SIMULTANEOUS ESCAPE ROUTING AND LAYER ASSIGNMENT FOR DENSE PCBS

Muhammet Mustafa Ozdal

Dept. of Computer Science
Univ. of Illinois at Urbana-Champaign
Urbana, IL 61801
ozdal@uiuc.edu

Martin D. F. Wong

Dept. of Electrical and Computer Engineering
Univ. of Illinois at Urbana-Champaign
Urbana, IL 61801
mdfwong@uiuc.edu

ABSTRACT

As die sizes are shrinking, and circuit complexities are increasing, the PCB routing problem becomes more and more challenging. Traditional routing algorithms can not handle these challenges effectively, and many high end designs in the industry require manual routing efforts. In this paper we propose a problem decomposition that distinguishes routing within dense components from routing in the intermediate area. In particular, we propose 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, we model the problem as a *longest path with forbidden pairs* (LPFP) problem, and propose two algorithms 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 we use these algorithms to assign the maximal subset of planar nets to each layer, and then distribute the remaining nets at the end. We demonstrate the effectiveness of these algorithms through experiments on industrial circuits.

1. 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 caused the circuits become more and more dense. So, boards and packages have reduced in size, while the pin counts have been increasing. Today, high-density fine-pitch packages typically contain pin counts on the order of thousands, while they occupy only a minimal board space [11]. This brings a significant increase in routing challenges for current PCB designs [10]. Traditional routing algorithms can not handle those challenges effectively, and many high-end designs require manual efforts for routing. In a typical design cycle of a high-end board, manual routing efforts take about a month [8], and new effective routing tools are necessary to significantly reduce this time. In this paper, we focus on routing within dense components, and we propose algorithms that address these challenges effectively.

A typical PCB routing problem contains a number of dense pin arrays corresponding to different chip components such as MCM, memory, etc. The routing area within such pin arrays is 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

This work was partially supported by the National Science Foundation under grants CCR-0244236 and CCR-0306244, and by an IBM Faculty Partnership Award.

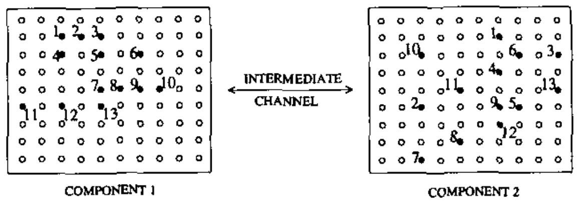


Figure 1: A sample escape problem with 13 nets on two components. Each terminal pin is labeled with its net index. The problem is to find a conflict-free routing solution within components, and to minimize crossings in the channel.

from their terminal pins to the corresponding component boundaries. On the other hand, the intermediate routing area 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 components separately from the intermediate area routing. In other words, two separate problems are distinguished here: (1) routing nets from pin terminals to chip boundaries (escape routing), and (2) routing nets between chip boundaries (area routing). In this paper, we propose algorithms to handle the problem of escape routing.

It is important here to note that escape routing problem for different components should not be considered independent of each other. That is, we can not just apply a traditional escape routing algorithm [4] 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. Figure 1 illustrates a sample problem, and Figure 2 gives a 2-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 increase the manufacturing costs, and adversely affect routability and signal delay characteristics, we try to minimize 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 components, because the scarcity of routing resources does not allow us to perform additional tasks such as length extension for length matching, as in [9].

In Section 2, 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 3. Mainly, we process one layer at a time, and try to route as many non-crossing nets as possible on each layer. In Section 4.1, we model the maximal planar routing problem as a *longest path with forbidden pairs* (LPFP) problem. Although the general case of this problem is NP-complete, the special structure of our problem allows us to propose a polynomial-time exact algorithm in Section 4.3. Then, we propose a fast and effective randomized algorithm in Section 4.4 for large circuits. In Section 5, we discuss generalizations of our models and algorithms. Finally, we demonstrate the effectiveness of our algorithms through experiments in Section 6.

2. PROBLEM FORMULATION AND RELATED WORK

Let a (chip) component be defined as a 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 2-terminal net specifies two pins as its endpoints, which are assumed to be in different components by definition. *Escape route* for a given net is defined as the route from its terminal pins (within components) to the respective chip boundaries. Given an input circuit and a set of 2-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, 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, our objective is to minimize number of vias through crossing minimization.

Figure 1 illustrates a sample escape problem with 13 nets in two components, and Figure 2 gives a 2-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, 6 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 Figure 2 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 [7, 2, 12]. 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 [3], where the objective is to determine the topological

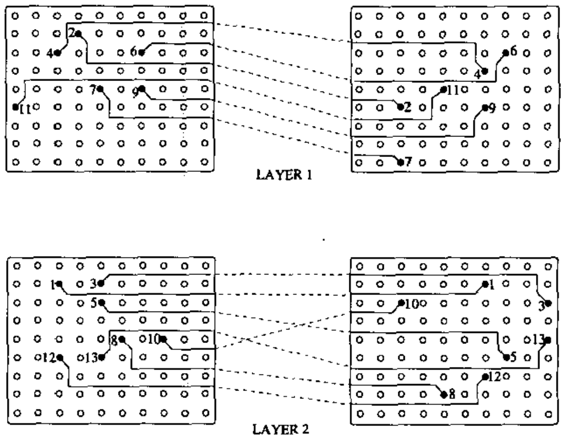


Figure 2: A sample solution for the problem given in Figure 1. Escape routes are illustrated with solid lines within components. Channel segments are shown with dashed lines.

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 [3]. 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 chip boundaries, but they are determined based on the escape routes. For instance in the example of Figure 2, the ordering of nets within chips is not necessarily the same as the ordering on chip boundaries[1], since this ordering further reduces the number of crossings.

3. METHODOLOGY

We use a two-phase approach for this problem: (1) for each layer l, pack as many non-crossing routes as possible on l, (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 Figure 2 is an example output of this phase. Specifically, the maximum non-crossing 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 4.

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 Figure 2 that although it has a conflict-free routing solution within the components, it crosses with nets 5 and 13 in the channel.

[1]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.

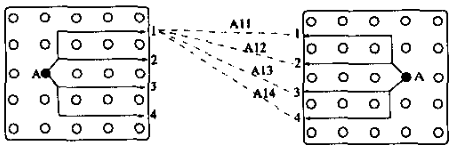


Figure 3: Routing patterns considered for net A. Only 4 out of 16 patterns are shown here for clarity.

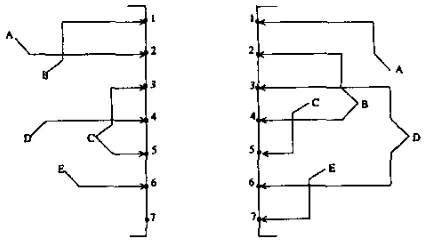


Figure 4: A sample escape routing problem for 5 nets. For clarity, only one or two routing patterns are defined for each net (instead of 16 as in the actual algorithm).

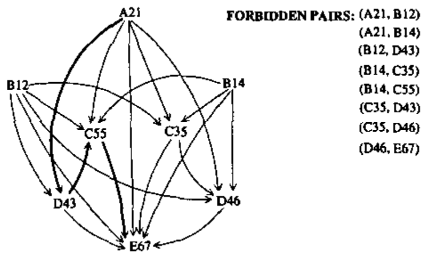


Figure 5: The graph model corresponding to the problem given in Figure 4. The longest path with forbidden pairs is illustrated with the thick lines.

For the second phase, we use a negotiated congestion based net-by-net approach similar to Pathfinder [6]. 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.

4. MAXIMAL PLANAR ROUTING

4.1. 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 Figure 2, where an escape route spans at most 2 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[2], as shown in Figure 3. Namely, we consider 4 escape routes for a net within each component, so that it can escape from one of the 4 neighboring rows of its terminal pin. Note that either one of the 4 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 Figure 3, some sample routing patterns are illustrated.

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 non-crossing nets [3]. 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 non-conflicting and non-crossing nets. For this purpose, we will define a graph model $\mathcal{G}$, and a set of forbidden pairs $\mathcal{F}$ (such that $\mathcal{F}$ contains pairs of vertices from $\mathcal{G}$) as follows:

- For each routing pattern, a vertex exists in $\mathcal{G}$.
- Let u, v be vertices in $\mathcal{G}$ corresponding to the routing patterns U_{ij} and V_{kl}, respectively. 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 Figure 3 would be strictly above A_{34}.)
- Let u, v be vertices in $\mathcal{G}$. Forbidden pair (u, v) exists in $\mathcal{F}$ iff at least one the following conditions is the case:
  1. u and v correspond to the same net.
  2. The routing patterns corresponding to u and v conflict with each other 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:

[2]In Section 5, we discuss possible extensions to relax this assumption.

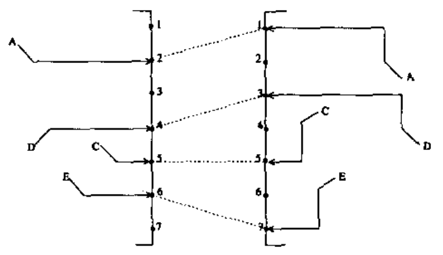


Figure 6: The actual maximal planar routing solution corresponding to the path given in Figure 3.

Theorem 4.1. *The problem of finding the maximum subset of non-crossing and non-conflicting routing patterns is equivalent to the longest path with forbidden pairs (LPFP) problem on $\{\mathcal{G}, \mathcal{F}\}$.*

LPFP problem [5] 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 $\mathcal{G}$ can not 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 4.3:

Lemma 4.2. *For any pair $(u, v) \in \mathcal{F}$, the maximum distance between u and v in $\mathcal{G}$ is guaranteed to be less than or equal to 3.*

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 can not escape to rows separated by more than 3 rows if $(u, v) \in \mathcal{F}$. So, the maximum distance between conflicting vertices in $\mathcal{G}$ can be at most 3. □

Figure 4 gives a sample problem with a limited number of patterns defined for each net[3]. The graph model corresponding to these patterns is illustrated in Figure 5. Observe that the longest path with 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 Figure 6.

4.2. Checkerboard Graph Model

In the graph model described in Section 4.1, 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 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 (chip) 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 Figure 7. We can

[3]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.

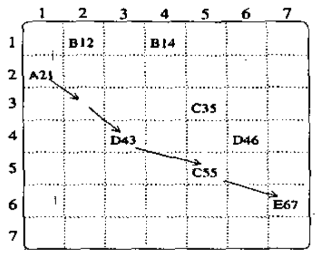


Figure 7: The checkerboard structure corresponding to the graph of Figure 5. For clarity, only the edges on the longest path are illustrated.

formally define a graph model $\mathcal{G}$ based on this conceptual structure as follows:

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

Figure 7 shows the checkerboard structure corresponding to the graph given in Figure 5. For clarity, the vertices with zero weights, and the edges between adjacent rows and columns are omitted in this figure. The corresponding longest path with 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 Figure 6.

This graph structure is in fact very similar to the one proposed in Section 4.1. The main difference is that edges exist only between neighboring routing patterns here. This reduces the number of edges from $O(n^2)$ to $O(nr)$, which will be helpful to reduce the complexity of the exact algorithm we propose in Section 4.3. Furthermore, the structured view of a checkerboard will help us to propose a very effective randomized algorithm in Section 4.4.

4.3. 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 4.2. 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 4.3; however to give an intuition about this process, we will first describe simpler versions of this transformation in Definitions 4.1 and 4.2.

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 directed acyclic graph, 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

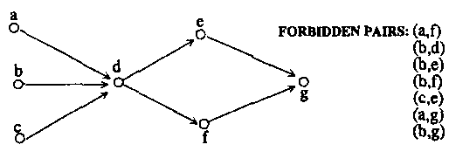


Figure 8: A sample graph $\mathcal{G}$, and a set of forbidden pairs.

$u \rightarrow w$ and $w \rightarrow v$ exist in $\mathcal{G}$. For consistency, we assume that each vertex has a parent-grandparent pair of NULL-NULL.

Definition 4.1. *First-order transformation of $\mathcal{G}$ (denoted as $\mathcal{G}^1$) is defined as follows:*

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

Remark 4.3. *If the maximum distance 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 4.2. *Second-order transformation of $\mathcal{G}$ (denoted as $\mathcal{G}^2$) is defined as follows:*

- *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 i^{th} parent of u. In other words, number of vertices in U is equal to the number of parents of u.*
- *There exists an edge from $U[i]$ to $V[j]$ in $\mathcal{G}^2$ iff:*
  1. *u is the j^{th} parent of v in $\mathcal{G}$.*
  2. *(u, v) is not a forbidden pair.*
  3. *(i^{th}-parent-of-u, v) is not a forbidden pair.*

As an example, consider graph $\mathcal{G}$ with forbidden pairs in Figure 8. Second order transformation of this graph is shown in Figure 9. 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 4 vertices in Figure 9 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:

Lemma 4.4. *Consider two vertices w and v in $\mathcal{G}$ such that the maximum distance from w to v is at most 2. 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 distance from w to v is 1, then the proof is straightforward. Otherwise, consider any path of the form $w \rightarrow u \rightarrow v$. Assume that w is the i^{th} parent of u, and u is the j^{th} parent of v. Due to rule (1) in Definition 4.2, edges from vertex set W to vertex set U in $\mathcal{G}^2$ can only be to $U[i]$. Due to rule (3), 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 can not exist. □

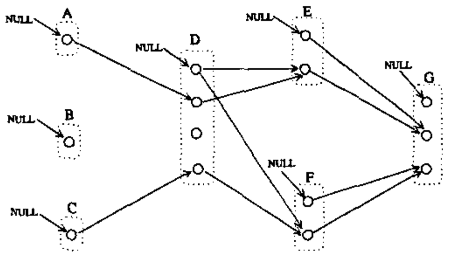


Figure 9: Second-order transformation of graph $\mathcal{G}$ in Figure 8. A set of vertices indicated with dotted lines correspond to each vertex of $\mathcal{G}$.

As an example, consider the forbidden pairs (a, f), (b, d), (b, e), (b, f), (c, e) in Figure 8, each having a maximum distance of 2 between the pairs. Observe that there are no paths in the transformed graph of Figure 9 between the corresponding set of vertices.

Lemma 4.5. *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.*

Theorem 4.6. *If the maximum distance 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 4.4 and 4.5. □

Definition 4.3. *Third-order transformation of $\mathcal{G}$ (denoted as $\mathcal{G}^3$) is defined as follows:*

- *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 i^{th} parent of u and the j^{th} parent of i^{th} parent of u. In other words, for each parent-grandparent pair of u, there exists a corresponding vertex in set U.*
- *There exists an edge between $U[i][j]$ and $V[k][l]$ in $\mathcal{G}^3$ iff:*
  1. *u is the k^{th} parent of v in $\mathcal{G}$.*
  2. *$l = i$.*
  3. *(u, v) is not a forbidden pair.*
  4. *(i^{th}-parent-of-u, v) is not a forbidden pair.*
  5. *(j^{th}-parent-of-i^{th}-parent-of-u, v) is not a forbidden pair.*

Figure 10 illustrates the third-order transformation of the graph given in Figure 8. Here, it is again assumed that the first parent of each vertex is NULL. For instance, $G[2][1]$ (i.e. the first vertex 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.

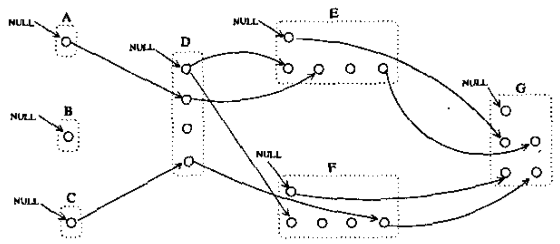


Figure 10: Third-order transformation of graph $\mathcal{G}$ in Figure 8. A set of vertices indicated with dotted lines correspond to each vertex of $\mathcal{G}$.

Lemma 4.7. *Consider two vertices w and v in $\mathcal{G}$ such that the maximum distance from w to v is at most 3. 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 distance from w to v is 1 or 2, then the proof is similar to that of Lemma 4.4. Otherwise, consider any path of the form $w \rightarrow y \rightarrow u \rightarrow v$, where w is the j^{th} parent of y, y is the i^{th} parent of u, and u is the k^{th} parent of v. Any edge in $\mathcal{G}^3$ from vertex set W to vertex set Y can only be to $Y[j][.]$ due to rule (1) in Definition 4.3. Similarly, any edge from $Y[j][.]$ to vertex set U can only be to $U[i][j]$ due to rules (1) and (2). Finally, an edge from $U[i][j]$ to vertex set V exists only if (w, v) is not a forbidden pair, due to rule (5). So, a path from w to v can not exist if w and v conflict with each other. □

Observe in Figure 10 that, there is no path between vertex sets corresponding to the forbidden pairs in Figure 8. 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 4.8. *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.*

Theorem 4.9. *If the maximum distance between any forbidden pair (u, v) in $\mathcal{G}$ is at most 3, then the longest path in $\mathcal{G}^3$ is the exact solution to LPFP problem on $\mathcal{G}$.*

PROOF. It follows directly from Lemma 4.7 and 4.8. □

Due to Lemma 4.2, we can apply a third-order transformation on the directed acyclic graph described in Section 4.1, and obtain the exact solution to LPFP problem by using a linear-time longest path algorithm [4]. From Theorem 4.1, this solution corresponds to the maximal planar routing solution to our original problem.

We can show by amortized analysis that the complexity of this algorithm is $O(nc^3 + rc^4)$, where n is the number of nets; c and r are the number of columns and rows of the (chip) component, respectively. Note here that c and r are typically on the order of the square-root of component size; so the complexity can be rewritten as $O(ns^{3/2} + s^{5/2})$, where s is the component size. Although this complexity would be acceptable for moderate chip 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.

```
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 $(u, v)$ such that
        $u \in P_j^i$ and $v \in P_k^{i+1}$
Return the longest path in $\mathcal{G}_R$
```

Figure 11: Randomized algorithm for LPFP problem on a checkerboard graph where the maximum distance between any forbidden pair is at most 3.

4.4. Randomized algorithm for LPFP

As stated by Lemma 4.2, 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.

Figure 11 gives the outline of the randomized algorithm we propose for the checkerboard graph model described in Section 4.2. The first step here is to define subproblems on the checkerboard structure as shown in Figure 13. Then, we randomly generate a predefined number of *permissible* subpaths for each subproblem. Figure 12 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 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 directed acyclic graph $\mathcal{G}_R$, which is defined in Figure 11. The following lemma explains the rationale behind this model:

Lemma 4.10. *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, v) such that $u \in P_j^i$ and $v \in P_k^l$, 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 Figure 11. Otherwise, assume that $l \geq i + 2$, and 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 3 checkerboard rows, since there is at least one subproblem between them. Furthermore due to rule (2), there are at least 3 columns between the last element of P_j^i, and the first element of P_k^l. Since the maximum distance between a forbidden pair can be at most 3 in the original graph (as stated in Lemma 4.2), there exists no forbidden pair (u, v) such that $u \in P_j^i$ and $v \in P_k^l$. □

Due to this lemma, we can use a simple longest path algorithm

```
GENERATE-SUBPATHS(Subproblem $i$: between rows $T_i$ and $B_i$)
for a fixed number of 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 $v$
    If $P$ is one of the $K$ longest subpaths ending at $C$
        record $P$
    else
        discard $P$
```

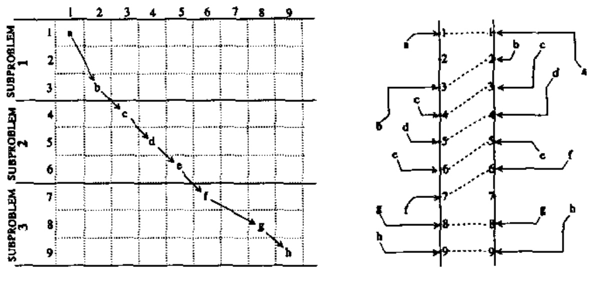
Figure 12: Algorithm to generate a set of random subpaths between rows T_i and B_i of the checkerboard.

on $\mathcal{G}_R$ without the need of checking forbidden pairs. This longest path will correspond to the optimal combination of the 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. Note here that the operation of generating one random subpath can be done in constant time, since each subpath contains at most 3 routing patterns. If we generate K random paths for each subproblem, then $\mathcal{G}_R$ will contain $O(Kr)$ vertices (where r is the number of rows on the chip), and $O(K^2r)$ edges, since edges exist only between adjacent subproblems. The longest path for a dag can be calculated in linear time [4]; hence the complexity will be $O(K^2r)$. Here, we can set K to a large value (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 6.

Figure 13 illustrates a sample checkerboard with 9 rows, and 3 subproblems. For each subproblem, a subpath is selected, and they are merged to obtain a path of 8 routing patterns. The solution corresponding to this path is illustrated in part (b).

5. GENERALIZING THE MODELS

In the algorithms of Section 4, we have considered only 16 routing patterns for each net. The rationale behind this assumption has been discussed in Section 4.1. 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 4.1 can be used with small modifications for different V values. However, for the exact maximal planar routing algorithm in Section 4.3, 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 4.4 can easily be generalized for arbitrary V values. Namely, only two modifications are needed in the algorithm described in Figure 11. 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



(a) (b)

Figure 13: (a) A sample checkerboard structure with 3 subproblems. The selected subpaths in each subproblem are $\{a11, b32, c43\}$, $\{d54, e65, f76\}$, and $\{g88, h99\}$, respectively. (b) The corresponding escape routing solution.

that P_k^{i+1} and P_j^i are separated by $V - 2$ columns, instead of 2 columns. Hence the randomized algorithm would still be scalable for large V values.

Another assumption we have made in the previous sections is that the problem consists of two components separated by a channel. So for a general circuit, we can apply these algorithms on different pairs of components independently. However, it is also possible to merge different components to obtain two (conceptual) super-components, and apply the algorithms on all components simultaneously. Those details are omitted here due to page limitations.

6. 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. We have implemented all our algorithms in C++, and performed our experiments on an AMD Athlon 1.3 GHz system with 512MB memory, and a Linux operating system.

First, we have performed experiments to evaluate the effectiveness of the randomized maximal planar routing algorithm given in Section 4.4. Table 1 gives comparison of this algorithm with the exact algorithm described in Section 4.3. 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 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 3. 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 a net-by-net approach based on Pathfinder [6]. We have fine-tuned this algorithm such that the number of crossing nets (in the channel) is minimized. Table 2 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

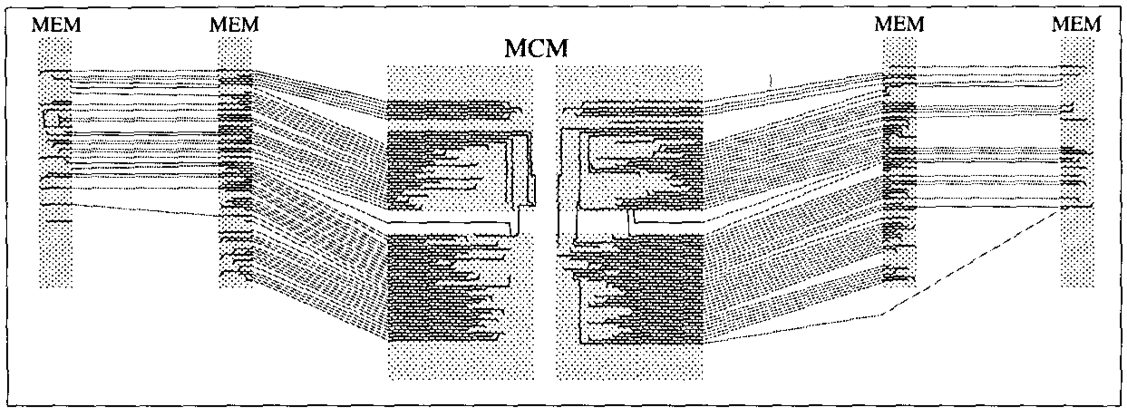


Figure 14: A sample solution for one layer (out of 8) of a problem containing an MCM and 4 memory units. The non-crossing channel connections are illustrated as straight (dotted) lines between components, while the escape routing solutions are shown with solid lines inside the components. 120 (out of 906 total) nets have been assigned to this layer, and 109 of them have non-crossing channel segments.

Table 1: 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 2: 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

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 Figure 14. Actually, this figure contains two separate problems: (1) the memory units on the left and MCM, (2) the memory units on the right and MCM. As mentioned in Section 5, we have grouped multiple components together to obtain two *super-components* separated by a channel, for each problem. Although the exact area routing will be determined by a later stage, we also display the non-crossing channel segments in this figure.

7. CONCLUSIONS

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.

8. REFERENCES

[1] P. Berman and G. Schnitger. On the complexity of approximating the independent set problem. *Inform. and Comput.*, 96:77–94, 1992.

[2] H. N. Brady. An approach to topological pin assignment. *IEEE Trans. on Computer Aided Design*, 3:250–255, 1984.

[3] J. Cong and C. L. Liu. On the k-layer planar subset and topological via minimization problems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10, 1991.

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

[5] P. Crescenzi and V. Kann. A compendium of np optimization problems. http://www.nada.kth.se/ viggo/problemlist.

[6] C. Ebeling, L. McMurchie, S. A. Hauck, and S. Burns. Placement and routing tools for the triptych fpga. *IEEE Trans. on VLSI*, pages 473–482, 1995.

[7] N. L. Koren. Pin assignment in automated printed circuit board design. In *Proc. of the ninth design automation workshop on Design automation*, pages 72–79, 1972.

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

[9] M. M. Ozdal and M. D. F. Wong. Length matching routing for high-speed printed circuit boards. In *Proc. of IEEE Intl. Conf. on Computer-Aided Design*, Nov. 2003.

[10] B. Voss. The package routing challenge. *StilwellBaker, Inc.*, June, 2003.

[11] D. Wiens. Printed circuit board routing at the threshold. *White Paper, Mentor Graphics*, 2000.

[12] H. Xiang, X. Tang, and D. F. Wong. An algorithm for simultaneous pin assignment and routing. In *Proc. of Intl. Conf. on Computer Aided Design*, 2001.