Ratio Cut Partitioning for Hierarchical Designs

Yen-Chuen Wei, *Student Member, IEEE*, and Chung-Kuan Cheng, *Member, IEEE*

***Abstract*—Circuit partitioning is an important process for hierarchical designs. A good partitioning can significantly improve circuit performance and reduce layout costs. This paper proposes a partitioning approach called *ratio cut*. We demonstrate that the ratio cut algorithm can locate the clustering structures in the circuit. Finding the optimal ratio cut is *NP*-complete. However, in certain cases the ratio cut can be solved by linear programming techniques via the multicommodity flow formulation. We also propose a fast heuristic algorithm running in linear time with respect to the number of pins in the circuit. Experiments show good results in all tested cases.**

I. Introduction

1.1. Hierarchical Designs and Circuit Partitioning

With MODERN fabrication technology, VLSI circuitry has become so dense that a single silicon chip may contain thousands of transistors. As a consequence, the sizes of many VLSI design problems have become very large. Using the concept of "divide-and-conquer," a huge and complicated problem is first divided into small subproblems, so that the complexity of the original problem can be dramatically reduced. Therefore, partitioning plays an important role in attacking VLSI design problems.

Many partitioning approaches have been applied to various VLSI design problems. In silicon compilation, for example, partitioning operates on a behavioral or functional hardware description and guides the design of the data-path structure [17]. In placement and routing, hierarchical design has been proposed and generated excellent results [6]. In logic synthesis, decomposition operations were proposed to cluster or factorize the Boolean expression and thus reduce the production cost [2]. In PLA design, partitioning is used to reduce total area and improve the circuit speed [11].

In particular, given a system, partitioning is used to divide the whole circuit at all levels of the design hierarchy: from the system level to board level, from board level to chip level, and from chip level to the macrocell [7]. This kind of circuit partitioning results in a hierarchical design problem. Since different partitionings correspond to different system implementations, a good partitioning can significantly improve system performance and reduce layout costs.

Manuscript received January 23, 1990. This was supported in part by the National Science Foundation under Grant MIP-9009260 and by grants from MICRO, Amdahl, Cadence, Data General, Elite, Hughes, and Intergraph. This paper was recommended by Associate Editor R. H. J. M. Otten.

The authors are with the Department of Computer Science and Engineering, University of California at San Diego, La Jolla, CA 92093.

IEEE Log Number 9143306.

1.2. Previous Work

Many approaches have been proposed for attacking circuit partitioning problems. They include clustering [3], [20], eigenvector decomposition [10], network flow [9], group swapping [8], [13], [15], [19], [21], [22], simulated annealing [14], etc.

Clustering [3], [20] is an intuitive method for building up clusters based on interconnections among components. Usually, certain seeds are given from input specification as starting points. Neighboring components with strong connections to each seed are merged with that seed. This approach is limited by the lack of a global view taking into account the connectivity of the entire system. Therefore, clustering is commonly used to yield an initial partitioning.

In eigenvector decomposition [10], connections are represented in a matrix. The eigenvectors of the matrix define the locations of all components and thus derive partitioning results. This method requires the transformation of every multipin net into several twopin nets in real circuits before establishing the matrix.

The maximum-flow-minimum-cut algorithm was presented by Ford and Fulkerson [9]. They transformed the minimum cut problem into the maximum flow problem. In order to separate a pair of nodes into two subsets, the minimum number of crossing edges is equal to the maximum amount of flow from one node to the other. Although this algorithm can find the optimal solution between any pair of nodes in a network, there is no constraint on the sizes of resultant subsets. In practice, the result will be useless if two very unevenly sized subsets are generated.

Kernighan and Lin proposed a two-way partitioning algorithm [13] with constraints on the subset size. They randomly started with two subsets, and iteratively applied pairwise swapping on all pairs of nodes. Subsequently, many improvements have been made to this method. Schweikert and Kernighan [21] proposed the use of a net cut model so that the algorithm can handle multipin net cases. Fiduccia and Mattheyses [8] improved this algorithm by reducing time complexity to $O(P)$ with respect to the number of pins P, and Krishnamurthy [15], [19] further added in lookahead ability. Recently, Sechen and Chen [22] proposed a new multipin net model and

0278-0070/91/0700-0911$01.00

achieved excellent results. Generally speaking, the Kernighan–Lin based algorithm is quite efficient but it needs a predefined subset size to start with.

Simulated annealing [14] is another successful example of the iterative improvement method. The objective function in simulated annealing is analogous to energy in a physical system, and each move is analogous to changes in the energy of the system. The Metropolis Monte Carlo method is introduced to decide whether a move is accepted. Simulated annealing usually produces good results at the expense of very long running time.

1.3. Motivation

The partitioning of a system can be viewed as assigning operations hierarchically from a high-level circuit to boards, from boards to chips, and from chips to modules. The resultant hierarchical partitioning information can be stored in a tree structure. Fig. 1 shows such a configuration. The root of the tree represents the system, each node of the tree represents a partitioned subcircuit of its parent, and the leaves of the tree correspond to the subcircuits partitioned in the last level. Suppose each node of the tree has at most N branches and the tree has M levels. We say that the tree represents an *N-way* and *M-level* partition. The evaluation of the partition result depends on the final integration of all partition levels, from the basic subcircuits to the whole system. A greedy approach to find a minimum cut at one level may sacrifice the quality of cuts for the following levels. Therefore, instead of finding a minimum cut at a specific level, we should locate the clustering structure in the circuit.

The maximum-flow-minimum-cut algorithm seems to be a good tool for finding the clustering structure since it derives a minimum cut of the circuit. However, in practice the cut usually generates two subcircuits of very uneven sizes. We observed this phenomenon after implementing the algorithm and applying it to the partition of several circuits [23]. We found that almost all of the partitions separated only one of three components from the rest of the circuit. On the other hand, the Kernighan–Lin algorithm can yield a partition of two comparable sizes, but these sizes are fixed or predefined. Observing that almost all circuit layouts have the tendency to put components of similar functionality together to form a strongly connected group [7], one concludes that clustering structures in circuits are quite natural. Predefining the partition size may not be well suited for hierarchical designs, since there is no way to know cluster size in circuits before partitioning. Fig. 2 shows a ten node example, which will be partitioned as in Fig. 2(a) if the Kernighan–Lin algorithm is applied with subset size predefined as one half of its total size. The reader could easily identify a more reasonable partition like the one in Fig. 2(b). We propose to adopt an approach termed **ratio cut** as a new metric in order to locate natural clusters in the circuit like those in Fig. 2(b).

In the following section, we first use a random graph model to derive the concept of the ratio cut. We also explain the ratio cut through a closely related communication problem by a multicommodity flow formulation. Then in Section III, we propose a fast heuristic algorithm which handles multipin nets in real VLSI circuitry. The ratio cut algorithm with size constraints is also considered. Experimental results are presented in Section IV. We compare the ratio cut algorithm with other state-of-the-art algorithms by using fifteen real circuits from different resources. Finally, we give our conclusions and future research directions in Section V.

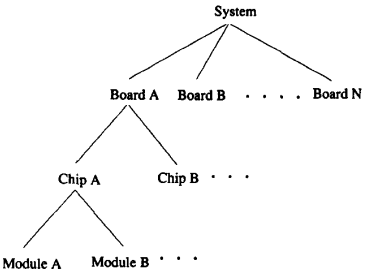

Fig. 1. A tree structure for hierarchical design.

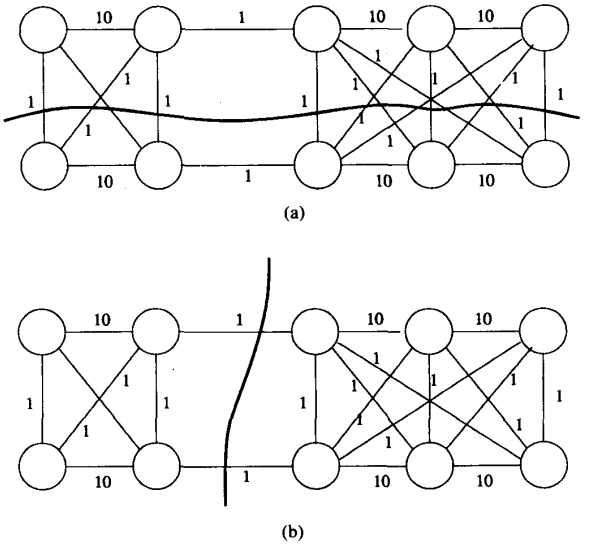

Fig. 2. A ten node example. (a) The Kernighan–Lin approach. (b) A better partitioning.

II. The Ratio Cut

Given a network $N = (V, E)$ where V is the set of nodes and E is the set of edges, let c_{ij} be the capacity of an edge connecting node i and node j. (A, A') denotes a cut that separates a set of nodes A from its complement $A' = V - A$. The capacity of this cut is equal to $C_{AA'} = \Sigma_{i \in A} \Sigma_{j \in A'} c_{ij}$. The **ratio** of this cut is defined as $R_{AA'} = (C_{AA'}/|A| \times |A'|)$, where $|A|$ and $|A'|$ denote the cardinalities of subsets A and A', respectively. The **ratio cut** is the cut that generates the minimum ratio among all cuts

in the network, i.e., $\min_A (C_{AA'}/|A| \times |A'|)$ ($A \subset V$ and $A \neq \emptyset, A' \neq \emptyset$).

2.1. The Clustering Property of the Ratio Cut

The clustering property of the ratio cut can be interpreted by a random graph [1] model. Fig. 3 shows a *uniformly distributed random graph* with n nodes. The probability of having an edge connecting each pair of nodes in this graph is equal to an identical value f. Consider a cut CUT_1 in Fig. 3 which partitions the network into two subsets A and A' with comparable sizes $\alpha \times n$ nodes and $(1 - \alpha) \times n$ nodes, respectively, where $0 < \alpha < 1$. The expected capacity $C_{AA'}$ of cut CUT_1 equals the probability f multiplied by the number of possible edges between A and A':

$$E(C_{AA'}) = f \times |A| \times |A'| = f \times \alpha \times (1 - \alpha) \times n^2. \tag{1}$$

On the other hand, if another cut CUT_2 in Fig. 3 separates only one node s from the rest of the nodes, the expected number of edges crossing CUT_2 is

$$E(C_{\{s\}\{s\}'}) = f \times (n - 1). \tag{2}$$

As n approaches infinity, the value of (1) becomes much larger than (2). This derivation explains why the maximum-flow-minimum-cut method tends to generate very uneven sized subsets in the experimental results mentioned earlier. The very uneven sized subsets naturally give the lowest cut value. Therefore, a ratio value $(C_{AA'}/|A| \times |A'|)$ is proposed to alleviate the hidden size effect.

As a consequence, the expected value of this ratio is a constant with respect to different cuts:

$$E(R_{AA'}) = E\left(\frac{C_{AA'}}{|A| \times |A'|}\right) = \frac{f \times |A| \times |A'|}{|A| \times |A'|} = f. \tag{3}$$

Thus if the edges of the graph are uniformly distributed, all cuts have the same ratio value. In other words, the choice of the cuts does not make a difference in such a uniformly distributed random graph. However, in a general network different cuts generate different ratios. Cuts that go through weakly connected groups correspond to smaller ratios. The minimum of all cuts according to their corresponding ratios defines the sparsest cut [16] since this cut deviates the most from the cuts of a uniformly distributed graph.

2.2. The Ratio Cut Via A Multicommodity Flow Formulation

The ratio cut can be explained by an analogy to certain communication problems [4], [16]. In a communication system (Fig. 4), suppose each user wants to communicate with every other user with identical demand f. We can use nodes to represent the users, and edges with weights to represent the capacities for the communication. The objective is to increase the demand f as much as possible.

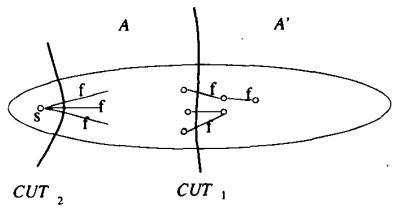

Fig. 3. A random graph with n nodes and probability f.

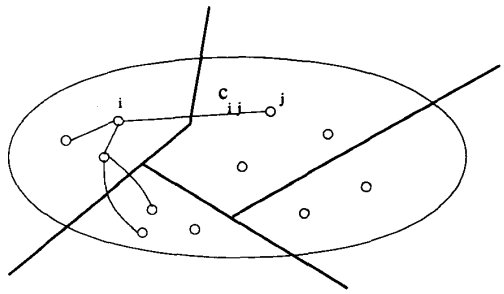

Fig. 4. A communication system.

As f is increased, some of the edges become saturated. If the saturated edges constitute a cut in the system, the cut forms a communication bottleneck, and no more communication demand can be added.

This communication problem can be formulated as a *uniform multicommodity flow* problem [4], [16]. In a multicommodity flow problem [5], each flow (communication) which starts from a distinct source node (user) is considered as a distinct commodity. While each commodity has its own flow conservation constraints, the commo 'ities interact when they flow on the same edges either by competing for edge capacity or by causing congestion effects.

Let commodity p denote the flow that starts from node p as the source to the other nodes as the destinations. Let x_{ij}^p be the flow for commodity p on edge (i, j), n be the total number of nodes and commodities, and c_{ij} be the capacity of edge (i, j); the uniform multicommodity flow can be formulated as a linear programming problem [4], [16]:

$$\text{Obj: } \max f \tag{4}$$

subject to the flow demand of commodity p from every node p to the other nodes i:

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

and the constraints of the edge capacities,

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

where $x_{ij}^p \geq 0$ for all $1 \leq i, j, p \leq n$.

The above linear programming problem can be transformed into its dual formulation by assigning dual variables as prices associated with the constraints of the problem [5]. Let us assign dual variables λ_i^p for the vertex $i(i = 1, \cdots, n)$ with respect to commodity p, and d_{ij} to edge (i, j). We can derive the dual problem [5]:

$$\text{Obj: } \min \sum_{ij} c_{ij} d_{ij} \tag{5}$$

subject to

$$d_{ij} \geq |\lambda_i^p - \lambda_j^p|, \quad 1 \leq i, j, p \leq n \tag{6}$$

$$\sum_{p=1}^{n} \sum_{i \neq p, i=1}^{n} (\lambda_i^p - \lambda_p^p) \geq 1. \tag{7}$$

Let d_{ij} be an undirected distance function on the edge connecting vertex i and vertex j. From (6), d_{ij} can be interpreted as a metric system with the triangular inequality [12], [18]:

$$d_{ij} + d_{jk} \geq d_{ik}, \quad 1 \leq i, j, k \leq n. \tag{8}$$

To minimize (5), we can derive that $\lambda_i^p - \lambda_p^p = d_{ip}$. Thus constraint (7) can be rewritten as a function of d_{ij}:

$$\sum_{p=1}^{n} \sum_{i \neq p, i=1}^{n} d_{ip} = 1. \tag{9}$$

Therefore, the uniform concurrent flow problem can also be stated as an objective function of (5) with constraints (8) and (9).

When the edges with positive distance d_{ij} form a two-way partition, we can show that the partition defined the ratio cut. When the edges with positive distances form a k-way partition with $k \leq 4$, we also find that there exists a two-way partition that again defines the ratio cut [4].

Theorem: Let $D = \{(i, j) \mid d_{ij} > 0\}$ define a partition that separates the network into k disconnected subsets. If $k \leq 4$, then there exists a ratio cut that is a subset of D.

Proof:

1) The case when $k = 2$:

a. First of all, we want to prove that for all $d_{ij} \in D$, $d_{ij} = e$, where e is a constant.

Let (A, A') be the partition made by D. Let i and j be the nodes in A, and k be the node in A'. From (8) we have $d_{ij} + d_{jk} \geq d_{jk}$ and $d_{ij} + d_{ik} \geq d_{jk}$. Since $d_{ij} = 0$ by assumption, we have $d_{ik} = d_{jk}$. Similarly, let l be a node in A', then we have $d_{jk} = d_{jl}$. Hence, we have $d_{ik} = d_{jk} = d_{jl}$ for all i and j in A, and k and l in A'.

b. Secondly, we prove that $e = 1/2\ (1/|A| \times |A'|)$.

Since $d_{ij} = e$ for all $d_{ij} \in D$, from (9), we have $e = (1/2|A| \times |A'|)$.

c. From a and b, we have the object function $\Sigma\ c_{ij} d_{ij} = (C_{AA'}/2|A| \times |A'|)$. Since this is a minimum solution, D is the optimal ratio cut.

2) The case when $k = 3$:

a. Let A_1, A_2, A_3 be the disjoint node sets partitioned by D. Similarly to the proof of 1)a, we have $d_{kl} = e_{ij}$ for all $k \in A_i$, $l \in A_j$, and $i \neq j$, $i, j \in \{1, 2, 3\}$, where e_{ij} is a constant.

b. From a., (5), (8), and (9) can be expressed as functions of e_{12}, e_{13}, and e_{23}. The constraint (8) is expressed with the following three equations:

$$\begin{aligned} e_{12} + e_{13} + S_1 &= e_{23} \\ e_{12} + e_{23} + S_2 &= e_{13} \\ e_{13} + e_{23} + S_3 &= e_{12} \end{aligned} \tag{10}$$

where S_1, S_2, and S_3 are non-negative slack variables. Thus we have four constraint equations and six variables. Deriving from the fundamental theorem of linear programming, we can have two variables equal zero in the optimal solution. If $e_{ij} = 0$ for $i, j \in \{1, 2, 3\}$, then by definition D becomes a two-way partitioning. Let us set two slack variables, S_i, S_j equal zero. From (10), we then can derive $e_{ij} = 0$. Consequently, we reduce D to a two-way partitioning case 1).

3) The case when $k = 4$:

The proof is similar to the proof of 2). For the case of $k = 4$, we have thirteen constraint equations and eighteen variables. Five slack variables can be zero in the optimal solutions. Likewise, we can derive that an e_{ij} should be zero. Consequently, the problem is reduced to a three-way partitioning: case 2). Q.E.D.

III. A Heuristic Ratio Cut Algorithm

3.1. Overview of the Algorithm

Like many other partitioning problems, finding the ratio cut in a general network belongs to the class of *NP*-complete problems [16]. Therefore, a good and fast heuristic algorithm is needed for dealing with complicated VLSI circuitry. We adopt the strategy presented by Fiduccia and Mattheyses [8] in our algorithm. The advantages of their algorithm include: 1) it is efficient due to linear running time; 2) it obtains relatively promising results; and 3) it is able to handle multipin nets and fix certain modules on certain sides.

The ratio cut algorithm consists of three major phases: 1) initialization, 2) iterative shifting, and 3) group swapping. These phases are discussed in more detail below.

3.1.1. Initialization: We release the constraint on subset size. The algorithm dynamically establishes its own subsets which are close to clusters in the circuit. A module s is randomly chosen as a seed. Another seed t is found at the end of a longest path by breadth-first search starting from the seed s. We start the cluster with either s or t and each time merge one best candidate with the seed until all modules (except the other seed) are exhausted. Since each merging step results in a new ratio value, we record every ratio value even if it is increasing, so that the algorithm can jump out of local minima. The cut giving the minimum ratio forms an initial cluster for the circuit. Suppose

a network has a set of modules M, the procedure is as follows.

1) Randomly choose a module s. Find the other seed module t at the end of a longest path by breadth-first search starting from s. Let $X = \{s\}$, and $Y = M - \{s, t\}$.
2) Choose a module i in Y whose movement to X will generate the best ratio among all the other competing modules. Move module i from Y to X; update $X = X \cup \{i\}$, and $Y = Y - \{i\}$.
3) Repeat step 2) until $Y = \varnothing$.
4) Repeat steps 2) and 3) with $X = \{t\}$ and $Y = M - \{s, t\}$ until $Y = \varnothing$.
5) The cut giving the minimum ratio found in the procedure forms the initial partitioning.

Suppose we fix seed s at the left end, and seed t at the right end. Fig. 5(a) shows the direction of module movements from s to t, and Fig. 5(b) shows the direction from t to s. This configuration is used throughout this paper. We define a **right (left) shifting operation** as shifting the modules with the best resultant ratio value from $s(t)$ toward $t(s)$. We will discuss the implementation of the left/right operation so that the whole procedure runs in linear time later.

3.1.2. Iterative Shifting: Once we have an initial partitioning, we repeat the shifting operations in the opposite direction to achieve further improvement. The same procedure is iteratively applied to the latest partition position. If the direction of the initial partitioning is found from s to t (Fig. 6), then the shifting process begins with the following operations.

1) Repeat right shifting operations until all the modules are exhausted.
2) Choose the minimal ratio value obtained in step 1). If the new ratio value is reduced from step 1), then the cut producing this ratio forms a new starting partition; otherwise, output the previous partition and exit the process.
3) Repeat steps 1) and 2) with left shifting operations.
4) Repeat steps 1)–3).

Fig. 7 is a two-pin net example to show the effect of iterative shifting operations after the initialization phase. Suppose that the two seeds s and t have been chosen. From s to t, a cut with the ratio $1/26$ can be found in Fig. 7(a); from the other direction t to s, a cut with a ratio value $1/28$ is located in Fig. 7(b). Since the ratio value of the first cut is lower than the second cut, we use the first cut as our initial partitioning. Apply steps 1) and 2) of the iterative shifting procedure to the cut in Fig. 7(a) in the opposite direction, we will obtain a new cut with a better ratio value $3/(8 \times 16)$ indicated by the first arrow in Fig. 7(c). Followed by step 3) in the opposite direction indicated by the second arrow in Fig. 7(c), we finally reach a cut with the ratio $3/(12 \times 12) = 1/48$, which is the optimal ratio cut in this example.

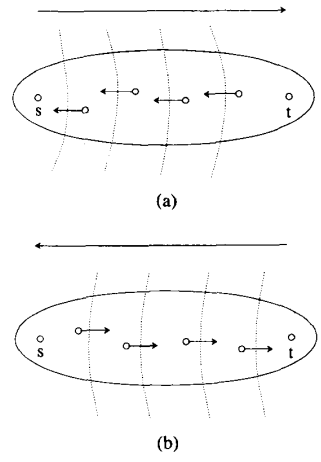

Fig. 5. Initial partitioning. (a) From s to t. (b) Fron t to s.

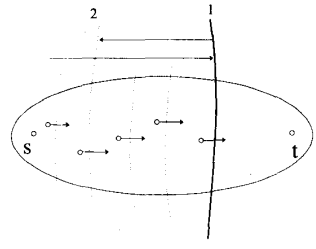

Fig. 6. Iterative shifting.

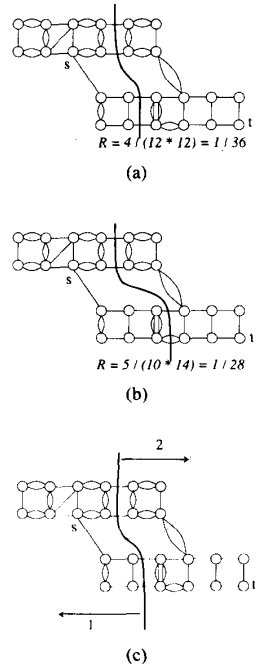

Fig. 7. An example for the first two phases. (a) The initialization phase from s to t. (b) The initialization phase from t to s. (c) The iterative shifting.

3.1.3. Group Swapping: After the iterative shifting is completed, a local minimal partitioning is reached in the sense that a single module movement from its current subset to the other cannot reduce the ratio value. Thus a group swapping technique is utilized to make further improvements.

As shown in Fig. 8, the network is partitioned into two subsets after the iterative shifting phase. We define the *ratio gain* $r(i)$ of a module i to be the ratio decreased if module i(except the two seeds s and t) were moved from its current subset to the other. The ratio gain could be a negative real number if the module movement increases the ratio value of the cut. The group swapping process is as follows.

1) Calculate the ratio gain $r(i)$ for every module i, and set all modules to the "unlocked" state.
2) Select an unlocked module i with the largest ratio gain from two subsets.
3) Move module i to the other side, and lock it.
4) Update the ratio gains for the remaining affected and unlocked modules.
5) Repeat steps 2)–4) until all modules are locked.
6) If the largest accumulated ratio gain during this process is positive, swap the group of modules corresponding to the largest gain, and go to step 1); otherwise, output the previous partition and stop.

The implementation of the group swapping technique is based on the the *bucket list data structure* proposed by Fiduccia and Mattheyses [8]. Let (A, A') be the cut derived from the iterative shifting, the ratio gain for moving module i with size $s(i)$ from A' to A is expressed by $r(i) = g(i)/[|A| + s(i)] \times [|A'| - s(i)]$, where $g(i)$ is the decrement of the number of nets crossing the cut if module i were moved to the other side, i.e., the *net gain* defined in [8]. Since every $g(i)$ is an integer in the range of $-p(i)$ to $p(i)$, where $p(i)$ is the number of pins in module i, the bucket can be represented by a finite array whose kth entry contains a double-linked list of modules with net gains currently equal to k. We maintain two such buckets in the ratio cut algorithm, one for each subset. The ratio gain calculations are only needed in step 2). The detailed procedure for selecting a module to be moved in step 2) is as follows.

a) Consider all modules with the highest net gain from two bucket arrays.
b) Among those modules returned in step a), choose a module whose movement will give us the best ratio gain.

The reason we only consider modules with highest net gains in the bucket is that unless the size of a module is unusually large, it will not contribute much in the denominator to the ratio value $g(i)/[|A| + s(i)] \times [|A'| - s(i)]$ as the net gain appearing in the numerator. Therefore, the effect of size is only considered in step b). However, if there are huge modules in the circuit, such as macroblocks in the standard cell layout, then the algorithm handles the huge modules as a special case since the number of huge modules is proportionally much smaller than the total number of modules in the circuit.

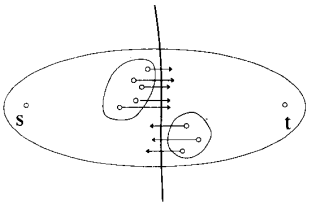

Fig. 8. Group swapping.

3.2. Complexity

The reader may notice that the initialization and iterative shifting can be implemented as special cases of the group swapping. These two phases move in only one direction all the way to the end, while the group swapping has the flexibility of moving modules in both directions. The group swapping using the bucket list data structures, as derived in [8], has the complexity of $O(P)$, where P is number of pins. In addition, finding a longest path by breadth-first search is also bounded by $O(P)$. It is straightforward to show that the initialization and iterative shifting phases can also be done in $O(P)$ time complexity. Thus the time complexity of the whole ratio cut algorithm remains linear in the number of pins.

3.3. The Algorithm with Size Constraints

The philosophy of the ratio cut is to identify the clusters in the circuit. We have removed the constraint on subset sizes, and let the ratio cut approach automatically provide us with the subsets which are natural clusters in the circuit. Thus the previous algorithm has the full freedom in terms of the resultant subset sizes. However, sometimes we must force a size constraint on the resultant subsets due to the physical limitation of system designs. A typical example is to divide an entire system into several cabinets, and each cabinet contains several hundred boards. Since the number of cabinets and the size of each cabinet are fixed by physical limitation, an upper bound on every partitioned subset size should be introduced so that the number of subsets is equal to the number of cabinets, and each subset can fit into the size of the cabinet simultaneously.

In order to force a partitioning with the size constraint, an upper bound of the subset size is given from the input specification. Each time we apply the original ratio cut algorithm to the circuit and obtain a partitioning result, two subsets return from the algorithm. If one of the subset sizes is larger than the upper bound, we just take out the small subset and keep on applying the ratio cut algorithm to the remaining largest subset in a reduced network until all subset sizes are less than the upper bound.

The final step is to put back all small groups we have taken out and distribute them into proper subsets. We can assign the larger subset from the last partitioning as one subset, and all other subsets generated during the process as another subset. The algorithm applies one pass of left/right shifting operations to move a certain number of modules from the large subset to the small subset, and locate the best possible ratio cut satisfying the size constraint. Note that the shifting operations in this final step are limited by the upper bound of the physical size. Thus the partitioning quality may be sacrificed in order to make the size constraint.

Given a network N and an integer *upperbound*, the algorithm with size constraints is as follows.

1) Initialize $\Psi = \{N\}$.
2) Choose $N' \in \Psi$ such that $|N'| = \max |A_i|, A_i \in \Psi$. Set $\Psi = \Psi - \{N'\}$. Apply the ratio cut algorithm to N' to obtain a cut (A, A') where $N' = A \cup A'$, assuming $|A| \geq |A'|$.
3) If ($|A| \leq$ *upperbound*), go to 4); otherwise, let $\Psi = \Psi \cup \{A, A'\}$, and go to 2).
4) Let A be one subset, $A'' = N - A$ be the other subset. Do one pass of right/left shifting operations between A and A'' to find the best possible ratio cut making the size constraint.
5) Output the final result.

If the network contains all small strongly connected groups such that only a few modules are taken out at one time, it takes $O(n)$ iterations for steps 2) and 3) to identify all small groups. This worst case results in the algorithm running in $O(np)$. Fortunately, according to our experience (see Section IV-4.2), steps 2) and 3) are repeated at most five times. This observation matches our assumption that logic designs do contain clustering structure.

IV. Experimental Results

There was a lack of a standard benchmark for comparing partitioning algorithms as noted by [7]. Thus we use three boards of the ILLIAC IV computer [23]: IC67, IC116, IC151, three circuits from Hughes Aircraft Company: bm1, 19ks, 19kstw, and nine benchmark examples from Microelectronics Center of North Carolina (MCNC): PrimGA1, PrimGA2, PrimSC1, PrimSC2, Test02, Test03, Test04, Test05, Test06 (Table I) for testing. Furthermore, we test six circuits from Sechen and Chen [22] and compare the results.

We implemented both the Fiduccia–Mattheyses and the ratio cut algorithms for the comparison of two-way partitioning. The cut capacity is extended to a net cut model for handling multipin nets. All algorithms were written in C and run on a Celerity C1260 dual processor machine. The experiments show that the proposed ratio cut obtains better or comparable partitioning results even in terms of the cut capacity.

4.1. Ratio Cut Without Size Constraints

In order to show that the ratio cut algorithm does identify the natural clusters in the circuit, we first remove the constraints on subset size. We also predefine the subset size in the Fiduccia–Mattheyses algorithm to half of the original problem size, with a tolerance the size of the largest basic module (excluding macroblocks) for observation. Table II lists the results from the Fiduccia–Mattheyses and ratio cut algorithms. Since the sizes of modules are irregular, we let $|A|$ and $|A'|$ denote the total sizes of modules in A and A', respectively. For both algorithms, we make twenty trials for each circuit and report the best result according to the ratio value R.

TABLE I
The Sizes of the Tested Examples

Example	# Nets	# Modules	# iopads
IC67	115	52	15
IC116	329	101	14
IC151	419	136	15
bm1	904	752	130
19ks	3343	2684	160
19kstw	3731	3079	164
PrimGA1	904	752	81
PrimGA2	3029	2907	107
PrimSC1	904	752	81
PrimSC2	3029	2907	107
Test02	1866	1663	70
Test03	1699	1607	65
Test04	1738	1515	36
Test05	2910	2595	63
Test06	1745	1691	69

The average ratio improvement of these 15 tested examples is 40%. The best result, 70% improvement, is derived from example Test02. In this case, the ratio cut identifies the cluster structure of an approximate 1:2 of $|A| : |A'|$ size ratio instead of even sized partition. However, in Test04-06 three MCNC test cases, the ratio cut derives an even sized partitioning. Still, the results are better than the Fiduccia–Mattheyses method. Since we remove the constraint of subset size in the initialization and iterative shifting phases, this freedom lets the ratio cut approach explore larger domains of the solution space and thus derive better results. Test04 is the only case in which the ratio cut approach generates no improvement over the Fiduccia–Mattheyses method. This is a special case because it contains a huge block of the size 19 928, which is almost half of the total circuit size. A partition separating the block from most of the rest modules is quite natural. Therefore, both methods derive similar results.

As stated earlier, our ratio cut algorithm is efficient due to its linear time complexity. The machine took only 3 s to finish the smallest example, IC67, while 200 s were used to the largest example, 19kstw.

We manually analyzed the IC67 example in detail. Two strongly connected groups were discovered, one of 11 modules, and the other of 56 modules, which exactly matched the results for IC67 from the ratio cut algorithm. We display the partitioning results of the Fiduccia–Mattheyses (Fig. 9(a)) and ratio cut (Fig. 9(b)) algorithms, respectively, for circuit IC116. The placement of modules is random within each partitioned group. Since some modules may overlap due to random placement, the

TABLE II
THE COMPARISON OF BEST RESULTS OBTAINED FROM TWO ALGORITHMS

Example	Fiduccia-Mattheyses			Ratio Cut			$\frac{C^f - C^r}{C^f}$ %	$\frac{R^f - R^r}{R^f}$ %
	$\|A\| : \|A'\|$	Net Cut C^f	$R^f \times 10^{-5}$	$\|A\| : \|A'\|$	Net Cut C^r	$R^r \times 10^{-5}$		
IC67	34 : 33	37	3298	56 : 11	14	2272	62	31
IC116	58 : 57	38	1149	64 : 51	29	888	24	23
IC151	76 : 75	49	860	136 : 15	14	686	71	20
bm1	1745 : 1735	70	2.31	2535 : 945	19	0.79	73	66
19ks	5535 : 5467	148	0.49	9596 : 1406	38	0.28	74	42
19kstw	3954 : 3938	111	0.71	6065 : 1827	54	0.49	51	32
PrimGA1	1742 : 1707	67	2.28	2929 : 502	11	0.75	84	67
PrimGA2	4193 : 4180	233	1.33	5886 : 2487	89	0.61	62	54
PrimSC1	1383 : 1370	54	2.85	1682 : 1071	35	1.94	35	32
PrimSC2	3853 : 3853	266	1.79	5374 : 2332	89	0.71	67	60
Test02	28 613 : 28 437	156	0.02	37 146 : 19 904	43	0.006	72	70
Test03	11 166 : 11 063	95	0.08	13 609 : 8620	57	0.05	40	37
Test04	21 083 : 20 956	44	0.01	21 062 : 20 977	44	0.01	0	0
Test05	36 005 : 35 890	72	0.005	37 187 : 34 708	44	0.003	39	39
Test06	8502 : 8466	90	0.13	8557 : 8411	61	0.08	32	32

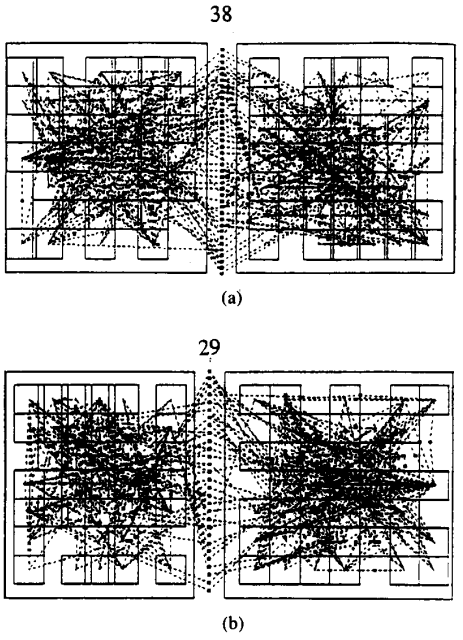

Fig. 9. The 2-way partitioning results of IC116 from different algorithms (a) The Fiduccia-Mattheyses algorithm. (b) The ratio cut algorithm.

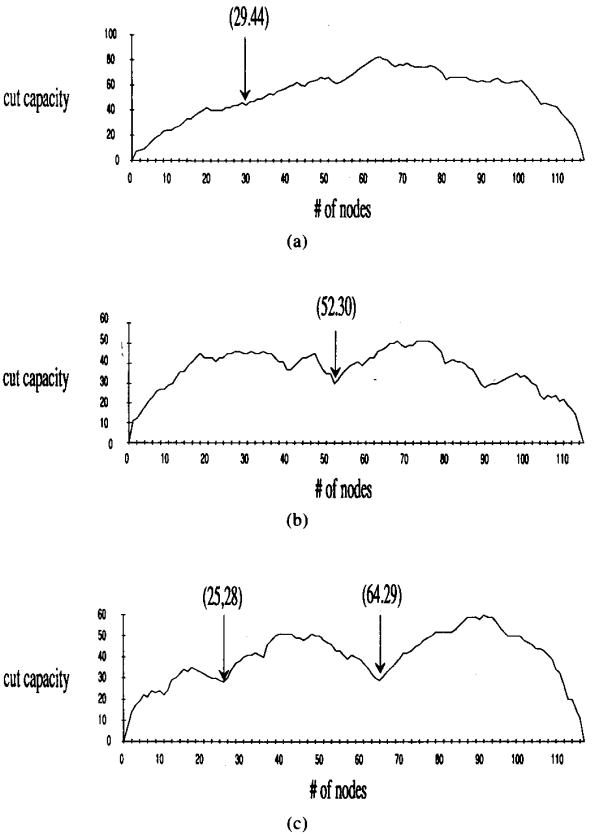

Fig. 10. Cut capacity versus number of modules for IC116 during the ratio cut algorithm. (a) Initialization phase from one direction. (b) Initialization phase from the other direction. (c) Iterative shifting phase.

boundary for each partitioned group shows the exact size of all modules in that group. Each square represents a module of the circuit which, for IC116, is uniform in size. Each small dark box represents a net connecting all the centers of its modules by dotted lines. The Fiduccia-Mattheyses algorithm generates a net cut of 38 while the ratio cut generates 29.

Fig. 10 shows three plots of circuit IC116 where we display the cut capacity against the number of modules in one subset for the ratio cut algorithm. Fig. 10(a) and (b) illustrate the variation of cut capacities in the initialization phase starting from the first seed to the other and vice versa. Fig. 10(c) corresponds to the iterative shifting phase. We can observe some facts from the plots. The cut capacity is equal to zero whenever we put all modules in one subset and none for the other subset; as the number of modules in the subset is increased, so is the cut capacity. We can see the lowest cut capacity occurs at either

TABLE III
THE COMPARISON OF BEST RESULTS WITH 1 : 3 SIZE RATIO FROM TWO ALGORITHMS

Example	Fiduccia-Mattheyses			Ratio Cut			$\frac{C^f - C^r}{C^f}$ %	$\frac{R^f - R^r}{R^f}$ %
	$\|A\| : \|A'\|$	Net Cut C^f	$R^f \times 10^{-5}$	$\|A\| : \|A'\|$	Net Cut C^r	$R^r \times 10^{-5}$		
IC67	50 : 17	24	2823	50 : 17	20	2352	17	17
IC116	64 : 51	29	888	64 : 51	29	888	0	0
IC151	110 : 41	36	798	108 : 43	36	775	0	3
bm1	2234 : 1246	35	1.26	2532 : 948	19	0.79	46	37
19ks	6985 : 4017	169	0.60	7827 : 3175	122	0.49	28	18
19kstw	4978 : 2914	82	0.56	5981 : 1911	55	0.48	33	15
PrimGA1	2209 : 1222	34	1.26	2106 : 1325	34	1.21	0	4
PrimGA2	4567 : 3806	140	0.80	5837 : 2536	77	0.52	45	35
PrimSC1	1729 : 1024	46	2.60	1682 : 1071	35	1.94	24	25
PrimSC2	4999 : 2707	166	1.23	5331 : 2375	77	0.61	54	51
Test02	36 716 : 20 334	66	0.009	37 146 : 19 904	43	0.006	35	34
Test03	15 221 : 7008	84	0.078	14 869 : 7360	49	0.044	42	43
Test04	21 062 : 20 977	44	0.01	21 062 : 20 977	44	0.01	0	0
Test05	37 192 : 34 703	42	0.003	37 192 : 34 703	42	0.003	0	0
Test06	9958 : 7010	66	0.094	8551 : 8417	62	0.086	6	9

end of the curves, and the highest cut capacity occurs somewhere in the middle. In the profile of Fig. 10(a), the algorithm found the cut with ratio 0.0176 occurred at coordinate (29, 44), meaning that it separated 29 modules in one subset with capacity 44; while ratio 0.0092 occurred at (52, 30) in Fig. 10(b). These two ratio values correspond to some local minima of the cut capacity with quite evenly sized subsets. Because the ratio value in Fig. 10(b) is better than Fig. 10(a), we choose the partitioning from Fig. 10(b) as our new starting point for further improvement. Fig. 10(c) shows another local minimum of cut capacity at (64, 29), which generates a ratio value of 0.0089. This is better than the one in Fig. 10(b) after one iterative shifting. In this case, no more iterative shifting is needed and the group swapping will not improve the ratio value further for IC116, Fig. 10(c) is the final result from Fig. 10(b). Our arguments against other algorithms are once more supported in these plots. If we use the max-flow-min-cut algorithm, very unevened subset sizes will be generated because of the lowest cut position at the curves. If we apply the Kernighan–Lin algorithm with a predefined subset size, say one half of the total circuit size, then the cut is in the middle, which is worse than the ratio cut. Even if we give a larger tolerance in the Kernighan–Lin algorithm, the approach may find another cut at (25, 28) in Fig. 10(c). Even though the cut capacity is lowered by one, two resultant subsets are quite uneven in size. Comparing the last cut with the ratio cut, the latter one is still preferred.

4.2. Ratio Cut with Size Constraints

In this subsection, we conduct the experiments by making the same constraint on subset size for both the ratio cut and the Fiduccia–Mattheyses algorithms. We allow the maximum size of subsets to be three quarters of the total circuit. We test all 15 circuits in Table I again. For each case, we still give it twenty trials and report the best result according to our ratio metric in Table III.

Comparing Tables II and III on the results of Fiduccia–Mattheyses algorithm, almost all net cut values are reduced in Table III because the maximum size constraint is released from a strict half to three quarters of the original problem size. For instance, circuit IC67 was divided into 34 and 33 modules with cut capacity 37 in Table II. However, the cut capacity result was reduced to 24 in Table III after removing the strict size constraint. The average improvement of the cut capacities is 28% for the Fiduccia–Mattheyses algorithm. On the other hand, the cut capacities generated by the ratio cut algorithm in Table III is on average 12% higher than the data shown in Table II due to the enforcement of a size constraint.

As shown in Table III, the ratio cut algorithm with size constraints produces better results than the Fiduccia–Mattheyses algorithm for most cases. The average improvement for the fifteen tested examples is 20% in terms of the ratio. The best result is from circuit PrimSC2, which has a 51% ratio improvement. The ratio cut approach can derive better solutions in evenly sized partitioning, such as circuits PrimSC1 and Test06. This is consistent with our previous experiments. Only three cases show no improvements, namely, IC116, Test04, and Test05.

In all tested cases, the ratio cut algorithm finds a comparable partitioning after taking out at most five small subsets. This caused the ratio cut algorithm to spend one to three times more CPU time than the Fiduccia–Mattheyses algorithm.

4.3. A Comparison with the Sechen–Chen Algorithm

A recent paper by Sechen and Chen [22] proposed a new objective function for multipin net model and obtained excellent results for 2-way partitioning. Because the ratio value is not available from [22], we modify the ratio cut algorithm with size constraints in Section III-3.3 so that the algorithm can adjust its final result according to the cut capacity. Instead of the shifting operations in

TABLE IV
THE COMPARISON OF THREE DIFFERENT ALGORITHMS IN SIX CIRCUITS

Circuits	# Nets	# Modules	# Io	Fiduccia-Mattheyses Average	Fiduccia-Mattheyses Best	Sechen-Chen Average	Sechen-Chen Best	Ratio Cut Average	Ratio Cut Best
example	235	183	97	13.53	4	7.67	3	10.50	3
fifo	407	346	0	29.93	19	21.70	16	18.23	15
8870	349	286	23	25.27	18	19.00	11	17.53	13
linear	453	364	0	38.63	24	26.47	14	21.03	14
sd2	494	469	37	30.77	14	17.37	6	16.53	9
5655	843	801	120	98.47	83	63.87	45	57.63	49

step 4), we apply the Fiduccia-Mattheyses algorithm to a condensed network consisting of nodes that are the groups of modules produced during the process, i.e., A, A' in the last partitioning, and all elements in Ψ. Finally, another run of the Fiduccia-Mattheyses algorithm is applied to the original network N with its initial partitioning from the previous condensed network result.

We tested six released circuits from their ten examples. Since they also set the maximum subset size to be three quarters of the whole circuit size, the comparison of the average and best cut capacities after 30 trials is listed in Table IV. Again, our algorithm is far better than the Fiduccia-Mattheyses algorithm, and competitive with the Sechen-Chen method in either the average or the best results even in terms of the cut capacity.

V. CONCLUSION

In this paper, we have proposed a new circuit partitioning method. Using the concept of the **ratio cut**, we can locate the clustering structure buried in very complicated circuits. Ratio cut partitioning is more natural than the minimum cut which requires predefined subset sizes. Therefore, the ratio cut algorithm is quite suitable for hierarchical designs or multiway partitions.

Although finding the ratio cut in general is an *NP*-complete problem, in certain cases, it can be obtained by linear programming techniques using a polynomial number of operations through the analogous multicommodity flow problem.

We also propose a fast algorithm for VLSI circuitry. The implementation of the heuristic ratio cut algorithm takes advantage of the Fiduccia-Mattheyses algorithm, and runs in linear time. Experiments show good results for all tested cases. Table II lists the improvements from 20% up to 70% in terms of the proposed ratio.

Future research includes the considerations of other partitioning constaints, such as the time delay consideration in high-speed circuitry, noise isolation if there are analog circuits in the design, thermal effects, and so on. Placement can also be based on applying the ratio cut algorithm recursively to a large circuit until an expected number of smaller subcircuits are partitioned. This high-level partitioning reduces the size of the placement solution space sufficiently so that an exhaustive search can be used to find further improvements.

ACKNOWLEDGMENT

The authors would like to acknowledge the encouragement given by Prof. E. S. Kuh and Dr. M. Marek-Sadowska.

REFERENCES

[1] B. Bollobas, *Random Graphs*. New York: Academic, 1985, pp. 11-12, 31-53.
[2] R. Camposano and R. K. Brayton, "Partitioning before logic synthesis," in *Proc. Int. Conf. on Computer-Aided Design*, 1987, pp. 324-326.
[3] H. R. Charney and D. L. Plato, "Efficient partitioning of components," *IEEE Design Automation*, pp. 16.0-16.21, July 1968.
[4] C. K. Cheng and T. C. Hu, "Maximum concurrent flow and minimum ratio cut," Tech. Rep. CS88-141, Dec., 1988.
[5] V. Chvatal, *Linear Programming*. San Francisco, CA: Freeman, 1983.
[6] W. M. Dai and E. S. Kuh, "Simultaneous floor planning and global routing for hierarchical building-block layout," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 828-837, 1987.
[7] W. E. Donath, "Logic partitioning," in *Physical Design Automation of VLSI Systems*, B. Preas and M. Lorenzetti, ed. Menlo Park, CA: Benjamin/Cummings, 1988, pp. 65-86.
[8] C. M. Fiduccia and R. M. Mattheyses, "A linear time heuristic for improving network partitions," in *Proc. 19th Design Automation Conf.*, 1982, pp. 175-181.
[9] L. R. Ford and D. R. Fulkerson, *Flows in Networks*. Princeton, NJ: Princeton University, 1962.
[10] J. Frankle and R. M. Karp, "Circuit placement and cost bounds by eigenvector decomposition," in *Proc. Int. Conf. on Computer-Aided Design*, 1986, pp. 414-417.
[11] J. Hennessy, "Partitioning programmable logic arrays summary," in *Proc. Int. Conf. on Computer-Aided Design*, 1983, pp. 180-181.
[12] M. Iri, "On an extension of the maximum flow minimum cut theorem to multicommodity flows," *J. Oper. Res. Soc. Japan*, vol. 5, no. 4, pp. 697-703, Dec. 1967.
[13] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, vol. 49, no. 2, pp. 291-307, Feb. 1970.
[14] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Sci.*, vol. 220, pp. 671-680, 1983.
[15] B. Krishnamurthy, "An improved min-cut algorithm for partitioning VLSI networks," *IEEE Trans. Comput.*, vol. C-33, pp. 438-446, May 1984.
[16] D. W. Matula and F. Shahrokhi, "The maximum concurrent flow problem and sparsest cuts," Tech. Rep., Southern Methodist Univ., 1986.
[17] M. C. McFarland, "Computer-aided partitioning of behavioral hardware description," in *Proc. 20th Design Automation Conf.*, 1983, pp. 474-478.
[18] K. Onaga and O. Kakusho, "On feasibility conditions of multicommodity flows in networks," *IEEE Trans. Circuit Theory*, vol. CT-18, pp. 425-429, 1971.
[19] L. A. Sanchis, "Multi-way network partitioning," *IEEE Trans. Comput.*, vol. 38, pp. 62-81, Jan. 1989.
[20] D. M. Schuler and E. G. Ulrich, "Clustering and linear placement," in *Proc. 9th Design Automation Workshop*, 1972, pp. 50-56.

[21] D. G. Schweikert and B. W. Kernighan, "A proper model for the partitioning of electrical circuits," in *Proc. 9th Design Automation Workshop*, 1972, pp. 57-62.
[22] C. Sechen and D. Chen, "An improved objective function for mincut circuit partitioning," in *Proc. Int. Conf. on Computer-Aided Design*, 1988, pp. 502-505.
[23] J. E. Stevens, "Fast heuristic techniques for placing and wiring printed circuit boards," Ph.D. dissertation, Comp. Sci. Dep. Univ. Illinois, 1972.

Yen-Chuen Wei (S'89) received the B.S. degree in electrical engineering from National Taiwan University, Taiwan, in 1982, and the M.S. and Ph.D. degrees in computer science from University of California, San Diego, in 1987 and 1990, respectively.

He is currently a senior member of technical staff at Cadence Design Systems, Inc., where he works on the design and development of VLSI CAD systems. His research interests include hierarchical partitioning, floorplanning, and placement and routing algorithms for automatic VLSI design tools.

Dr. Wei is a member of the ACM Special Interest Group in Design Automation.

Chung-Kuan Cheng (S'82-M'84) received the B.S. and M.S. degrees in electrical engineering from National Taiwan University, and the Ph.D. degree in electrical engineering and computer sciences from University of California, Berkeley, in 1984.

From 1984 to 1986 he was a senior CAD engineer at Advanced Micro Devices Inc. Since 1986 he has been an Assistant Professor at Computer Science and Engineering Department, University of California, San Diego. His research interests include design automation on microelectronic circuits, network flow optimization, circuit partitioning, and physical design of multichip modules for hybrid circuit packaging.