Simultaneous Escape-Routing Algorithms for Via Minimization of High-Speed Boards

Muhammet Mustafa Ozdal, Martin D. F. Wong, *Fellow, IEEE*, and Philip S. Honsinger

***Abstract*—Shrinking transistor sizes, increasing circuit complexities, and high clock frequencies bring new board-routing challenges that cannot be handled effectively by traditional routing algorithms. Many high-end designs in the industry today require manual routing efforts, which increases the design-cycle times considerably. In this paper, we propose an escape-routing algorithm to route nets within multiple dense components simultaneously so that the number of crossings in the intermediate area is minimized. We also show how to handle high-speed-design constraints within the framework of this algorithm. Experimental comparisons with a recently proposed algorithm show that our algorithm reduces the via requirements of industrial test cases on average by 39%.**

***Index Terms*—Design constraints, escape routing, package routing, printed circuit board, randomized algorithms, via minimization.**

I. Introduction

DUE TO THE shrinkage of transistor sizes and the increase of functional complexities, the densities of very large scale integrated (VLSI) circuits have been increasing rapidly. In parallel to these developments, boards and packages have been reducing in size, while the pin counts have been increasing [2], [3]. For example, a multichip module (MCM) used in IBM eServer z900 [4] (introduced in 2000) contains 20 processor chips, eight L2 cache chips, two system control chips, four memory-bus adapter chips, and a clock chip. On the bottom of this MCM, there are 4224 I/O pins, within an area of 127×127 mm. In the subsequent generation of the same series, IBM eServer z990 [5] (introduced in 2003), the corresponding number of pins in an MCM has increased by about 20%, with a decrease of almost 50% in the substrate area. With the increasing pin densities of this pace, routing nets on boards beneath the component areas (escape routing) is increasingly becoming the main bottleneck in terms of overall routability [5]. Since the traditional routing algorithms cannot handle designs with such complexities, many high-end boards in the industry today require manual efforts for routing, which take about a month in a typical design cycle [6]. Therefore, new routing algorithms that can handle recent challenges are needed to significantly reduce this time.

In a typical high-end board, there are a number of components corresponding to the MCM, memory, and I/O modules. A component is typically in the form of a 2-D pin array, with routing tracks between adjacent rows and columns of pins. The routing resources within a component are extremely limited due to the blockage of pins and tight clearance rules. Furthermore, a large number of nets need to be routed from their terminal pins to the component boundaries using these limited resources. On the other hand, there are relatively few blockages in the area between the different components, and the amount of available routing resources is relatively larger.

In a recent work [1], a problem decomposition has been given to distinguish the following two different problems: 1) routing nets from pin terminals to component boundaries (escape routing) and 2) routing nets between component boundaries (area routing). Here, escape-routing problems for different components need to be considered simultaneously to reduce the number of crossings in the intermediate area. In other words, we cannot just apply a traditional escape-routing algorithm [7]–[11] on each component independently. The reason is that such an approach would ignore the connections between the different components and increase the buried via requirements of the next routing stage (area routing) substantially. Particularly in high-speed designs, these vias seriously degrade the signal characteristics, add an additional delay, decrease the routing area, and lower the manufacturing yields. Furthermore, for some board designs, no buried vias are allowed for the purpose of limiting manufacturing costs [6]. For such designs, the nets need to be routed in a planar fashion on every layer. Hence, an escape-routing algorithm that tries to minimize (or completely avoid) crossings in the intermediate areas is crucial to handle the recent challenges encountered in board-routing problems.

In this paper, we propose an algorithm for solving the escape-routing problem in multiple components simultaneously so that the number of crossings in the intermediate area is minimized and the high-speed-design constraints are satisfied. Fig. 1 shows a one-layer escape-routing solution for two components. In this figure, the nets have been routed from their terminal pins to the corresponding component boundaries. Here, only one net (net D) crosses with the others in the intermediate area. For this net, the area router will need to use a via to resolve the crossing. We can say that the number of crossings in the intermediate area is a good measure for the via requirements of an escape-routing solution.

Manuscript received December 4, 2006; revised April 12, 2007. This work was supported in part by the National Science Foundation under Grant CCR-0306244 and in part by an IBM Faculty Award. The preliminary version of this paper was presented in IEEE/ACM ICCAD in November 2005. This paper was recommended by Associate Editor L. Scheffer.

M. M. Ozdal was with the Department of Electrical and Computer Engineering, University of Illinois, Urbana, IL 61801 USA. He is now with the Design and Technology Solutions Department, Intel Corporation, Hillsboro, OR 97124 USA.

M. D. F. Wong is with the Department of Electrical and Computer Engineering, University of Illinois, Urbana, IL 61801 USA.

P. S. Honsinger is with IBM Corporation, Hopewell Junction, NY 12533 USA.

Digital Object Identifier 10.1109/TCAD.2007.907274

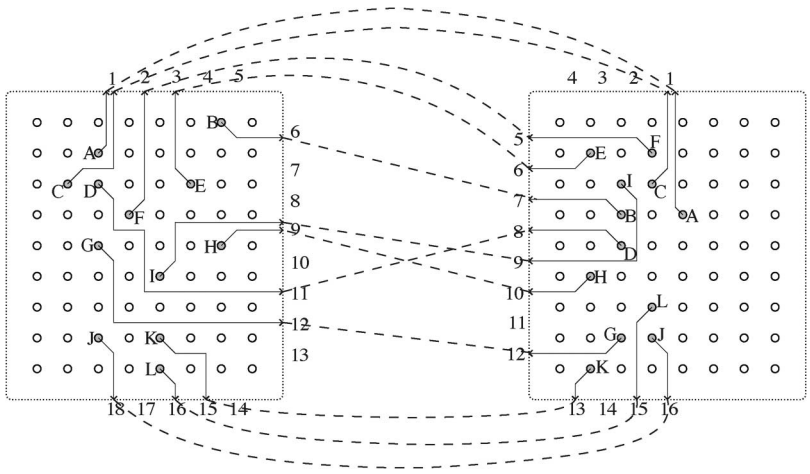


Fig. 1. Escape-routing solution for 12 nets. The escape slots are identified on the boundaries of components. The connections in the intermediate area are shown by dashed lines.

Compared to the algorithm proposed in [1], our algorithm has the following three main advantages. 1) More general escape patterns are considered within the framework, instead of simple straight connections (Section III). 2) An improved maximal planar-route-selection algorithm is proposed, which is general enough to handle multicapacity escape slots and high-speed-design constraints (Section IV). 3) An explicit discussion about how to handle various high-speed-design constraints is given for this framework (Section V). Our experiments in Section VII show that our algorithm reduces the via requirements of industrial test cases on average by 39%, compared to the recent algorithm [1].

The rest of this paper is organized as follows. We give a formal description of this problem in Section II. Our methodology to solve this problem is based on generating a number of different routing alternatives for each net and selecting the maximum planar subset of escape patterns on each layer. In Section III, we propose an algorithm to generate escape patterns based on congestion levels within the components, and the crossings in the intermediate area. Then, we propose a randomized algorithm in Section IV for the problem of maximum planar-route selection. In Section V, we discuss how to handle high-speed-design constraints within the framework of this algorithm. In Section VI, we present how our algorithms can be generalized to handle real-life problems. In particular, we discuss how to handle boards with multiple components and boards utilizing blind-via technology for component pins. Finally, we demonstrate the effectiveness of this algorithm on industrial test cases in Section VII.

II. Problem Formulation and Methodology

Let a component be defined as a 2-D array of pins, where each pin spans multiple layers and routing tracks are defined on each layer between the adjacent rows and columns of the pins. An escape segment is defined to be a route from a pin inside the component to an escape slot on the component boundary. For a component, a set of escape slots are defined on its boundary, defining the permissible end-points of escape segments originating from the pins. Due to limited routing resources, buried vias are not allowed inside the components. Therefore, routing within the component area needs to be planar on every layer. Two escape segments corresponding to two different nets are defined to have a conflict inside the component if they cannot be routed together on the same layer in a planar fashion. The number of routing tracks at each row and column is predetermined based on the pin diameters, wire widths, pin spacings, and clearance constraints. In a feasible solution, the number of escape segments passing through a row/column of the component cannot exceed the corresponding capacity of that row/column.

Let us assume that the input problem consists of only two components, which are denoted as the left and right components, respectively, for simplicity of presentation. A net is assumed to have two terminals, one in each component. An escape pattern P_i for net i is defined to be the combination of two escape segments originating from the terminals of net i in the left and right components. Two escape patterns P_i and P_j corresponding to nets i and j are defined to have a conflict iff their escape segments have conflicts within at least one component. Note here that a pair of nonconflicting escape patterns P_i and P_j can have a crossing in the intermediate area between the components, depending on the relative ordering of their escape slots. Since buried vias are allowed in the intermediate area between the components, these crossings are allowed in a feasible solution. However, the number of crossings need to be minimized for the objective of via minimization.

Based on these definitions, the simultaneous escape-routing problem for a set of nets can be stated as follows. Find an escape pattern P_i for each net i and assign it to a layer such that: 1) no pair of escape patterns on the same layer conflict with each other; 2) the capacity constraints on all the rows and columns of the components are satisfied; and 3) the number of crossings in the intermediate area is minimized.

Fig. 1 shows a sample one-layer solution for 12 nets. The number of escape slots in the left and right components are 18 and 16, respectively. While the slots on the left component are numbered increasing in the clockwise direction, the slots on the right component are numbered increasing in the counterclockwise direction. Among the 12 nets routed on this layer, only one

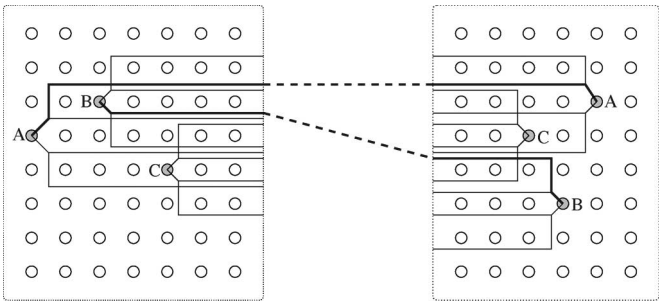


Fig. 2. Output of a simple pattern-generation technique for three nets. The maximal planar subset is highlighted with bold lines.

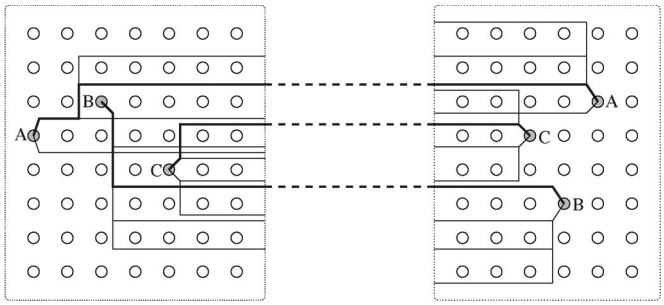


Fig. 3. Pattern generation with the objective of low congestion levels and small number of crossings. The maximal planar subset is highlighted with bold lines.

of them (net D) crosses with the others in the intermediate area. Some of the escape slots (slots 1 and 9 in the left component, slot 1 in the right component) are used by more than one escape segment. This is allowed in a feasible solution as long as the capacity constraints are not violated.

Our methodology to solve this problem is to process one layer at a time and route as many number of the noncrossing nets as possible on each layer. After finding a maximal planar-routing solution for all the layers, we distribute the remaining nets to the available layers, this time allowing crossings in the intermediate area. In the rest of the paper, we will focus on the problem of maximal planar routing. The details of the postprocessing phase to distribute the remaining nets to available layers can be found in [1].

Our algorithm for maximal planar routing consists of the following two phases: 1) generate a number of different routing alternatives for each net and 2) select the maximal subset of routing patterns that will give a feasible planar-routing solution for the current layer. A similar algorithm has been proposed for this purpose recently [1]. Our main contributions in this paper can be summarized as follows. For the first phase, we propose an algorithm to generate routing patterns based on the congestion levels inside the components and the number of crossings in the intermediate area (Section III). For the second phase, we propose a more sophisticated randomized algorithm, which can also be generalized to handle high-speed-design constraints (Sections IV and V).

III. Escape-Pattern Generation

A. Motivation

In this section, we describe an algorithm to generate a number of different routing alternatives for each net. In a recent work [1], a simple pattern-generation methodology has been used for this purpose. In particular, four escape segments are generated for each net within each component, for a total of $4 \times 4 = 16$ escape patterns. The escape segments generated in that algorithm have vertical spans of at most two rows, as shown in Fig. 2. The justification here is that escape patterns with large vertical spans block other patterns; therefore, small vertical spans are needed for maximal planar routing. However, nonregular escape patterns with larger vertical spans, as shown in Fig. 3, may be helpful in some situations. For example, if we use the simple pattern-generation technique of the recent work [1], as in Fig. 2, then only two out of three nets will be routed in a planar fashion, as highlighted in the figure. However, if we use a more intelligent pattern-generation algorithm as in Fig. 3, we can route all the three nets in a planar fashion. With this motivation, we propose an algorithm in this section that generates escape patterns based on the congestion levels within the components and the crossings in the intermediate area.

Our objective here is to generate escape patterns with low congestion levels inside the components and small number of crossings in the intermediate area. However, the patterns generated need to satisfy the following two properties.

Consider any pair of the escape segments S_i and S_j generated within the same component. Let V be a constant input parameter.

Property 3.1: If S_i and S_j correspond to the same net (i.e., S_i and S_j originate from the same terminal), then it must be the case that $|S_i.\mathrm{slot} - S_j.\mathrm{slot}| < V$.

Property 3.2: If S_i and S_j have a conflict, then it must be the case that $|S_i.\mathrm{slot} - S_j.\mathrm{slot}| < V$.

Here, the notation $S.\mathrm{slot}$ denotes the index of the escape slot of segment S, as defined in Section II. Intuitively, the segments belonging to the same net and the conflicting segments must escape to the slots that are close to each other on the component boundary. In the examples of Figs. 2 and 3, these two properties hold for $V = 4$. As will be discussed in detail in Section IV, our maximal planar-route-selection algorithm will be based on the assumption that these two properties hold. Furthermore, we will show in Section IV that a polynomial-time optimal algorithm exists for maximal planar-route-selection problem if these two properties hold for a constant V value.

B. Algorithm

Given a simultaneous escape-routing problem, as defined in Section II, we start with sorting all the net terminals based on their distances to the closest escape slots on the component boundaries. Then, we process these terminals in a sorted order, starting from the terminal closest to an escape slot. Originating from each terminal, we generate a number of escape segments, by using the algorithm shown in Figs. 4 and 5.

Fig. 4 shows the high-level algorithm used to generate a number of routing segments originating from a given terminal t. Here, graph G is used to model the routing resources of the input component. As described in Section II, a

```
GENERATE-ESCAPE-SEGMENTS(G, t, V, T)
  // G: the grid graph corresponding to the component
  // t: the terminal from which the segments will be generated
  // V: the input parameter
  // T: the set of target escape slots
  for index ← 1 to V do
    S ← GENERATE-ONE-ESCAPE-SEGMENT(G, t, T)
    add S to the set of escape segments originating from t
    T ← T ∩ ({s : S.slot−V < s < S.slot+V}\{S.slot})
      // limit the target slot range to satisfy Property 3.1
```

Fig. 4. High-level algorithm to generate a number of V escape segments originating from terminal t.

```
GENERATE-ONE-ESCAPE-SEGMENT(G, t, T)
  pQ ← an empty priority queue
  for each vertex v ∈ G that is adjacent to terminal t do
    v.label ← 0
    v.targetSlots ← T
    pQ ← pQ ∪ {v}
  while pQ not empty do
    u ← pQ.extractMin()
    if u corresponds to an escape slot then terminate loop
    for each edge (u → v) ∈ G do
      if (u.targetSlots ∩ v.reachableSlots ≠ ∅)
          && (u.label + cost(u → v) < v.label) then
        v.label ← u.label + cost(u → v)
        v.targetSlots ← u.targetSlots ∩ v.reachableSlots
          // limit the target slot range to satisfy Property 3.2.
        v.parent ← u
        pQ ← pQ ∪ {v}
  construct escape segment S by backtracing from escape slot u
```

Fig. 5. Low-level algorithm to generate one escape segment originating from terminal t.

component is assumed to be a 2-D array of pins, with rows and columns of routing tracks between the adjacent pins. In addition, a set of target escape slots $\mathcal{T}$ is specified for terminal t as the input to the algorithm of Fig. 4. Although $\mathcal{T}$ can be set such that it contains all the escape slots on the component boundary, it is also possible to set it based on the length constraints of the corresponding net, as will be discussed in Section V. Observe in Fig. 4 that after an escape segment S is generated from terminal t, the set $\mathcal{T}$ is restricted to the escape slots that are within the neighborhood of the escape slot of S. The purpose here is to make sure that Property 3.1 is maintained for the segments generated from terminal t.

Before describing the low-level algorithm, we need to make the following definition.

Definition 3.1: Let $v.\mathrm{segments}$ denote the set of escape segments passing through vertex v. An escape slot s is defined to be reachable from vertex v iff for each escape segment $S \in v.\mathrm{segments}$, it is the case that $|S.slot - s| < V$, where V is the input parameter specified in Property 3.2. The set of reachable escape slots from vertex v is denoted as $v.\mathrm{reachableSlots}$.

Remark 3.1: Consider a path P in grid graph G that starts at terminal t and ends at escape slot s. If $s \in v.\mathrm{reachableSlots}$ for each $v \in P$, then it is guaranteed that path P satisfies Property 3.2.

The low-level algorithm used to generate one escape segment is shown in Fig. 5. This is basically a variant of Dijkstra's shortest path algorithm [8]. As an additional constraint, we make sure that Property 3.2 is satisfied, by restricting the target slot range when a conflict with an existing pattern is possible. The cost of edge $(u \rightarrow v)$ is computed by the following formula:

$$\mathrm{cost}(u \rightarrow v) = \alpha.\mathrm{dist}(u \rightarrow v) + \beta.\mathrm{cong}(v) + \gamma.\mathrm{cross}(v). \tag{1}$$

Here, α, β, and γ are scaling factors for distance, congestion, and crossing cost metrics, respectively. The congestion cost for vertex v is computed based on the number of escape segments passing through v. Before generating any escape pattern, we first estimate the congestion values for individual vertices through path analysis. As we generate escape segments, we gradually replace these estimations with actual congestion values. If v is a vertex corresponding to an escape slot, we also compute a crossing cost based on the estimated number of crossings in the intermediate area.

IV. Maximal Planar-Route Selection

In this section, it is assumed that a number of escape patterns that maintain Properties 3.1 and 3.2 have been generated. The objective now is to select the maximum number of escape patterns such that: 1) at most, one pattern for each net is selected; 2) the segments of the selected patterns do not conflict with each other within the components (i.e., they are routable in a planar fashion on the same layer); and 3) the selected patterns have no crossing in the intermediate area.

A. Problem Modeling

Let $P.\mathrm{slotL}$ and $P.\mathrm{slotR}$ denote the escape slots of escape pattern P in the left and right components, respectively. Furthermore, let us assume that a unique rank is assigned for each escape segment within a component, indicating the relative ordering between the different segments. As an example, consider the segments of nets I and H in the left component of Fig. 1. Although these two segments escape to the same slot (slot 9), the rank of net I's segment must be less than the rank of the corresponding segment of net H. Let $P.\mathrm{rankL}$ and $P.\mathrm{rankR}$ denote the rank of pattern P in the left and right components, respectively. For simplicity of presentation, we will first consider the problem with unit slot capacities in the following definitions.

Definition 4.1: The less-than predicate for two escape patterns is defined as follows: $P_i \prec P_j$ iff $P_i.\mathrm{rankL} < P_j.\mathrm{rankL}$ and $P_i.\mathrm{rankR} < P_j.\mathrm{rankR}$.

Note here that the precedence relation defined above is transitive, i.e., if $P_i \prec P_j$ and $P_j \prec P_k$, then $P_i \prec P_k$. Based on this property, we can give the following definition.

Definition 4.2: A pattern sequence $\mathcal{S}$ is defined to be an ordered set of patterns $\{P_1, P_2, \ldots, P_n\}$ such that if $i < j$, then $P_i \prec P_j$.

Definition 4.3: A pattern sequence $\mathcal{S}$ is defined to be permissible iff it contains no pair of conflicting patterns.[1]

[1]We denote two patterns P_i and P_j as conflicting iff they cannot occur together in a valid planar-escape-routing solution.

Theorem 4.1: For a given set of escape patterns, the longest permissible pattern sequence $\mathcal{S}$ is equivalent to the maximum subset of patterns that can be routed on one layer in a planar fashion.

Theorem 4.2: For a given set of escape patterns, assume that Properties 3.1 and 3.2 are satisfied for a constant V value. Then, there is a polynomial-time optimal algorithm to solve the maximal planar-route-selection problem.

Proof: As given in Theorem 4.1, the maximal planar-route-selection problem is equivalent to finding the longest permissible pattern sequence among a given set of patterns. If Properties 3.1 and 3.2 are satisfied, then we can use a dynamic-programming algorithm to solve this problem. As an example, consider the simplified problem where $V = 1$, i.e., there is only one pattern corresponding to each net and no pair of patterns conflict with each other. In this case, a simple dynamic-programming-based algorithm that computes the longest sequence ending at each pattern will be sufficient. We can use the same intuition to devise an algorithm for the general case, where V has an arbitrary constant value. Let $\mathcal{P}_V$ be the set of all the different permutations of the given patterns with size V. The main idea here is to compute each longest sequence that has its last V patterns the same as an element of set $\mathcal{P}_V$. For example, let us consider $p_v \in \mathcal{P}_V$ (where p_v consists of V patterns). We can find the longest sequence that has its last V elements the same as p_v in $O(n)$ time (we need to consider the longest subsequences that have their last $V-1$ patterns the same as the first $V-1$ patterns in p_v). Based on these ideas, we can show that such an algorithm will have a time complexity of $o(n^{V+1})$, where n is the number of nets and V is a constant value. Note here that this is a loose upper bound, and more efficient algorithms can be devised for the fixed V values. In particular, an exact algorithm with time complexity $O(nc^3 + rc^4)$ has been proposed for $V = 4$ [1], where r and c are the number of rows and the number of columns of the input components, respectively. ■

Although a polynomial-time optimal algorithm exists for this problem, its high time complexity makes it impractical for large circuits. For this reason, we propose a much faster randomized algorithm, which gives solutions that are very close to optimum in practice, in the next section. As mentioned before, we will also discuss how to handle high-speed-design constraints within the framework of this algorithm in Section V. The algorithm proposed in the next section can also handle multicapacity slots, as given by the following definition.

Definition 4.4: Assume that each escape slot is defined to have a particular capacity, as defined in Section II. A sequence $\mathcal{S}$ is defined to be the capacity constrained iff the number of patterns in $\mathcal{S}$ that use a particular escape slot is less than or equal to the corresponding slot capacity.

B. Randomized Algorithm

In this section, we propose a randomized algorithm to solve the capacity-constrained longest permissible sequence problem for a given set of escape patterns. The high-level algorithm is shown in Fig. 6. Compared to the algorithm given in [1], the main improvement is our randomized subsequence-generation algorithm, as shown in Fig. 9. This algorithm not only improves the routing results considerably (Section VII), but is also general enough to handle multicapacity escape slots and typical high-speed-design constraints (Section V). We also prove later in this section that the average-time complexity of this algorithm is linear in the component sizes (Theorem 4.6).

PLANAR-ROUTE-SELECTION($\mathcal{P}$: a set of patterns)
map each pattern in $\mathcal{P}$ to a cell of checkerboard $\mathcal{C}$
rowwise partition $\mathcal{C}$ into subproblems with $V-1$ rows each
randomly generate a set of subsequences in each subproblem
create a graph $\mathcal{G}_R$ as follows:
– A vertex v_j^i exists in $\mathcal{G}_R$ corresponding to each subsequence S_j^i in subproblem i.
– The weight of v_j^i is equal to the number of patterns in S_j^i.
– Let x_j^i and x_k^{i+1} denote the x coordinates of the checkerboard cells of the last patterns in subsequences S_j^i and S_k^{i+1}.
An edge from v_j^i to v_k^{i+1} exists in $\mathcal{G}_R$ iff:
(1) all patterns in S_j^i are to the left of all patterns in S_k^{i+1}
(2) $x_k^{i+1} > x_j^i + V - 2$,
(3) no pattern in S_j^i conflict with a pattern of S_k^{i+1}.
return the longest path in $\mathcal{G}_R$

Fig. 6. High-level description of the randomized planar-route-selection algorithm.

The (conceptual) checkerboard model introduced in the algorithm of Fig. 6 is defined as follows [1].

Definition 4.5: Let $\#s\mathrm{L}$ and $\#s\mathrm{R}$ denote the number of escape slots defined on the left and right components, respectively. Let $\mathcal{C}$ be a (conceptual) checkerboard with $\#s\mathrm{L}$ rows and $\#s\mathrm{R}$ columns. An escape pattern P is defined to be mapped to cell (i, j) of checkerboard $\mathcal{C}$ iff $P.\mathrm{slotL} = i$ and $P.\mathrm{slotR} = j$.

Fig. 7 shows a sample escape problem and the corresponding checkerboard model. Let us consider the two patterns P_i and P_j on this checkerboard. If P_j is below and to the right of P_i (e.g., $P_i = B12$, $P_j = C33$), then $P_i \prec P_j$, as defined in Definition 4.1. If P_j is above and to the right of P_i (e.g., $P_i = D43$, $P_j = A35$), then neither $P_i \prec P_j$ nor $P_j \prec P_i$. Otherwise, if P_i and P_j are on the same row (e.g., $P_i = D44$, $P_j = A45$), the same column (e.g., $P_i = C33$, $P_j = D43$), or the same cell (e.g., $P_i = E56$, $P_j = F56$), then we need to check the ranks of P_i and P_j to determine the relationship between these patterns. For instance, ranks of $E56$ in both the left and right components are less than those of $F56$ (since the corresponding escape segments of net E are above those of net F); hence, $E56 \prec F56$.

After mapping each pattern to a checkerboard cell, $\mathcal{C}$ is rowwise partitioned into subproblems. Then, a set of capacity-constrained permissible subsequences is randomly generated within each subproblem, as will be described in detail later in this section. After that, these subsequences are merged together to obtain the capacity-constrained longest permissible sequence. For this purpose, a graph model $\mathcal{G}_\mathrm{R}$ is shown in Fig. 6, which satisfies the following lemma.

Lemma 4.3: Consider two subsequences S_j^i and S_k^l in subproblems i and l, respectively. If there is a path between the corresponding vertices v_j^i and v_k^l in $\mathcal{G}_\mathrm{R}$, then it is guaranteed that S_j^i and S_k^l contain no patterns that conflict with each other.

A more restricted version of this lemma has been given previously for a similar graph model [1]. Based on this lemma, we can compute the longest path in an acyclic graph of $\mathcal{G}_\mathrm{R}$ to find

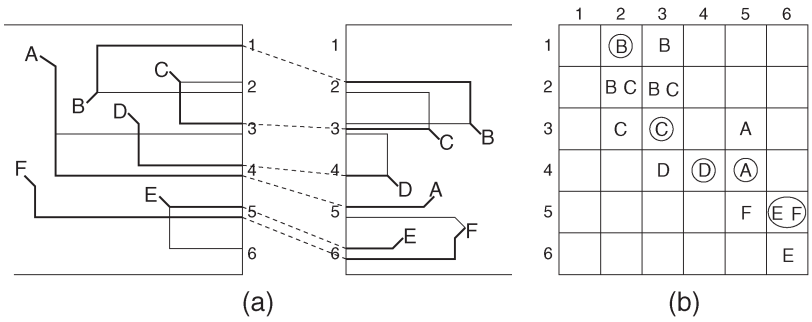


Fig. 7. (a) Set of routing patterns defined for six nets. (b) Corresponding checkerboard model. For clarity, only one or two escape segments are illustrated for each net. The maximum planar subset is highlighted in both figures.

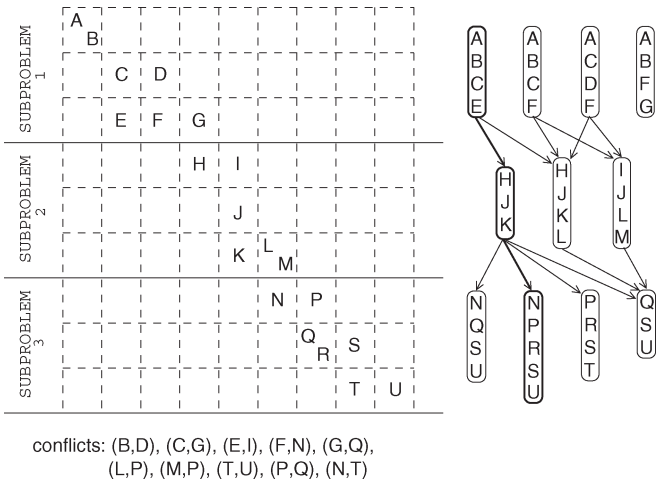


Fig. 8. Illustration of the randomized algorithm given in Fig. 6 on a sample checkerboard. For clarity, ranks of the patterns are not displayed. The set of subsequences generated for each subproblem are shown on the right, together with the corresponding graph $\mathcal{G}_{\mathrm{R}}$ and the (highlighted) longest path. It is assumed here that each escape slot has a capacity of two.

```
GENERATE-SUBSEQ(x, y, subseq)
// (x,y): coordinate of the current checkerboard cell
// subseq: the partial subsequence generated so far
if cell (x, y) is not within subproblem boundaries
    terminate recursion
if subseq is good
    store subseq in candidate set of the subproblem
Let P' be the last pattern in subseq
T ← {P : P' ≺ P (see Definition 4.1) AND
        ((x ≤ P.slotR ≤ x + Δ AND P.slotL = y) OR
        (y ≤ P.slotL ≤ y + Δ  AND P.slotR = x)) AND
        capacity of (P.slotR, P.slotL) not fully used AND
        P has no conflict with subseq}
for each pattern P ∈ T do
    randomly determine whether to accept or reject P
    if P is accepted
        GENERATE-SUBSEQ(P.slotR, P.slotL, subseq ∪ {P})
GENERATE-SUBSEQ(x + 1, y + 1, subseq)
```

Fig. 9. Algorithm to generate a set of random subsequences.

the best combination of subsequences generated in different subproblems. Then, we can merge these subsequences to obtain the longest permissible sequence. Fig. 8 shows the execution of the randomized algorithm on a sample board. Here, assume that a number of patterns have already been mapped to this checkerboard, and the conflicts between patterns are as listed in this figure. A small set of randomly generated subsequences[2] is shown for each subproblem on the right. Corresponding to each subsequence, there is a vertex in $\mathcal{G}_{\mathrm{R}}$, and edges between them are created based on the rules shown in Fig. 6. For instance, there is no edge from $\{A, B, C, E\}$ to $\{I, J, L, M\}$ because patterns E and I are conflicting. Similarly, there is no edge from $\{A, B, F, G\}$ to $\{I, J, L, M\}$ because it violates rule (2) in Fig. 6. The longest path in $\mathcal{G}_{\mathrm{R}}$, corresponding to the capacity-constrained longest sequence is also highlighted in this figure.

The algorithm we used to generate a set of random subsequences is shown in Fig. 9. In the beginning, this recursive function is called for each cell on the first row of the given subproblem, with argument subseq set to $\emptyset$. In one recursive call, first, it is checked whether the partial subsequence generated so far is good enough to store. This decision is made by comparing the weight of the current subsequence subseq with the weights of the subsequences already stored for this subproblem. Let tx denote the x-coordinate of the checkerboard cell corresponding to the last pattern in subseq. The weight of subseq is compared with only the subsequences that end at column tx of the checkerboard. An input parameter determines how many subsequences can be stored corresponding to each column.[3] If the partial subsequence subseq is to be stored, a previously stored subsequence with less weight may need to be replaced. Note here that our purpose is to generate a large variety of good subsequences for the given subproblem, instead of generating only the best ones. The variety among subsequences is obtained by making sure that a subsequence ending at a particular checkerboard column does not replace another subsequence ending at a different column.

The next step of the recursive algorithm is to find the set of patterns $\mathcal{T}$ that can be added to the partial subsequence subseq. Here, this selection is done based on the invariant that subseq remains permissible (Definition 4.3) and capacity constrained (Definition 4.4). In one recursive iteration, we consider the patterns that are: 1) on cell (x, y); 2) on column x; and 3) on row y of the checkerboard. To limit the search space, we only consider patterns that are within the Δ-neighborhood of (x, y), where Δ is an input parameter typically set to a value less than

[2]For clarity, only three or four subsequences are given in this example. Normally, hundreds or even thousands of subsequences are generated for each subproblem to obtain a good variety.

[3]In our experiments, the maximum number of subsequences that can be stored corresponding to each column is set to 50.

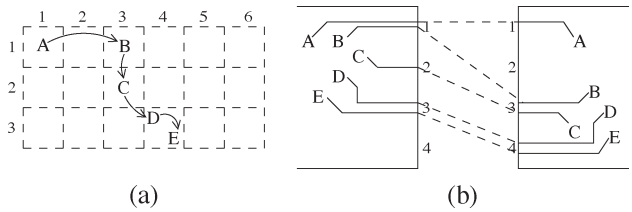


Fig. 10. (a) Subsequence on the checkerboard. (b) Corresponding escape patterns.

five. Fig. 10 shows the physical meaning of selecting patterns from the same cell, row, or column of the checkerboard.

After finding the candidate pattern set $\mathcal{T}$, we consider each P in $\mathcal{T}$ and randomly decide whether to accept or to reject P. Here, the probability of accepting pattern P is set so that the expected number of escape patterns that can be selected from set $\mathcal{T}$ is equal to a fixed input parameter.[4] In other words, this probability is inversely proportional to the number of candidate patterns in $\mathcal{T}$. If P is accepted, then another recursive call is made starting from the current checkerboard cell. After all the patterns in $\mathcal{T}$ are considered, a recursive call to cell $(x+1, y+1)$ is made to continue the subsequence generation without selecting any pattern from the current level. The main purpose here is to have a good variety in the generated subsequences.

For the following complexity analysis, we assume that parameter V, given in Properties 3.1 and 3.2, and all the slot capacities are constants [i.e., have complexity $O(1)$].

Lemma 4.4: Let $\mathcal{R}$ be the recursion tree of the function GENERATE-SUBSEQ shown in Fig. 9. The following two properties hold for $\mathcal{R}$: 1) The maximum depth of $\mathcal{R}$ is $O(1)$. 2) The number of recursive calls made from a node in $\mathcal{R}$ is $O(1)$ on the average.

Proof: At each recursive call, either a pattern P is added to the partial subsequence or the x and y coordinates are both incremented by 1. Since each subproblem consists of V rows and the escape slot capacities are constants, the maximum length of any subsequence is $O(1)$. Hence, the maximum recursion depth is $O(1)$. Furthermore, we randomly decide whether to accept or to reject pattern P such that the expected number of patterns selected in each iteration is constant. As a result, the number of recursive calls made from a node in $\mathcal{R}$ is $O(1)$ on the average. ■

Lemma 4.5: The recursive function GENERATE-SUBSEQ(x, y, subseq) is invoked only a constant number of times for each checkerboard cell (x, y).

Proof: Our proof is based on the induction on the depth of the recursive tree $\mathcal{R}$. Obviously, the checkerboard cell at the root of $\mathcal{R}$ is called only a constant number of times (base case). Let us consider a grid cell (x, y), and let us assume that the induction hypothesis holds for all the cells called before (x, y). From the algorithm of Fig. 9, we know that only the cells that are in the Δ-neighborhood of (x, y) can make a call to (x, y). Since Δ is constant, the lemma follows due to the induction hypothesis. ■

[4]We have set the expected number of patterns that can be selected at each recursive iteration to seven in our experiments. The execution time of the subsequence-generation phase can be controlled by this parameter.

Theorem 4.6: The total average-time complexity of the subsequence generation for all the subproblems is $O(n + s^2)$, where n is the number of nets and s is the number of escape slots on the component boundaries.

Proof: We will first prove that the average-time complexity for subproblem i is $O(n_i + s)$, where n_i is the number of patterns mapped to a cell within subproblem i. In one recursive call, all patterns P which are mapped to cells in the Δ-neighborhood of cell (x, y) are processed to determine the set $\mathcal{T}$. Since Δ is constant, and due to Lemma 4.5, each pattern is processed only a constant number of times. Furthermore, the average number of nodes in a recursion tree $\mathcal{R}$ is $O(1)$, due to Lemma 4.4. Since there are s separate recursion trees (each root corresponding to a cell on the first row of the current subproblem), the average-time complexity for one subproblem is $O(n_i + s)$. Based on this, the total average-time complexity for all the subproblems can be written as $\sum_{1 \le i \le s/V} O(n_i + s) = O(n + s^2)$. ■

Theorem 4.7: Let K denote the maximum number of subsequences that can be stored for each subproblem. The average-time complexity for the proposed randomized-planar-route-selection algorithm is $O(n + s^2 + K^2 s)$, where n is the number of nets and s is the number of escape slots on component boundaries.

Proof: In graph $\mathcal{G}_{\mathrm{R}}$ (defined in the beginning of this section), there is a vertex corresponding to each subsequence generated. Since there are $s/V = O(s)$ subproblems, the number of vertices in $\mathcal{G}_{\mathrm{R}}$ is $O(Ks)$. The edges in $\mathcal{G}_{\mathrm{R}}$ are only between the vertices that correspond to adjacent subproblems. Hence, the number of edges in $\mathcal{G}_{\mathrm{R}}$ is $O(K^2 s)$. Since $\mathcal{G}_{\mathrm{R}}$ is acyclic, computing the longest path has linear time complexity in the graph size [8], which is $O(K^2 s)$. As given in Theorem 4.6, the average-time complexity of the subsequence generation is $O(n + s^2)$; therefore, the proof is complete. ■

V. HANDLING HIGH-SPEED-DESIGN CONSTRAINTS

In the following sections, we discuss how to generalize the algorithms given in Sections III and IV to handle different high-speed-design constraints.

A. Maximum-Length Constraints

Board designers specify maximum-length constraints for critical nets to limit the maximum arrival times. We can handle these constraints during the pattern-generation phase of our framework. Specifically, we can restrict the set of the target escape slots (parameter $\mathcal{T}$ in Fig. 4) such that the escape segments with long detours are avoided. Furthermore, remember that an escape pattern is created by merging two escape segments from the left and right components. It is possible to check the maximum length constraints during this step and to eliminate the patterns that violate the corresponding constraints.

B. Minimum-Length Constraints

Minimum-length constraints are typically enforced for nets belonging to a bus structure, with the objective of matching

the signal-arrival times. Recently, routing algorithms have been proposed to handle these constraints [12]–[15]. Typically, the length of a short net needs to be extended to satisfy its minimum-length constraint.[5] Since the routing resources within the components are extremely limited, it makes more sense to perform length extension in the intermediate area between the components in a later stage of the routing system. However, we can also modify our randomized planar-route-selection algorithm (Section IV) such that the patterns that satisfy the minimum bounds are preferred over the others. For this purpose, we can assign a weight to each escape pattern, based on its length and the corresponding minimum-length constraint. Then, the randomized algorithm given in Section IV can be used to select the permissible pattern sequence with the largest weight.

C. Adjacency Constraints for Noise Avoidance

Adjacency constraints between different nets are defined by designers to avoid crosstalk problems. A typical adjacency constraint between nets n_i and n_j can be stated as follows [6]. If n_i and n_j are routed adjacent to each other on the same layer, then their routes need to be separated by at least k routing tracks. Such a constraint is enforced typically on signal nets that belong to different bus structures. In the context of the model defined in Section IV-A, we can restate this constraint as follows: If the patterns corresponding to n_i and n_j are adjacent in a permissible pattern sequence $\mathcal{S}$, then the escape slots of these patterns need to be separated by at least k routing tracks. This constraint can be handled effectively in the subsequence-generation algorithm shown in Fig. 9 by comparing the last pattern in the partial subsequence subseq with the candidate pattern P. Specifically, the following line needs to be added immediately after set $\mathcal{T}$ is defined in Fig. 9:

$\mathcal{T} \leftarrow \mathcal{T} \cap \{P :$ if (P', P) has a k-adjacency constraint, then there are k empty tracks between P' and P in both the left and right components$\}$

By adding this line, we make sure that only the subsequences that do not violate adjacency constraints are generated. In addition, we also need to check these constraints for subsequences in the neighboring subproblems. Specifically, we need to add the following rule while defining the edges of $\mathcal{G}_{\mathrm{R}}$ in Fig. 6.

Let v_j^i and v_k^{i+1} denote two vertices in $\mathcal{G}_{\mathrm{R}}$ corresponding to subsequences S_j^i and S_k^{i+1}, which have been generated in subproblems i and $i+1$, respectively. Let P_j^i denote the last pattern in subsequence s_j^i and P_k^{i+1} denote the first pattern in subsequence S_k^{i+1}. If (P_j^i, P_j^{i+1}) has an adjacency constraint of at least k tracks, then an edge from v_j^i to v_k^{i+1} (in $\mathcal{G}_{\mathrm{R}}$) exists only if there are at least k tracks between P_j^i and P_k^{i+1} in both components.

These two modifications are sufficient to ensure that the output of our algorithms satisfies all the adjacency constraints.

[5]These minimum-length constraints can be derived from the length-matching constraints of a set of nets, as described in [12] and [15]. Here, matching the lengths of the escape routes exactly is not crucial since the length-matching objective can be effectively realized during area routing.

D. Differential Pairs

A differential pair is a complementary pair of nets that provide noise immunity. The two nets within a differential pair, which is separated by a specific distance as long as possible, need to be routed parallel to each other. Let us consider two nets n_i and n_j that belong to a differential pair. During pattern generation, we can identify the pairs of escape segments corresponding to n_i and n_j that adhere to these constraints. In the context of the model defined in Section IV-A, a pattern corresponding to n_i can exist in a permissible sequence $\mathcal{S}$ only if it is adjacent to an acceptable segment of n_j. This constraint can be explicitly checked in the subsequence-generation algorithm of Fig. 9 by comparing the last pattern in the partial subsequence subseq with the candidate pattern P. Specifically, the following code segment needs to be added immediately after set $\mathcal{T}$ is defined in the algorithm of Fig. 9:

Let P'' be the second-to-the-last pattern in subseq
if P' belongs to a differential pair AND
  (P'', P') is not a differential pair, then
    $\mathcal{T} \leftarrow \mathcal{T} \cap \{P : (P', P)$ is a differential pair

By adding these lines, we make sure that the patterns belonging to a differential pair always occur together in a subsequence. However, we also need to check the differential pairs that are in two adjacent subproblems. For this purpose, we need to add the following rule while defining the edges of $\mathcal{G}_{\mathrm{R}}$ in Fig. 6.

Let P_{j-1}^i and P_j^i denote the second-to-the-last and the last patterns in subsequence s_j^i; let P_k^{i+1} denote the first pattern in subsequence S_k^{i+1}. Assume that P_j^i is part of a differential pair and that (P_{j-1}^i, P_j^i) is not a differential pair. If this is the case, then an edge from v_j^i to v_k^{i+1} (in $\mathcal{G}_{\mathrm{R}}$) exists only if (P_j^i, P_k^{i+1}) is a differential pair.

These two modifications are sufficient to handle the differential-pair constraints.

VI. Generalizations of the Algorithms

In this section, we discuss applicability of our algorithms in real-life scenarios. We also present discussions about how to utilize these algorithms in more general routing frameworks.

A. Routing Multiple Components

In a typical industrial board, there are numerous bus structures between a number of components, and routing needs to be done for all of them. In the algorithms that we have proposed, we have assumed a simple model of routing between two components. However, our algorithms can also be utilized within a larger routing framework where multiple components need to be routed to each other.

Fig. 11 shows the basic structure of a sample industrial board from IBM. Here, there are four quadrants of an MCM at the center and several memory and I/O modules around it. Although not explicitly shown in this figure, there are several bus structures that need to be routed from the MCM to the other modules around it. A straightforward way to handle this

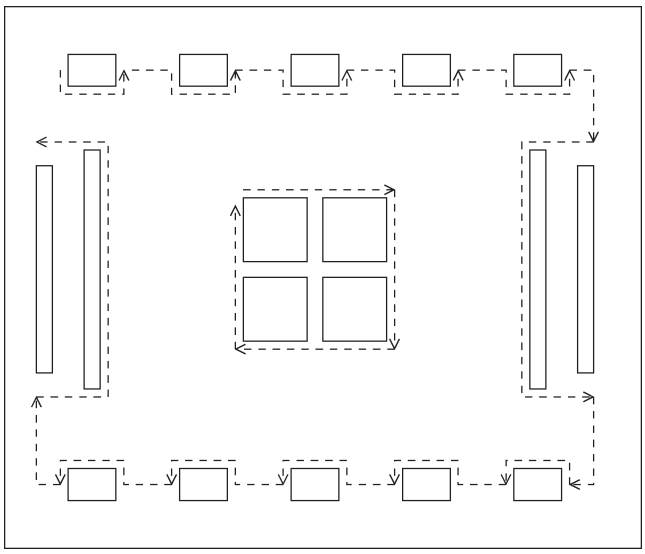


Fig. 11. Basic structure of an industrial board. There are four MCM quadrants at the center and memory and I/O modules around it. A strict ordering of the escape slots can be defined as shown with the dashed arrows.

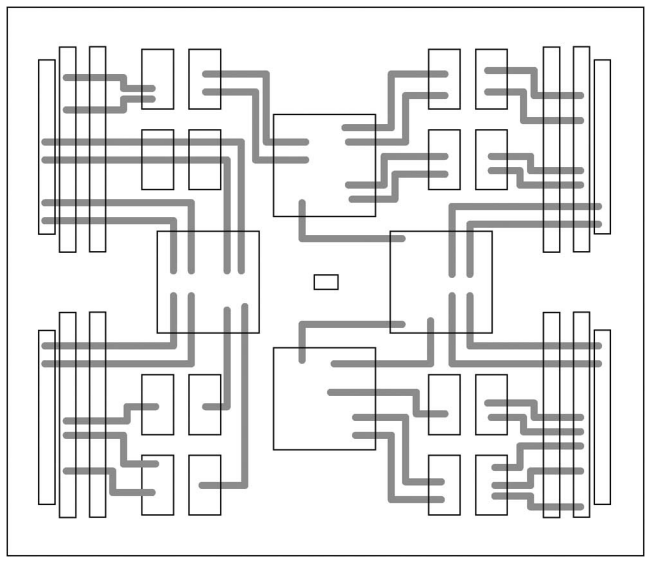


Fig. 12. Industrial board from IBM with complex bus structures. The set of buses to be assigned on one layer in a planar fashion are shown with thick lines.

problem is to group the components into supercomponents and to apply our algorithms on pairs of the supercomponents. For example, it is possible to conceptually merge the two memory modules on the left to obtain one supercomponent. Similarly, we can merge the MCM's two quadrants on the left to obtain another supercomponent. Then, we can apply our algorithms on these two supercomponents to find the escape-routing solution. After that, we can merge all the modules at the top to obtain another supercomponent and continue in this way. Observe here that this method enables the routing of one supercomponent pair at a time, as opposed to routing individual nets one by one.

Another way to handle the sample problem in Fig. 11 is to model only the following two supercomponents: 1) the MCM at the center and 2) all other modules around it. Remember from Section IV that the only assumption our algorithms make about the component structures is that it is possible to define a strict ordering between the ranks of the escape slots around the component boundaries. In particular, our models assume that a less-than predicate can be defined between any two escape patterns P_i and P_j as stated in Definition 4.1. In Fig. 11, the dashed lines represent a strict ordering between the escape slots on the boundaries of the MCM and another strict ordering between the escape slots of the other modules. Given this ordering, our algorithms are directly applicable to these two supercomponents.

It is also possible to handle boards with more complex bus structures, as shown in Fig. 12. Here, we first need a bus-planning algorithm to identify the set of buses that can be routed on each layer in a planar fashion. The output of the bus-planning algorithm for one layer is shown in Fig. 12 with thick lines. Given the planar bus structures for one layer, we can assign a strict ordering for the escape slots of each component. Furthermore, we can also group smaller components into bigger supercomponents as in the example of Fig. 11. After that, we can apply our algorithms iteratively on supercomponent pairs.

The bus-planning phase is similar to global routing in the sense that routing is determined roughly for all nets. Here, an algorithm similar to the one in [16] can be used to perform bus planning. We are currently working on such an algorithm within the larger framework of a routing tool for high-performance packages. Given the output of bus planning, the algorithms in this paper can be utilized to perform simultaneous escape routing in pairwise supercomponents.

B. Escape Routing for Ball Grid Arrays (BGA)

For the models and the algorithms that we propose in this paper, we have assumed that each component pin can be accessed from every layer of the input printed circuit board (PCB). For example, in IBM eServer z900 [4], the pin-grid-array (PGA) connectors are used to connect components or daughtercards into the PCBs. PGA-based connectors have a grid of metal leads as their pins, which are plugged into the PCBs, making each pin directly accessible from the inner layers of the board. However, our algorithms are also applicable to surface-mount-type packages, if plated through holes (PTHs) [17] (also known as through vias) are used to connect component pins to inner board layers. For example, the MCMs in IBM eServers p690 and z990 use land-grid-array connectors [18], which are mounted on the board surface and connected to PTHs on the board. Similar statements can be made for a BGA-type package that is mounted on a grid of PTHs, where the through-via pitch is equal to the ball pitch of the BGA. Typically, dog-bone-pattern-type routing is used in such cases to connect the component balls to the through vias on the board surface [19], as shown in Fig. 13(a). In these cases, we can regard each through via as a component pin in the context of board-level routing.

However, it is also possible to extend our algorithms to boards that utilize the blind-via technology, as shown in Fig. 13(b). If the blind vias are used, then the component pins do not necessarily have uniform grid structures in the lower routing

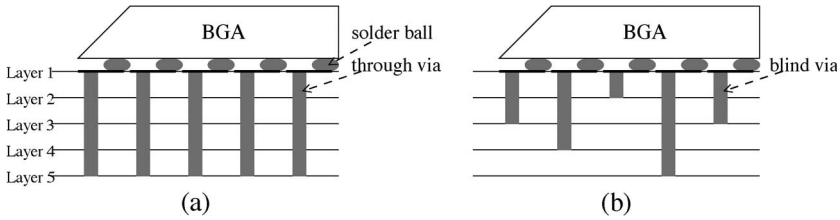


Fig. 13. BGA connected to a board (a) using through vias and (b) using blind vias.

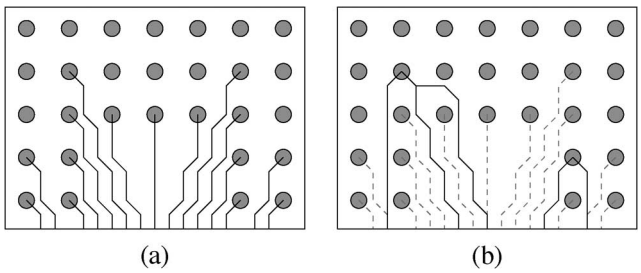


Fig. 14. (a) Escape-routing solution that maximizes the resource utilization. (b) Patterns generated for two pins using the previous solution as baseline. Patterns generated for other pins are not shown here for clarity.

TABLE I
COMPARISON OF OUR FRAMEWORK WITH A RECENTLY PROPOSED APPROACH

			OUR ALGORITHM		ALGORITHM IN [1]	
Input	# layers	# nets	nonplanar nets	time (m:s)	nonplanar nets	time (m:s)
IBM1	5	426	25	1:56	50	0:37
IBM2	5	428	35	0:58	44	0:30
IBM3	4	352	22	1:04	48	0:33
IBM4	5	312	47	1:22	68	0:54
IBM5	3	226	6	0:25	18	0:18
IBM6	5	441	35	1:01	50	0:32

layers. If the escape routing for pin p is performed on layer ℓ, then the blind via corresponding to this pin will be between layer ℓ and the top layer. Therefore, the area corresponding to pin p below layer ℓ will be available for routing other pins. Inherently, our algorithms support this scenario, mainly because we perform escape routing one layer at a time. Once escape routing for layer ℓ is completed, we can simply update the pin structures and the component boundaries for layer $\ell + 1$. The basic idea of generating escape patterns in two components and then choosing the maximal planar subset is still applicable here. However, the pattern-generation algorithms may need to be modified to adapt to the technology being used, since the extra routing resources below blind vias open up different possibilities for escape routing, as shown in Fig. 14(a).

In fact, it is still possible to use the pattern-generation algorithms proposed in Section III in the presence of blind vias since our algorithms do not assume that the component pins have uniform grid structures. However, if the main objective is to maximize resource utilization on every escape layer, instead of minimizing the number of vias, a slightly different methodology can be adapted. First, an escape-routing algorithm that is best for the underlying technology (such as [20]) can be used to find the routing solution that gives the best resource utilization, as shown in Fig. 14(a). Then, we can use this as our baseline, and we can generate alternative escape patterns for each pin, using the algorithms proposed in Figs. 4 and 5. As long as Properties 3.1 and 3.2 are satisfied, we can use our maximal planar-pattern-selection algorithm proposed in Section III. Fig. 14(b) shows how to generate alternative routing patterns for two pins, given the baseline escape routing of Fig. 14(a). In this figure, three alternative escape patterns are generated for the pin on the left and two alternative escape patterns are generated for the pin on the right. The objective here is to generate escape patterns that have the minimum number of conflicts with the baseline solution, while providing alternative orderings on the boundary. The cost metrics of our pattern-generation algorithms can be modified in a straightforward way to handle this objective.

VII. EXPERIMENTAL RESULTS

We have performed experiments on the escape problems extracted from a real industrial-board design, for which current industrial tools fail to produce a routing solution. We have implemented our algorithms in C++ and performed the experiments on an Intel Pentium-4 2.4-GHz system with 1-GB memory and a Linux operating system. The input parameter V given in Properties 3.1 and 3.2 is set to four in our experiments.

As mentioned in Section IV-A, the optimal algorithm for maximal planar-route selection has a high time complexity, and it is impractical for large circuits. Our experiments on relatively small-sized problems indicated that the solutions given by our randomized algorithm are within about 3% of the optimal solution in terms of the number of planar routes selected. However, the optimal algorithm was not applicable to larger problems due to its large time complexity. For this reason, we have used the algorithm proposed in [1] for comparison with the algorithm that we propose in this paper. Table I gives the results obtained on industrial test cases. As mentioned before, layers are processed one by one, and the maximal planar-routing solution is found for each layer. The number of nets that could not be routed in a planar fashion is given for each problem under the columns nonplanar nets. These nets will be distributed to the available layers later, allowing crossings in the intermediate channel. As discussed before, a crossing net will need to use a via during the later stages of the routing system. The results in Table I indicate that our algorithm reduces the via requirements on the average by 39%, for the given industrial test cases. Note that the CPU times have been collected using the same computer system for both algorithms. Due to the more

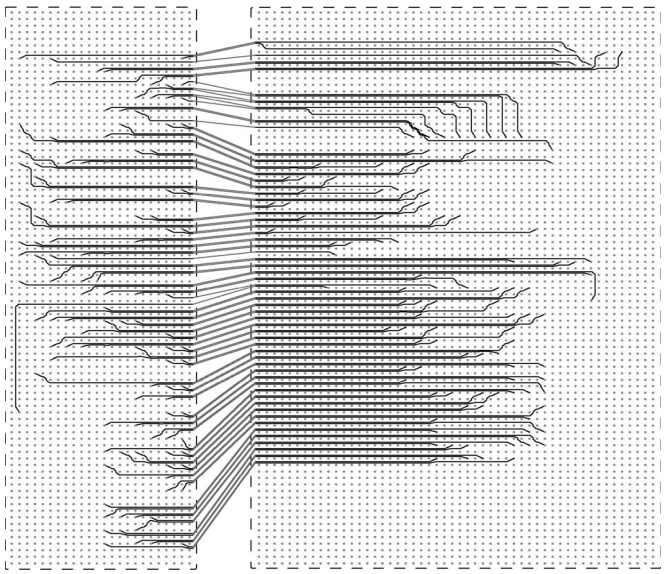


Fig. 15. Planar-escape-routing solution is illustrated for two components. The 111 nets have been routed on this layer. The connections in the intermediate area are shown as straight lines between components.

aggressive optimization, the execution times of our algorithms are longer than [1]. However, the significant reduction in the number of vias justifies this runtime increase. A sample output of our maximal planar-routing algorithm for one layer is shown in Fig. 15.

VIII. Conclusion

We have proposed an algorithm to solve the escape-routing problem in multiple components simultaneously. Compared to the algorithm proposed in [1], our main contributions can be summarized as follows. First, we propose a more intelligent pattern-generation algorithm that is based on congestion levels in the components and the number of crossings in the intermediate area. Then, we propose a more sophisticated randomized algorithm for the maximal planar-routing problem. We also show how to handle typical high-speed-design constraints within the framework of this algorithm. Our experiments show that our algorithm can reduce the via requirements significantly.

References

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

[2] W. D. Brown, *Advanced Electronic Packaging With Emphasis on Multichip Modules*. Piscataway, NJ: IEEE Press, 1999. IEEE Press Series on Microelectronic Systems.

[3] E. H. Laine and P. M. O'Leary, "IBM chip packaging roadmap," in *Proc. Int. Packag. Strategy Symp.*, 1999.

[4] H. Harrer, H. Pross, T.-M. Winkel, W. D. Becker, H. I. Stoller, M. Yamamoto, S. Abe, B. J. Chamberlin, and G. A. Katopis, "First- and second-level packaging for the IBM eServer z900," *IBM J. Res. Develop.*, vol. 46, no. 4/5, pp. 397–420, 2002.

[5] T.-M. Winkel, W. D. Becker, H. Harrer, H. Pross, D. Kaller, B. Garben, B. J. Chamberlin, and S. A. Kuppinger, "First- and second-level packaging of the z990 processor cage," *IBM J. Res. Develop.*, vol. 48, no. 3/4, pp. 379–394, May 2004.

[6] J. Ludwig, 2004, IBM Systems Group. Private communication.

[7] W.-T. Chan, F. Y. L. Chin, and H.-F. Ting, "Escaping a grid by edge-disjoint paths," in *Proc. 11th Annu. ACM-SIAM Symp. Discr. Algorithms*, 2000, pp. 726–734.

[8] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge, MA: MIT Press, 1992.

[9] J. Hershberger and S. Suri, "Efficient breakout routing in printed circuit boards," in *Proc. 13th Annu. Symp. Comput. Geom.*, 1997, pp. 460–462.

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

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

[12] M. M. Ozdal and M. D. F. Wong, "Length matching routing for high-speed printed circuit boards," in *Proc. IEEE/ACM ICCAD*, Nov. 2003, pp. 394–400.

[13] M. M. Ozdal and M. D. F. Wong, "A provably good algorithm for high performance bus routing," in *Proc. IEEE/ACM ICCAD*, Nov. 2004, pp. 830–837.

[14] M. M. Ozdal and M. D. F. Wong, "A two-layer bus routing algorithm for high-speed boards," in *Proc. IEEE ICCD*, Oct. 2004, pp. 99–105.

[15] M. M. Ozdal and M. D. F. Wong, "A length-matching routing algorithm for high-performance printed circuit boards," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 12, pp. 2784–2794, Dec. 2006.

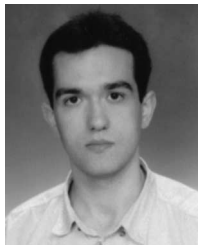
[16] D. Staepelaere, J. Jue, T. Dayan, and W. W.-M. Dai, "SURF: Rubber-band routing system for multichip modules," *IEEE Des. Test Comput.*, vol. 10, no. 4, pp. 18–26, Dec. 1993.

[17] T. Cohen, "Practical guidelines for the implementation of back drilling plated through hole vias in multi-gigabit board applications," in *Proc. IEC DesignCon*, 2003, pp. 1–10.

[18] J. S. Corbin, C. N. Ramirez, and D. E. Massey, "Land grid array sockets for server applications," *IBM J. Res. Develop.*, vol. 46, no. 6, pp. 763–778, Nov. 2002.

[19] G. Capwell, "High density design with MicroStar BGAs," Texas Instrum., Application Rep. SPRA471A, 1998.

[20] R. Shi and C.-K. Cheng, "Efficient escape routing for hexagonal array of high density I/Os," in *Proc. Des. Autom. Conf.*, 2006, pp. 1003–1008.



Muhammet Mustafa Ozdal received the B.S. degree in electrical engineering and the M.S. degree in computer engineering from Bilkent University, Ankara, Turkey, in 1999 and 2001, respectively. He received the Ph.D. degree in computer science from the University of Illinois, Urbana, in 2005.

He is currently with the Design and Technology Solutions Department, Intel Corporation, Hillsboro, OR. His current research interests include algorithms for computer-aided design of VLSI circuits, with a primary focus on physical design algorithms.

Martin D. F. Wong (F'06) received the B.Sc. degree in mathematics from the University of Toronto, Toronto, ON, Canada in 1979, and the M.S. degree in mathematics and the Ph.D. degree in computer science from the University of Illinois (UIUC), Urbana, in 1981 and 1987, respectively.

He is currently a Professor of Electrical and Computer Engineering with the UIUC. Before he was with UIUC, he was a Bruton Centennial Professor of Computer Sciences with the University of Texas (UT-Austin), Austin, TX. His research interests are computer-aided design (CAD) of very large scale integrated (VLSI) circuits, design and analysis of algorithms, and combinatorial optimization. He has published over 300 technical papers and has graduated 34 Ph.D. students.

Dr. Wong received the 2000 IEEE CAD Transactions Best Paper Award for his work on interconnect optimization. He also received best paper awards at DAC-86 and ICCD-95 for his work on floorplan design and routing, respectively. His ICCAD-94 paper on circuit partitioning has been included in the book *The Best of ICCAD—20 Years of Excellence in Computer Aided Design* (Kluwer, 2003). He has served as the Technical Program Chair, the General Chair, and a Steering Committee member for the annual ACM International Symposium on Physical Design. He has also served on the technical program committees of numerous VLSI/CAD conferences. He has served as an Associate Editor for IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN (2002–2005) and IEEE TRANSACTIONS ON COMPUTERS (1995–2000). He has also served as a Guest Editor of four special issues for IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN. He is currently on the Editorial Board of ACM Transactions on Design Automation of Electronic Systems. He is an IEEE Distinguished Lecturer (2005–2006).

Philip S. Honsinger received the Ph.D. degree in theoretical particle physics from Ohio University, Athens, and the M.S. degree in computer science from Ohio State University, Columbus.

After several years as an Assistant Professor with the West Virginia Weslyan College, Buckhannon, he came to IBM Corporation, Hopewell Junction, NY. During his 25 years at IBM Corporation, he has been focused on the physical design of chips and on packaging. Currently, he is working on package routing.