Algorithms for Simultaneous Escape Routing and Layer Assignment of Dense PCBs

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

***Abstract*—As die sizes are shrinking, and circuit complexities are increasing, the printed circuit board routing problem becomes more and more challenging. Traditional routing algorithms cannot handle these challenges effectively, and many high-end designs in the industry require manual routing efforts. This paper proposes a problem decomposition that distinguishes routing under dense components from routing in the intermediate area. In particular, it proposes an effective methodology to find the escape routing solution for multiple components simultaneously such that the number of crossings in the intermediate area is minimized. For this, the problem is modeled as a longest path with forbidden pairs problem, and two algorithms are proposed for it. The first is an exact polynomial-time algorithm that is guaranteed to find the maximal planar routing solution on one layer. The second is a randomized algorithm that has good scalability characteristics for large circuits. Then, these algorithms are used to assign the maximal subset of planar nets to each layer, and then the remaining nets are distributed at the end. This paper demonstrates the effectiveness of these algorithms through experiments on industrial circuits.**

***Index Terms*—Algorithms, escape routing, layer assignment, longest path with forbidden pairs, printed circuit board.**

I. Introduction

DURING the past several years, we have seen dramatic advances in the IC technology. The shrinkage of die sizes and the increase in functional complexities made the circuits become more and more dense. So, boards and packages have reduced in size, while the pin counts have been increasing. For example, a multichip module (MCM) used in IBM eServer z900 [10] (introduced in 2000), contains 20 processor chips, eight L2 cache chips, two system control chips, four memory bus adapter chips, and a clock chip—a total of 35 chips in one package. On the bottom of this MCM, there are 4224 I/O pins, within an area of 127 mm × 127 mm. In the subsequent generation of the same series, IBM eServer z990 [16] (introduced in 2003), the corresponding number of pins in an MCM has increased about 20%, with a decrease of almost 50% in the substrate area. With 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 [16]. Traditional board routing algorithms cannot handle designs with such complexities, and many high-end boards in the industry today require manual efforts for routing. In a typical design cycle of a high-end board, manual routing efforts take about a month [12], and new effective routing tools are necessary to significantly reduce this time. In this paper, we focus on board-level routing beneath dense components, and we propose algorithms that address these challenges effectively.

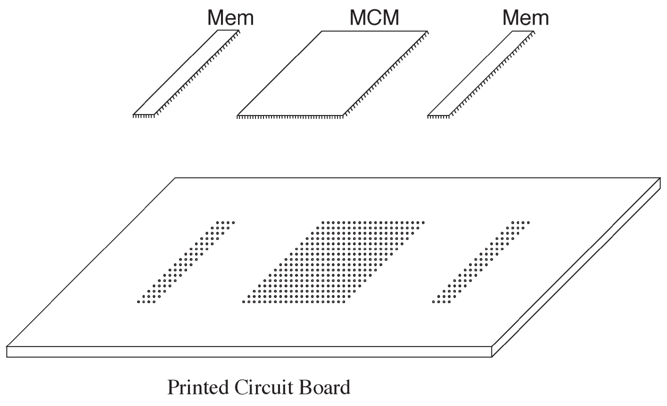


Fig. 1. Different components are mounted on or plugged into a PCB. A pin array is created on the board corresponding to each component.

A typical printed circuit board (PCB) contains a number of different components such as MCMs, memory, or I/O modules. These components are mounted on or plugged in to the board, forming a set of dense pin arrays, as shown in Fig. 1. In this paper, we will focus on the type of board designs where each component pin is accessible from all layers, as will be discussed in detail below. The routing resources within such pin arrays are extremely limited due to the large number of pins, and tight clearance rules. Furthermore, there are large number of nets that need to be routed from their terminal pins to the corresponding component boundaries. On the other hand, the intermediate routing area on the board between different components has relatively few blockages, and the amount of available routing resources is relatively larger.

In accordance with this characteristics, we propose a problem decomposition that handles routing within dense pin arrays separately from the intermediate area routing. In other words, two separate problems are distinguished here: 1) routing nets from pin terminals to component boundaries (escape routing) and 2) routing nets between component boundaries (area routing). In this paper, we propose algorithms to handle the problem of escape routing.

It is important here to note that the escape routing problem for different components should not be considered independent

Manuscript received June 30, 2004; revised February 5, 2005. The preliminary version of this paper was presented in IEEE/ACM International Conference on Computer Aided Design (ICCAD) in November 2004. This work was supported in part by the National Science Foundation under Grants CCR-0244236 and CCR-0306244, and in part by an IBM Faculty Partnership Award. This paper was recommended by Associate Editor T. Yoshimura.

M. M. Ozdal is with the Intel Corporation, Hillsboro, OR 97124 USA.

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

Digital Object Identifier 10.1109/TCAD.2005.857376

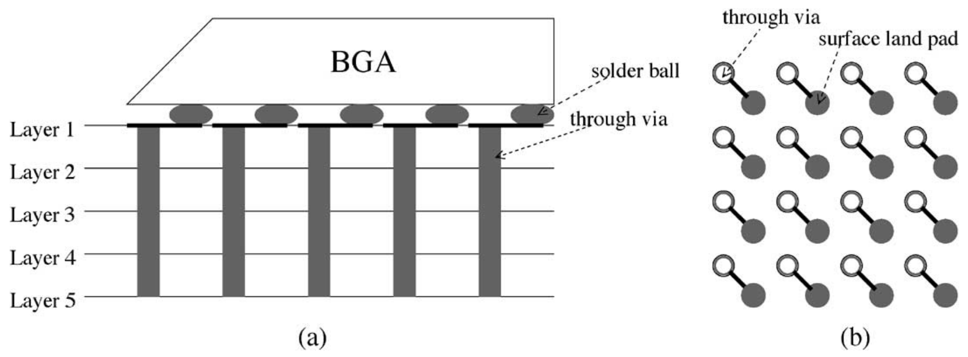


Fig. 2. BGA package is mounted on a PCB, and through vias are used to connect component pins to inner layers of the board: (a) Cross-sectional view. (b) Top-side view. In the context of board-level routing, each through via here will be regarded as a component pin.

of each other. That is, we cannot just apply a conventional escape routing algorithm [7] on different components independently. The reason is that such an approach would ignore the connections between different components, and would make the next phase (area routing) more difficult. Instead, we propose algorithms to find the escape routing solutions of different components simultaneously such that the number of crossings in the intermediate area is minimized. For multilayer designs, the best layer assignment also needs to be determined during this process. Fig. 3 illustrates a sample problem, and Fig. 4 gives a two-layer solution.

Since the routing resources inside dense components are extremely limited, we assume that via usage is not allowed within components. So, the escape routing solution has to be conflict-free within components on every layer. On the other hand, via usage is possible in the intermediate areas, where there are relatively few routing blockages. However, since vias adversely affect routability and signal delay characteristics, and they lower manufacturing yield, we try to minimize the number of vias through crossing minimization. So, our objective is to find the best conflict-free escape routing solution inside components that will minimize the number of crossings in the intermediate area.

Note here that the exact routing solution for the intermediate area will be determined by the next stage (i.e., area routing stage) of the routing system. Additional requirements (such as length matching for high-speed designs) can be handled during that stage, since there are more routing resources available in the intermediate area. On the other hand, we mainly focus on the objective of routability inside the dense pin arrays, because the scarcity of routing resources does not allow us to perform additional tasks such as length extension for length matching, as in [13]–[15].

As mentioned before, for the models and algorithms we propose in this paper, we assume that each component pin can be accessed from every layer of the input PCB. For example in IBM eServer z900 [10], 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) [4] (a.k.a 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 (LGA) connectors [6], which are mounted on the board surface, and connected to PTHs on the board. Similar statements can be made for a ball grid array (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 [3], as illustrated in Fig. 2. In these cases, we will regard each such through via as a component pin in the context of board-level routing. We will not go into further details of these issues in this paper; instead we will focus on escape problem from the perspective of board-level routing where each component pin is accessible from every layer, either directly or by through vias.

The rest of the paper is organized as follows. In Section II, we give a formal description of this problem, and discuss how it relates to the existing work in the literature. Then, we outline our solution approach in Section III. Mainly, we process one layer at a time, and try to route as many noncrossing nets as possible on each layer. In Section IV-A, we model the maximal planar routing problem as a longest path with forbidden pairs (LPFP) problem, and propose an efficient checkerboard-based graph model for it in Section IV-B. Although the general LPFP problem is NP-complete, the special structure of our problem allows us to propose a polynomial-time exact algorithm in Section IV-C. Then, we propose a fast and effective randomized algorithm in Section IV-D for large circuits. In Section V, we discuss generalizations of our models and algorithms. Finally, we demonstrate the effectiveness of our algorithms through experiments in Section VI.

II. Problem Formulation and Related Work

Let a component be defined as a two-dimensional (2-D) array of pins that span multiple layers. The input circuit is assumed to contain two components separated by a channel between them. A two-terminal net specifies two pins as its endpoints, which are assumed to be in different components by definition. For simplicity of presentation, we will assume that only one net can be routed between adjacent rows and columns of component pins.[1] An escape route for a given net is defined as the route from one

[1]In Section V-B, we will discuss how to handle components with multiple routing tracks between adjacent pins.

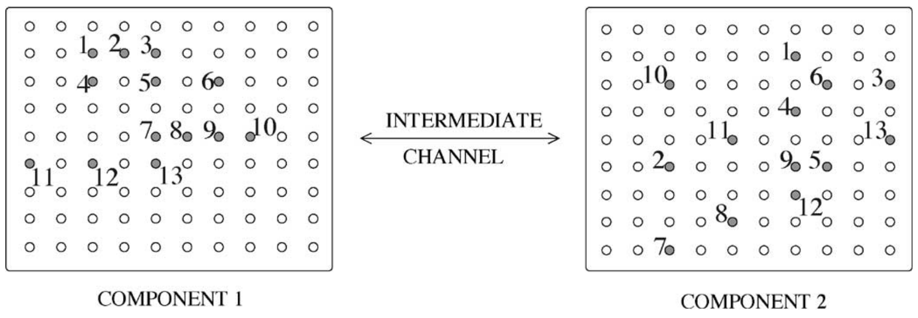


Fig. 3. Sample escape problem with 13 nets on two components. Each terminal pin is labeled with its net index. The problem is to find a conflict-free routing solution within components, and to minimize crossings in the channel.

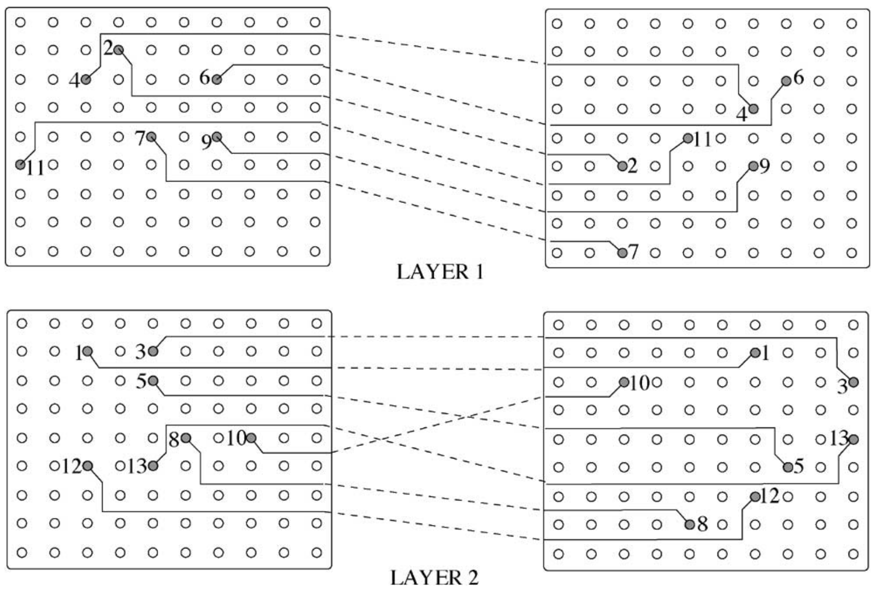


Fig. 4. Sample solution for the problem given in Fig. 3. Escape routes are illustrated with solid lines within components. Channel segments are shown with dashed lines.

of its terminal pins to the respective component boundary. Two escape routes r_i and r_j within the same component are defined to conflict with each other iff r_i and r_j cannot exist together in a permissible one-layer planar routing solution. Given an input circuit and a set of two-terminal nets, the problem is to find an escape routing solution for each net, and assign them to different layers such that: 1) conflict-free routing solution is obtained within each component at every layer and 2) the number of crossings in the intermediate channel is minimized. Here, routing conflicts are not allowed inside the components, because routing resources within components are too scarce to allow via usage. On the other hand, via usage is allowed in the intermediate channel between components; hence crossings are allowed here. However, since vias have adverse affects on routability and signal delay characteristics, and they lower manufacturing yield, our objective is to minimize the number of vias through crossing minimization.

Fig. 3 illustrates a sample escape problem with 13 nets in two components, and Fig. 4 gives a two-layer solution. As mentioned earlier, it is assumed that each pin spans multiple layers; so it is possible to assign the route for each net to any layer. In the given solution, six nets are routed on layer 1 without any crossings in the channel. On the other hand, the channel segment of one net (net 10) on layer 2 crosses with others. This crossing can be avoided in the later stages of the routing system by using a via for only net 10. So, we can state that the escape routing solution given in Fig. 4 helps the objective of via minimization since it minimizes the crossings in the channel.

A related problem in the literature is the pin assignment problem [2], [11], [17]. Its objective is to determine the positions of pins on chip boundaries such that a cost function is minimized. However, this problem ignores escape routing inside the components. Another related problem is the k-layer topological via minimization problem [5], where the objective is to determine the topological routing of a set of nets on k routing layers such that the total number of vias is minimized. It has been shown that the general case of this problem is NP-complete, and an algorithm has been proposed for the case of a crossing channel, where nets have fixed pin positions on chip boundaries [5]. However, escape routing is not considered also in this problem. On the other hand in our problem, we need to find the escape routes simultaneously while assigning nets to different layers for via minimization. In other words, the pin positions of nets are not fixed on component boundaries, but they are determined based on the escape routes. For instance in

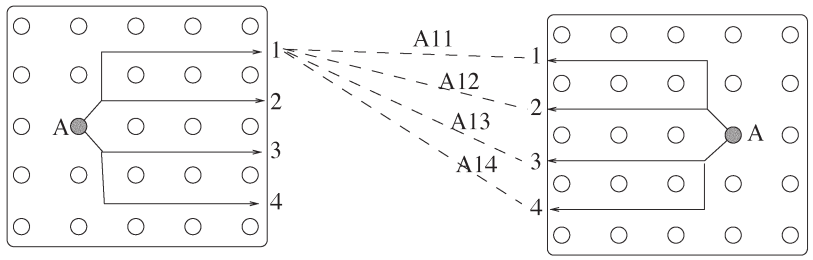


Fig. 5. Routing patterns considered for net A. Only four out of 16 patterns are shown here for clarity.

the example of Fig. 4, the ordering of nets within components is not necessarily the same as the ordering on component boundaries,[2] since this ordering further reduces the number of crossings.

III. Methodology

We use a two-phase approach for this problem: 1) for each layer l, pack as many noncrossing routes as possible on l and 2) distribute the remaining nets to available layers, this time allowing crossings in the intermediate channel.

In the first phase, we process one layer at a time, and try to find the maximum subset of available nets that can be routed without any crossings on that layer. The first layer in Fig. 4 is an example output of this phase. Specifically, the maximum noncrossing subsets for layer 1 and layer 2 have been found to be {2,4,6,7,9,11}, and {1,3,5,8,12,13}, respectively, for this problem. The details of the algorithm we propose for this phase are presented in Section IV.

Then, in the second phase, the nets that have not been routed are distributed to available layers. In our sample problem, net 10 does not belong to any of the planar subsets of phase 1. So, an escape routing solution is found for it in the second phase on layer 2. Observe in Fig. 4 that although it has a conflict-free routing solution within the components, it crosses with nets 5 and 13 in the channel.

For the second phase, we use a negotiated congestion-based net-by-net approach similar to Pathfinder [9]. The main idea is to allow routing conflicts in the beginning, and then to iteratively rip up and reroute nets, while gradually increasing the costs of conflicted routing resources. By doing so, nets with alternative routes are forced not to use the conflicted resources, and eventually a conflict-free routing solution is obtained. Note here that we discourage ripping up the nets routed in the first phase by using relatively higher costs for conflicts with these nets.

IV. Maximal Planar Routing

A. Algorithm Outline

Given a set of nets, our objective is to find the maximum subset that can be routed on one layer without any conflicts. For this purpose, we define a number of routing patterns for each net, and we propose algorithms to choose the best possible combination of these patterns. For simplicity of presentation, we will focus on a horizontal problem, where one component is to the right of another. It is straightforward to extend the algorithm to a vertical problem.

Our main assumption in the following algorithm is that the vertical span of escape routes within components will be limited in a typical solution, as in Fig. 4, where an escape route spans at most two rows. The main reason is that large vertical spans within components block other escape routes; so we need small vertical spans for maximal routing. Furthermore, we have observed this behavior for a great majority of nets in typical manual industrial solutions. Based on this, we define 16 possible configurations for each net,[3] as shown in Fig. 5. Namely, we consider four escape routes for a net within each component, so that it can escape from one of the four neighboring rows of its terminal pin. Note that any one of the four escape routes within each component can be selected, and so there are $4 \times 4 = 16$ possible routing patterns for each net. Let A_{ij} denote the configuration where net A escapes to row i in the first component, and to row j in the second component. In Fig. 5, some sample routing patterns are illustrated. As seen in this figure, we consider only simple escape routes, each of which has a single horizontal segment. This assumption can easily be generalized for more general patterns, as will be discussed in Section V-A.

Now, the problem can be stated as to select the maximum subset of patterns for a given set of nets such that:

1) at most one pattern is selected for each net;
2) there are no conflicts within components; and
3) there are no crossings in the channel.

Note that even though we consider only a limited number of routing patterns for each net, there are exponential number of possible ways of selecting patterns for a set of nets. However, we will propose a polynomial-time algorithm to select the best combination that gives the maximal planar routing solution.

If every net had only one possible routing pattern (instead of 16), and if there were no conflicts between different nets within components, then we could use a longest path algorithm to find the maximal subset of noncrossing nets [5]. However, we have to consider escape routes within components, and try to find the best possible escape route for each net simultaneously while finding the optimal subset of nonconflicting and noncrossing nets. For this purpose, we will define a graph model $\mathcal{G}$, and a

[2]e.g., In the left component of first layer, net 4 escapes to row 1, and net 2 escapes to row 3, although the terminal of net 2 is above net 4 within the component.

[3]In Section V-A, we discuss possible extensions to relax this assumption.

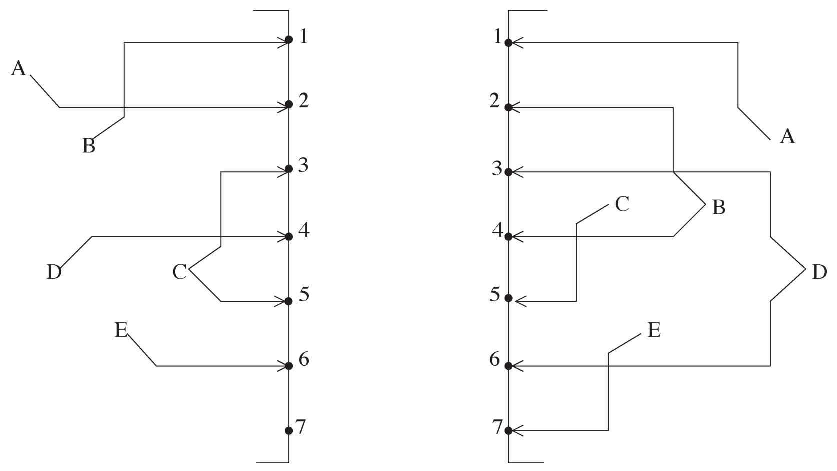


Fig. 6. Sample escape routing problem for five nets. For clarity, only one or two routing patterns are defined for each net (instead of 16 as in the actual algorithm).

set of forbidden pairs $\mathcal{F}$ (such that $\mathcal{F}$ contains pairs of vertices from $\mathcal{G}$) as follows.

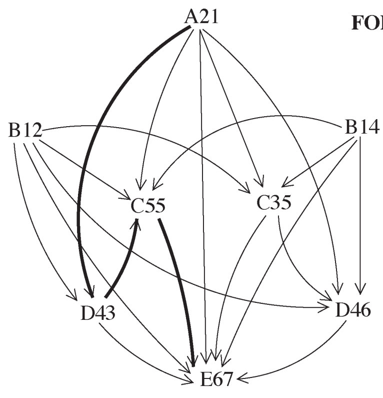
1) For each routing pattern, a vertex exists in $\mathcal{G}$.
2) Let u and v be two vertices in $\mathcal{G}$ corresponding to routing patterns U_{ij} and V_{kl}, respectively.[4] An edge from u to v exists in $\mathcal{G}$ iff the channel segment of U_{ij} is strictly above the channel segment of V_{kl}, i.e., $i < k$ and $j < l$ (e.g., A_{12} in Fig. 5 would be strictly above A_{34}).
3) Let u and v be vertices in $\mathcal{G}$. Forbidden pair (u, v) exists in $\mathcal{F}$ iff at least one the following conditions is the case.
   a) u and v correspond to the same net.
   b) The routing patterns corresponding to u and v conflict with each other (as defined in Section II) in at least one component.

It is straightforward to show that $\mathcal{G}$ is in fact a directed acyclic graph (DAG). We can state that if a path exists from vertex u to vertex v in $\mathcal{G}$, then it is guaranteed that the channel segments of the corresponding routing patterns do not cross with each other. Hence, the longest path in $\mathcal{G}$ will correspond to the maximum set of routing patterns that have no crossings in the channel. However, we also need to consider the conflicts within components, as defined by the forbidden pair set $\mathcal{F}$. The following theorem gives a formal description of this problem.

Theorem 1: The problem of finding the maximum subset of noncrossing and nonconflicting routing patterns is equivalent to the longest path with forbidden pairs (LPFP) problem on $\{\mathcal{G}, \mathcal{F}\}$.

LPFP problem [8] for a graph $\mathcal{G}$, and a vertex pair set $\mathcal{F}$ is defined as finding the longest path P in $\mathcal{G}$ such that P contains at most one vertex from each pair of vertices in $\mathcal{F}$. In other words, if $(u, v) \in \mathcal{F}$, then a permissible path in $\mathcal{G}$ cannot contain both u and v. The general LPFP is known to be an NP-complete problem [1]. However, the following property of our problem will enable us to propose a polynomial time algorithm in Section IV-C.

Lemma 1: For any forbidden vertex pair $(u, v) \in \mathcal{F}$, if v is reachable from u in $\mathcal{G}$, then the maximum path length (in terms of number of edges) from u to v is guaranteed to be less than or equal to three.



FORBIDDEN PAIRS: (A21, B12)
(A21, B14)
(B12, B14)
(B12, D43)
(B14, C35)
(B14, C55)
(C35, C55)
(C35, D43)
(C35, D46)
(D43, D46)
(D46, E67)

Fig. 7. Graph model corresponding to the problem given in Fig. 6. The longest path without forbidden pairs is illustrated with the thick lines.

Proof: An edge from w to t exists only if the corresponding routing pattern of t escapes to rows strictly below those of w (by definition). Furthermore, the vertical spans of routing patterns are limited. Hence, if u and v conflict with each other within a component, then this means that their escape routes are on *nearby* rows. It is possible to show by case-by-case analysis that u and v cannot escape to rows separated by more than three rows if $(u, v) \in \mathcal{F}$. So, the maximum path length between conflicting vertices in $\mathcal{G}$ can be at most three. ■

Fig. 6 gives a sample problem with a limited number of patterns defined for each net.[5] The graph model corresponding to these patterns is illustrated in Fig. 7. Observe that the longest path without forbidden pairs on this graph is given as $A_{21} \rightarrow D_{43} \rightarrow C_{55} \rightarrow E_{67}$. The actual solution corresponding to this path is also shown in Fig. 8.

B. Checkerboard Graph Model

In the graph model described in Section IV-A, an edge exists from vertex u to every vertex v of which channel segment is strictly below u. So, the number of edges in $\mathcal{G}$ is $O(n^2)$, where

[4]As before, U_{ij} denotes the routing pattern where net U escapes to row i in the first component, and row j in the second component.

[5]Only one or two patterns are defined for each net for clarity of the figure. In our actual algorithm, there are 16 patterns defined for each net.

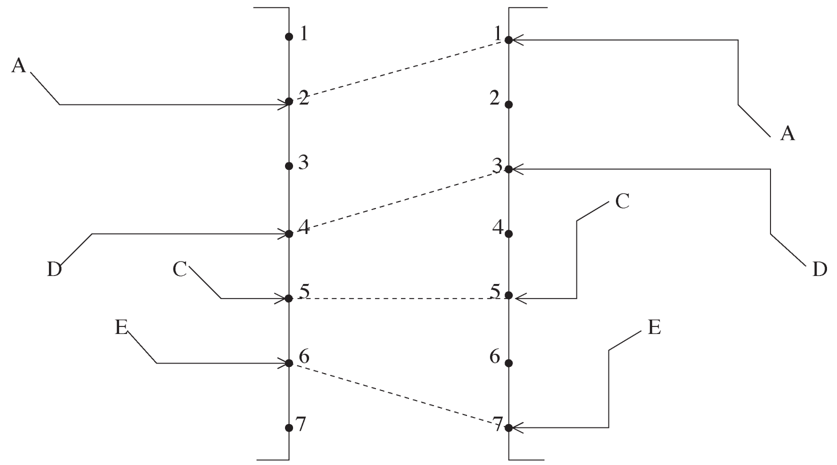


Fig. 8. Actual maximal planar routing solution corresponding to the path given in Fig. 7.

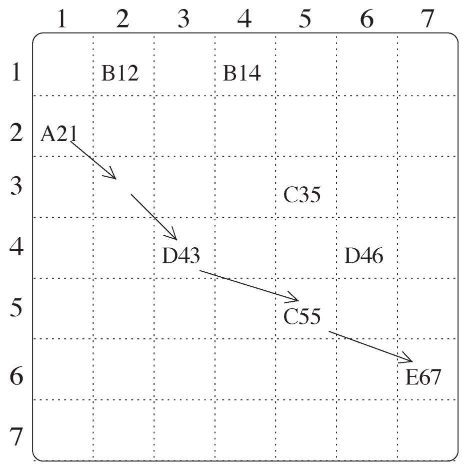


Fig. 9. Checkerboard structure corresponding to the graph of Fig. 7. For clarity, only the edges on the longest path are illustrated.

n is the number of nets. In this section, we will describe a more structured graph model with less number of nets.

Let us consider a (conceptual) checkerboard structure with size $r \times r$, where r is the number of rows in a component. As before, let A_{ij} denote the routing pattern where net A escapes to row i in the first component, and to row j in the second component. The main idea here is to (conceptually) assign each routing pattern A_{ij} to cell (i, j) of the checkerboard, as shown in Fig. 9. We can formally define a graph model $\mathcal{G}_{\mathrm{C}}$ based on this conceptual structure as follows.

1) For each cell (i, j) of the checkerboard, a vertex e_{ij} with zero weight exists in $\mathcal{G}_{\mathrm{C}}$.
2) For each routing pattern U_{ij}, a vertex u_{ij} with unit weight exists in $\mathcal{G}_{\mathrm{C}}$.
3) Let u_{ij} and v_{kl} be vertices in $\mathcal{G}_{\mathrm{C}}$. An edge from u to v exists in $\mathcal{G}_{\mathrm{C}}$ iff ($k = i + 1$ AND $l > j$) OR ($l = j + 1$ AND $k > i$). In other words, an edge exists only between adjacent rows or adjacent columns of the checkerboard, and the direction is always towards southeast.

Fig. 9 shows the checkerboard structure corresponding to the graph given in Fig. 7. For clarity, the vertices with zero weights, and the edges between adjacent rows and columns are omitted in this figure. The corresponding longest path without forbidden pairs is also illustrated here. Observe that this path traverses the empty cell (3,2) on the checkerboard in addition to the selected routing patterns. Intuitively, this empty cell corresponds to the unused connection from row 3 to row 2 of the channel illustrated in Fig. 8.

This graph structure is in fact very similar to the one proposed in Section IV-A. The main difference is that edges exist only between neighboring routing patterns here, which brings an asymptotic reduction in the problem complexity. For the following analysis, let r and c denote the number of rows and columns of the components, respectively; let s denote the size of the components (i.e., $s = rc$); and let n denote the number of nets. Furthermore, let us assume that the components have constant aspect ratios, i.e., $r = \Theta(c) = \Theta(s^{1/2})$.

Lemma 2: The total number of vertices in $\mathcal{G}_{\mathrm{C}}$ that are assigned to row i of the checkerboard is $O(s^{1/2})$; similarly, the number of vertices assigned to column j is $O(s^{1/2})$, where $1 \leq i, j \leq r$.

Proof: Remember that each net has a constant number of routing patterns, and each routing pattern has a limited (constant) vertical span (as shown in Fig. 5). So, the total number of nets escaping to row k of a component is $O(c)$. Furthermore, the routing patterns assigned to row i of the checkerboard are the ones escaping to row i of the first component, by definition. Similarly, the patterns assigned to column j of the checkerboard are the ones escaping to row j of the second component. In addition, each row and column of the checkerboard contains $O(r)$ zero-weighted vertices, corresponding to the cells of the checkerboard. Hence, the total number of vertices assigned to any row or column of the checkerboard is $O(r + c) = O(s^{1/2})$. ■

Lemma 3: The number of vertices in $\mathcal{G}_{\mathrm{C}}$ is $O(n + s)$, and the number of edges is $O(ns^{1/2} + s^{3/2})$.

Proof: For each net, a constant number of routing patterns are defined, and there is a zero-weighted vertex corresponding to each checkerboard cell. Hence, the number of vertices in $\mathcal{G}_{\mathrm{C}}$ is $O(n + s)$. The edges are only between adjacent rows and columns of the checkerboard structure; so each vertex has $O(s^{1/2})$ incoming edges (due to Lemma 2). As a result, the number of edges in $\mathcal{G}_{\mathrm{C}}$ is $O(n + s) \times O(s^{1/2}) = O(ns^{1/2} + s^{3/2})$. ■

Assuming that the component pins are densely populated [i.e., $n = \Theta(s)$], the number of edges in graph $\mathcal{G}_C$ is in fact $O(n^{3/2})$. This is an asymptotic reduction in complexity, compared to the graph model described in Section IV-A, which has $O(n^2)$ number of nets. Hence, the checkerboard structure will be helpful to reduce the complexity of the exact algorithm we propose in Section IV-C. Furthermore, the structured view of a checkerboard will help us to propose a very effective randomized algorithm in Section IV-D.

C. Exact Algorithm for LPFP Problem

As mentioned earlier, the exact algorithm is possible due to the special property of the input graph as given in Lemma 1. Our approach will be to perform a graph transformation such that the longest path on the transformed graph will be equivalent to the solution of the LPFP problem on the original graph. This transformation will be described in Definition 3; however to give an intuition about this process, we will first describe simpler versions of this transformation in Definitions 1 and 2.

The notations we will use in this section are as follows: The input problem is given in the form $\{\mathcal{G}, \mathcal{F}\}$, where $\mathcal{G}$ is a DAG, and $\mathcal{F}$ is the set containing forbidden vertex pairs. Consider two vertices u and v in $\mathcal{G}$. We denote u as a parent of v if there is an edge $u \rightarrow v$ in $\mathcal{G}$. On the other hand, u is denoted as a grandparent of v if there is a vertex w such that the edges $u \rightarrow w$ and $w \rightarrow v$ exist in $\mathcal{G}$. For consistency, we assume that each vertex has a parent-grandparent pair of NULL-NULL.

Definition 1: First-order transformation of $\mathcal{G}$ (denoted as $\mathcal{G}^1$) is defined as follows.

1) For each vertex u in $\mathcal{G}$, there is a vertex u' in $\mathcal{G}^1$.
2) There exists an edge $u' \rightarrow v'$ in $\mathcal{G}^1$ iff:
    a) the edge $u \rightarrow v$ exists in $\mathcal{G}$; and
    b) (u, v) is not a forbidden pair.

Remark 1: If the maximum path length between any forbidden pair (u, v) in $\mathcal{G}$ is at most 1, then the longest path in $\mathcal{G}^1$ is the exact solution to LPFP problem in $\mathcal{G}$.

Definition 2: Second-order transformation of $\mathcal{G}$ (denoted as $\mathcal{G}^2$) is defined as follows.

1) For each vertex u in $\mathcal{G}$, there is a set of vertices U in $\mathcal{G}^2$ such that $U[i]$ corresponds to the ith parent of u. In other words, number of vertices in U is equal to the number of parents of u.
2) There exists an edge from $U[i]$ to $V[j]$ in $\mathcal{G}^2$ iff:
    a) u is the jth parent of v in $\mathcal{G}$;
    b) (u, v) is not a forbidden pair; and
    c) (ith-parent-of-u, v) is not a forbidden pair.

As an example, consider graph $\mathcal{G}$ with forbidden pairs in Fig. 10. Second-order transformation of this graph is shown in Fig. 11. Observe that there is a group of vertices in the transformed graph corresponding to each vertex in $\mathcal{G}$. For instance, there is set D, containing four vertices in Fig. 11, corresponding to vertex d in $\mathcal{G}$. Here, each vertex in set D corresponds to one parent of d, and it is connected to that parent if they are not forbidden pairs. As mentioned earlier, we assume that each vertex in $\mathcal{G}$ has a (pseudo) parent of NULL; hence, an extra vertex with no parent is created in each set. For instance, the extra vertex in set D corresponds to the case where the path starts with d in $\mathcal{G}$, i.e., a NULL parent. The following lemma gives the rationale behind this transformation.

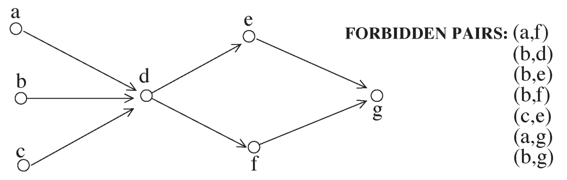


Fig. 10. Sample graph $\mathcal{G}$ and a set of forbidden pairs.

Lemma 4: Consider two vertices w and v in $\mathcal{G}$ such that the maximum path length from w to v is at most two. If (w, v) is a forbidden pair, then there exists no path from vertex set W to vertex set V in $\mathcal{G}^2$.

Proof: If the maximum path length from w to v is one, then the proof is straightforward. Otherwise, consider any path of the form $w \rightarrow u \rightarrow v$. Assume that w is the ith parent of u, and u is the jth parent of v. Due to rule a) in Definition 2, edges from vertex set W to vertex set U in $\mathcal{G}^2$ can only be to $U[i]$. Due to rule c), an edge from $U[i]$ to $V[j]$ exists only if (w, v) is not a forbidden pair. Hence, if (w, v) is a forbidden pair, a path from W to V cannot exist. ■

As an example, consider the forbidden pairs (a, f), (b, d), (b, e), (b, f), and (c, e) in Fig. 10, each having a maximum path length of 2 between the pairs. Observe that there is no path in the transformed graph of Fig. 11 between the corresponding set of vertices.

Lemma 5: If there is a path from w to v in $\mathcal{G}$ such that no pair of vertices on the path is a forbidden-pair, then there will be at least one path of the same length in $\mathcal{G}^2$ from vertex set W to vertex set V.

Proof: Since there are no forbidden pairs in the path from w to v in $\mathcal{G}$, only the first rule of Definition 2 will apply, and the proof of the lemma follows directly. ■

Theorem 2: If the maximum path length between any forbidden pair (u, v) in $\mathcal{G}$ is at most 2, then the longest path in $\mathcal{G}^2$ is the exact solution to LPFP problem on $\mathcal{G}$.

Proof: It follows directly from Lemma 4 and 5. ■

Definition 3: Third-order transformation of $\mathcal{G}$ (denoted as $\mathcal{G}^3$) is defined as follows.

1) For each vertex u in $\mathcal{G}$, there is a 2-D array of vertices U in $\mathcal{G}^3$ such that $U[i][j]$ corresponds to the ith parent of u and the jth parent of ith parent of u. In other words, for each parent–grandparent pair of u, there exists a corresponding vertex in set U.
2) There exists an edge between $U[i][j]$ and $V[k][l]$ in $\mathcal{G}^3$ if:
    a) u is the kth parent of v in $\mathcal{G}$;
    b) $l = i$;
    c) (u, v) is not a forbidden pair;
    d) (ith-parent-of-u, v) is not a forbidden pair; and
    e) (jth-parent-of-ith-parent-of-u, v) is not a forbidden pair.

Fig. 12 illustrates the third-order transformation of the graph given in Fig. 10. Here, it is again assumed that the first parent of each vertex is NULL. For instance, $G[2][1]$ (i.e., the first vertex

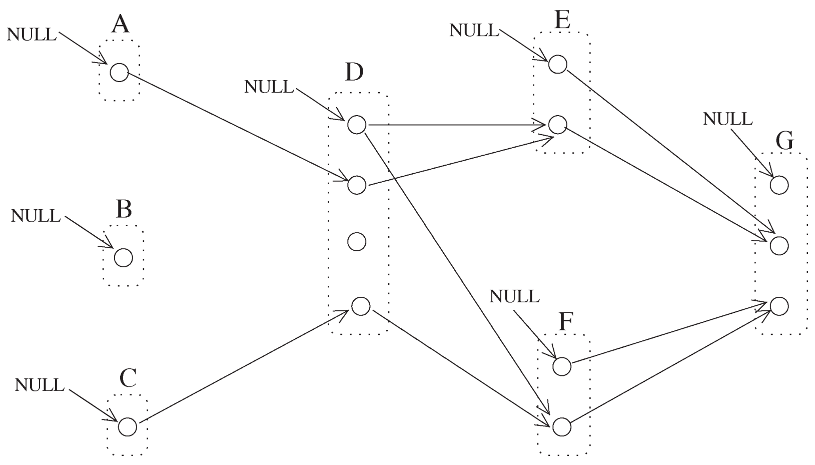


Fig. 11. Second-order transformation of graph $\mathcal{G}$ in Fig. 10. A set of vertices indicated with dotted lines correspond to each vertex of $\mathcal{G}$.

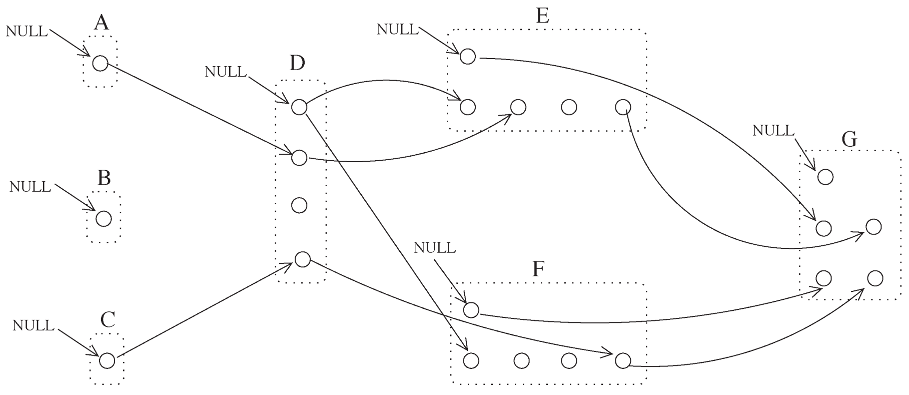


Fig. 12. Third-order transformation of graph $\mathcal{G}$ in Fig. 10. A set of vertices indicated with dotted lines correspond to each vertex of $\mathcal{G}$.

on the second row of vertex set G) corresponds to the vertex pair (e, NULL) in the original graph, since e is the second parent of g, and NULL is the first parent of e.

Lemma 6: Consider two vertices w and v in $\mathcal{G}$ such that the maximum path length from w to v is at most three. If (w, v) is a forbidden pair, then there is no path from vertex set W to vertex set U in $\mathcal{G}^3$.

Proof: If the maximum path length from w to v is 1 or 2, then the proof is similar to that of Lemma 4. Otherwise, consider any path of the form $w \rightarrow y \rightarrow u \rightarrow v$, where w is the jth parent of y, y is the ith parent of u, and u is the kth parent of v. Any edge in $\mathcal{G}^3$ from vertex set W to vertex set Y can only be to $Y[j][\cdot]$ due to rule a) in Definition 3. Similarly, any edge from $Y[j][\cdot]$ to vertex set U can only be to $U[i][j]$ due to rules a) and b). Finally, an edge from $U[i][j]$ to vertex set V exists only if (w, v) is not a forbidden pair, due to rule e). So, a path from w to v cannot exist if w and v conflict with each other. ■

Observe in Fig. 12 that there is no path between vertex sets corresponding to the forbidden pairs in Fig. 10. For example, (a, g) is a forbidden pair, and there is no path between vertex set A and vertex set G in the transformed graph.

Lemma 7: If there is a path from w to v in $\mathcal{G}$ such that no pair of vertices on the path is a forbidden pair, then there will be at least one path of the same length in $\mathcal{G}^3$ from vertex set W to vertex set V.

Proof: Since there are no forbidden pairs in the path from w to v in $\mathcal{G}$, only the first and second rules of Definition 3 will apply, and the proof of the lemma follows directly. ■

Theorem 3: If the maximum path length between any forbidden pair (u, v) in $\mathcal{G}$ is at most three, then the longest path in $\mathcal{G}^3$ is the exact solution to LPFP problem on $\mathcal{G}$.

Proof: It follows directly from Lemma 6 and 7. ■

Let $\mathcal{G}_{\mathrm{C}}$ denote the acyclic checkerboard graph structure described in Section IV-B. Due to Lemma 1, we can apply a third-order transformation on $\mathcal{G}_{\mathrm{C}}$ to obtain $\mathcal{G}_{\mathrm{C}}^3$, and find the exact solution to LPFP problem by using a linear-time longest path algorithm [7] on $\mathcal{G}_{\mathrm{C}}^3$. From Theorem 1, this solution corresponds to the maximal planar routing solution to our original problem.

The following theorem gives the asymptotic time complexity of this algorithm.

Theorem 4: Let s and n denote the size of the components, and the number of nets, respectively. The time complexity of the exact algorithm proposed in this section is $O(ns^{3/2} + s^{5/2})$.

Proof: Each vertex v in $\mathcal{G}_{\mathrm{C}}$ has $O(s^{1/2})$ parents, and $O(s^{1/2})$ grandparents, due to Lemma 2. Since there is a vertex $V[i][j]$ in $\mathcal{G}_{\mathrm{C}}^3$ corresponding to each parent i and grandparent j of v in $\mathcal{G}_{\mathrm{C}}$, the number of vertices in $\mathcal{G}_{\mathrm{C}}^3$ will be $O(s) \times O(n + s) = O(ns + s^2)$. Note here that $O(n + s)$ is the number of vertices in $\mathcal{G}_{\mathrm{C}}$, as stated in Lemma 3. Furthermore, closer

```
RANDOM-LPFP
Define horizontal subproblems with 3 rows on the checkerboard
Randomly generate subpaths $P_j^i$ within each subproblem $i$
Create a graph $\mathcal{G}_R$ as follows:
  –A vertex $v_j^i$ exists in $\mathcal{G}_R$ corresponding to each subpath $P_j^i$
  –Weight of $v_j^i$ is equal to size of $P_j^i$
  –An edge from $v_j^i$ to $v_k^{i+1}$ exists iff:
    (1) $P_k^{i+1}$ is completely to the south-east of $P_j^i$
    (2) The last element of $P_k^{i+1}$ is separated from the last
        element of $P_j^i$ by at least 2 columns
    (3) There exists no forbidden pair $(u, w)$ such that
        $u \in P_j^i$ and $w \in P_k^{i+1}$
Return the longest path in $\mathcal{G}_R$
```

Fig. 13. Randomized algorithm for LPFP problem on a checkerboard graph where the maximum path length between any forbidden pair is at most three.

examination of Definition 3 will reveal that the number of edges entering to each vertex $V[k][l]$ in $\mathcal{G}_C^3$ is $O(s^{1/2})$. Hence, the total number of edges in $\mathcal{G}_C^3$ will be $O(ns^{3/2} + s^{5/2})$. Since the longest path algorithm used on acyclic $\mathcal{G}_C^3$ has linear time complexity in terms of the input graph size, the total time complexity of our algorithm is $O(ns^{3/2} + s^{5/2})$. ■

Although this time complexity is acceptable for moderate component sizes, the algorithm might not be scalable for very large circuits. In the next subsection, we propose a scalable randomized algorithm as an effective alternative for large circuits.

D. Randomized Algorithm for LPFP

As stated by Lemma 1, the vertices that conflict with each other are always close to each other in graph $\mathcal{G}$. Intuitively, if we somehow generate subpaths by grouping the nearby vertices together, then we can obtain a graph where there are no conflicts between groups that are far away from each other. The algorithm we propose in this section makes use of this idea, and uses randomization to group the nearby vertices together, and handle forbidden pairs accordingly.

Fig. 13 gives the outline of the randomized algorithm we propose for the checkerboard graph model described in Section IV-B. The first step here is to define subproblems on the checkerboard structure , as shown in Fig. 15. Then, we randomly generate a predefined number of permissible subpaths for each subproblem. Fig. 14 gives the algorithm we use to generate random subpaths for one subproblem. Observe that for each checkerboard cell C at the last row of a subproblem, we keep the K/r longest subpaths ending at C. Note that our purpose here is not just to find the best possible subpath, but instead to find various (possibly on the order of thousands) good subpaths for each subproblem. After that, we merge them in an optimal way by applying a longest path algorithm on the DAG $\mathcal{G}_R$, which is defined in Fig. 13. The following lemma explains the rationale behind this model.

Lemma 8: Consider two subpaths P_j^i and $P_k^l (i < l)$ in subproblems i and l, respectively. If there is a forbidden pair (u, w) such that $u \in P_j^i$ and $w \in P_k^l$, then there exists no path between the corresponding vertices v_j^i and v_k^l in $\mathcal{G}_R$.

Proof: If $l = i + 1$, this check is done explicitly by rule 3), as given in Fig. 13. Otherwise, assume that $l \geq i + 2$, and

```
GENERATE-SUBPATHS(Subprob $i$: between rows $T_i$ and $B_i$)
for a fixed number of $M$ iterations do:
  $u \leftarrow$ a random vertex at row $T_i$
  $P \leftarrow \{u\}$        // initialize the subpath
  repeat:
    $v \leftarrow$ a random vertex for which edge $u \rightarrow v$ exists,
        and $(w, v)$ is not a forbidden pair for any $w \in P$
    $P = P \cup \{v\}$
    $u \leftarrow v$
  until $v$ is not at row $B_i$
  Let $C$ be the checkerboard cell that contains the last $v$
  Let $\mathcal{P}_C$ denote the set of recorded subpaths ending at $\mathcal{C}$.
  If $|\mathcal{P}_C| < K/r$, where $r$ is the number of component rows
    record $P$
  else if there is a $P' \in \mathcal{P}_C$ such that $P'$ is shorter than $P$
    replace $P'$ with $P$
  else
    discard $P$
```

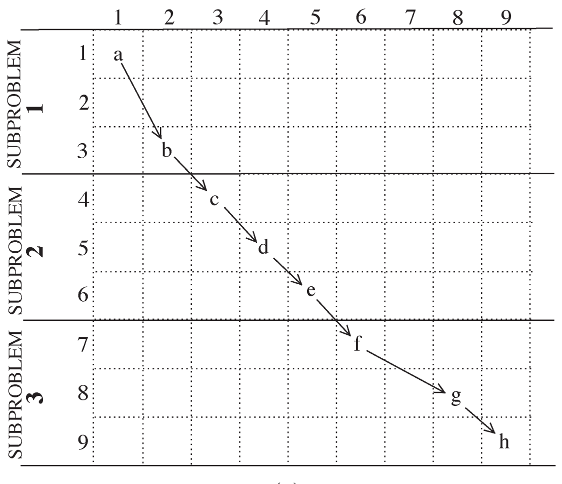
Fig. 14. Algorithm to generate a set of $O(K)$ random subpaths between rows T_i and B_i of the checkerboard.

there is a path from P_j^i to P_k^l in $\mathcal{G}_R$. It is obvious that P_j^i and P_k^l are separated by at least three checkerboard rows, since there is at least one subproblem between them. Furthermore, due to rule 2), there are at least three columns between the last element of P_j^i, and the first element of P_k^l. Since the maximum path length between a forbidden pair can be at most 3 in the original graph (as stated in Lemma 1), there exists no forbidden pair (u, w) such that $u \in P_j^i$ and $w \in P_k^l$. ■

Due to this lemma, we can use a simple longest path algorithm on $\mathcal{G}_R$ without the need of checking forbidden pairs. This longest path will correspond to the optimal combination of subpaths that were randomly generated. If we can generate a large variety of random paths, we can expect the final solution to be sufficiently close to the optimal planar routing solution.

For the complexity analysis of this randomized algorithm, let us first focus on the subpath generation phase given in Fig. 14. In one iteration of this algorithm, a subpath P is generated and evaluated. Generation of one subpath P takes constant time, since P can contain at most three routing patterns, by definition. The evaluation of P can also be performed in constant time by using efficient bucket-based data structure.[6] Since M iterations are performed in Fig. 14 for one subproblem, and there are $O(r)$ subproblems (where r is the number of rows in the components), the total time complexity of the subpath generation phase is $O(Mr)$. Note here that, the number of subpaths recorded at the end of this phase for each subproblem is $O(K)$, where $K \leq M$. After this phase, a graph $\mathcal{G}_R$ is created, as shown in Fig. 13. The number of vertices in $\mathcal{G}_R$ is equal to the total number of recorded subpaths, which is $O(Kr)$. The edges in $\mathcal{G}_R$ are only between vertices that correspond to adjacent subproblems. Hence, the number of edges in $\mathcal{G}_R$ is $O(K^2r)$. As mentioned before, computing the longest path for a DAG has linear time complexity in terms of the input graph size [7]. As a result, the total time complexity of this

[6]Namely, we can create three buckets for each checkerboard cell C, each bucket corresponding to a linked list of subpaths having the same length. Using this, we can find a subpath $P' \in \mathcal{P}_C$ such that P' is shorter than P, and replace it with P in constant time.



(a)

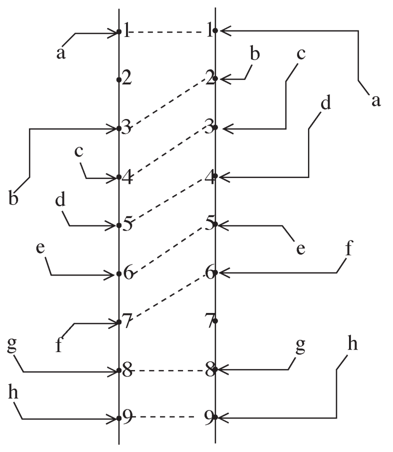

(b)

Fig. 15. (a) A sample checkerboard structure with three subproblems. The selected subpaths in each subproblem are $\{a11, b32, c43\}$, $\{d54, e65, f76\}$, and $\{g88, h99\}$, respectively. (b) The corresponding escape routing solution.

algorithm is $O(Mr + K^2r)$. Here, we can set K and M to large values (possibly on the order of thousands) so that a large number of subpaths are generated for each subproblem, and various path combinations are explored for the solution. Yet the algorithm will still have good run-time characteristics, as will be demonstrated in Section VI.

Fig. 15 illustrates a sample checkerboard with nine rows, and three subproblems. For each subproblem, a subpath is selected, and they are merged to obtain a path of eight routing patterns. The solution corresponding to this path is illustrated in part (b).

V. Generalizations of the Algorithms

A. Different Escape Styles

In the algorithms of Section IV, we have considered only 16 routing patterns for each net. The rationale behind this assumption has been discussed in Section IV-A. However, it is also possible to extend our algorithms such that more routing patterns are considered. Assume that a net is allowed to escape from one of the V neighboring rows of its terminal. (We have assumed that $V = 4$ in the previous sections). The graph model described in Section IV-A can be used with small modifications for different V values. However, for the exact maximal planar routing algorithm in Section IV-C, we would need a $(V-1)$st-order transformation on the input graph. Note that the size of the transformed graph would be exponential in V, and this approach could be impractical for large V values. However, the randomized algorithm we propose in Section IV-D can easily be generalized for arbitrary V values. Namely, only two modifications are needed in the algorithm described in Fig. 13. First, the subproblem sizes need to be $V-1$, instead of 3. Then, the second rule for edge creation in $\mathcal{G}_R$ needs to be changed such that P_k^{i+1} and P_j^i are separated by $V-2$ columns, instead of two columns. Hence, the randomized algorithm would still be scalable for large V values.

Furthermore, it is also possible to generalize the types of escape patterns used in the proposed algorithms. As illustrated in Fig. 5, only a certain class of escape patterns have been considered here. However, the algorithms we propose can handle different escape styles, as long as the property stated in Lemma 1 is satisfied for some constant ℓ value (ℓ is equal to three in Lemma 1). As future research, we are working on a more sophisticated escape pattern generation algorithm, which considers different factors, such as estimated congestion levels within the component areas. The escape selection algorithms described in Section IV will still be applicable for these types of escape patterns, as long as the generated patterns satisfy Lemma 1.

B. Multicapacity Components

For simplicity of presentation, we have described our models and algorithms in the context of the assumption that only a single net can be routed between adjacent pins. However, it is also possible to generalize these algorithms for components with multiple capacities.

A straightforward way to handle this problem is to (conceptually) replace the multicapacity rows with multiple single-capacity tracks. For example, if row r of a component has a capacity of 2, then we can define two single-capacity routing tracks corresponding to r, as shown in Fig. 16. After that, the algorithms proposed in Section IV can be applied without a change. If all the capacities have constant [i.e., $O(1)$] values, then the asymptotic complexity of the proposed algorithms will remain the same.

C. Multiple Components

Another assumption we have made in the previous sections is that the problem consists of two components separated by a channel. For a general design, we can apply these algorithms on different pairs of components independently. As future work, we are working on an algorithm that automatically identifies the best pairs of components to be routed on each layer of a complex design. Once the component pairs are identified, the algorithms given in this paper can be applied on each pair independently. For a typical industrial board today, it is reasonable to expect large bus structures (each containing a

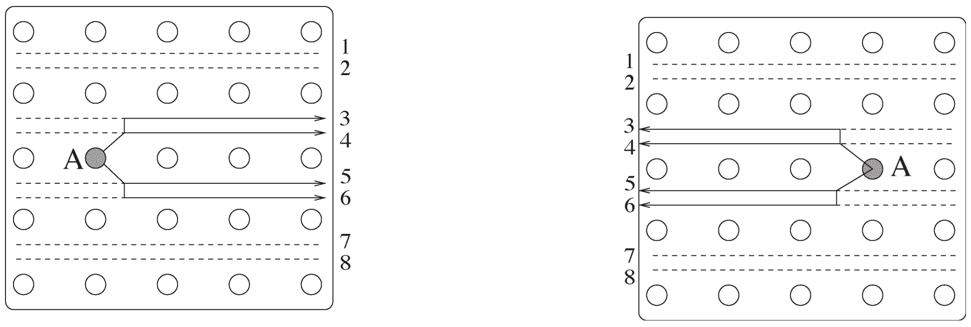


Fig. 16. Routing patterns of Fig. 5 are redrawn for multi-capacity components. Dashed lines illustrate the routing tracks available between adjacent rows of pins.

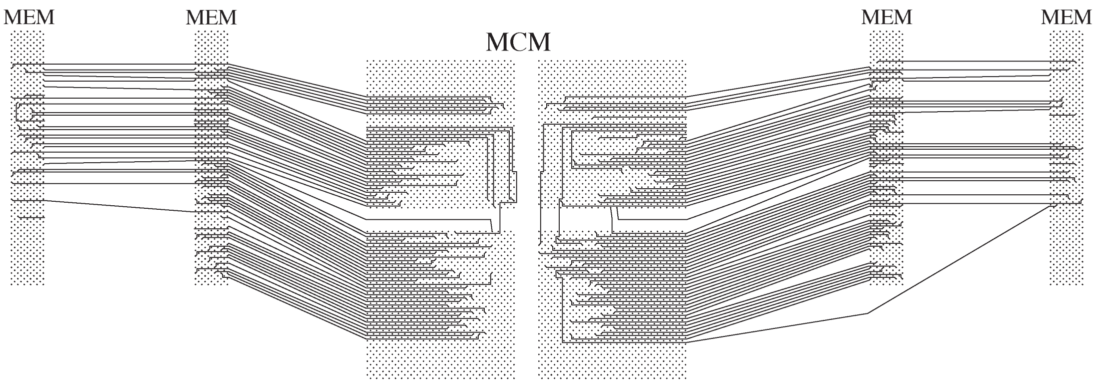


Fig. 17. Sample solution for one layer (out of 8) of a problem containing an MCM and four memory units. The noncrossing channel connections are illustrated as straight lines between components, while the escape routing solutions are shown with solid lines inside the components. 120 (out of 906 total) nets have been assigned to this layer, and 109 of them have noncrossing channel segments.

large number of nets) between different pairs of components. Furthermore, in high-speed designs, there are additional spacing requirements between nets belonging to different buses due to noise considerations. For such designs, it is highly preferable to route nets belonging to the same bus together, and minimize adjacencies between nets belonging to different bus structures. So, identifying different component pairs, and solving the escape problem for each pair separately will be an effective approach.

It is also possible to merge more than one component to obtain a (conceptual) supercomponent, and apply our algorithms on supercomponent pairs. As an example, consider Fig. 17, which illustrates components from a real industrial design. For this circuit, we can define two separate subproblems: 1) two memory modules on the left, and the left half of the MCM, and 2) two memory modules on the right, and the right half of the MCM. For the first subproblem, we can define one supercomponent as the concatenation of the two memory units on the left, and the other supercomponent as the two quadrants on the left half of the MCM. Applying our algorithms on these supercomponents will give an escape routing solution for the three components simultaneously. Section VI gives further details about our experiments on this and other industrial problems.

VI. Experimental Results

For evaluation of our algorithms, we have extracted escape problems corresponding to different components of an industrial circuit from IBM, for which the current industrial routers fail to produce a routing solution. Typically, the industrial tools do not use the problem formulation proposed in this paper. A common approach used to route board designs in the industry is to perform global routing, followed by iterations of rip-up and reroute techniques. However, such an approach fails when there are a large number of nets of which terminals are inside very dense pin arrays. As discussed before, applying escape routing on each component independently is not an effective approach, either.

For experimental comparisons, we have used a special implementation of the Pathfinder [9] algorithm that recognizes the special property of this problem. In particular, a special graph structure is used so that the dense component areas are modeled as detailed grids, and the intermediate areas between components are modeled as coarse-grain connections. The purpose here is to find the detailed escape routing solutions inside the dense components, while minimizing the number of crossings in the intermediate areas. We have implemented all our algorithms in C++, and performed our experiments on an AMD Athlon 1.3 GHz system with 512 MB memory, and a Linux operating system. For the randomized algorithm, we have used a fixed random seed throughout our experiments. We have observed that changing the random seed does not have a considerable effect on the routing results.

First, we have performed experiments to evaluate the effectiveness of the randomized maximal planar routing algorithm given in Section IV-D. Table I gives a comparison of this algorithm with the exact algorithm described in Section IV-C. Note that the exact algorithm is guaranteed to route maximum number of planar nets on one layer. However, it does not guarantee the optimal result on multiple layers, since we

TABLE I
COMPARISON OF RANDOMIZED AND EXACT ALGORITHMS

			EXACT-PLANAR		RANDOM-PLANAR	
Input	# nets	# layers	# planar nets	time (min:sec)	# planar nets	time (min:sec)
IBM_MEM1	213	4	196	3:34	198	0:31
IBM_MEM2	213	4	191	3:33	190	0:34
IBM_STI	352	5	319	21:52	313	1:44

TABLE II
COMPARISON OF OUR METHODOLOGY WITH A NET-BY-NET APPROACH

			OUR METHOD		NET-BY-NET	
Input	# nets	# layers	# crossing nets	time (min:sec)	# crossing nets	time (min:sec)
IBM_MEM1	213	4	8	0:38	41	5:14
IBM_MEM2	213	4	19	0:43	32	4:33
IBM_STI	352	5	24	0:27	62	11:23
IBM_MEMG1	452	8	82	2:59	164	62:51
IBM_MEMG2	454	8	101	2:24	174	52:32

process one layer at a time. As can be seen from this table, the randomized algorithm gives almost as good results as the exact algorithm, requires less running time, and is more scalable for larger circuits. So, we have used the randomized algorithm as the underlying maximal planar routing algorithm in the next set of experiments.

Then, we have implemented the methodology described in Section III. Namely, the maximal planar routing solution is found for each layer, and then the remaining nets are distributed to all layers at the end. For comparison purposes, we have used the Pathfinder [9]-based algorithm described above. We have fine-tuned this algorithm such that the number of crossing nets (in the channel) is minimized. Table II gives comparison of the results. Here, the number of crossing nets can also be viewed as the number of nets that need to use vias in the area routing stage. Observe that our methodology results in substantially less number of crossing nets for all problems. On average, 14% and 28% of all nets are crossing in the solution of our methodology, and the net-by-net approach, respectively. So, we can say that our algorithms reduce the via requirements significantly. Furthermore, the execution times of our method are much lower, since we calculate the best set of planar nets simultaneously in an efficient way. On the other hand, the net-by-net approach requires multiple iterations to negotiate routing resources among different nets.

We also illustrate a sample solution for one layer of a circuit in Fig. 17. Actually, this figure contains two separate problems: 1) the memory units on the left and MCM, and 2) the memory units on the right and MCM. As mentioned in Section V-C, we have grouped the multiple components together to obtain two supercomponents separated by a channel, for each problem. Although the exact area routing will be determined by a later stage, we also display the noncrossing channel segments in this figure.

VII. CONCLUSION

We have proposed an exact and a randomized algorithm for simultaneous escape routing and layer assignment problem for boards with dense components. The experimental results show that the randomized algorithm gives as good results as the exact algorithm, and is much faster. We also show that the methodology we propose produces considerably better results than a net-by-net approach.

ACKNOWLEDGMENT

The authors would like to thank P. S. Honsinger, D. S. Miller (IBM Microelectronics), and J. Ludwig (IBM Systems) for their valuable discussions, and for providing the industrial data used in our experiments.

REFERENCES

[1] P. Berman and G. Schnitger, "On the complexity of approximating the independent set problem," *Inf. Comput.*, vol. 96, no. 1, pp. 77–94, Jan. 1992.

[2] H. N. Brady, "An approach to topological pin assignment," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. CAD-3, no. 3, pp. 250–255, Jul. 1984.

[3] G. Capwell, "High density design with MicroStar BGAs," Texas Instruments, Dallas, TX, SPRA471A, 1998.

[4] T. Cohen, "Practical guidelines for the implementation of back drilling plated through hole vias in multi-gigabit board applications," in *Proc. IEC DesignCon*, Santa Clara, CA, 2003.

[5] J. Cong and C. L. Liu, "On the k-layer planar subset and topological via minimization problems," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 10, no. 8, pp. 972–981, Aug. 1991.

[6] 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.

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

[8] P. Crescenzi and V. Kann, *A Compendium of np Optimization Problems*. [Online]. Available: http://www.nada.kth.se/ viggo/problemlist

[9] C. Ebeling, L. McMurchie, S. A. Hauck, and S. Burns, "Placement and routing tools for the triptych fpga," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 3, no. 4, pp. 473–482, Dec. 1995.

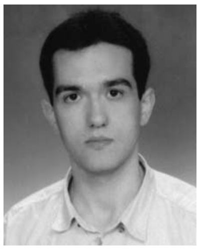
[10] H. Harrer, H. Pross, T.-M. Winkel, W. D. Becker, H. I. Stoller, M. Yamamoto, S. Abe, B. J. Chamberlin, and G. A. Katopis, "First- and second-level packaging for the IBM eServer z900," *IBM J. Res. Develop.*, vol. 46, no. 4–5, pp. 397–420, Jul./Sep. 2002.

[11] N. L. Koren, "Pin assignment in automated printed circuit board design," in *Proc. 9th Design Automation Workshop Design Automation*, Dallas, TX, 1972, pp. 72–79.

[12] J. Ludwig, IBM Systems Group, *Private Communication*, 2004.

[13] M. M. Ozdal and M. Wong, "Length matching routing for high-speed printed circuit boards," in *Proc. IEEE Int. Conf. Computer-Aided Design*, San Jose, CA, Nov. 2003, pp. 394–400.

[14] M. M. Ozdal and M. D. F. Wong, "A provably good algorithm for high performance bus routing," in *Proc. IEEE/ACM ICCAD*, San Jose, CA, Nov. 2004, pp. 830–837.
[15] ——, "A two-layer bus routing algorithm for high-speed boards," in *Proc. IEEE ICCD*, San Jose, CA, Oct. 2004, pp. 99–105.
[16] 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.
[17] H. Xiang, X. Tang, and D. F. Wong, "An algorithm for simultaneous pin assignment and routing," in *Proc. Int. Conf. Computer Aided Design*, San Jose, CA, 2001, p. 232.



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–Champaign, 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 (CAD) of very-large scale integration (VLSI) circuits, with primary focus on physical design algorithms.

Martin D. F. Wong (M'88–SM'04–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 Ph.D. degree in computer science from the University of Illinois, Urbana–Champaign (UIUC), in 1981 and 1987, respectively.

He is currently Professor of Electrical and Computer Engineering at UIUC. Before he joined UIUC, he was a Bruton Centennial Professor of Computer Sciences at University of Texas at Austin (UT-Austin). His research interests are CAD of VLSI circuits, design and analysis of algorithms, and combinatorial optimization. He has published over 250 technical papers and has graduated 31 Ph.D. students. He is a coauthor of *Simulated Annealing for VLSI Design* (Norwell, MA: Kluwer, 1988) and two invited articles in the Wiley Encyclopedia of Electrical and Electronics Engineering, in 1999. 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* published in 2002.

Dr. Wong received the 2000 IEEE CAD Transactions Best Paper Award for his work on interconnect optimization. He also received Best Paper Awards at Digital-to-Analog Converter (DAC)-86 and the International Conference on Computer Design (ICCD)-95 for his work on floorplan design and routing, respectively. He was the General Chair of the 1999 ACM International Symposium on Physical Design (ISPD-99) and was the Technical Program Chair of the same conference in 1998 (ISPD-98). He is on the Steering Committee of ISPD (ISPD-01, ISPD-02, and ISPD-05). He also regularly serves on the technical program committees of many other VLSI conferences (e.g., DAC, ICCAD, ISPD, DATE, ASPDAC, ISCAS, FPGA, SASIMI, GLS-VLSI, SSMSD). He has served as an Associate Editor for IEEE TRANSACTIONS ON COMPUTERS in 1985–2000 and Guest Editor of four special issues for IEEE TRANSACTIONS ON COMPUTER AIDED DESIGN. He is currently on the Editorial Boards of ACM TRANSACTIONS ON DESIGN AUTOMATION OF ELECTRONIC SYSTEMS and IEEE TRANSACTIONS ON COMPUTER AIDED DESIGN.