Via Design Rule Consideration in Multilayer Maze Routing Algorithms

Jason Cong, Jie Fang, and Kei-Yong Khoo

***Abstract*—Maze routing algorithms are widely used for finding an optimal path in detailed routing for very large scale integration, printed circuit board and multichip modules In this paper, we show that finding an optimal route of a two-pin net in a multilayer routing environment under practical via design rules can be surprisingly difficult. A straightforward extension to the maze routing algorithm that disallows via-rule incorrect routes may either cause a suboptimal route to be found, or more seriously, cause the failure to find any route even if one exists. We present a refined heuristic to this problem by embedding the distance to the most recently placed via in an extended connection graph so that the maze routing algorithm has a higher chance of finding a via-rule correct optimum path in the extended connection graph. We further present efficient data-structures to implement the maze routing algorithm without the need to preconstruct the extended connection graph. Experimental results confirmed the usefulness of our algorithm and its applicability to a wide range of CMOS technologies.**

***Index Terms*—Routing, via design rule.**

I. Introduction

FINDING an optimal point-to-point path is the fundamental operation in the area-based detailed routing for very large scale integration, printed circuit board (PCB) and multichip modules (MCM's). The most common approach is to represent the routing area with a routing grid and perform routing over the grid. The grid-points in the routing grid represent the permissible locations that the center-line of a path can pass through, and the edges between the grid-points determine the permissible routing patterns. In general, the grid-points and grid-edges can be represented as a set of nodes and edges, respectively, in an undirected graph $G = (V, E)$ called the *connection graph*. The edges are usually weighted to reflect the routing cost, such as the actual length of the grid-edge; and the cost of a path $F(p)$ is the sum of edge-weights along the path p. In this paper, we will assume that the edge weights are uniform in each direction for each layer. For example, all horizontal paths on the first routing layer have the same cost per unit length. We also define $F(p_e) = \infty$ if p_e is an invalid path due to design rule violations.

A. Practical Via Design Rules

The layout design rules specify a set of spacing and width constraints on layout geometries to ensure both the yield and the electrical performance of the manufactured design. For instance, minimum wire spacings and widths primarily prevent electrical shorts and opens, respectively. Minimum spacings in vias ensure good yields as well as good connections between the connecting metal layers. Table I shows some design rules, (also illustrated in Fig. 1), related to the metal and cut layers for a three-level-metal 0.5-μm CMOS process. While a cut is clearly defined as the connection between two adjacent conducting layers, a "via" is less well defined and commonly meant as the connecting object between *metal* layers. In this paper, this distinction is immaterial and we will use "cut" and "via" interchangeably when referring to the connection between two routing layers.

TABLE I
DESIGN RULES FOR A 0.5-μm CMOS PROCESS

	Rules	Dimension (μm)
w1	Minimum MET1 and MET2 width	0.6
w2	Minimum MET3 width	1.2
w3	CONTACT, VIA1 and VIA2 size	0.8×0.8
e1	Minimum metal enclosure of CONTACT, VIA1 and VIA2	0.2
s1	Minimum MET1 to MET2 spacing	0.8
s2	Minimum MET3 spacing	1.2
s3	Minimum CONTACT to CONTACT spacing	0.6
s4	Minimum CONTACT to VIA1 spacing	0.3
s5	Minimum VIA1 to VIA1 spacing	0.6
s6	Minimum VIA1 to VIA2 spacing	0.3
s7	Minimum VIA2 to VIA2 spacing	0.6

There are three properties regarding the design rules that are generally true in practice:

Property 1: The minimum spacing for a cut affects only the same or adjacent cut layer. For example, there is no minimum spacing requirement between the CONTACT and VIA2 layers since they are not adjacent cut layers.

Property 2: The minimum spacing for cuts on adjacent layers is smaller than or equal to the minimum spacing for cuts on the same layer.

For example, the minimum spacing between VIA1 and VIA2 is 0.3-μm, whereas the minimum spacing between VIA1 and VIA1 or between VIA2 and VIA2 is 0.6 μm.

Manuscript received June 1, 1999; revised October 1, 1999. This work was supported in part by DARPA/ETO under Contract DAAL01-96-K-3600 managed by the U.S. Army Research Laboratory. This paper was recommended by Associate Editor M. Wong.

The authors are with the Computer Science Department, University of California, Los Angeles, CA 90095 USA.

Publisher Item Identifier S 0278-0070(00)01799-1.

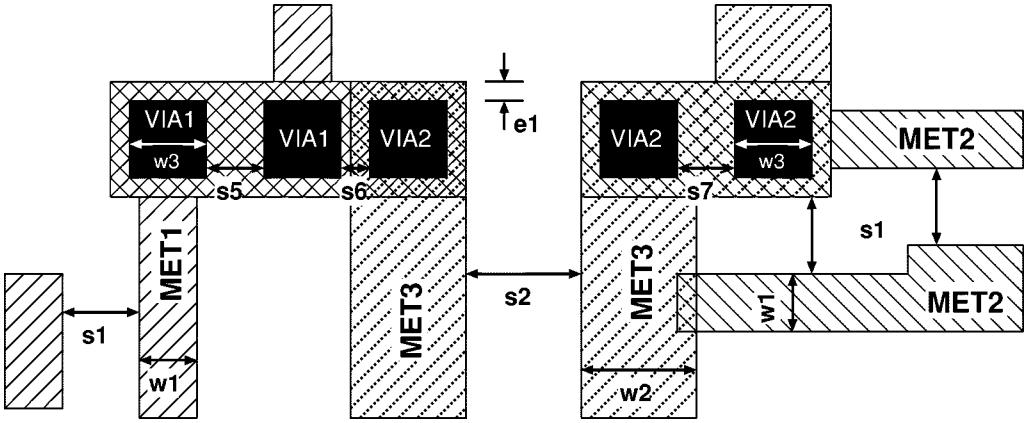

Fig. 1. Illustration of the 0.5-μm CMOS design rules.

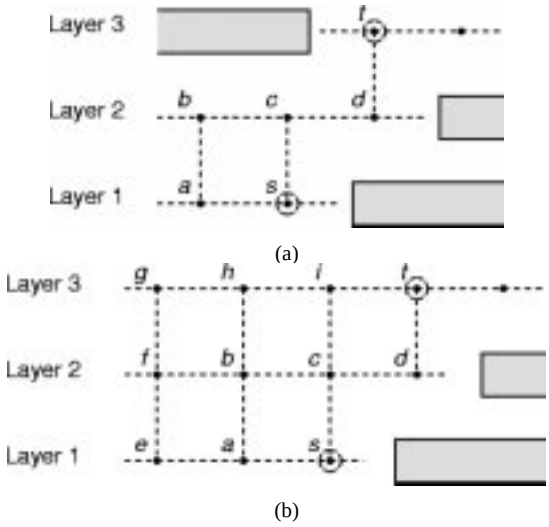

Fig. 2. Examples where no path can be found from s to t, as shown in (a), or, a suboptimal path, $s-c-b-f-g-h-i-t$, is found in a slightly different scenario, as shown in (b), by a traditional maze router due to the via spacing rule that requires a minimum via-to-via grid-spacing of two. The optimal via-rule correct path in both cases is $s-a-b-c-d-t$.

Property 3: The minimum spacing for two cuts, on either the same or adjacent cut layers, is smaller than the minimum wire width plus two times the wire spacing (W + 2S) of either of its connecting metal layers.

For example, the minimum spacing between VIA2 and VIA2 is 0.6-μm, but the W + 2S for MET2 is $(0.6+2\times0.8)=1.4\mu$m, and the W + 2S for MET3 is$(1.2+2\times1.2)=3.6\mu$m.

B. The Classical Maze Routing Algorithm and its Limitations

Given a connection graph $G=(V,E)$, a source node s and a destination t (where $s,t\in V$), the minimum-cost path problem is to find a path p^* in G such that $F(p^*)$ is minimal among all feasible paths from s to t in G. It is clear that the path p^* corresponds to a detailed routing solution in the routing region represented by G. The minimum-cost path problem can be solved using the maze routing algorithm [1], [2] [Fig. 3] which finds the minimum-cost path using a point-by-point expansion strategy based on the dynamic programming principle. It maintains a priority queue Q of candidate nodes for expansion, ordered according to their priority. The priority determines the expansion

Algorithm: MAZE ROUTING ALGORITHM (G,s,t)

```
1   Q ← s;
2   while Q ≠ ∅
3     p ← POP(Q);
4     if (p = t) then path found;
5     for-each q ∈ NEIGHBORS(p) do
6       EXPAND(p, q, Q);
7     end for-each
8   end while
end
```

Fig. 3. The maze routing algorithm finds a minimum-cost path s to t in the graph G.

strategy of the algorithm. For example, using the actual costs to the nodes in Q as the priorities will result in a breadth-first search. Using the actual costs plus the estimated costs to the destination will result in an A^* search. At each iteration, the highest priority node p is retrieved from Q and *expanded* into each of its feasible neighbors q. EXPAND(p,q,Q) updates node q (and adds q to Q if necessary) if the path $s\rightarrow p\rightarrow q$ is better than the path to q (if there is one).

The optimality of the maze algorithm is predicated on a cost function $F(p)$ that is *monotone* and satisfies the *principle of optimality* [3] in dynamic programming defined as follows.

A monotone cost function $F(p)$ implies that $F(p)\geq F(p')$ for all subpaths $p'\subset p$, for all paths in G. This is easily satisfied by having only positive weights for the edges in G. Intuitively, a monotone cost function allows the path searching process to always progress *away* from the scource s. Therefore, each node in G is expanded *at most once* in the maze routing algorithm. The principle of optimality in dynamic programming [3] states that:

Principle of Optimality: An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

In the maze algorithm, the best paths found to all visited nodes so far constitute a state, and how to update the best path to a visited node constitute a decision. The principle of optimality implies that at any given point in the maze expansion (line 6 in

Fig. 3) process, a partial routing towards the destination is *independent* of the partial routings that have been already been found. However, this is usually not true in practical layout design since a partial routing that has been completed immediately imposes possible design rule restrictions around its vicinity. A solution to this problem is to ensure that the grid spacing is greater than or equal to the largest applicable spacing rule. However, in "gridless" routing, the routing grid is smaller than the worst-case via-to-via spacing and can be as fine as the "manufacturing" grid (The actual grid size is determined by the resolution of the technology and/or the design database). In this case, the placement of a via *will* restrict where the next via can be placed.

The problem of via-rules on the maze routing algorithm can now be illustrated with the following simplified examples for clarity. Let us suppose that the routing grid has uniform spacing and uniform edge cost, and the minimum via spacing is two grid-spacings. Fig. 2(a) shows an example with a cross-sectional (i.e., a vertical two-dimensional (2-D) plane) routing region. With the maze routing algorithm, the source node s will be expanded first into nodes a and c. If node c is expanded next (since the paths $s-a$ and $s-c$ have the same cost), then c will be expanded into nodes b and d. Now node d can be expanded to t but the solutions $s-c-d-t$ will be design rule incorrect! If invalid paths are disallowed during maze expansion, then node d will be discarded and node a will be expanded into node b. But since a node can be expanded at most once in the maze routing algorithm, node b cannot be expanded into node c because node c has been expanded before. Therefore, the feasible path $s-a-b-c-d-t$ will not be found. Notice that even if node a is expanded before c (say by weighing the via edges with higher cost), the feasible path still cannot be found since node c will always be expanded *before* node b. In a slightly different example, as shown in Fig. 2(b), the maze routing algorithm will be able to find a via-rule correct path $s-c-b-f-g-h-i-t$ because of the removal of layer-3 obstacle in Fig. 2(a). However, this path is a suboptimal solution since the optimal s to t path is $s-a-b-c-d-t$. So with the presence of via-rules, traditional maze routing algorithm may fail to find a solution or find a suboptimal solution.

C. Problem Formulation

In the previous example, the failure to find a path is due to the existence of node c which is on the optimal path from s to t but the subpath is not an optimal s to c path. Such a node is called a β-node and is defined formally as follows:

Definition 1: A node u in G is a β-node if 1) there exists an optimal via-rule-correct path $p^*_{s,u}$ from s to u (β–path) that is via-rule correct up to u, *and* 2) $p^*_{s,u}$ has a smaller cost than the subpath from s to u in $p^*_{s,t}$.

The β-node is illustrated in Fig. 4. For the example shown in Fig. 2, node c is a β-node since 1) there is an optimal via-rule-correct path $s-c$ and 2) whose cost of one is smaller than the cost of three of the subpath $s-a-b-c$ in the via-rule-correct path to t The existence of a β-node prevents an optimal path from being discovered by the maze routing algorithm:

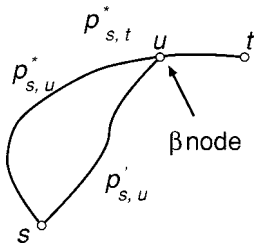

Fig. 4. Illustration of a β node: the path $p^*_{s,u}$ is a subpath of optimal via-rule-correct s to t path $p^*_{s,t}$. However, there exists a minimum-cost path $p'_{s,u}$ from s to u, that is via-rule correct and has a smaller cost than $p^*_{s,u}$.

Property 4: If there is an optimal via-rule-correct path $p^*_{s,t}$ from s to t which has a β-node along the path, then the path will not be found by the maze routing algorithm.

Proof: The maze routing algorithm will always expand the β-node along the β-path because it is the smallest cost path among all the paths from s to β. Since a node can only be expanded once in the maze routing algorithm, the optimal path $p^*_{s,t}$ that passes through β will not be found.

Followed by Property 4, we can show that if all the optimal paths have a β-node along their paths, the maze algorithm might not find the optimal via-rule-correct path. Notice that this does *not* prevent the maze routing algorithm from returning a suboptimal path, as shown in Fig. 2(b). However, if *every* via-rule-correct path from s to t has a β-node, then the maze routing algorithm will not find a via-rule-correct path from s to t even if such a path exists, as shown in Fig. 2(a).

We want to stress that in practice, the via-rules are not a serious problem in the early phases of routing where rip-up-and-reroutes, and local modifications can effectively handle many of the issues with via-rules. It is in the later phases of routing when the routing region is extremely congested or compacted, and the free spaces are narrow and irregular, that careful consideration of the via rules becomes critical. The proposed routing algorithm is meant to function as an auxiliary but more accurate router that seeks a *partial path* when the traditional maze routing algorithm fails or being suboptimal. This will be described in more details in Section III.

Previous works in detailed routing have considered the interaction of vias with other objects [4], [5] but not between vias within the same route. The problem of design rule interactions within the same route has been acknowledged in [4] but not solved except for some easily identifiable special cases. Notice that the routers in [4], [5] are actually gridless routers using area expansion. This is so because early gridded routers have used a large enough grid spacing and evaded the via-spacing problems. It is also interesting to note that Property 3 implies that the via spacing is not a problem for two-layer routing as illustrated in Fig. 5. Therefore, the via spacing problem is limited to three or more layer routings which does not apply to many early works in detailed routing. In industrial routers, heuristics are often used to make the maze routing algorithm more robust so that it is unlikely to fail to find a route but the route may be suboptimal. The general optimality of path searching in a graph that does not satisfy the principle of optimality of dynamic programming is also discussed in [6]. However, the scheme proposed in [6] is too general and cannot exploit many properties in a practical design.

Fig. 5. Via spacing violations in a two-layer routing problem can be easily corrected because Property 3 implies that there cannot be any obstacles between the two vias. Therefore, the vias (either one or both depending on other connections to the vias) can be replaced with a wire segment at the metal layer.

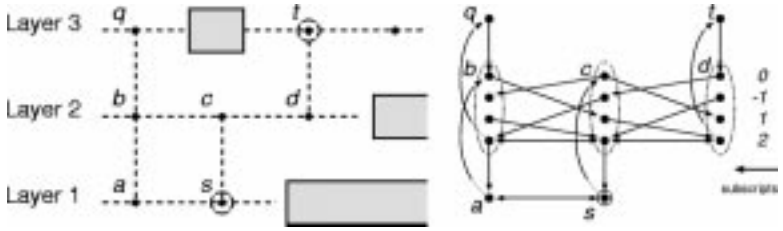

Fig. 6. The extended connection graph G', example in (a), is shown in (b).

In this paper, we will present a heuristic to solve this problem in Section II by using an extended connection graph that embeds the distance to the most recently placed via in a path. While a straightforward implementation of the maze routing algorithm on the extended connection graph can solve the problem, we present in Section III efficient data-structures to implement the maze routing algorithm without the need to preconstruct the extended connection graph. Section IV shows our experimental results. We first show an actual routing example where our algorithm can find a solution, whereas a traditional maze routing algorithm cannot. Next, we show the applicability of our algorithm to a variety of CMOS technologies. Finally, to validate the advantages of our approach, an experiment is set up to compare our algorithm with traditional maze routing algorithm in randomly generated examples. We conclude our paper in Section V. An extended abstract of this work was published earlier in the *Proceedings of the 1999 International Symposium on Physical Design* [8].

II. Extended Connection Graph

There are two basic problems caused by the via rules. One is that a grid position may need to be expanded more than once to find the path. The other is that we need to maintain the distance to the most recently placed via along the path to determine when the next via can be placed. Our solution to this problem is to conceptually create an extended connection graph that embeds the via distance as well as to provide multiple nodes at *each* grid-position so that each grid position can effectively be expanded more than once during the maze routing.

We let K be the minimum number of unit grid-spacings, defined by the maximum common divisor of all minimum length in the design rule, between the vias on *adjacent layers* (e.g., $K = 2$ in Fig. 2). Our idea is to transform each node v in the original connection graph G into $2K$ extended nodes $v_{-(k-1)}, \ldots, v_{-1}, v_0, v_1, \ldots, v_{K-1}, v_K$ in an extended directed connection graph $G' = (E', V')$. The nodes in G at the top and bottom layers are the exceptions, and they are added to G' without being transformed. Each extended node v_i ($|i| < K$) captures the best path that is $|i|$ grid-spacings away from the most recently placed via that is to the left (if $K > i > 0$) or to the right (if $i < 0$) of the current node. The extended node v_K captures the best path that is K or more grid-spacings away from the most recently placed via. The edges in G' are added as follows: 1) if v can traverse to its neighbor u in G, then v_K can traverse to u_K, and $v_i|_{|i|<K}$ can traverse to u_{i+1} if u is to right of v or u_{i-1} if u is to the left of v, and 2) the extended nodes that can be connected *from* a via are those with subscript 0 and the extended nodes that can be connected *to* a via are those with subscript K. Based on the two rules, we can construct the extended connection graph for the example in Fig. 2 as shown in Fig. 6. Notice that the optimal via-rule-correct path is embedded in G' as $s-a-b_0-c_1-d_2-t$. Also notice that the minimum-cost but via-rule-incorrect path is not in G'; i.e., there exists no indexes i, j such that $s-c_i-d_j-t$ is in G'.

The extended connection graph shown in Fig. 6 is valid only for a cross-sectional (2-D) routing region. For multilayer general area routing, the extended nodes must encode the distance from the most recently placed via in both the x-direction and the y-direction. Therefore, each node v in G, that is not in the top or bottom layers, is tranformed to $(2K-1)^2+1$ extended nodes $v_{K,K}$ and $v_{i,j}$ for $i, j = 0, \pm1, \pm2, \ldots, \pm(K-1)$. The first and second indexes (subscripts) represent the (either positive or negative) distance in the x-direction and the y-direction from the most recently palced via, respectively. For each edge $e = (u, v)$ in the original graph G that represents a wire segment (i.e., u and v represent grid-points in the same routing layer) in the x-direction, we add the following edges to G': 1) an undirected edge $(u_{K,K}, v_{K,K})$; 2) directed edges in the positive x-direction: $(u_{i,j}, v_{i+1,j})$ for all possible j, and $i = 0, \ldots, K-2$ and the directed edges $(u_{K-1,j}, v_{K,K})$ for all possible j; and 3) directed edges edges in the negative x-direction: $(v_{i,j}, u_{i-1,j})$ for all possible j, and $i = 0, -1, \ldots, -(K-2)$, and $(v_{-(K-1),j}, v_{K,K})$ for all possible j. Similarly, edges in the y-direction are added. Therefore, an extended node $v_{i,j}$ captures

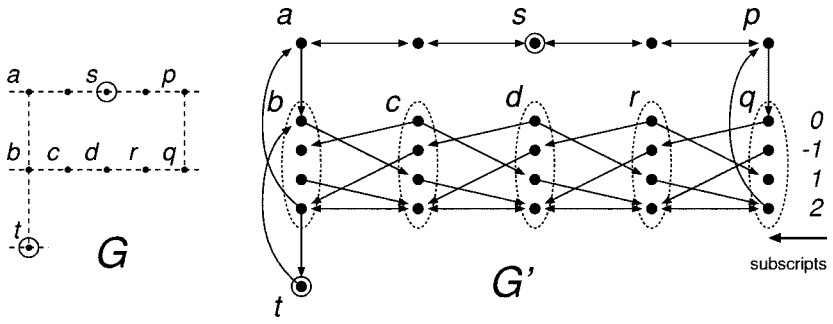

Fig. 7. Example showing the failure of finding the minimum-cost via-rule-correct path in G'.

the best path that is i and j grid-points away from the most recently placed via in the x and y direction, respectively. For each edge $e = (u, v) \in E$ representing a via, (i.e., u and v represent grid-points on adjacent routing layers), we add the edges $(u_{0,0}, v_{K,K})$ and $(v_{0,0}, u_{K,K})$ to G'. If u is on the bottom or top layers, then the edges added to G' are $(u, v_{0,0})$ and $(v_{K,K}, u)$. Finally, notice that K can be different for different adjacent layers. For example, in a four-level routing region, K_1 and K_2 can correspond to the spacing of VIA1 to VIA2 and VIA2 to VIA3, and applied to the transformation of graph nodes on layer 2 and layer 3, respectively.

The number of nodes and edges in G' are $|V'| = O(K^2|V|)$ and $|E'| = O(K^2|E|)$, respectively. However, the via-spacing K need not be very large in practice. For example, the 0.5-μ CMOS technology shown in Table I has a VIA1 to VIA2 minimum spacing of 0.3-μm. Therefore, using a manufacturing grid-spacing of 0.1-μm gives us a reasonably small K value of only 3. Furthermore, we will present efficient data-structures to implement the maze routing algorithm on G' without the need to construct G' before routing begins in the next section.

The extended connection graph G' alone does not *prevent* finding a via-rule incorrect path. For example, the path $s - a - b_0 - c_1 - d_2 - c_2 - b_2 - q$ in Fig. 6 is a feasible path in G' but *not* a via-rule-correct path since the via $a - b_0$ is *not* at least two grid-points away from the via $b_2 - q$. Therefore, we need to apply the following restriction when expanding a node to its neighbors in G'.

Restriction 1: Given the nodes $v_{K,K}, u_{K,K}$, and $q_{0,0}$ in G' where 1) $v_{K,K}$ and $u_{K,K}$ are adjacent nodes, and 2) $q_{0,0}$ is at the position of the most recently placed via along the search path to $v_{K,K}$, then $v_{K,K}$ can only be expanded into $u_{K,K}$ if $u_{K,K}$ is at least K grids away from $q_{0,0}$.

For example, this restriction will prevent exploring the paths $s-a-b_0-c_1-d_2-c_2-\cdots$ in Fig. 6 since d_2 will not be allowed to expand into c_2 due to the presence of b_0. This restriction can be easily implemented when the search path is stored explicitly as a path (described in the next section) rather as "back links" in a traditional maze router. We can now apply the maze routing algorithm in Fig. 3 to G' to find the corresponding minimum-cost via-rule-correct path in G. With the restriction, G' will always return a via-rule-correct path if one is found. However, the restriction places a conditional rule on G' and in effect violates the Principle of Optimality. This will prevent a valid via-rule-correct path to be found in G' even if one exists. For example, Fig. 7 shows a cross-sectional connection graph G and its corresponding extended graph G' (similar to Fig. 6) that contains the via-rule-correct paths $s-p-q_0-r_{-1}-d_2-c_2-b_2-t$. If the edges are uniformed weighted, then node d_2 will be visited by the equal cost paths $s-a-b_0-c_1-d_2$ and $s-p-q_0-r_{-1}-d_2$. If the path $s-a-b_0-c_1-d_2$ is chosen, then no solution will be found in G'. Therefore, node d_2 is a β-node and the via-rule-correct path *may not* be found in G'. In fact, the via-rule-correct path *will not* be found if the via $p-q$ is *more than* $2K$ grid-spacings away from the via $a-b$. On the other hand, if the via $p-q$ is *within* $2K$ grid-points from the via $a-b$, then the via-rule-correct path *will* be found. Therefore, our algorithm is effective in resolving the problem of via-design-rule interactions for vias that are close by but it does not solve the problem of via-design-rule interaction *at a distance*.

III. Searching on the Extended Connection Graph

A straightforward approach to find the optimal via-rule-correct path is to construct G' and apply the maze routing algorithm on G'. This is inefficient because of the overhead in constructing G', and many extended nodes will never be visited. Furthermore, searching on G' is only useful around congested areas. The application of the proposed router is to take over the task of finding short partial paths (i.e., to squeeze through congested areas) when a traditional maze router fails to find a path. Therefore, it is not desirable to construct G' ahead of time. We now show a maze expansion algorithm that does not require G' to be constructed ahead of time. That is, we shall represent G' implicitly, and construct G' on-the-fly during the maze expansion. A similar idea of implicit routing graph representation was also used in a very recent work in [9] for gridless ECO routing.

The algorithm is outlined in Fig. 8 and we will explain the implementation details in the rest of this section.

The maze expansion process traverses G' to find the optimal path. The data needed at each node $v \in V'$ (we have dropped the subscripts here for brevity) during the expansion are: 1) the cost of the best path found to v, and 2) the trace back code to generate that path. Realizing that only a very small subset of the nodes in the expanding wavefront are actively involved in the maze expansion operation, Soukup proposed in [7] to use a *separate* data-structure to maintain the maze expansion information. This way, the size of connection graph nodes can be significantly reduced. This is particularly useful because the extended connection graph nodes can be computed on-the-fly based on the original connection graph and need not be realized at all. Only the

Algorithm: EFFICIENT MULTI-PATH ROUTING ALGORITHM (G, s, t)

```
1   Q ← s;
2   M ← ⟨s, s⟩;
3   while Q ≠ ∅
4       p ← POP(Q);
5       if (p = t) then path found;
6       On-the-fly compute neighboring nodes of p in G';
7       for-each q ∈ NEIGHBORS(p) do
8           EXPAND(p, q, Q, M);
9           M.insert(⟨q, q.path⟩);
10      end for-each
11      M.remove(p);
12  end while
end
```

Fig. 8. By computing the expanded routing graph G' on-the-fly, the algorithm searches for a via-rule-correct path s to t in the original routing graph G using efficient data structure for priority queue Q and dictionary M.

Step	Expands	Priority queue Q $\langle cost, node\rangle$	Hash Dictionary M $\langle pos, path\rangle$
Start		$\langle 0, s\rangle$	$\langle s, s\rangle$
1	s	$\langle 1, c_0\rangle\langle 1, a\rangle$	$\langle c_0, sc\rangle\langle a, sa\rangle$
2	c_0	$\langle 1, a\rangle\langle 2, b_{-1}\rangle\langle 2, d_1\rangle$	$\langle c_0, sc\rangle\langle a, sa\rangle$
			$\langle b_{-1}, scb\rangle\langle d_1, scd\rangle$
3	a	$\langle 2, b_{-1}\rangle\langle 2, d_1\rangle\langle 2, b_0\rangle$	$\langle b_{-1}, scb\rangle\langle d_1, scd\rangle$
			$\langle b_0, sab\rangle$
4	b_{-1}	$\langle 2, d_1\rangle\langle 2, b_0\rangle$	$\langle d_1, scd\rangle\langle b_0, sab\rangle$
5	d_1	$\langle 2, b_0\rangle$	$\langle b_0, sab\rangle$
6	b_0	$\langle 3, c_1\rangle$	$\langle c_1, sabc\rangle$
7	c_1	$\langle 4, d_2\rangle$	$\langle d_1, sabcd\rangle$
8	d_2	$\langle 5, t\rangle$	$\langle t, sabcdt\rangle$
Done			

Fig. 9. On-the-fly expansion of G' for the example in Fig. 2. We show the positions for entries in H symbolically with the node names, and have omitted the cost and visitation information for brevity. Notice that the extended nodes are created only they are visited. For instance, at Step 2, the extended node c_0 is created when s is expanded. Obviously, only a small subset of nodes in G' is created.

information needed during expansion needs to be created and stored in temporary nodes called the *maze* nodes.

The data-structures for implementing the maze expansion information are as follows. We let m be a *maze* node for an extended node v and m consists of a cost (*m.cost*), the position of v (*m.pos*), flags to mark visitations by neighbors (*m.visit*) and the best path (*m.path*) from s to *m.pos*. The path is a sequence of segments and each segment is represented using a two-tuple $\langle d, l\rangle$ where d is the direction and l the length, respectively. The overall path is constructed by tracing from s using the direction and length in each segment. Let M be the set of maze nodes organized as a hash dictionary using *m.pos* as the search key. An entry in the priority queue Q for a maze node m consists of *m.pos* and the computed priority based on *m.cost*. During the maze expansion, an entry is popped from Q that provides the position of the maze-node n that is being expanded so n can be retrieved from M. The position *pos'* of a feasible neighbor in G' that has not been visited, found using *n.visit*, is computed *on-the-fly* based on the original connection graph G. If *pos'* is not a search key in M, then a new maze node m with *m.pos* $=$ *pos'* is added to M; otherwise m where *m.pos* $=$ *pos'* is retrieved from M. If the path to m through n is better, then *m.cost* and *m.path*, and the priority queue are updated accordingly. The visitation flag in m is also marked appropriately. Finally, when n is fully expanded by visiting all its feasible neighbors, it is deleted from M. This is illustrated in Fig. 9 for the example shown in Fig. 2. Our scheme differs from that described in [7] in that M is not organized in [7] and that we store the optimal path directly at each maze node instead of generating back-trace code for every visited node.

Our algorithm effectively allocates memory only for the wavefront nodes in the maze expansion. Therefore, the memory requirement is dependent on the maximum number of wavefront nodes and the size of each path in these nodes. The number of wavefront nodes is influenced by both the maze expansion strategy and the presence of obstacles in the routing region. The size of a path in the maze node is the number of segments. In practice, in a congested layout, the number of wavefront nodes are fairly constant due to the very limited search space.

IV. EXPERIMENTAL RESULTS

We have successfully implemented our maze routing algorithms in comparison with two other routers. One is Iroute [4] in the Magic IC layout editor and the other is a traditional maze router that performs on-the-fly via-rule check and discards all incorrect paths. In Fig. 10, we show a three-layer example similar to the cross-sectional diagram in Fig. 2. The traditional maze router failed to find any path because it expanded c first. Iroute returned via-rule-incorrect path $s-c-d-t$ while our algorithm found the optimal via-rule-correct path $s-b-c-d-t$. Fig. 11 shows a four layer example using MOSIS SCMOS design rule. The path $s-a-b-c-t$ found by Iroute is via-rule incorrect because the via-to-via spacing between b and c is too small. By detecting this incorrect path, traditional maze router found a detoured path $s-a-b_1-c_1-t$. With the ability to expand node b more than once, our algorithm found the optimal path $s-a_1-b_2-c-t$.

The validity of our algorithm depends on Properties 1, 2, and 3 (described in Section I) being true. A survey of a few IC technologies show that they are in fact true as shown in Table II. The size of K is also given in the table. It is very likely that in advanced technology, the minimum spacing rule between adjacent cut layers will remain small, so that the size of the extended connection remains reasonable. Notice that with the on-the-fly expansion scheme described in Section III, it is the number of *visited* extended nodes that impacts the memory and runtime performance of our algorithm. As a result, the impact of K is less than quadratic in practice. We implemented our algorithm and tested it with three examples on a Sun Ultra 1 workstation, as shown in Table III. The x/y dimension for each example ranges

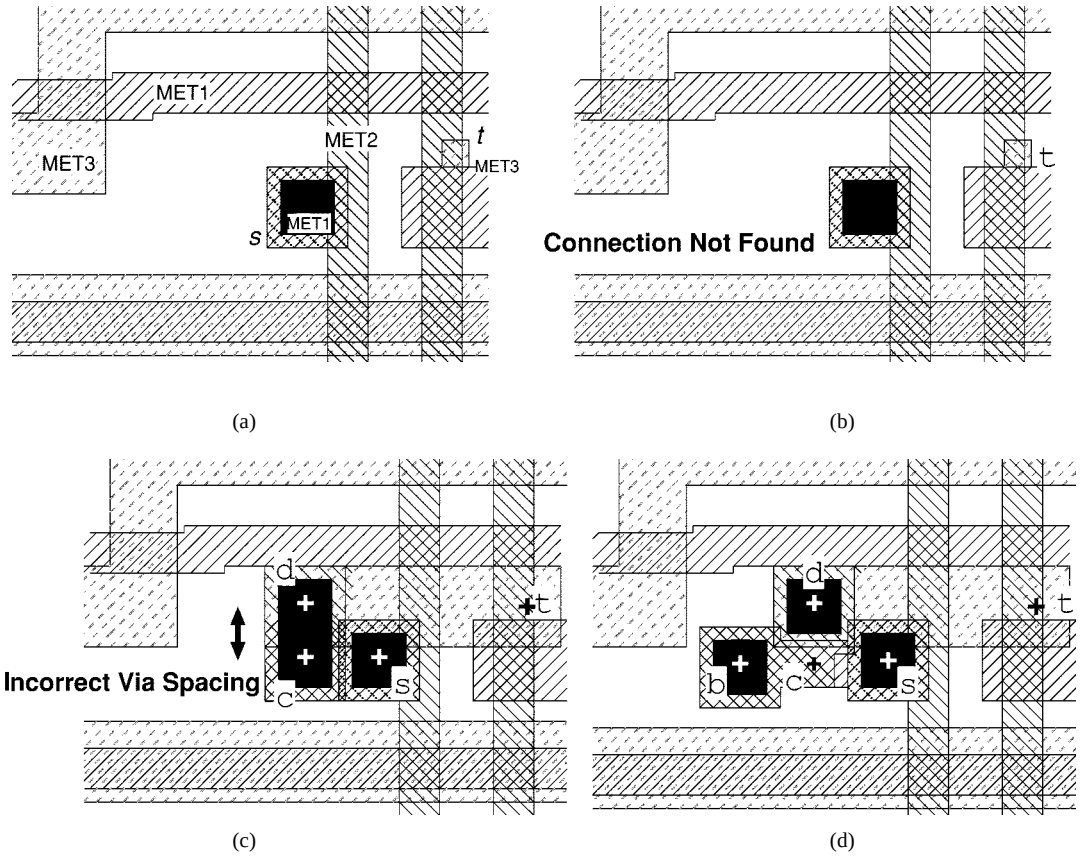

Fig. 10. A case using the 0.5-μm CMOS technology shown in Table II where our proposed routing algorithm found the via-rule correct optimal path. The starting point s is a polysilcon-to-MET1 contact and the target is somewhere to the right of the figure. The layouts are drawn to scale. (a) Routing problem. (b) Traditional maze algorithm result. (c) Iroute result. (d) Our result.

from 200 $\times$ 200 to 200 $\times$500. Our experimental result shows that the run time increase is subquadratic with respect to K.

Our proposed algorithm, although optimized in searching extended graph G', is not intended to replace the traditional maze routing algorithm in current detailed routers. The obvious application of this router is to supplement a traditional router when it fails to find a route in a local congested region. In addition, since our router may be able to find a better route, it can be used, during the "optimized mode" of a routing session, to reroute a path in a region where the traditional router has used too long a detour due to via-rules. Finally, in applications where the region is small and the routing density is high, such as routing within a cell, this router can actually be used for finding all the routes. To determine how much we can improve over the traditional maze algorithm and how often our algorithm can find the best via-rule-correct path, we have designed a set of experiment to compare our algorithm with the traditional maze algorithm. In this experiment, the test cases are examples which contain shortest paths, generated by adding random obstacles in a restricted routing region. By applying real SCMOS design rule, as shown in Table II, the routing results of our multipath algorithm are compared with two other algorithms: traditional maze algorithm without considering via-rules and traditional maze algorithm with consideration of via-rules, as shown in Table IV. Although our algorithm does not guarantee to find an optimal via-rule-correct path, comparing with maze algorithm with consideration of via-rules gives us a rough estimation of how often

TABLE II
EXAMPLE OF CMOS DESIGN RULES

	Feature	Micron rule (μm)		λ rule
		0.5-μm	0.8-μm	SCMOS
w1	Min MET1 width	0.6	1.2	3
w2	Min MET2 width	0.6	1.4	3
w3	Min MET3 width	1.2	2.1	6
w4	Min MET4 width	N/A	N/A	6
e1	Min encl. of VIA1	0.2	0.5	1
e2	Min encl. of VIA2	0.2	0.6	1
e3	Min encl. of VIA3	N/A	N/A	1
s1,s2	Min MET1, MET2 sp.	0.8	1.2	3
s3	Min MET3 spacing	1.2	1.6	4
s4	Min MET4 spacing	N/A	N/A	6
s5	Min CONTACT to VIA1 spacing	0.6	0.8	3
s6	Min VIA1 to VIA1 sp.	0.6	1.2	3
s7	Min VIA1 to VIA2 sp.	0.3	1.2	2
s8	Min VIA2 to VIA2 sp.	0.6	1.5	3
s9	Min VIA2 to VIA3 sp.	N/A	N/A	3
s10	Min VIA3 to VIA3 sp.	N/A	N/A	4
	Unit grid spacing	0.1	0.1	1
	K	3	12	2, 3

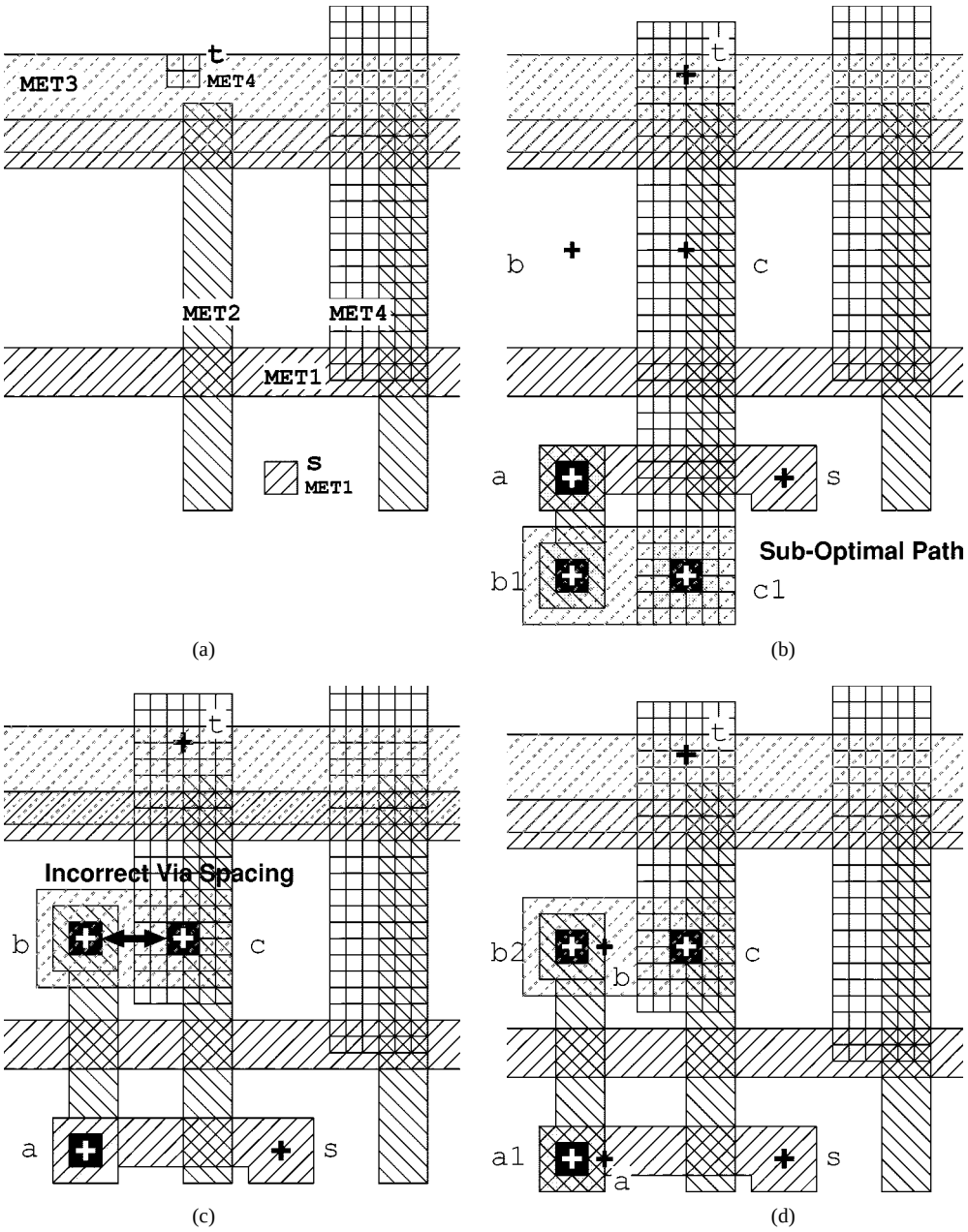

Fig. 11. A case using the four layer SCMOS technology shown in Table II where our proposed routing algorithm found the via-rule correct optimal path. The starting point s is a polysilcon-to-MET1 contact and the target point t is on MET4. The layouts are drawn to scale. (a) Routing problem. (b) Traditional maze algorithm result. (c) Iroute result. (d) Our result.

TABLE III
RUN TIMES ON DIFFERENT K's

Ex.	Search Window (X/Y grids)	# Nets	Run Time (sec) K=3	K=5	K=12
T1	210×560	2	0.45	0.43	3.25
T2	210×210	4	14.9	24.2	41.7
T3	210×560	4	103.9	178.1	344.5

TABLE IV
ROUTING RESULTS ON RANDOMLY GENERATED EXAMPLES

Ex.	Num. Layers	Search Windown x/y grids (λ)	# nets	Improved Examples Total #	Optimal #
test1	3	96 × 30	907	142 / 15.7%	94 / 66.2%
test2	3	96 × 96	807	144 / 17.8%	81 / 56.3%
test3	4	96 × 30	507	201 / 39.6%	160 / 79.6%
test4	4	96 × 96	596	214 / 35.9%	109 / 50.9%

and how much we can improve in a random scenario. Also, comparing with the maze algorithm without considering via-rule tells us how often we can get the optimal solution since such a maze algorithm returns the shortest s to t path in G, with or without regards to via-rules. Our experiment shows that multiple path algorithm finds better solutions than traditional maze routing algorithm in 15%–40% of the random examples. What is more, among these improved cases, there are 50%–80% optimal via-rule-correct paths.

V. Conclusion

We have shown that solutions of the traditional maze routing algorithm can violate practical via-rules in a multilayer routing environment. Furthermore, a straightforward extension to the maze routing algorithm that disallows via-rule incorrect routes may either cause a suboptimal route to be found, or more seriously, cause the failure to find any route even if one exists. We present a heuristic to this problem by embedding the distance to the most recently placed via in an extended connection graph so that the maze routing algorithm has a higher chance of finding a via-rule correct optimum path in the extended connection graph. We further present efficient data-structures to implement the maze routing algorithm without the need to preconstruct the extended connection graph.

References

[1] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[2] C. Lee, "An algorithm for path connections and its applications," *IRE Trans Electron. Computers*, vol. EC-10, pp. 346–365, Jan. 1961.

[3] R. Bellman, *Dynamic Programming*. Princeton, NJ: Princeton Univ. Press, 1957.

[4] M. Arnold and W. Scott, "An interactive maze router with hints," in *Proc. 25th ACM/IEEE Design Automation Conf.*, 1988, pp. 672–676.

[5] W. Schiele, T. Kruger, K. Just, and F. Kirsch, "A gridless router for industrial design rules," in *Proc. 27th ACM/IEEE Design Automation Conf.*, June 1990, pp. 626–631.

[6] A. Tetelbaum, "Generalized optimum path search," *IEEE Trans. Computer-Aided Design*, vol. 14, pp. 1586–1590, Dec. 1995.

[7] J. Soukup, "Maze router without a grid map," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, Nov. 1992, pp. 382–385.

[8] J. Cong, J. Fang, and K. Y. Khoo, "Via design rule consideration in multilayer maze routing algorithms," in *Proc. Int. Symp. Physical Design*, Apr. 1999, pp. 214–220.

[9] ——, "An implicit connection graph maze routing algorithm for ECO routing," in *Proc. ACM/IEEE Int. Conf. Computer Aided Design*, Nov. 1999, pp. 163–167.

Jason Cong received the B.S. degree in computer science from Peking University, Peking, China, in 1985 and the M.S. and Ph.D. degrees in computer science from the university of Illinois at Urbana-Champaign, Urbana, in 1987 and 1990, respectively.

Currently, he is a Professor and Co-director of the very large scal inegration (VLSI) computer-aided design (CAD) Laboratory in the Computer Science Department of University of California, Los Angeles. His research interests include layout low-power VLSI circuits, design and optimization of high-speed VLSI interconnects, FPGA synthesis, and reconfigurable computing. He has published more than 100 research papers and led more than 20 research projects supported by DARPA, NSF, and a number of industrial sponsors in these areas. He served as the General Chair of the 1993 ACM/SIGDA Physical Design Workshop, the Program Chair and General Chair of the 1997 and 1998 International Symposium on FPGA's, respectively, and on program committees of many VLSI CAD conferences, including DAC, ICCAD, and ISCAS. He is an Associate Editor of *ACM Transactions on Design Automation of Electronic Systems*.

Dr. Cong received the Best Graduate Award from the Peking University, in 1985, and the Ross J. Martin Award for Excellence in Research from the University of Illinois at Urbana-Champaign, in 1989. He received the NSF Research Initiation Award and NSF Young Investigator Award in 1991 and 1993, respectively. He received the Northrop Outstanding Junior Faculty Research Award from UCLA in 1993, and IEEE Transactions on Computer-Aided Design Best Paper Award in 1995. He received the ACM Recognition of Service Award in 1997.

Jie Fang received the B.S. degree in computer science and engineering from Tsinghua University, Beijing, China, in 1995. He is currently pursuing the Ph.D. degree in VLSI CAD laboratory of Computer Science Department University of California, Los Angeles.

His research interests are physical design of VLSI circuits.

Kei-Yong Khoo received the M.S. degree in electrical engineering from University of California, Los Angeles, in 1994 and the B.S. degree in electrical engineering and computer science from the Oregon State University, Corvallis, in 1988. He is currently pursuing the Ph.D. degree in the electrical engineering at the University of California, Los Angeles.

From 1988 to 1990, he was a member of technical staff at Mentor Graphic Co., Warren, NJ, where he engaged in the development of the datapath compiler. His research interests include computer-aided design of VLSI circuits and design of high-speed circuits.