A New Strategy for Simultaneous Escape Based on Boundary Routing

Lijuan Luo, Tan Yan, Qiang Ma, Martin D. F. Wong, *Fellow, IEEE,* and Toshiyuki Shibuya

***Abstract*—Simultaneous escape routing on dense circuit boards is a very challenging task and a great amount of manual effort is still needed in order to achieve high routability. In this paper, we present a new simultaneous escape routing algorithm which is based upon a novel boundary routing approach. Our algorithm can solve complicated escape problems in a very short time. For a set of industrial escape problems, our algorithm successfully solved all of them while Cadence Allegro PCB router was only able to complete the routing of half of the problems. In addition, we propose a clustering strategy targeting at large escape routing problems. Experimental results show that this clustering strategy can significantly cut down the runtime of our router when solving large problems.**

***Index Terms*—Boundary routing, computer-aided design, escape routing, printed circuit board (PCB).**

I. Introduction

PRINTED circuit board (PCB) routing is the problem of determining wiring connections between components on the circuit board. This problem has become extremely difficult due to the rapid increase in pin count and density, the presence of differential pairs, and tight length-matching requirements. There are many papers about packaging technologies as well as routing technologies to achieve higher pin density and fewer routing layers [1]–[4]. In this paper, we focus on the problem of routing one layer assuming a reasonable layer assignment has been done. For today's high-end complex PCBs, the routing problem can only be solved by a substantial amount of tedious and time-consuming manual effort. Thus, more research on the design automation of PCB routing is greatly needed. In this paper, we focus on a key problem in PCB routing called escape routing. The objective of escape routing is to route all terminal pins to the component boundaries; i.e., the pins are "escaping" from the components. Fig. 1 shows that pins connecting two components are escaped to their respective component boundaries. (The inter-component connections are shown in dotted lines and their detailed routing paths will be determined in a subsequent detailed routing phase.)

There are several types of escape routing problems. The most widely studied one is single-component escape [5]–[9], which is the problem of routing all pins within a component to the component boundary without any constraint on the pin ordering on the boundary. Note that its application to routing nets between two different components is limited as the inconsistency of pin ordering on the two component boundaries will lead to many crossings of nets during the inter-component detailed routing phase (see Fig. 2). To address this issue, the ordered escape routing problem [10]–[12] was introduced by imposing an ordering of the escape pins for single-component escape. This way, one can first perform an un-ordered escape from one component and then do an ordered escape from the other one (matching the escape pin ordering from the first component). Unfortunately, the pin ordering from the first component may be the wrong pin ordering for the second component, resulting in an infeasible escape problem. A better way is to simultaneously escape from both components, and this simultaneous 2-component escape routing problem is the subject of this paper. Fig. 1 illustrates the simultaneous escape problem where we are given pins connecting two components and we are asked to route all pins to their respective component boundaries such that inter-component connections are planar, i.e., no net crossing. Existing published algorithms [13], [14] for this problem are based on pattern routing where each pin is given a small number of fixed escape patterns (e.g., L-shaped routes escaping no more than two tracks above/below the pin). These algorithms would perform well if the pins on the two components are reasonably well aligned. However, more complicated escape problems cannot be solved by escape algorithms based on fixed/limited escape patterns. For example, the routing solution in Fig. 1 cannot be captured by pattern routing.

In this paper, we present a new simultaneous escape routing algorithm, B-Escape, which is based on a novel boundary routing approach. B-Escape can solve complicated escape problems in a very short time. We tested our algorithm on a set of industrial escape problems. These problems were previously successfully routed by experienced layout experts taking about

Manuscript received June 5, 2010; revised August 16, 2010; accepted October 5, 2010. Date of current version January 19, 2011. This work was supported in part by the National Science Foundation, under Grants CCF-0701821 and CCF-1017516, and by a grant from the Fujitsu Laboratories. This paper was recommended by Associate Editor P. Saxena.

L. Luo, Q. Ma, and M. D. F. Wong are with the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA (e-mail: lluo3@uiuc.edu; qiangma1@uiuc.edu; mdfwong@uiuc.edu).

T. Yan is with the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA, and also with Synopsys, Inc., Mountain View, CA 94043 USA (e-mail: tanyan@synopsys.com).

T. Shibuya is with the Fujitsu Laboratories of America, Inc., Sunnyvale, CA 94085 USA (e-mail: shibu@us.fujitsu.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2010.2097173

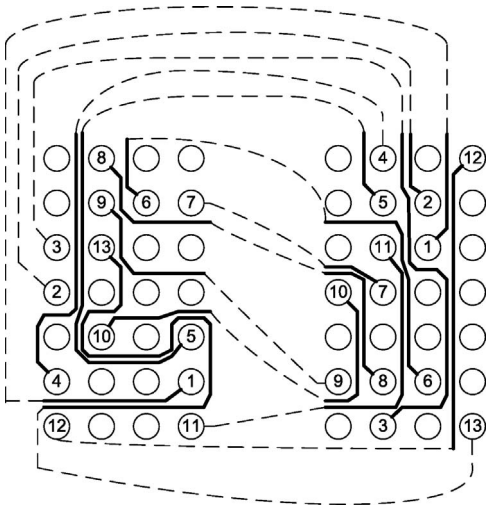


Fig. 1. Simultaneous escape routing.

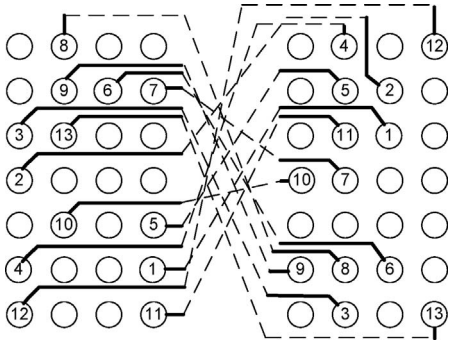


Fig. 2. Separately escaping each component causes many crossings.

8 h per problem. B-Escape successfully solved all of them while Cadence Allegro PCB router was only able to complete the routing of half of the problems. In addition, we propose a clustering strategy targeting at large escape routing problems. Experimental results show that this clustering strategy can significantly cut down the runtime of our router when solving large problems.

The rest of this paper is organized as follows. Section II introduces our boundary routing approach. Section III introduces the strategy to decide net ordering for the simultaneous escape routing problem. We limit our discussion to 1-side escape, because as shown in Fig. 3, 4-side escape can always be transformed into 1-side escape by adding more rows or columns. Also, to better present our ideas, we use a grid structure in Sections II and III. Then, we will discuss some special implementation issues on circuit boards in Section IV. Section V presents our clustering strategy for solving large problems. Experimental results are shown in Section VI. Finally, Section VII concludes this paper.

II. Boundary Routing

Before we present our simultaneous escape routing algorithm, we first introduce the foundation of our algorithm: boundary routing. To better present the idea of boundary routing, we assume that we are given a rectangular routing

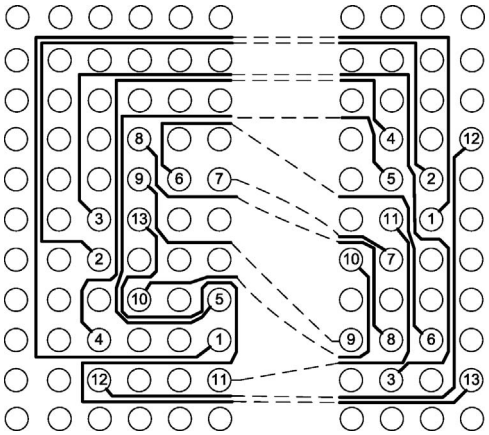


Fig. 3. Conversion from 4-side escape (Fig. 1) to 1-side escape.

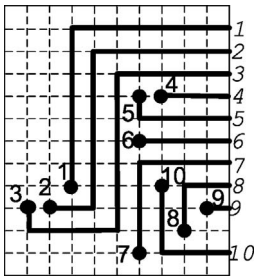


Fig. 4. Fixed-order escape problem.

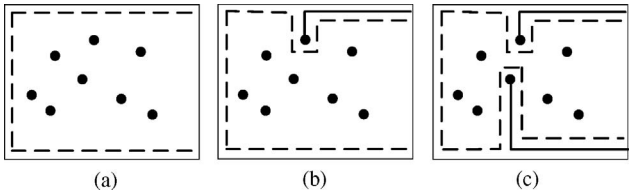


Fig. 5. (a) Initial routing boundary. (b) Boundary after routing one net. (c) Boundary after routing two nets.

area, which we call the *component*. We also assume that there is a grid structure inside the component, like in Fig. 4. (In later figures, the underlying grid is hidden for conciseness of illustration.) Selected grid points, which we call *pins*, are labeled $1, 2, \ldots, n$, and are expected to be routed to the right side of the component. The ordering of the routes must be increasing along the right side from top to bottom. We call such a routing problem the *fixed-ordering escape problem*. In the following, we will present the boundary routing methodology which is very effective on such fixed ordering escape problems.

We define the *routing boundary* as the boundary of the maximum routable region for the unrouted pins. Initially, the routing boundary is just the boundary of the component. After we route a pin, the boundary needs to be shrunk to exclude the routing of that pin. See Fig. 5 for an illustration of the routing boundary (shown as a dashed line).

We can make one observation from Fig. 5: when we route a pin, we will make more space available for later pins if we follow the routing boundary as much as possible. This observation leads to our boundary routing strategy: whenever we route a pin, we first route it to the routing boundary and

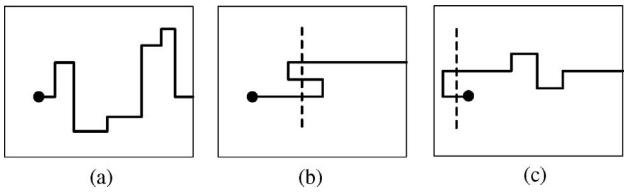


Fig. 6. (a) Monotonic routing. (b), (c) Non-monotonic routing.

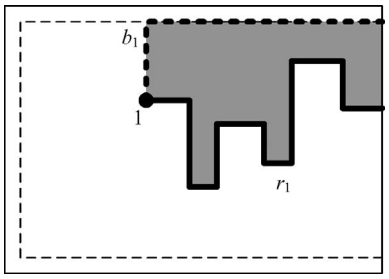


Fig. 7. Illustration for the proof of Theorem 1. The gray region should contain no pin or route.

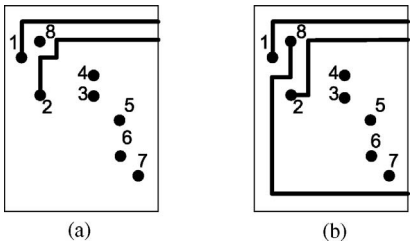


Fig. 8. Up-down mode versus upward mode. (a) Upward. (b) Up-down.

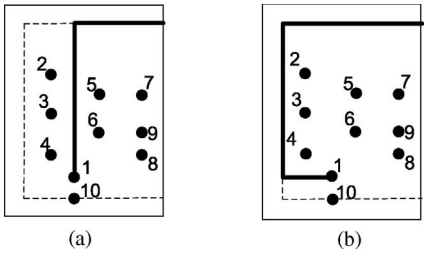


Fig. 9. Detour versus non-detour. The routing of Pin 1 in (a) blocks pins 2, 3, 4. Leftward detour in (b) resolves this issue.

then follow the boundary. Such a simple strategy turns out to be very powerful in solving fixed-ordering escape problems. The following statements show that the strategy is optimal for a certain type of escape problems.

Definition 1: A routing path is said to be *monotonic* if the intersection of any vertical line with the routing path is either empty, or a single point or a single line segment.

Fig. 6(a) shows a monotonic routing path while (b) and (c) give two non-monotonic routing paths because we can find a vertical line that intersects the routing at two points.

Theorem 1: If all routing paths in a fixed-ordering escape routing solution are monotonic, then this routing solution can be captured by the boundary routing strategy.

Sketch: Consider pin 1 with route r_1 in the solution (see Fig. 7). We draw a path from pin 1 straight up to the upper boundary of the component and then follow the upper boundary to the right side of the component. We call this path b_1.The region between b_1 and r_1 should contain no other pin or routing segment. This is because such pin or segment has a label larger than 1 and therefore a destination below r_1. Then, the monotonic routing from the pin or segment must intersect r_1 in order to reach that destination, causing a conflict.

Since the region contains no other pin or routing, we can replace r_1 with b_1 without changing the topology of the routing solution. By repeating such a procedure for each pin, we can transform a solution into a boundary-routing solution. This means we can obtain a solution with the same topology through the boundary routing strategy. ■

The ability of boundary routing is not limited to monotonic solutions. By varying the way we route from the pin to the routing boundary and the direction we follow the boundary, we can derive various routing styles. We found six routing modes simple and effective in our experiments. The first mode is *upward* mode: we route the pin straight up until it meets the boundary and then follow the boundary clockwise. The proof of Theorem 1 shows that this mode captures the monotonic routing solution. We can also route the pin straight down to the boundary and follow the boundary in a counter-clockwise direction. We call this routing mode *downward* mode. Obviously, when we use upward mode, we need to route the pins in increasing order of their labels, and for downward mode, we need to route the pins in decreasing order. Sometimes, pins may be trapped and become unroutable if we route the pins in pure increasing or decreasing order as shown in Fig. 8(a).

The *up-down* mode can resolve this issue by alternating between the upward mode and downward mode. In this mode, whenever we find that routing in upward mode will completely block another pin, we will switch to downward mode and route the unrouted pin with the largest label. In Fig. 8, we found that routing pin 2 in upward mode completely blocks pin 8, so we switch to downward mode and route pin 8 instead. Since all the routing modes introduced so far route straight up or down to the routing boundary without detouring to the left, we call them *non-detour upward/downward/up-dowm* modes. Note that the detour of net 8 in Fig. 8(b) is generated by following the routing boundary, which is not conflicting with the definition of non-detour mode.

Routing straight up or down to reach the boundary might not be the best choice for some cases. For example, routing a pin straight up may block other pins behind it and make those pins difficult to escape (see Fig. 9).

Therefore, we also introduce the *detour* style, which is to route each pin leftward to reach the boundary and then follow the boundary. The detour style is combined with the upward mode, downward mode and up-down mode to form three new modes: *detour upward* mode (route each pin leftward to reach the boundary and then follow the boundary clockwise), *detour downward* (route each pin leftward to reach the boundary and then follow the boundary counter-clockwise) and *detour up-down* mode (alternate between detour upward and detour downward). Fig. 10 shows an example of routing the same problem using the six modes. Notice that non-detour up-down

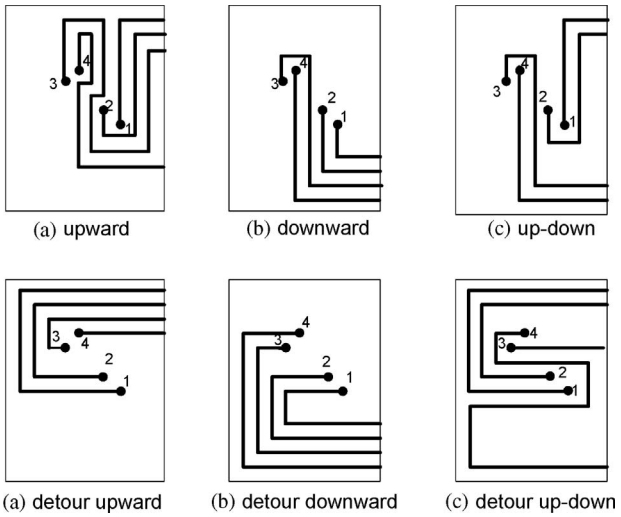


Fig. 10. Different routing modes.

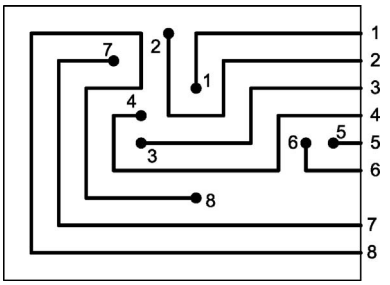


Fig. 11. Routing solution that can be captured by up-down boundary routing mode.

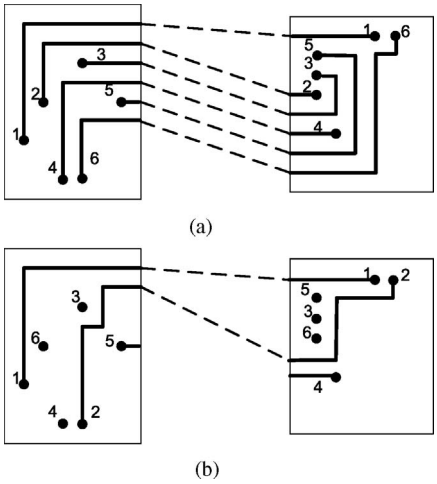


Fig. 12. Importance of correct ordering. (a) Correct ordering. (b) Wrong ordering.

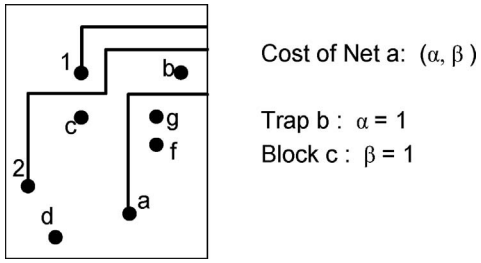


Fig. 13. Cost function.

mode and all the detour modes can capture non-monotonic routing solutions.

Although the boundary routing strategy and the six modes we use seem simple, they can cover a large number of problems we encounter in industrial data. Many routing solutions that seem complex can also be captured by one of the six modes. For example, the routing solution in Fig. 11 seems very complex. However, the same topology can be captured by boundary routing under up-down mode.

III. Dynamic Net Ordering

Given two side-by-side components and a set of two-pin nets with one pin in the left component and the other pin in the right component, a *simultaneous escape* is to find escape solutions in both components so that they are honoring the same escape ordering. An example of a simultaneous escape solution is shown in Fig. 12(a). If we already know the escape ordering, a simultaneous escape problem can be transformed into two separate fixed-ordering escape problems. However, it is very difficult to get the correct ordering. Only a slight difference in the ordering can cause huge difference in routability. Fig. 12 shows the escape results of one problem under different net orderings. We use upward mode for both cases. When the ordering is correct, we can get all six nets routed. However, if we switch Net 2 with Net 6, we can only route two nets.

Owing to the complicated escape situations on dense boards, it is always difficult to find the correct ordering beforehand. Therefore, we use a dynamic strategy to solve this ordering problem. The idea is to gradually determine the routing order as we route the nets. Whenever routing a new net, we tentatively route each remaining net, evaluate the routing cost and pick the one with minimum cost. Here α is the number of pins trapped (unroutable) by routing current net. β is the number of pins blocked (but still routable) by current routing. A net is *blocking* a pin if it is blocking the projection from the pin to the escape boundary. The blockage will cause a twisted route, which usually costs more routing resource. Since trapped pins have more impact on the routability than blocked pins, α is the dominating factor when comparing two costs. β is compared only when the α values of the two costs are the same. Fig. 13 is an example illustrating the vector cost function. Suppose we are using upward mode and the first two nets have been routed. Now we tentatively route Net *a* and calculate its cost. Note that although we are only showing one component to demonstrate the cost idea, each cost element takes both components into account.

The pseudo code of B-Escape is shown in Fig. 14. B-Escape loops through six routing modes. Under each routing mode, we define the process of routing one net (Lines 3–15) as one step. In each step, we first do trial route (Lines 3–8), i.e., tentatively route each remaining net, get the routing cost and then clear the route. Then we pick the net with minimum cost (Line 9) and actually route it (Lines 13, 14). Note that cost vectors are

```
1: for each of the six routing modes do
2:   repeat
3:     for each unrouted net i do
4:       route net i in the left component
5:       route net i in the right component
6:       calulate the cost vector for net i
7:       clear the routes generated for net i
8:     end for
9:     choose the net j with minimum cost
10:    if net j traps other nets then
11:      backtrack and reorder
12:    else
13:      route net j in the left component
14:      route net j in the right component
15:    end if
16:  until all routed or exceed the backtrack limit
17:  store the solution for this routing mode
18: end for
19: output the solution with the best routability
```

Fig. 14. B-Escape routing algorithm.

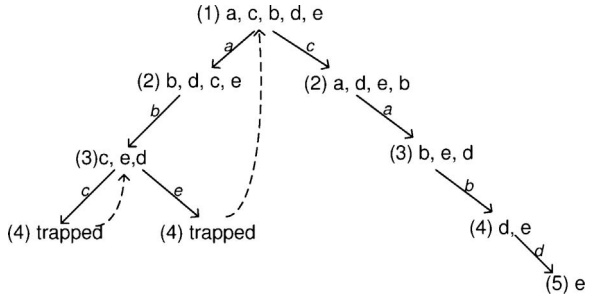


Fig. 15. Backtrack and reorder.

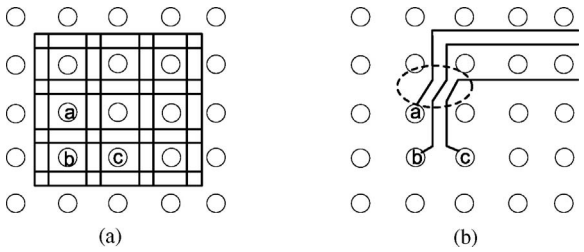


Fig. 16. (a) Regular grid structure on a PCB component. (b) Diagonal routing cannot be represented by the regular grid.

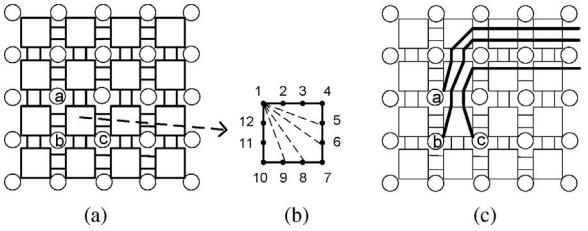


Fig. 17. (a) Grid structure with switch boxes. (b) Structure of a switch box. (c) Representation of diagonal routing.

compared with each other in lexicographical order, meaning that α is the most significant followed by β. Although most of the time this flow works pretty well, it is possible that we may find one or two nets trapped at a certain step. Lines 10 and 11 are used to detect the trap and get a new ordering.

We use the following method to reorder. For each step, we keep a list of candidate nets with non-decreasing routing cost. At first, we always choose the net with minimum cost. Once we reach the reordering point, we backtrack to the step where the cost difference between the chosen net and the next candidate net is minimum. Then we choose the next candidate net instead and continue the routing process. If more than one step has the minimum cost difference, we choose the step nearest to the reordering point. Fig. 15 demonstrates this process. At the first step, the nets are ordered as *a*, *c*, *b*, *d*, *e* according to their costs. Thus, we choose Net *a* as the first net and go to Step 2. This process continues until we reach Step 4, where if we route Net *c*, some nets will get trapped. Therefore, we need to backtrack and find a new ordering. We calculate the cost difference between the current net and the next candidate net for each step, i.e., $|cost(c) - cost(e)|$ of Step 3, $|cost(b) - cost(d)|$ of Step 2, and $|cost(a) - cost(c)|$ of Step 1. Suppose Step 3 has the minimum cost difference. We go back to Step 3 and route Net *e* instead. Continue this process until either we find the solution as in this figure or the times of backtracking have exceeded the limit. The latter case means that current routing mode cannot solve this particular escape problem; therefore, we switch to the next routing mode.

IV. Application to PCB Routing

The components on a printed circuit board also present a regular grid structure, as shown in Fig. 16(a). The labeled pins need to escape to the component boundary. There are usually two routing tracks between each pair of neighboring pins. These tracks present a grid structure similar to the one in Section II. The only difference is that here the pins are not on the routing grid. But we can simply connect a pin to a nearest grid point in the NW, NE, SW, or SE direction. The limitation of grid structure is that it cannot represent the diagonal routing on boards, like the one in Fig. 16(b). Therefore, we add a switch box inside each tile formed by four adjacent pins. Fig. 17(a) shows the grid structure with switch boxes. Each switch box has 12 points. Each corner point corresponds to a neighboring pin; e.g., Point 1 corresponds to Pin *a*, Point 7 corresponds to Pin *c*, and Point 10 corresponds to Pin *b*. The other points correspond to track segments. Fig. 17(b) shows all the possible connections between Point 1 and the other points on the switch box. In the routing process, whenever a new connection needs to be made within a switch box, we check whether this connection is crossing with previous connections; if not, whether the spacing rule is satisfied. For example, if Point 1 and Point 8 are already connected by previous routes, the connection between Point 5 and Point 12 is disabled; otherwise, it will cause crossing within the switch box. Fig. 17(c) is the routing solution obtained by using switch boxes.

B-Escape can easily handle differential pairs. A *differential pair* is a pair of nets that are required to be routed together.

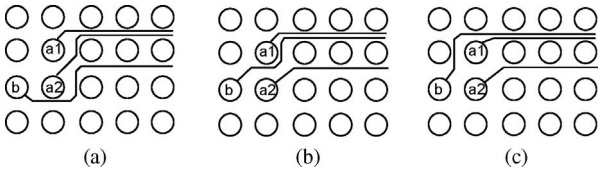


Fig. 18. Pair-routing solutions. (a) Legal pair-routing. (b) Illegal pair-routing. (c) Illegal pair-routing.

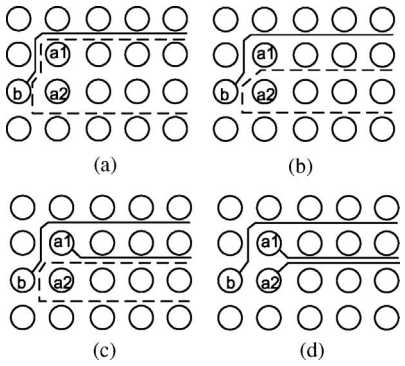


Fig. 19. Routing boundary for pair-routing. (a) Single-net boundary. (b) Paired-net boundary. (c) Rout net a1. (d) Rout net a2.

The routing of a differential pair is called *pair routing*. Fig. 18 shows three routing solutions, where Net $a1$ and Net $a2$ are a differential pair. (Note that for conciseness of illustration, we omit the underlying routing grid and switch boxes in the figure.) The first one satisfies the pair constraint. The second one is illegal because the paired nets are separated by another net. The third one is also illegal because the paired nets are separated by a row of pins. In B-Escape, as long as the paired nets are consecutively indexed, they are always routed side by side because the second net is basically following the boundary defined by the first net. To avoid the situation in Fig. 18(c), we maintain two routing boundaries. For non-paired nets or single nets, we use the boundary exactly defined by the previously routed nets as shown in Fig. 19(a). Fig. 19(b) shows how to adjust the single-net boundary for pair-routing. Here the upper boundary is lowered so that there is one track right beneath the boundary. Fig. 19(c) shows the routing of Net $a1$ obtained by tracing the paired-net boundary. Once $a1$ is routed, Net $a2$ can be treated as a single net and its route is finished in Fig. 19(d).

V. Solve Large Problems by Clustering

Our router performs very well for small and medium sized problems. For large routing problems, our router would require more time to find a solution because the dynamic reordering mechanism (Section III) takes longer to figure out a proper routing order. On the other hand, pins in large routing problems usually present a clustering structure due to the internal bus structure of the problem. Pins belonging to the same cluster are located close to each other. By taking advantage of such clustering structure, we can significantly reduce the runtime for large problems. Our main idea is as follows (see also Fig. 20): first, we use a clustering algorithm to discover the pin clusters. Then we plan the global routes for each cluster. Finally, we detail route each cluster by the B-Escape router following the global routing.

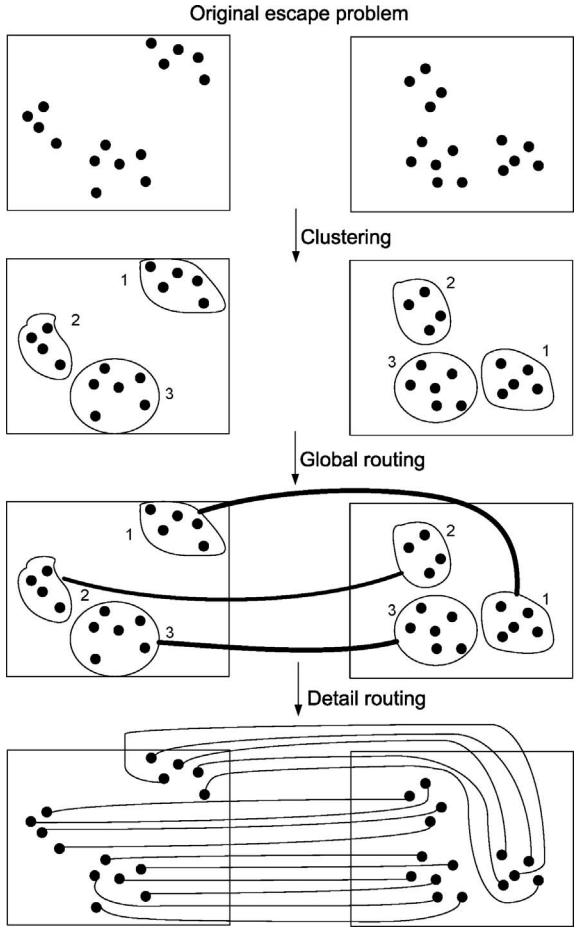


Fig. 20. Routing a large problem by clustering.

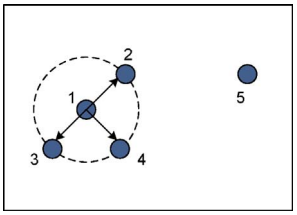


Fig. 21. Building 2-CNG (V, E) on one component. Only the edges from Pin 1 are shown.

A. Clustering

Our clustering algorithm is based on the idea of K-closest neighbor graph. A *K-closest neighbor graph* K-CNG (V, E) is a directed graph with V being the vertex set and E being the edge set. Each vertex corresponds to a pin in one component. An edge (p, q) is included in the graph if and only if there are less than K vertices in V which are closer to p than q. In this case, q is called p's *K-closest neighbor*. Fig. 21 illustrates an example of 2-closest neighbor. In the figure, Pins 2, 3, and 4 have the smallest distance to Pin 1. Therefore, they are all Pin 1's 2-closest neighbors. Pin 5 is not a 2-closest neighbor of Pin 1 because there are already three pins closer than it to

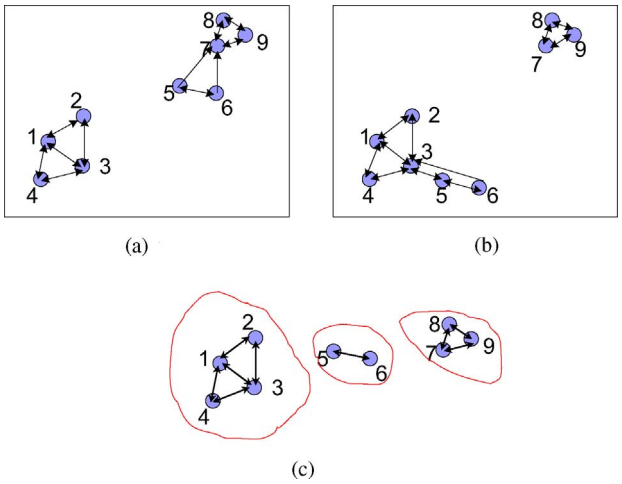


Fig. 22. Building an intersection graph. Each connected component in the graph corresponds to one cluster. (a) 2-CNG (V, E_L). (b) 2-CNG (V, E_R). (c) 2-CNG (V, $E_L \cap E_R$) and clusters.

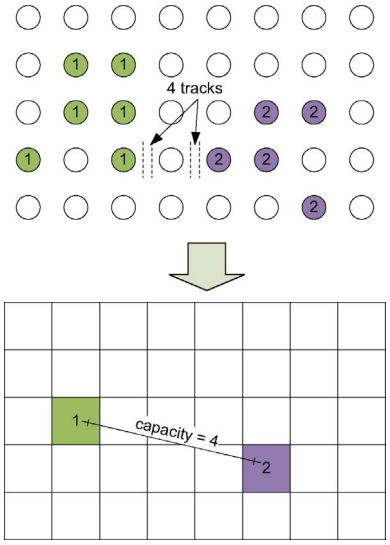


Fig. 23. Contracting clusters into representative pins and adding capacity constraints between the representative pins. We assume two routing tracks between adjacent pins.

Pin 1. As a result, we would have edges from 1 to 2, 3 and 4, respectively, but no edge from 1 to 5. After we build the K-CNG (V, E_L) and K-CNG (V, E_R) for the left and right components, respectively, we compute the intersection graph K-CNG (V, $E_L \cap E_R$) (see Fig. 22). If two pins are connected in the intersection graph, it means that they are relatively close to each other in both components and they should belong to the same cluster. We consider each connected component in the intersection graph as a cluster. For example, there are three clusters in Fig. 22(c).

B. Global Routing

After the clusters are discovered by the previous step, we need to find out a global routing of the clusters to guide the detail routing. We abstract the global routing problem into a regular pin-to-pin routing problem. First, we contract each cluster into a representative pin which is located at the centroid

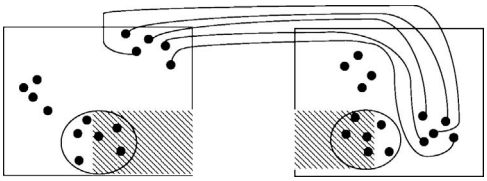


Fig. 24. Projection of cluster 3 onto its escape boundaries.

TABLE I
BENCHMARK PROPERTY AND COMPARISON RESULTS WITH ALLEGRO

Data	#Nets	Left Component #Row #Col	Right Component #Row #Col	Routability Allegro	Routability B-Escape
Ex1	39	44 × 20	22 × 28	100%	100%
Ex2	36	30 × 18	42 × 16	100%	100%
Ex3	18	18 × 24	18 × 16	100%	100%
Ex4	26	28 × 32	28 × 26	95%	100%
Ex5	52	24 × 26	60 × 10	80%	100%
Ex6	40	16 × 14	22 × 16	100%	100%
Ex7	64	38 × 16	36 × 12	90%	100%
Ex8	32	34 × 12	34 × 12	100%	100%
Ex9	41	24 × 14	30 × 30	100%	100%
Ex10	37	18 × 28	24 × 10	95%	100%
Ex11	58	54 × 12	62 × 12	96%	100%
Ex12	36	34 × 28	20 × 26	100%	100%
Ex13	38	26 × 28	36 × 30	70%	100%
Ex14	39	38 × 36	32 × 26	80%	100%
# of routed problems				7/14	14/14

of the cluster (that is, by averaging the coordinates of all the pins in the cluster). Then we obtain the wiring capacity between the two representative pins by counting the number of wiring tracks between the nearest two pins from the two clusters. This wiring track number limits the number of wires that can pass between the two clusters and thus becomes a capacity constraint. Fig. 23 shows how we contract pins and compute the capacity.

Now we have a pin-to-pin routing problem: connect the corresponding representative pins while satisfying the capacity constraint. We solve this routing problem by a negotiated congestion based router [15].

C. Detail Routing

Once we have a global routing for the clusters, we will perform detail routing for each cluster following the global routing. The global routing gives us two essential pieces of information: the escape ordering of the clusters around the component boundary and which boundary each cluster escapes toward. In our detail routing phase, the order in which the clusters are routed is consistent with the order of the global routing around the boundary. In Fig. 20, the order of the global routing around the boundary would be 1 followed by 2 and then 3.

When we route one cluster, other clusters that are already routed are considered as blockages. In addition, we also need to reserve areas that might be used by the unrouted clusters in the future. To estimate the possible routing area of an unrouted cluster, we project the unrouted cluster to the component boundary it escapes toward in the global routing. Such a projection creates a rectangular area that will be considered as blockage for the current cluster. For example, the shaded

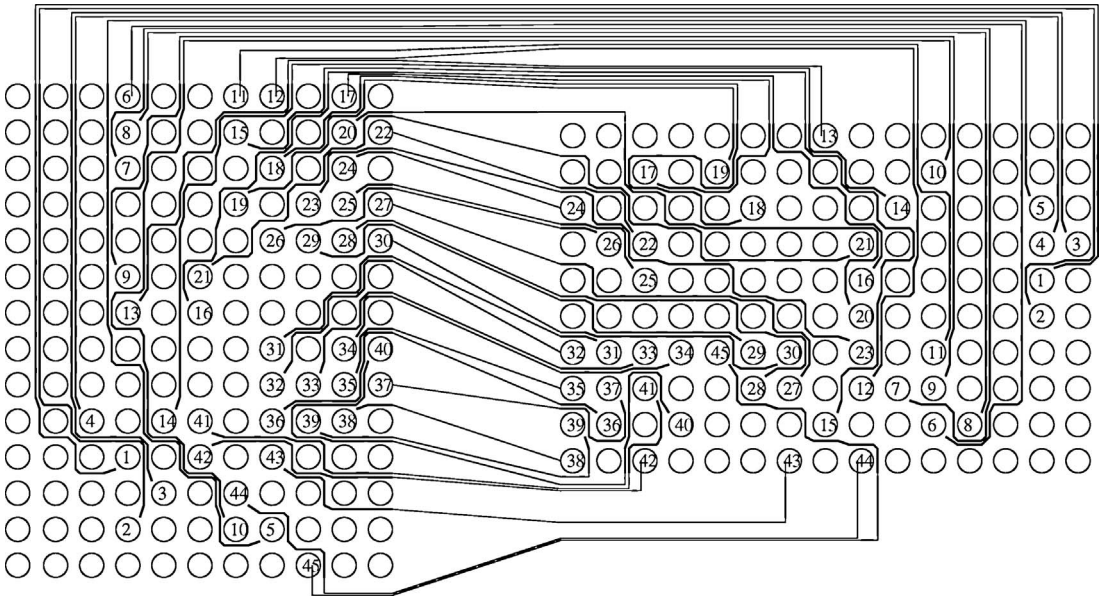


Fig. 25. Escape solution by B-Escape.

TABLE II
RUNNING TIMES (S) OF B-ESCAPE AND NRC

	Ex1	Ex2	Ex3	Ex4	Ex5	Ex6	Ex7	Ex8	Ex9	Ex10	Ex11	Ex12	Ex13	Ex14	Avg.
B-Escape	106.8	87.4	0.2	81.0	185.1	82.3	691.3	0.8	8.7	96.2	44.0	74.1	46.5	289.0	137.73
NRC	2872.0	520.5	55.8	403.5	5442.0	940.2	2881.0	18.5	3610.0	6189.0	629.8	2827.0	1005.0	2580.0	2065.87

rectangle in Fig. 24 represents the projection of Cluster 3 of Fig. 20, which is blocked when we are routing Cluster 2.

The clustering strategy presented in this section is very effective for solving large escape routing problems. As will be shown in the experimental results, the use of this strategy greatly reduces the runtime of our escape router.

VI. EXPERIMENTAL RESULTS

We have implemented B-Escape in C++ and carried out the experiments on a Pentium 4 2.8 GHz system with 4 GB memory. We tested B-Escape on 14 industrial benchmark escape problems. These problems were previously successfully routed manually by experienced layout engineers taking about 8 h per problem. Table I gives detailed information on the benchmark problems as well as the comparison results with Cadence Allegro PCB router. (#Row and #Col are the number of rows and columns of the routing grid.) B-Escape successfully solved all 14 problems but Cadence Allegro router only completed the routing of half of them. (Note that if an auto router fails to route even just one single net, the manual effort needed to finish the routing could be as high as re-routing the entire problem from scratch.) B-Escape is also very fast. Unfortunately, we could not directly compare the runtime with Allegro since there is no way to separate the escape routing runtime from the total execution time, which includes runtime for the detailed routing of inter-component connections. Since industrial PCB routers are typically based on the framework of cost-driven negotiated-congestion routing (NCR), we implemented an escape router based on NCR for runtime comparison. Our NCR router is comparable to Allegro in performance, solving 7 out of the 14 problems. On average, B-Escape is 15 times faster than NCR. (See Table II.) Fig. 25 shows the output routing solution of a simultaneous escape routing problem by B-Escape. (Note that this problem is not among the 14 industrial problems we used in the experiments since we do not have the permission to display any of them. Instead we display a problem which is derived from an industrial problem.) Due to the nature of the various detouring modes of B-Escape, the original routing solution contains many long detouring wires. We implemented a compaction procedure to shorten the long wires. The compaction simply searches for the shortest paths while maintaining the topology achieved by B-Escape and it is usually done in seconds. The solution displayed in Fig. 25 is obtained after compaction. Note that even after compaction, B-Escape routing may still generate longer routes than the shortest-path based routing scheme. However, compared with inter-component connection, escape routing only contributes to a small portion of the total wire-length. Moreover, it is typical that there are min-max length constraints requiring wires to meander to increase its wire-length. Longer escape routes can actually be beneficial since they reduce the number of length extensions in the inter-component connections.

We also implemented the clustering strategy presented in Section V. We ran our router on three large escape routing problems with and without using the clustering strategy. Both approaches achieved 100% routability on all three problems. However, we observed a substantial speedup when the clustering strategy was used. Table III summarizes the experimental

TABLE III
SPEEDUP OF THE CLUSTERING STRATEGY OVER THE FLAT RUN

	Left Component		Right Component				Runtime		
Data	#Rows	#Cols	#Rows	#Cols	#Clusters	#Nets	With Clustering	W/O Clustering	Speedup
Large1	35	35	35	35	4	94	116 s	1648 s	14.2x
Large2	30	20	35	11	4	93	155 s	1228 s	10.6x
Large3	40	40	40	40	3	71	106 s	4740 s	40.9x

results. It can be seen that the speedup can be as great as 40x. The reason why the clustering strategy can greatly cut down the runtime is that by decomposing the routing problem into smaller clusters, the solution space for proper routing order is greatly limited. As a result, the backtracking based dynamic ordering step takes much less time to find a good routing order. We also tested the negotiated-congestion router on the three large escape problems. The routability is equal to or less than 55% and the running time ranges from 6 h to 11 h.

Note although all of our test cases have two routing tracks between adjacent pins, our algorithm is not limited to the particular number of tracks. More/fewer tracks only result in larger/smaller routing grids. And our switch-box structure provides the flexibility to handle any spacing rule for the 45 degree routing segments.

VII. CONCLUSION

We have presented a new simultaneous escape routing algorithm in this paper. It dynamically decides net ordering according to real time congestion information. Each net is routed by a novel boundary routing engine, which always leaves as much space as possible to the unrouted nets and therefore guarantees great routability. Experimental results show that our escape router performs better than the Cadence PCB router on industrial benchmarks. We also proposed a clustering strategy for solving large escape routing problems. The strategy can discover the clustering structure in large problems, and by utilizing such structure, it can greatly reduce the runtime. The effectiveness of our clustering strategy is verified by experiments.

REFERENCES

[1] T. Chiba, M. Yamada, and F. Kobayashi, "Limitation of the signal pin density on wiring boards," *IEEE Trans. Compon. Packag. Manufact. Technol.*, vol. 19, no. 2, pp. 391–396, May 1996.

[2] C. S. Patel, K. P. Martin, and J. D. Meindl, "Optimal printed wiring board design for high I/O density chip size packages," *Circuit World*, vol. 25, no. 4, pp. 25–27, 1999.

[3] J. Cong, J. Fang, and K.-Y. Khoo, "Via design rule consideration in multilayer maze routing algorithms," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 19, no. 2, pp. 215–223, Feb. 2000.

[4] A. Titus, B. Jaiswal, T. Dishongh, and A. Cartwright, "Innovative circuit board level routing designs for BGA packages," *IEEE Trans. Adv. Packag.*, vol. 27, no. 4, pp. 630–639, Nov. 2004.

[5] W.-T. Chan and F. Y. Chin, "Efficient algorithms for finding maximum number of disjoint paths in grids," *J. Algorithms*, vol. 34, no. 2, pp. 337–369, 2000.

[6] J. W. Fang, I. J. Lin, P. H. Yuh, Y. W. Chang, and J. H. Wang, "A routing algorithm for flip-chip design," in *Proc. IEEE/ACM ICCAD*, Nov. 2005, pp. 753–758.

[7] R. Wang, R. Shi, and C.-K. Cheng, "Layer minimization of escape routing in area array packaging," in *Proc. IEEE/ACM ICCAD*, Nov. 2006, pp. 815–819.

[8] T. Yan and M. D. Wong, "A correct network flow model for escape routing," in *Proc. ACM/IEEE DAC*, Jul. 2009, pp. 332–335.

[9] M.-F. Yu and W. W.-M. Dai, "Single-layer fanout routing and routability analysis for ball grid arrays," in *Proc. IEEE/ACM ICCAD*, Nov. 1995, pp. 581–586.

[10] J. W. Fang, C. H. Hsu, and Y. W. Chang, "An integer linear programming based routing algorithm for flip-chip design," in *Proc. ACM/IEEE DAC*, Jun. 2007, pp. 606–611.

[11] L. Luo and M. D. Wong, "Ordered escape routing based on Boolean satisfiability," in *Proc. IEEE/ACM ASP-DAC*, Jan. 2008, pp. 244–249.

[12] Y. Tomioka and A. Takahashi, "Monotonic parallel and orthogonal routing for single-layer ball grid array packages," in *Proc. IEEE/ACM ASP-DAC*, Jan. 2006, pp. 642–647.

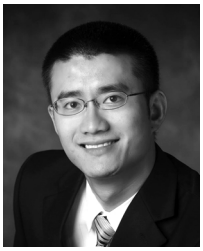
[13] M. M. Ozdal and M. D. Wong, "Simultaneous escape routing and layer assignment for dense PCBs," in *Proc. IEEE/ACM ICCAD*, Nov. 2004, pp. 822–829.

[14] M. M. Ozdal, M. D. Wong, and P. Honsinger, "Simultaneous escape-routing algorithms for via minimization of high-speed boards," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 1, pp. 84–95, Jan. 2008.

[15] L. McMurchie and C. Ebeling, "Pathfinder: A negotiation-based performance-driven router for FPGAs," in *Proc. ACM Int. Symp. FPGA*, Feb. 1995, pp. 111–117.

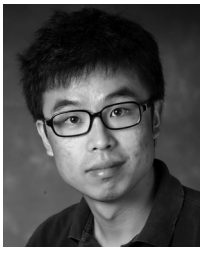
Lijuan Luo received the B.S. degree in computer science from Nankai University, Tianjin, China, in 2003, and the M.S. degree in computer science from Tsinghua University, Beijing, China, in 2006. She is currently working toward the Ph.D. degree in electrical and computer engineering from the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana.

She is currently working on parallel computer-aided design (CAD) algorithms in compute unified device architecture. Her main research interests include CAD for very large scale integration, primarily in the area of physical design, including placement and routing.



Tan Yan received the B.S. degree in computer science and technology from Fudan University, Shanghai, China, in 2003, and the M.Eng. degree in information engineering from the University of Kitakyushu, Kitakyushu, Japan, in 2005. He is currently pursuing the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana.

He is currently a member of the Detail Routing Team, Synopsys, Inc. His current research interests include combinatorial algorithms with applications in the computer-aided design of integrated circuits. His major research focus is in printed circuit board routing and integrated circuit routing.



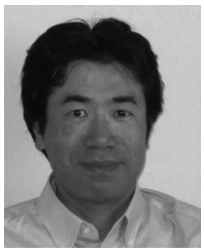
Qiang Ma received the B.Eng. degree in electrical engineering from Zhejiang University, Hangzhou, China, in 2006, and the M.Phil. degree in computer science from the Chinese University of Hong Kong, Shatin, Hong Kong, in 2008. He is currently working toward the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana.

His current research interests include physical design of chips, packages, and printed circuit boards.

Martin D. F. Wong (F'06) received the Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign (UIUC), Urbana, in 1987.

He is currently a Professor of Electrical and Computer Engineering with the Department of Electrical and Computer Engineering, UIUC. He has published over 350 technical papers and has graduated 39 Ph.D. Students in the area of computer-aided design (CAD) of very large scale integration.

Dr. Wong has won a few best paper awards in CAD. He has served on technical program committees of all leading CAD conferences and has served as an Associate Editor for several IEEE/ACM journals (e.g., IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS and *ACM TODAES*). He was an IEEE Distinguished Lecturer from 2005 to 2006.



Toshiyuki Shibuya received the B.E. degree in electrical engineering from Waseda University, Tokyo, Japan, in 1985.

He joined Fujitsu Laboratories, Ltd., Kawasaki, Japan, in 1985, and led the research and development of very large scale integration layout algorithms, design for manufacturability, statistical analysis, and parallel computing. He was a Visiting Scholar with the University of California, Los Angeles, from 1995 to 1996. He is currently the Director of Fujitsu Laboratories of America, Inc., Sunnyvale, CA.