A Global Router with a Theoretical Bound on the Optimal Solution

Robert C. Carden IV, *Member, IEEE*, Jianmin Li, *Member, IEEE*, and Chung-Kuan Cheng, *Senior Member, IEEE*

Abstract— **The global routing problem is formulated as a multiterminal, multicommodity flow problem with integer flows. An ϵ-optimal 2-terminal multicommodity flow algorithm with fractional flows is extended to handle multiterminal commodities. Our adaptation of this network flow algorithm seeks to maximize overall routability by minimizing edge congestion as opposed to conventional techniques which usually seek to minimize wire length. We show that under certain conditions, our approach derives an approximate optimal solution. We apply a randomized rounding procedure to derive an integer solution from the fractional multicommodity flow solution. Experimental results demonstrate that this network flow algorithm can be realistically used to route industrial sized circuits with reduced congestion.**

I. Introduction

WE INVESTIGATE the problem of routing given a placement and a list of pins to connect. The routing problem is traditionally broken down into two subproblems: global routing and detailed routing. Global routing gives the general paths for the signal nets; detailed routing assigns actual tracks and vias to the routes. Here we concern ourselves with the global routing problem.

Ting and Tien [1] employ an iterative improvement algorithm for global routing. Their algorithm starts by routing each net independently, allowing each net to find its natural interconnect pattern. It then proceeds to improve the solution by rerouting nets so that they avoid overflow boundaries. The authors of this algorithm make several comments. First, they acknowledge the problem of net ordering in the initialization step. Second, they indicate how to update overflow boundaries dynamically when rerouting each net.

Subsequently, much effort [2]–[8] has been expended on developing a linear programming or multicommodity flow formulation of the global routing problem. Hu and Shing [2] extend Ting and Tien's iterative improvement algorithm by formulating the global routing problem as a hierarchical linear program. Each column in the linear program represents a possible route for some particular net, while each row represents the utilization of channels. As a result, each column is a 0–1 vector specifying which channels to use for connecting some net. To solve the problem practically, they use linear programming to arrive at an initial solution, and then proceed to deduce an integer solution by selecting, for each net, the column with the highest fractional value. Because it is impractical to generate all the possible columns for each net in advance, Hu and Shing use column generating techniques [9] to generate potential candidates.

Manuscript received August 1, 1991; revised November 14, 1995. This work was supported in part under grants from NSF project number MIP-9009260 and MICRO. This paper was recommended by Associate Editor M. Marek-Sadowska.

R. C. Carden IV is with the Unisys Corporation, Mission Viejo, CA 92691 USA.

J. Li and C.-K. Cheng are with the Department of Computer Science and Engineering, University of California, San Diego, La Jolla, CA 92093 USA.

Publisher Item Identifier S 0278-0070(96)01841-6.

Global routers have been constructed using both multicommodity and single-commodity flow models. Shragowitz and Keel [7] extend Ting and Tien's global routing algorithm by formulating it as a multicommodity flow problem. They generate their initial global routing by solving a multicommodity flow problem with the capacity constraints removed. By dynamically adjusting edge weights with a cost function which heavily penalizes congested channels, they proceed to improve their initial solution by solving multicommodity flow problems which are in turn reduced to a series of shortest path problems. Meixner and Lauther [10] observe that multicommodity flow problem formulations cannot guarantee integer solutions because flows may be split into portions. Instead, they use a single-commodity flow formulation to improve an existing global routing solution by improving a group of adjoining nets all at once. Meixner and Lauther observe that an ordering problem still exists, though they claim that the problem of ordering graph nodes to process is far less severe than the net ordering problem.

Vannelli [3], [4] applies Karmarkar's linear programming algorithm [11] to Hu and Shing's linear programming formulation. Whereas Hu and Shing use column generating techniques to reduce the complexity of the problem, Vannelli reduces it by selecting only minimal rectilinear Steiner trees or near minimal rectilinear Steiner trees for his matrix. This approach limits the variety of routes for each net that may be used by the algorithm. The resulting program runs much faster than an equivalent simplex implementation.

Karp *et al.* [5] give an algorithm for global routing which uses linear programming followed by randomized rounding, a technique discussed in detail by Raghavan [12]. They are able to prove that given an integer program and a linear programming solution which is optimal, one can deduce an integer solution within a computable error bound. Raghavan and Thompson [8] also give a global router which is deterministic for 3-pin nets. They incorporate a randomized rounding algorithm derived from their previous work. Raghavan and Thompson's algorithm can be generalized to work on larger nets, but the total number of Steiner points grows exponentially as do the number of variables in the integer program. Because in practice many nets in a global routing problem can contain 50, 100, or even 200 pins, we see that this algorithm, as stated, is unsuitable for large practical problems. This follows from the observation that a single 20 pin net can require trillions of variables to be included in the integer program. Therefore, a heuristic algorithm is needed to reduce the complexity.

While Hu and Shing, as well as Vannelli, attempt to solve the global routing problem using combinatorial techniques, there are many papers [6], [13], [14] proposing fast hierarchical algorithms. By using a slicing structure, they decompose the problem so that its base case is either a simple 2×1 or 2×2 case. The slicing continues until the grid equals the basic unit size.

Lee and Sechen [15] have implemented a global router specifically designed for sea-of-gates circuits. They produce their routing by successively refining it, starting first by minimizing interconnect length, then proceeding to even out congestion, and conclude by applying a maze routing procedure which removes overflows and reduces congestion. They handle ordering problems by randomly selecting the next object to process.

Recently, Chiang *et al.* [16] have implemented a global routing algorithm which proceeds as follows. It first assigns a unique order number to each net such that nets with lower order numbers will be routed first. Then, it proceeds net by net, computing a min/max minimal rectilinear Steiner tree [17]–[19] for that net on a weighted graph. These minimal rectilinear Steiner trees attempt to minimize the maximum edge. The weight of an edge is a function of capacity and usage such that crowded edges have high weights. The resulting algorithm, which is essentially a greedy one pass algorithm, is very fast with good results against benchmark data. Its min/max minimal rectilinear Steiner tree algorithm is demonstrated to be effective in certain cases.

This paper describes a global router based on a multiterminal, multicommodity flow algorithm [20]. We utilize Shahrokhi and Matula's algorithm [21] to derive the fractional flow solution in a relatively short execution period. We extend this method to handle multiterminal Steiner trees instead of just shortest paths. Based on Shahrokhi and Matula's algorithm, we exhibit, at any stage, the error bound of the current result from an optimal solution of the linear programming formulation. We then use a randomized rounding technique to derive a discrete net connection with an error bound on the derivation from the optimal fractional solution. We finish by employing an iterative procedure to improve the final results. In the iteration, the cost of each edge is an exponential function of the congestion. Thus, on these cases, our procedures to handle net ordering and cost assignment on edges answer the problems proposed by Ting and Tien. The experimental results with benchmark data show significant improvements over previous work.

II. Problem Formulation

Given a layout problem, we want to generate the global routing that minimizes the maximum density. We can formulate the problem as a linear programming problem. Section 2.1 gives definitions and notation which are needed for the problem formulation in Section 2.2. In Section III, we present an approximate multicommodity multiterminal flow algorithm to solve this programming problem.

2.1 Definitions

Let $G = \langle V, E\rangle$ be a connected graph which is to model the connectivity of a global routing region. Each edge $e \in E$ has a capacity $c[e]$ associated with it. The capacity specifies the number of nets which may be routed through that edge. Let N be the set of nets to complete. Assume that these nets are numbered consecutively, i.e., 1 through r. Each net $n \in N$ contains a set of vertices $V_n \subseteq V$ *s.t.* $|V_n| \geq 2$. Subsequently, we will use n and e to represent either the elements or the indices of the nets and edges if no confusion arises.

Let $d(e)$ be a distance function giving the distance of edge e. The distance we define here should not be confused with Euclidean or Manhattan distance metrics that are normally used. If t is a tree, $d(t)$ denotes the distance of the tree, i.e., $d(t) = \Sigma_{e \in t}\, d(e)$. We also define $d(n)$ to be the length of the *shortest* tree that connects net n. The *demand* of a net specifies how many connections must be made for that net. Thus, each net n has a demand $b[n]$ associated with it. In general, this value is set to one. However, in problems where a net n represents a *super-net* containing s identical nets, $b[n]$ is set to be s, thus specifying that s distinct connections are required.

By assuming that G is a finite graph with finite sets V and E, we observe that any tree connecting a net is simply a subset of E. Thus, because the power set of E represents all possible trees within G, we can assume that we can enumerate and index all possible trees for any given net. We can henceforth define a matrix A to represent all possible paths for each net.

Let A be a 0–1 matrix specifying all possible trees for all nets within the graph. Each tree t is represented by a 0–1 column vector of dimension $|E|$, i.e.,

$$t = (a_1 \quad a_2 \quad \cdots \quad a_{|E|})^T.$$

If tree t uses edge e, then the corresponding row is one, otherwise it is zero. We assume that all possible enumerations for the first net are listed from left to right, followed by all possible enumerations for the second net, and so on. As a result, one may view A as a vector of columns specifying all the possible trees for net 1, net 2, and so on. Furthermore, given A, let A_n denote a submatrix of trees for net n. The number of columns within this submatrix is the total number of possible trees for net n.

Let $p_1, p_2, \cdots, p_r$ represent the number of trees for nets $1, 2, \cdots, r$, respectively. Let t_{nj} denote the jth tree of net n. Then, matrix A takes on the following form

$$A = (A_1 \quad A_2 \quad \cdots \quad A_r)$$

where

$$A_n = (t_{n1} \quad t_{n2} \quad \cdots \quad t_{np_n}).$$

Note that although we list all possible trees in the formulation, in calculation we only need a few throughout the process. We adopt a *column generating technique* [2] and [9] which only requires the generation of the *best* tree according to the *current distance function.*

Given a flow configuration, let f denote a function which returns the flow associated with its argument. Consequently, $f[t_{nj}]$ denotes the flow associated with tree t_{nj}; $f[n]$ denotes the total flow associated with this net, i.e., $f[n] = \Sigma_{j=1}^{p_n} f[t_{nj}]$. In addition, given an edge $e, f[e]$ denotes the total flow associated with this edge. By defining δ to be

$$\delta(t_{nj}, e) = \begin{cases} 1, & \text{if tree } t_{nj} \text{ uses edge } e; \\ 0, & \text{otherwise} \end{cases}$$

we observe $f[e] = \Sigma_{n=1}^{r} \Sigma_{j=1}^{p_n} f[t_{nj}]\delta(t_{nj}, e)$.

2.2 Linear Programming Formulation

Let X denote a vector with as many rows as there are columns in A with its element $x_{nj} \equiv f[t_{nj}]$. The resulting problem is how to maximize a throughput $\overline{z}$ subject to a set of constraints.

The following linear program specifies the global routing problem as a multicommodity multiterminal flow problem.

$$\begin{aligned} &\text{Minimize } g^* \text{ subject to} \\ &\qquad \forall n \in N, f[n] - b[n] = 0, \\ &\qquad \forall e \in E, f[e] - g^* c[e] \le 0, \\ &\qquad \forall n, j, x_{nj} \ge 0. \end{aligned}$$

Note that the above constraints can be expressed with variables g^* and X, i.e.,

$$\forall n, \sum_{j=1}^{p_n} x_{nj} = b[n]$$

$$\forall e \in E, \sum_{n=1}^{r} \sum_{j=1}^{p_n} x_{nj}\delta(t_{nj}, e) - g^* c[e] \le 0$$

$$\forall n, j, x_{nj} \ge 0.$$

The first constraint of the linear program specifies that the sum of all the flows must be equal to the demand for that net. Let g^* denote the bottleneck of the graph. Thus, $g^* = \max_{e \in E} (f[e]/c[e])$. We can see that $\overline{z} = 1/g^*$. Therefore, we can see that the second constraint specifies that the throughput $\overline{z}$ must be set so that no edges are overflowed. An equivalent matrix formulation of this constraint is $A \times (\overline{z}X) \le C$, where C is a vector of $c[e]$. In global routing, we require that the flows be integer valued so that: $\forall i, j, x_{ij} \in \{0, 1\}$. However, this transforms the problem into an integer program which is known to be NP-complete.

If the solution $\overline{z} \ge 1$, we have a fractional routing solution which is feasible under the current edge capacity constraint. Suppose $\overline{z} > 0$, then the maximum density of the fractional routing solution is reduced to $\max_{e \in E} c[e]/\overline{z}$. Therefore, we use the objective function of maximizing $\overline{z}$ to generate a minimal density solution.

III. Our Approach

An ϵ-optimal solution is one which differs from the *optimal* solution with a relative error of at most ϵ. Shahrokhi and Matula [21] propose an ϵ-optimal algorithm to solve the 2-terminal, multicommodity flow problem. This algorithm, if directly applied to global routing, requires 2-pin nets and fractional flow assignments. We present an algorithm which allows for n-pin nets and ultimately integer valued flows. This algorithm effectively generates a solution to the integer program given in Section 2.2 with an error bound. The six steps, R1 through R6, are described in the following sections.

3.1 Basic Algorithm

Steps R1 through R4 derive a fractional flow solution. Under certain assumptions (Section 5.2), we can claim an ϵ-optimal solution. Step R5 then integerizes this solution, identifying any overflow edges. A stochastic approach is adopted to obtain a probalistic error bound. Step R6 reroutes the nets to improve the resulting solution until either there are no more overflow edges or until it is impossible to improve the solution any further. Since R6 only improves the solution, we can claim a theoretical bound on the optimal solution when certain assumptions (Section 5.2) are met in R1–R5.

3.2 Fractional Flow Algorithm

The following pseudocode summarizes the flow of control of the fractional flow algorithm. The first step achieves an initial feasible solution. The second step updates edge weights and recomputes Steiner trees, and the third step determines whether the solution is ϵ-optimal. If so, it then breaks from the loop and proceeds on to the integerization Step R5. If not, the fourth step selects a net to reroute and the amount of flow to be rerouted.

Algorithm 3.1

```
R1—Initialize and compute initial Steiner trees
while not $\epsilon$-optimal do
    R2—compute edge distances and new Steiner trees
    R3—check if $\epsilon$-optimal
    R4—select and reroute net
done
```

3.2.1 R1—Initialization

The goal of this step is to initialize system wide parameters and to obtain an initial feasible solution. The solution is feasible in that all demands of each net shall be met. However, upon completing this step, the solution may not be *usable* because the *throughput* may be less than one.

Initialize System Wide Parameters: Given a graph $G = \langle V, E \rangle$ and netlist N, initialize all system wide parameters. Define $C_\Sigma \equiv \Sigma_{e \in E} c[e]$ and $C_* = \min_{e \in E} c[e]$. Define $b_\Sigma \equiv \Sigma_{n \in N} b[n]$. Define σ_0 in terms of a user defined constant ϵ, i.e., $\sigma_0 \equiv c_*^2 \epsilon / 16 \alpha c_\Sigma$. σ_0 will be used as the limit on the smallest fraction of flow to avoid iteration on the trees with tiny flow. The user input constant $\epsilon \in (0, 1]$ specifies the amount of error that the user can tolerate. The constant α, which is typically set between 0.01 and 100, is a user tunable parameter. Shahrokhi and Matula specify that α should be set to $2c_\Sigma^2 / c_* b_\Sigma \epsilon$. However, this is generally impractical on fixed precision computers.

Initialize Edge Distances: Establish initial distances $d_{R1}[e] = 1/c[e]$, for each edge $e \in E$. In this step, wider edges are assigned a lower distance.

Compute Initial Steiner Trees: For each net $n \in N$, compute the Steiner tree and make it active with its flow set to be $b[n]$. This Steiner tree algorithm is described in Section 4.1. All edges should be marked as to ensure that distances are calculated in Step R2.

Special Implementation Concerns: It may be desirable to update edge distances after routing each net. The distance function will be that of R2. However, in doing this, the order in which one routes nets becomes important. One heuristic we use is to route the net with smallest area bounding box first, and then to proceed onto the larger nets. This approach helps achieve an initial feasible solution with higher throughput than if edge distances are not updated. However, the wire length may become quite large.

3.2.2 R2—Edge Distances and Steiner Trees

Compute New Edge Distances: For each marked edge, i.e., any edge e whose utilization $f(e)$ has changed during Step R1 or R4, compute the new distance $d_{\mathrm{R2}}[e]$:

$$d_{\mathrm{R2}}[e] = \exp\left(\alpha(f[e]/c[e])\right).$$

For each active tree and for each Steiner tree affected by these edge changes, update their respective lengths.

Improve Steiner Trees: Using the distances computed by d_{R2}, recompute Steiner trees for each net. The Steiner tree algorithm used here is described in Section 4.2.

3.2.3 R3—Termination Criteria

Compute Error Bound: Let $d(e)$ and $d(n)$ be defined with respect to the current distance function d_{R2}. Define $g^* \equiv \max_{e \in E}(f[e]/c[e])$. Compute throughput $\overline{z} = 1/g^*$. Define upper bound $z_d \equiv \Sigma_{e \in E}\ d(e)c[e]/\Sigma_{n \in N}\ d(n)b[n]$. Define error bound $\Delta \equiv z_d - \overline{z}/z_d$.

Check if ϵ-Optimal: If $\Delta \le \epsilon$, the multicommodity solution is denoted as ϵ-optimal. For each $t \in T$, compute the final flow configuration function $\overline{f}(t) = f(t)/g^*$. Goto R5 to derive an integer solution.

Justification of Termination Criteria: In theory, z_d serves as an upper bound on the amount of flow that may be shipped. Similarly, $\overline{z}$ is the current value which we can achieve. Let $\hat{z}$ denote the optimal flow value. A later theorem will assert that $\overline{z} \le \hat{z} \le z_d$. Consequently, as $\Delta \to 0, \overline{z} \to \hat{z}$.

3.2.4 R4—Net Selection and Rerouting

Select Net To Reroute: Let W_n define the set of active trees for net n:

$$W_n \equiv \{t \in A_n | f(t) > 0\}$$

We first select a net to be rerouted. That is, for each net $n \in N$, let n^* denote the longest active tree and let n_* denote the Steiner tree for that net. Determine a net with trees t^* and t_* such that

$$d(t^*) - d(t_*) = \max\left(d(n^*) - d(n_*) | n \in N\right)$$

The net that maximizes the above quantity is to be rerouted.

Determine Flow Change: Compute the net change that results from shifting flow from d^* to d_*. Define $d_* \equiv \Sigma_{e \in t_* - t^*}\ d(e)$. Define $d^* \equiv \Sigma_{e \in t^* - t_*}\ d(e)$. Compute $\sigma = C_*/2\alpha \ln d^*/d_*$. For problems with constant edge capacities, this σ gives the optimal amount of flow to shift so that the throughput improves.

Partial Rerouting: If $f(t^*) \ge \sigma + \sigma_0$, then reroute σ units of flow from t^* to t_* and set $W_n = W_n \cup \{t_*\}$.

Total Rerouting: Otherwise, reroute all the flow on t^* to t_*, and set $W_n = W_n \cup \{t_*\} - \{t^*\}$.

Mark Changed Entities: Update flow values for t^* and t_*. Mark all edges whose flow has changed so as to ensure that their weights are calculated in Step R2. Goto R2.

3.3 Multicommodity Flow Integerization

The following pseudocode summarizes the flow of control of the multicmmodity flow integerization Steps R5 and R6. The first step integerizes the fractional flow solution. The last step impoves the solution until either no further improvement is possible, or the solution is valid.

```
Algorithm 3.2
R5—solution integerization
while not feasible
        R6—improve integerized solution
done
```

3.3.1 R5—Solution Integerization and Termination Criteria

Integerize Nets: Let us assume, for now, that all nets have a demand of one. For each net $n \in N$, select an active tree $t \in W_n$ to make permanent. One approach is some type of randomized rounding, using the current flow of each active tree as a probability. For each net n, we select $b[n]$ trees according to the distribution of the probability. Note that if $b[n] > 1$, some trees may be identical.

Identify Overflow Nets: When finished integerizing, identify overflow edges $e \in E$ such that $f(e) > c[e]$ and define $\xi \equiv \{n | n \in N \text{ and } \exists e \in t_n \text{ s.t. } f(e) > c(e)\}$, where t_n is the set of edges comprising some active tree of net n. Here, ξ denotes the set of overflow nets, i.e., nets which utilize overflow edges.

Check if Solution Is Feasible: If $\xi = \emptyset$, we are finished: report success. If $\xi \ne \emptyset$, goto R6 to improve the integer solution.

3.3.2 R6—Overflow Set Improvement

The iterative improvement scheme here is similar to [1] except that the edge distance is an exponential function of the congestion $f(e)/c[e]$.

Determine Net Ordering: Sort ξ so that the longest nets are first. For each net $n \in \xi$, we will reroute each overflow tree to avoid overflow edges. We do this by applying the following procedures to each overflow tree t of net n.

Reroute Overflow Trees: Define $\hat{f}[e] = f(e) - \delta(t, e)$. Thus, $\hat{f}[e]$ is the net flow after deleting the flow of tree t.

Compute New Edge Distances: This step will reroute exactly one unit of flow from the current tree to a better tree. To make this work, $\hat{f}$ adjusts the flow so that if tree t uses that edge, a new tree may use that edge without additional penalty. This simulates the computation of d^* and d_* in R4. Compute new edge distances $d_{\mathrm{R5}}[e, t]$:

$$d_{\mathrm{R5}}[e, t] = \exp\left(\alpha(\hat{f}[e]/c[e])\right).$$

Compute New Steiner Tree: Recompute the Steiner tree for this net using these new edge distances. The Steiner tree algorithm used here is described in Section 4.2. Decrement the flow of the old tree by one unit, deactivating it if its flow is reduced to zero. Make the Steiner tree active with a flow of one.

Check if Solution Is Feasible: Once we have rerouted all of the overflow nets, recompute ξ. If $\xi = \emptyset$, we are done (success). Otherwise, if there is no improvement, we are done (failure). Otherwise, repeat Step R6.

Improvement Scheme Summary: The following pseudocode summarizes the algorithm used by Step R6 to improve an integerized solution.

```
Algorithm 3.3
sort $\xi$ so that longest nets are first
for each net $n \in \xi$ do
    for each overflow tree $t$ of $n$ do
        compute new edge distances
        compute new steiner tree
        reroute one unit of flow
    done
recompute and repeat step if $\xi \neq \emptyset$
```

3.4 Comments

The cost function $\exp(\alpha(f[e]/c[e]))$ penalizes nets which use congested edges, thus encouraging nets to avoid the congested edges. In particular, our Steiner tree improvement procedure exploits this property by rerouting the path which utilizes the most expensive edge. The termination criteria is extended from the Japanese Theorem [22]. Once $\overline{z} = z_d$, then $\overline{z}$ is the optimum throughput. In R4, σ is the optimal flow with which to augment t_* and to decrement t^*. Thus, on each iteration, given a net to reroute, we are able to compute the optimal amount of flow to reroute so as to maximize the resulting increase of throughput. The constant σ_0 is used to ensure that the remaining flow is not too small, as the rerouting procedure in R4 guarantees that the flow of any tree is no smaller than σ_0.

IV. Steiner Tree Procedures

We adopt Prim's approach to search for Steiner trees. There are two procedures for creating and maintaining our Steiner trees. The first procedure creates a Steiner tree from scratch, while the second procedure improves an existing Steiner tree. The latter procedure takes advantage of the characteristic that the most congested edges in the graph are exponentially more expensive than the less congested edges. Section 4.1 describes how we build Steiner trees from scratch. Section 4.2 describes how we improve an existing Steiner tree. Our approach is similar to Pulleyblank's approach [23] and can be improved by applying some of his new techniques.

Note that the distance of an edge is a function of edge congestion, which may not correspond to a physical distance between the two incident nodes. Therefore, a Steiner tree algorithm on a rectilinear grid may not apply to our application even if $G\langle V, E\rangle$ is a gridded graph.

4.1 Creating a Steiner Tree from Scratch

Given a net $n \in N$, select a vertex $v \in V_n$ to serve as a source, and let $S = \{v\}$. Let $T = \{v \in V_n | v \notin S\}$. Use a variant of Dijkstra's shortest path algorithm to find the shortest path between sets S and T. Let v' denote the resulting vertex in T, and let S' denote the set of intermediate vertices used in the path between S and T. Set $S = S \cup S' \cup \{v'\}$, and $T = T - \{v'\}$. If $T \neq \emptyset$, repeat path finding procedure; otherwise, we are done.

4.2 Improving a Steiner Tree

Given an existing Steiner tree, find the most expensive path in the tree and delete all degree two edges within that path. Consequently, all temporary intermediate vertices should be removed. Let S and T be the sets of vertices from the two respective subtrees. These sets contain all terminals, Steiner and intermediate points that form the respective Steiner subtrees. Use a variant of Dijkstra's shortest path algorithm to find the shortest path between sets S and T. Reconnect the two trees along this new path, and merge sets S and T to contain any new Steiner points or internal vertices as well as the union of S and T.

This procedure for improving a Steiner tree is particularly appropriate for this algorithm. This is because the distance function used for establishing distances of edges heavily penalizes paths which use congested edges. Thus, by rerouting the path using the worst edge, we work on achieving the desired goal of minimizing overall congestion.

V. Special Cases and Properties of Algorithm

5.1 Two Pin Special Case

Let us assume that the number of pins in a net is restricted to two. Also, assume that fractional flows are allowed, that $\alpha \equiv 2c_\Sigma^2/c_* b_\Sigma \epsilon$, and that the edge capacities are equal to a given constant. Also, note that

$$\sigma_0 \equiv \frac{c_*^3 \epsilon^2 b_\Sigma}{32 c_\Sigma^3}.$$

Given these conditions, Shahrokhi and Matula [21] present a provably ϵ-optimal algorithm for this special case. Their algorithm simply uses Steps R1 through R4. The following theorems are proven for this special case by Shahrokhi and Matula [21].

Theorem 5.1: At Any Stage in the Algorithm: $\overline{z} \le \hat{z} \le z_d$.

Theorem 5.2: Given a User Defined Error Bound ϵ, *Let:* $\Delta \equiv z_d - \overline{z}/z_d$. Then the algorithm terminates in polynomial time with $\Delta \le \epsilon$.

With these conditions, we can obtain a solution which deviates from the optimum by ϵ in polynomial time.

5.2 N-Pin Multicommodity Flow with Optimal Steiner Tree and Fractional Flows

Instead of 2-pin nets, let us assume that we have n-pin nets. Also, assume that fractional flows are allowed, that $\alpha \equiv 2c_\Sigma^2/c_* b_\Sigma \epsilon$, and that the edge capacities are equal to a given constant. Finally, altbough it is NP-complete, let us assume that it is possible to compute a minimum Steiner tree. We concern ourselves with Steps R1 through R4.

Lemma 5.3: At any stage in the algorithm, the throughput $\overline{z}$ of any multicommodity multiterminal flow function f and the distance function d on G with $\Sigma_{n \in N} d(n)b[n] > 0$ satisfies

$$\overline{z} \leq z_d = \frac{\sum_{e \in E} d(e)c[e]}{\sum_{n \in N} d(n)b[n]}.$$

Remark 5.4: Let $\hat{z}$ denote the optimal throughput. Because lemma 5.3 is independent of the flow configuration, at any stage in the algorithm, $\overline{z} \leq \hat{z} \leq z_d$.

Theorem 5.5: Algorithm R is a fully polynomial time approximation scheme and terminates with an ϵ-optimal distance function.

With these conditions, we can obtain a solution which deviates from the optimum by ϵ in polynomial time.

5.3 Integerization Properties

We use a probabilistic approach to integerize the flow configuration. Let us assume that $b[n] = 1$ for all nets $n \in N$. For each net, the probability to select one of its active trees is equal to its flow. We apply the same general techniques for integerizing our fractional flow algorithm results as Raghavan and Thompson [12] use for integerizing their linear program results. As a result, we exhibit the same computable error bound as [12].

VI. Implementation and Experimental Results

The multicommodity flow routing algorithm presented earlier has been implemented in the C language on a Sun 4 Sparcstation under Berkeley UNIX. In order to apply our global router to large designs such as Primary1 and Primary2, we made a number of optimizations. First of all, we added an option in our router to update edge weights after routing each net in Step R1. Because this introduces dependency relationships, we route the net with the smallest bounding box first. We find that this improves our initial solution considerably. Second, on each iteration of R2–R4, we often choose to reroute more than one net at a time; we call this block pivoting. Potentially, half or more of the nets will be rerouted in a single step. We find that this helps speed convergence. Additionally, we added an option in our router to perform R5–R6 immediately after R1 to see if a feasible solution can be achieved without going through the pains of R2–R4; if not, we have an additional option where users can limit the total number of iterations to some number such as 100. The performance given represents times on a Sun Sparcstation/2.

Our test cases include difficult examples from [18], Primary1 and Primary2 from MCNC, and a multichip module design from MCC [25]. The difficult example test cases from [18] contain only two pin nets. A shortest path search can find the optimal Steiner tree solution for the two pin nets. Therefore, we can derive the ϵ-optimal solution from R2–R4. The multichip module design contains 7118 nets. Since there are many nets connecting the same set of nodes, the grouping of identical nets into super-nets reduces the number of nets to 319. It takes 908.9 s to derive an ϵ-optimal solution from R1–R4 with $\epsilon = 0.001$. Note that the optimality of this result is under the assumption that the net connection found relative to the current distance function is the best possible tree. Since our Steiner tree algorithm is a heuristic, an optimal Steiner tree search with an exponential complexity may further improve the results. Primary1 and Primary2 contain 915 and 3050 nets, respectively. The size of these two cases force the program to terminate after 100 iterations before the solutions converge in R2–R4. By comparing CPU times on the difficult examples and MCNC, we see that our approach takes about 3 times longer than the approach in [16].

TABLE I
Difficult Example Results

Name	Nets	Pins per Net	Bounding Length	Edge Capacity	ϵ	α	Total Length		Time (sec.)
							[16]	[CC]	
Diff–4/1	8	2	32	2	0.001	0.1	35	32	0.0
Diff–4/2	16	2	64	4	0.001	0.1	–	64	0.1
Diff–4/4	32	2	128	8	0.001	0.1	–	128	0.1
Diff–4/8	64	2	256	16	0.001	0.1	–	256	0.1
Diff–8	32	2	256	4	0.001	0.1	296	256	4.9
Diff–16	128	2	2048	9	0.001	0.1	2214	2048	2.8
Diff–32	512	2	16384	20	0.001	0.1	–	16384	90.8

6.1 Difficult Example Results

We have tested our algorithm against the examples provided in [18] with results listed in [16], i.e., particular instances of Difficult-4, Difficult-8, and Difficult-16. On these examples, we achieve the same densities. These results, therefore, are evaluated in terms of wire length. A result is optimal if its wire length equals the bounding length for the problem. In all three cases, i.e., Difficult-4, Difficult-8, and Difficult-16, we achieved results superior to [16], [18]. In particular, our results for Difficult-4, Difficult-8, and Difficult-16 are optimal. We also created an analogous instance of Difficult-32 to test. On that example we also achieved an optimal solution. The first five columns of Table I give the name, number of nets, number of pins, the bounding length, and the edge capacities of each difficult example. The next two columns give the values of ϵ and the initial α that we use. The next two columns give the total wire lengths achieved by [16] and our algorithm. The last column of Table I gives the total execution time of our program. Fig. 1 shows the resulting routing of Difficult-4, one which uses only 32 units of wire length.

We also tested our program on special instances of Difficult-4, i.e., ones which specified super-nets. In particular, we tested examples where each net is specified two, four, and eight times. We shall refer to these as Difficult-4/2, Difficult-4/4, and Difficult-4/8, respectively. The program identified that a particular vertex list of a particular net had already been specified, so it made the corresponding net a super-net. In all cases, the program achieved an optimal solution in terms of wire length. In each case, the initial fractional solution split the flows between two shortest nonoverlapping paths. Because the fractional solution is also a valid integer solution with one, two, and four units of flow assigned to each route, respectively, no changes except for roundoff error are made. Table I also shows the results for these examples.

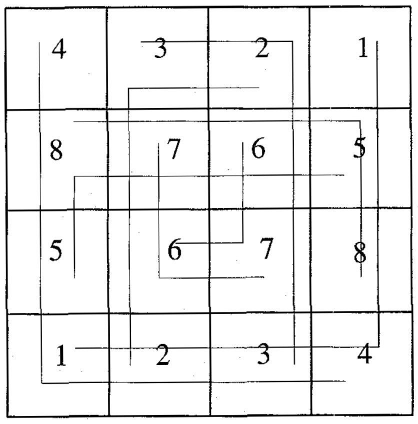


Fig. 1. A routing of Difficult-4 with wire length 32.

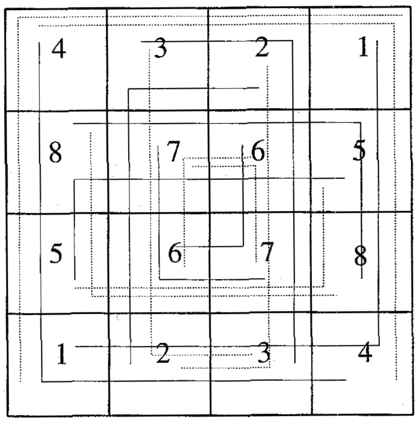


Fig. 2. A routing of Difficult-4 with super-nets.

Fig. 2 shows the resulting routing of Difficult-4 where each net has been specified two, four, or eight times. The solid route corresponds to the route specified in Fig. 1, while the dotted route corresponds to the other possible nonoverlapping route.

6.2 Gate Array Benchmark Example Results

We have also tested two gate array benchmark circuits, Primary1 and Primary2, from the 1988 IEEE Workshop on Placement and Routing. To compare our results to [17], [18], we used the placement as provided by [17]. Just as [17] did in their model, we set the width of the left and right boundaries to zero. To translate the physical design into the graph-based model, we impose a uniform grid on the gate array so that pins of modules are assigned to particular vertices on the grid. Fig. 3 illustrates how we impose a grid on the gate array when using approximately as many columns as rows. The resulting routing goes from vertex to vertex of the grid. Our grid is similar to the one used in [4].

We tested Primary1-GA with several different grids, one using 27 rows and 19 columns as in [17], one using 27 rows and 27 columns, one using 27 rows and 54 columns, and one using 27 rows and 106 columns. As the grid becomes finer, more nets must be routed by the global router, i.e., fewer nets fall under one grid, and many nets have more pins as fewer

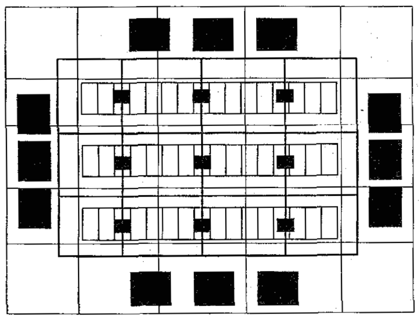


Fig. 3. Mapping a gate array to a graph.

TABLE II
BENCHMARK EXAMPLE RESULTS

Circuit	Cells	Nets	Grid Description			Density		Net Length		Params		Time
			Row	Col	Nets	[17]	CC	[17]	CC	α	I	(sec.)
Prim1	752	915	27	19	792	5	5	1.5162e+06	1.5641e+06	0.1	20	183.1
Prim1	752	915	27	28	822	–	5	–	1.7990e+06	0.5	20	226.1
Prim1	752	915	27	54	859	–	5	–	2.0078e+06	0.5	50	1101.5
Prim1	752	915	27	106	891	–	6	–	1.9301e+06	0.5	20	76.7
Prim2	3000	3050	37	37	2431	10	9	7.2095e+06	6.3262e+06	0.3	20	1081.7
Prim2	3000	3050	37	74	2698	–	9	–	7.2196e+06	0.5	50	4188.9

of the nets' pins fall under one grid. This explains why for Primary1-GA, we had to route 792, 822, 859, and 891 of 915 total nets, respectively.

We also tested Primary2-GA with two different grids, one using 37 rows and 37 columns as in [17], and one using 37 rows and 74 columns. In these examples, we had to route 2431 and 2698 of 3050 total nets, respectively.

Our results for Primary1-GA are very close to [17]. Our maximum densities are the same at five and our total net length is just 3% worse than [17]. Also, we both achieved the theoretical lower bound for this circuit. Our results for Primary2-GA tell a different story. On this much larger circuit, we achieved the theoretical lower bound of nine while [17] achieved ten, a 10.0% improvement. Our total net length is also 12.3% shorter than [17].

Table II summarizes our results. The first three columns of Table II give the name, the number of cells, and the number of nets in each of the benchmark examples tested. The next three columns describe the characteristics of the grid that we superimpose on the graph. In particular, the first two columns give the number of rows and columns, respectively. The third column gives the number of nets that the global router must actually route. The next two columns of Table II give the maximum densities achieved by [17] and our algorithm. The next two columns of Table II give the total net lengths achieved by [17] and our algorithm. The next two columns describe particular parameters passed to the program. In all test cases, we set ϵ to 0.001. The first column specifies the initial value of α. The second column specifies the total number of R2–R4 iterations the program was allowed to perform. The last column of Table II gives the total execution time of our program on a Sun SPARC 20.

We route Primary1 with two different channel capacities: 6 and 5, and route Primary2 with the channel capacities of 9. Table II shows only those tests which succeed; we did attempt to route with smaller channel densities, but these all

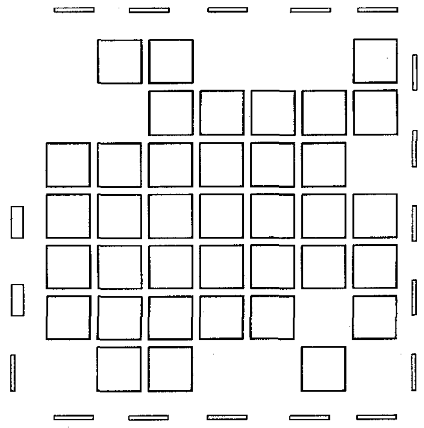


Fig. 4. A test case provided by MCC.

generated overflow nets. In all cases, we run the program with the option to update edge weights after routing each net in R1, then attempt to achieve a feasible integer solution by running Steps R5–R6, then if necessary proceed with Steps R2–R4 followed by Steps R5–R6. We also perform block pivoting.

6.3 Application to Multichip Module Designs

We have tested our program on a multichip module design provided to us from MCC [25]. This test case, illustrated in Fig. 4, models a next generation supercomputer on a 6×6 inch substrate with 37 $1.5 \times 1.5\,\mathrm{cm}^2$ gate arrays, and 18 pads. The netlist contains 7118 signals and 14 659 pins. By using the global router as an estimator, we can then determine whether a given pitch for channels will give a feasible routing. The design uses two layer metal routing with the same pitch for each layer. We use the placement provided to us by MCC. It arranges the cells within seven rows and seven columns. Our grid, therefore, is 7×7 with additional rows and columns provided for the perimeter.

The resulting graph model maps exactly one cell to a grid, and subsequently all the pins of that cell to that grid. All nets connect one distinct cell to another distinct cell, that is, no nets connect one pin of one cell to another pin of the same cell. The vast majority of the nets are two pin nets.

Because many nets in the design are identical when described in terms of graph vertices, we lump identical nets into a single super-net. To compensate, we set the demand of that super-net to be the number of nets it represents. For example, suppose nets 1–6 connect various pins of cells 1 and 4, respectively. We simply treat this as one net with a demand of 6. The multicommodity flow algorithm then assigns a flow of 6 units to that net and proceeds to reroute portions of that flow as dictated. Thus, in the integerization phase, only the fractional parts of each flow tree need to be redistributed.

With the test case from MCC (Table III), the 7118 nets are reduced to 319 nets, 279 of whom are super-nets. Twelve of these super-nets have cardinality exceeding 100 with one of them representing 280 nets. As each net in the MCC test case represents an average of 22 nets, we find that the error between the fractional solution and the subsequent integerized solution is, in practice, less than 1%.

TABLE III
ROUTING OF MCC TEST CASE WITH PITCH 65

ϵ	α	Fractional		Integerized $\bar{z}$		Iterations		Final Overflow		Time
		z_d	$\bar{z}$	Immed	Final	Fract	Tot	Nets	Edges	(sec.)
0.8	0.5	1.503	.689	.689	.935	1	64	11	3	232.4
	1.5	2.475	.736	.736	.935	3	22	255	39	425.1
0.5	0.5	1.449	.878	.875	.935	3	34	116	12	198.0
	1.5	1.117	.897	.896	.935	8	51	221	28	744.9
0.2	0.5	1.170	.936	.935	.935	21	41	25	5	58.5
	1.5	1.170	.937	.932	.935	61	64	249	39	81.1
0.1	0.5	1.039	.937	.932	.935	253	256	2	2	289.6
	1.5	1.039	.937	.935	.935	430	457	257	40	788.2
0.05	0.5	.985	.937	.935	.935	478	481	2	2	505.3
	1.5	.984	.937	.932	.935	632	635	267	43	550.6
0.01	0.5	.946	.937	.932	.935	657	660	2	2	692.6
	1.5	.946	.937	.935	.935	678	686	267	45	604.7
0.001	0.5	.937	.937	.932	.935	709	712	2	2	942.4
	1.5	.937	.937	.935	.935	750	764	268	46	908.9

Specifically, we attempt to do a global routing on this test case with a track pitch of 65 μm. Our global router, however, derives a lower bound on the solution by computing the upper bound on the routing throughput. We used an epsilon value of 0.01 for this test. The resulting fractional solution value of z_d is 0.945898 implying that at best, the solution is 94.5898% feasible, and that with a 1% margin of error, we can never hope to use a pitch larger than 61.4834 to achieve a feasible fractional solution, this with the assumption that the Steiner tree is limited to the domain of the searched active tree configurations. Our integer solution $\bar{z}$ is .934959, implying that a pitch of at most 60.7 will give a feasible routing. With a pitch of 60 μm, our global router successfully routes this test case. Consequently, we see that our global router can be used as an effective estimation procedure.

VII. CONCLUSION

This paper describes a global router based on a multiterminal, multicommodity flow algorithm. We utilize Shahrokhi and Matula's algorithm [21] to derive the fractional flow solution in a relatively short execution period. We extend this method to handle n-pin nets instead of just 2-pin nets. Based on the [21] algorithm, we exhibit, at any stage, the error bound of the current result from an optimal solution of the multicommodity flow problem. We then use a randomized rounding technique to derive a discrete net connection with an error bound on the derivation from the optimal fractional solution. We finish by employing an iterative procedure to improve the final results.

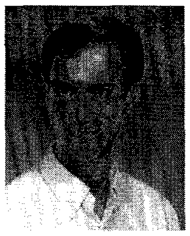
In the iteration, the cost of each edge is an exponential function of the congestion. Thus, the cost of highly congested edges dominate the cost of the whole net, which we conjecture explains why [18], [17] approach of [16] on min/max minimal rectilinear Steiner trees performs quite well in certain cases. Our adaptation of Shahrokhi and Matula's network flow algorithm [21] can effectively improve the overall routability of a circuit by minimizing edge congestion. In the iteration, our flow rerouting procedure can determine the optimal amount of flow to reroute. We show that our approach derives an approximate optimal solution under certain conditions. Thus, on these cases, our procedures to handle net ordering and cost assignment on edges answer the problems proposed by Ting and Tien.

ACKNOWLEDGMENT

The authors thank T. C. Hu for numerous stimulating conversations, and the suggestion of E. S. Kuh is highly appreciated. Also, they are very grateful to M. Sarrafzadeh and C. Sechen for providing test data. The authors are indebted to T. Hamada for his help in developing TimberWolf translators and in plotting data, and also appreciate A. Leung's manual help in preparing some of our test data. One of the authors (R. C. Carden) wishes to thank N. Dutt and D. Gajski at the University of California, Irvine, for allowing him access and time on their computer systems. One of the authors (R. C. Carden) conducted this research during his Ph.D. program at the University of California, San Diego.

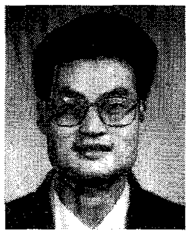
REFERENCES

[1] B. S. Ting and B. N. Tien, "Routing techniques for gate array," *IEEE Trans. Computer-Aided Design*, vol. CAD-2, pp. 301–312, Oct. 1983.
[2] T. C. Hu and M. T. Shing, "A decomposition algorithm for circuit routing," in *VLSI Circuit Layout: Theory and Design*. T. C. Hu and E. S. Kuh, Eds., New York: IEEE, 1985, pp. 144–152.
[3] A. Vannelli, "An interior point method for solving the global routing problem," in *IEEE Proc. Custom Integrated Circuits Conf.*, 1989, pp. 3.4.1–3.4.4.
[4] ——, "An adaptation of the interior point method for solving the global routing problem," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 193–203, Feb. 1991.
[5] R. M. Karp, F. T. Leighton, R. L. Rivest, C. D. Thompson, U. V. Vazirani, and V. V. Varizani, "Global wire routing in two-dimensional arrays," *Algorithmica*, vol. 2, pp. 113–129, 1987.
[6] M. Burstein and R. Pelavin, "Hierarchical wire routing," *IEEE Trans. Computer-Aided Design*, vol. CAD-2, pp. 223–234, Oct. 1983.
[7] E. Shragowitz and S. Keel, "A global router based on a multicommodity flow model," *Integration, the VLSI J.*, vol. 5, pp. 3–16, 1987.
[8] P. Raghavan and C. D. Thompson, "Multiterminal global routing: A deterministic approximation scheme," *Algorithmica*, vol. 6, pp. 73–82, 1991.
[9] T. C. Hu, *Integer Programming and Network Flows*. Reading, MA: Addison–Wesley, 1969.
[10] G. Meixner and U. Lauther, "A new global router based on a flow model and linear assignment," in *IEEE Proc. Int. Conf. Computer-Aided Design*, 1990, pp. 44–47.
[11] N. Karmarkar, "A new polynomial-time algorithm for linear programming," *Combinatorica*, vol. 4, pp. 373–395, 1984.
[12] P. Raghavan, "Lecture notes on randomized algorithms," Res. Rep. RC 15340 (#68237) 1/9/90, IBM Res. Div., T. J. Watson Res. Center, Yorktown Heights, NY, 1990.
[13] E. S. Kuh and M. Marek-Sadowska, "Global routing," in *Layout Design and Verification*, T. Ohtsuki, Ed. Amsterdam, The Netherlands: North Holland, 1986.
[14] T. M. Parng and R. S. Tsay, "A new approach to sea-of-gates global routing," in *IEEE Proc. Int. Conf. Computer-Aided Design*, 1989, pp. 52–55.
[15] K. W. Lee and C. Sechen, "A global router for sea-of-gates circuits," in *Proc. European Design Automation Conf.*, 1991, pp. 242–247.
[16] C. Chiang, M. Sarrafzadeh, and C. K. Wong, "A powerful global router: Based on Steiner min-max trees," in *IEEE Proc. Int. Conf. Computer-Aided Design*, 1989, pp. 2–5.
[17] C. Chiang, C. K. Wong, and M. Sarrafzadeh, "A weighted steiner tree-based global router with simultaneous length and density minimization," *IEEE Trans. Computer-Aided Design*, vol. 13, pp. 1461–1469, Dec. 1994.
[18] C. Chiang, M. Sarrafzadeh, and C. K. Wong, "Global routing based on steiner min-max trees," *IEEE Trans. Computer-Aided Design*, vol. 9, pp. 1318–1325, Dec. 1990.
[19] J. M. Ho, G. Vijayan, and C. K. Wong, "Constructing the optimal rectilinear steiner tree derivable from a minimum spanning tree," in *IEEE Proc. Int. Conf. Computer-Aided Design*, 1989, pp. 6–9.
[20] R. C. Carden and C. K. Cheng, "A global router using an efficient approximate multicommodity multiterminal flow algorithm," in *ACM/IEEE Proc. 28th Design Automation Conf.*, June, 1991, pp. 316–321.
[21] F. Shahrokhi and D. W. Matula, "The maximum concurrent flow problem," in *J. Assoc. Computing Machinery*, Apr. 1990, pp. 318–334.
[22] M. V. Lomonosov, "Combinatorial approaches to multiflow problems," *Discrete Appl. Mathematics*, vol. 11, pp. 1–94, 1985.
[23] W. R. Pulleyblank, "Two steiner tree packing problems," in *ACM Proc. 25th Ann. Symp. Theory Computing*, 1995, pp. 383–387.
[24] T. C. Hu, *Combinatorial Algorithms*. Reading, MA: Addison–Wesley, 1982.
[25] D. Cobb, private communication, Sept. 1990, MCC, Austin, TX.
[26] R. C. Carden and C. K. Cheng, "Feasibility estimation and cost optimization for multichip module technologies," in *IEEE Proc. 4th ASIC Conf.*, Sept. 1991, pp. 1–4.



Robert C. Carden IV (S'86–M'91) received the B.S. degree in mathematics from the University of California, Irvine in 1983, and the M.S. and Ph.D. degrees in computer science from the University of California, San Diego, in 1985 and 1991, respectively.

He worked part-time with Burroughs/Unisys Corporation during the summer of 1986 through April 1991. Since April 1991, he has been with the Unisys Corporation, initially in CAE Integration, and now as a Principal Engineer in the instruction processor design group. His current research interests include CAD, programming languages, and computer system emulation technologies.



Jianmin Li (M'95) received the B.S., M.E., and Ph.D. degrees in computer science and engineering from Tsinghua University, Beijing, China, in 1988, 1990, and 1993, respectively.

He is currently a post-Doctorate with the Department of Computer Science and Engineering, University of California, San Diego. His current research interests include circuit partitioning, placement, routing, network flow optimization, rapid prototyping systems, neural network, and speech recognition.

Chung-Kuan Cheng (S'82–M'84–SM'95) received the B.S. and M.S. degrees in electrical engineering from the National Taiwan University and the Ph.D. degree in electrical engineering and computer sciences from the University of California, Berkeley, in 1984.

From 1984 to 1986 he was a senior CAD engineer at Advanced Micro Devices, Inc. In 1986, he joined the University of California, San Diego, where he is currently an Associate Professor with the Computer Science and Engineering Department. His research interests include network optimization and design automation on microelectronic circuits.