) contains 94 nodes and 176 branches (Fig. 12(b)). Since each operational amplifier contains 19 internal nodes, let us choose $n_{max} = 27$. Applying our cluster algorithm, we obtain five clusters shown by the dotted lines in Fig. 14(e), where the first operational amplifier is split into two clusters. This solution is reasonably good unless we demand that each operational amplifier be included in a single cluster. A careful analysis of the tableau shows that the "local" character of our greedy strategy is responsible for the separation of the first operational amplifier into two clusters. On the other hand, if one is adamant about retaining each operational amplifier as an inseparable unit within each cluster, then we should transform this problem into the following *weighted cluster problem*: Transform each operational amplifier into a "super" node with weight 19 (i.e., the total number of internal nodes) and let all other nodes have weight 1. Find the set $\mathfrak{N}_{Y_2}$ with minimum total weight such that each cluster has weight $\leqslant n_{max}$.

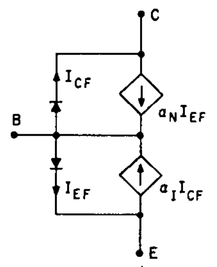
Observe that with some minor modifications, our cluster algorithm is still applicable in solving the above weighted cluster problem.



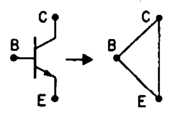
(a)


NON-INVERTING INPUT 2 (6)
INVERTING INPUT 1 (7)
14 (8) OFFSET NULL 3 (5)
OUTPUT 12 (10)
V+ 13 (9)
V- 4
The numbers without parenthesis are the pin numbers for 1/2 of the dual circuit, numbers in parenthesis are the pin numbers for the other half.

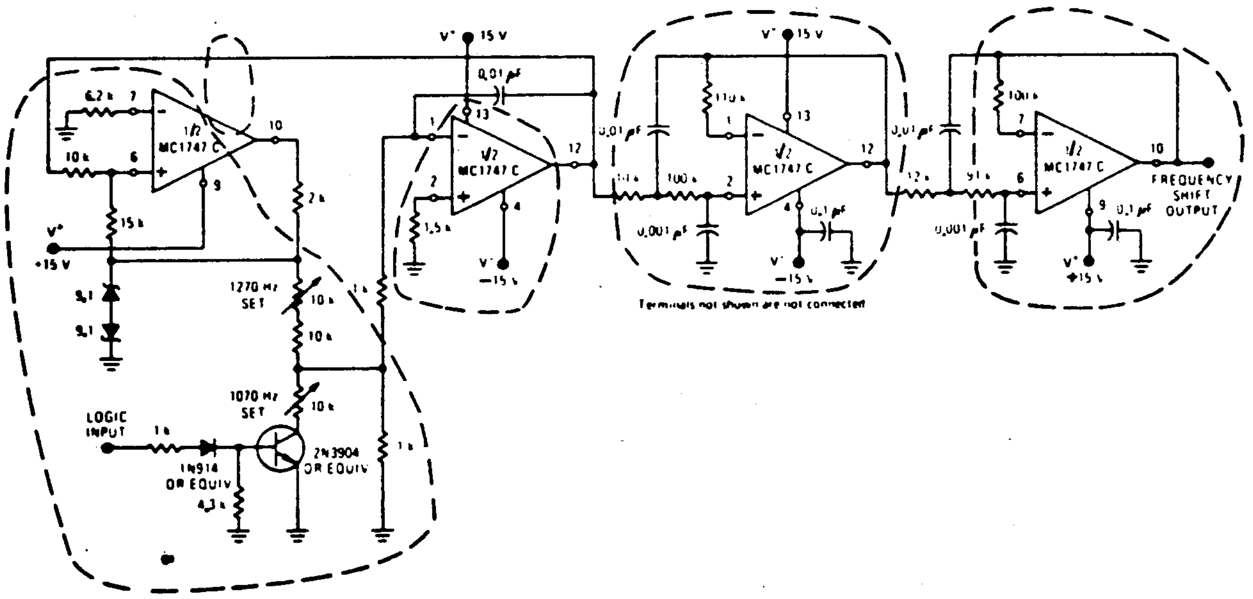
(b)



(c)



(d)



(e)

Fig. 14. A practical circuit cluster problem. (a) A frequency-shift keyer tone generator. (b) The operational amplifier circuit schematic. (c) The Ebers–Moll model for transistors. (d) The induced transistor sparsity subgraph. (e) The resulting five clusters. Note that due to the greedy strategy, the first operational amplifier is broken into two clusters.

III. Concluding Remarks

A heuristic algorithm for solving the cluster problem associated with the tearing of large-scale networks has been presented via the contour approach. First, the concept of a contour tableau was fully explored and utilized in developing our cluster algorithm. Then, several intuitive ideas such as the greedy strategy, the minimum-degree initial-node strategy, and the dynamic cutting strategy were employed to improve the efficiency of our algorithm.

The tradeoffs [15] involved in the strategies adopted were discussed together with the computational complexity of the algorithm. Finally, experimental results showed that our algorithm is highly efficient and yields near optimal solutions.

References

[1] G. B. Dantzig and P. Wolfe, "The decomposition algorithm for linear programming," *Econometrica*, vol. 29, no. 4, pp. 767–778, 1961.

[2] T. C. Hu, *Integer Programming and Network Flow.* Reading, MA: Addison-Wesley, 1969.

[3] G. Kron, *Diakoptics—Piecewise Solution of Large-Scale Systems.* London, England: MacDonald, 1963.

[4] L. O. Chua and L. K. Chen, "Diakoptic and generalized hybrid analysis," *IEEE Trans. Circuits and Systems*, vol. CAS-23, pp. 694–705, Dec. 1976.

[5] A. Sangiovanni-Vincentelli, L. K. Chen, and L. O. Chua, "Node-tearing nodal analysis," Electronics Research Laboratory, Univ. California, Berkeley, Memo. No. ERL-M582, Sept. 1976.

[6] ——"A new tearing approach—The node-tearing nodal analysis," in *Proc. 1977 IEEE Int. Symp. Circuit and Systems* (Phoenix, AR, Apr. 1977) pp. 143–148.

[7] G. Guardabassi and A. Sangiovanni-Vincentelli, "A two levels algorithm for tearing," *IEEE Trans. Circuits and Systems*, vol. CAS-23, pp. 783–791, Dec. 1976.

[8] F. M. Callier, W. S. Chan, and C. A. Desoer, "Input-output stability theory of interconnected systems using decomposition techniques," *IEEE Trans. Circuits and Systems*, vol. CAS-23, pp. 714–729, Dec. 1976.

[9] D. Ferrari, "Improving locality by critical working sets," *Comm. of ACM*, vol. 17, no. 11, pp. 614–620, Nov. 1974.

[10] W. E. Donath and A. J. Hoffman, "Lower bounds for the partitioning of graphs," *IBM J. Res. and Dev.*, vol. 17, no. 5, pp. 420–425, Sept. 1973.

[11] E. C. Ogbuobiri, W. F. Tinney, and J. W. Walker, "Sparsity-directed decomposition for Gaussian elimination on matrices," *IEEE Trans. Power, Appr. Syst.*, vol. PAS-89, pp. 141–150, Jan. 1970.

[12] F. Luccio and M. Sami, "On the decomposition of networks in minimally interconnected subnetworks,"*IEEE Trans. Circuit Theory*, vol. CT-16, pp. 184–188, May 1969.

[13] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell System Tech. J.*, vol. 49, no. 2, pp. 291–307, Feb. 1970.

[14] J. C. Gower, "Comparison of some methods of cluster analysis," *Biometrics*, vol. 54, pp. 623–637, Dec. 1967.

[15] S. Lin, "Heuristic programming as an aid to network design," *Networks*, vol. 5, no. 1, pp. 33–43, 1975.

[16] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms.* Reading, MA: Addison-Wesley, 1974.

[17] R. M. Karp, "On the computational complexity of combinatorial problems," *Networks*, vol. 5, no. 1, pp. 45–68, 1975.

[18] S. Parter, "The use of linear graphs in Gaussian elimination," *SIAM Review*, vol. 3, no. 2, pp. 119–130, Apr. 1961.

[19] R. A. Willoughby, "A survey of sparse matrix technology," IBM Research, Yorktown Heights, New York, NY, Rep. RC 3872, May 1972.

[20] L. K. Chen, B. S. Ting, and A. Sangiovanni-Vincentelli, "An edge-oriented adjacency list for undirected graphs," Electronics Research Laboratory, Univ. California, Berkeley, Memo. No. ERL-M589, May 1976; also *Int. J. Circuit Theory Appl.*, to be published.

[21] E. C. Cua, "Fundamental loops generation and clusters analysis—algorithm and implementation," Master's thesis (plan II) Department of Electrical Engineering and Computer Science, Univ. California, Berkeley, June 1976.

[22] *Linear Integrated Circuits Data Book*, Motorola, 1972, pp. (7-440)–(7-442).

[23] L. O. Chua and P. M. Lin, *Computer-Aided Analysis of Electronic Circuits: Algorithms and Computational Techniques.* Englewood Cliffs, NJ: Prentice-Hall, 1975.

[24] J. Edmonds, "Matroids and the greedy algorithm," *Math. Programming*, vol. 1, pp. 127–136, Nov. 1971.

Alberto Sangiovanni-Vincentelli (M'74) was born in Milano, Italy, on June 1947. He received the Dr. Eng. degree (Summa cum Laude) from the Politecnico di Milano, Milano, Italy, in 1971.

From 1974 to 1977 he held the position of Associate Professor at the Istituto di Elettrotecnica and Elettronica del Politecnico di Milano, Milano, Italy. In 1975 he was Research Associate at the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley. During the academic year 1976–1977, he was Visiting Assistant Professor at the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley. Since July 1977, he has been Assistant Professor at the same department. His research interests are in computer-aided design of electronic circuits and systems, large-scale systems, and layout of large-scale electronic circuits.

Dr. Sangiovanni is a member of IEEE. He is a member of the Large-Scale Networks Committee of the IEEE Circuits and Systems Society.

Li-Kuan Chen (S'73–M'77) was born in China, 1949. He received the B.S., M.S., and Ph.D. degrees in electrical engineering and computer sciences from the University of California, Berkeley, in 1970, 1974, and 1976, respectively.

He has since been associated with the Computer Applications Division of the American Electric Power Service Corporation. His current responsibility lies in the area of application of analytical and computational techniques in the operations of AEP river transportation system. His current interests also include large-scale networks and the application of modern circuit and system theory to power system problems.

Leon O. Chua (S'60–M'62–SM'70–F'74), for a photograph and biography please see page 117 of the March issue of this Transactions.