

# DUNE: A Multi-Layer Gridless Routing System with Wire Planning <sup>\*</sup>

Jason Cong  
UCLA Computer Science Dept  
Los Angeles, CA 90095  
cong@cs.ucla.edu

Jie Fang  
UCLA Computer Science Dept  
Los Angeles, CA 90095  
jfang@cs.ucla.edu

Kei-Yong Khoo<sup>†</sup>  
Synopsys Corp  
Mountain View, CA 94043  
khoo@ieee.org

## ABSTRACT

In this paper, we present a multi-layer gridless detailed routing system with wire planning for deep sub-micron (DSM) physical designs. It includes a multi-layer gridless detailed routing engine that can efficiently route point-to-point connections, and a wire planning algorithm that uses exact gridless design rules (variable width and variable spacing) to accurately estimate the routing resources and distribute nets into routing regions. The wire planning method also enables efficient rip-up and reroute in gridless routing. Unlike previous approaches which explore alternatives of blocked nets by gradually tightening the design rules, our planning based approach can take the exact gridless rules and resolve the congestion and blockage at a higher level. Our experimental result shows that using the wire planning algorithm in our detailed routing system can improve the routability and also speed up the run time by 3 – 17 times.

## 1. INTRODUCTION AND OVERVIEW

As VLSI technology reaches deep sub-micron dimensions and gigahertz clock frequencies, interconnect has become the dominating factor in determining the performance, power, reliability, and the cost of the overall system, as predicted in the 1997 National Technology Roadmap for Semiconductors (NTRS'97) [1]. Many optimization techniques, including wire sizing (for delay optimization), wire spacing (for delay and noise optimization), etc., have been proposed and shown to be very effective for interconnect optimization [2]. These optimizations impose variable width and variable spacing constraints on the wires of the interconnects.

A traditional routing system is partitioned into two stages: global routing and detailed routing, as shown in Figure 1(a). In global routing, the routing region is partitioned into tiles or channels and a rough route for each net is determined among these tiles to minimize the overall congestion in each

tile. This congestion driven global routing helps to distribute the routing resources and guides the detailed routing, which is carried out in stage two. In detailed routing the exact implementation of the conductive wires is determined for each net according to the design rules. The variable width and variable spacing design rules require a gridless detailed router which does not refrain the wires on predefined uniform grids. However, this two-level approach has two limitations in current VLSI designs. First, current designs may integrate over a hundred million transistors in a single chip. Traditional two-level design may not be able to handle such a big problem size. For example, even with 1000 tiles at the global level, we may still end up with over 100,000 objects in each tile. This presents a very high space and time complexity for the gridless detailed router. Therefore, additional levels of hierarchy are needed. Second, because of the deep sub-micron effects, the delay and noise due to the global interconnects need to be carefully considered during the routing [3]. The first-level tile size needs to match the global interconnect delay and noise so that interconnect optimization methods can be effectively applied at the global level. Traditional two-level routing framework does not address this problem properly.

Given these considerations, we at UCLA have been developing a routing system for high-performance DSM designs using three levels of routing hierarchy, as shown in Figure 1(b). The first stage is a performance-driven global routing that plans out nets according to the the delay and noise requirements with global congestion control. The works in this area include a performance-driven global router using high-performance routing topologies and optimal wire sizing [4] and a noise-constrained wire spacing and track assignment algorithm for global routing refinement [5; 6]. The second stage is a congestion-driven wire planning algorithm that plans the route of each net based on a detailed modeling of the routing resources and the individual requirement of each net (variable width and variable spacing). A gridless detailed routing algorithm is applied in the third stage to carry out the detailed implementation of the planning result from the second stage. In our three-level design flow, stage two and three are closely integrated. If a net cannot be routed, it can be sent back to wire planner to be re-planned. Thus, these two stages form a gridless detailed routing system. Most gridless detailed routing systems lack the wire-planning capability with exact routing resource modeling.

Most traditional detailed routing algorithms assume uniform underlying grids to simply the problem [7; 8; 9]. However,

<sup>\*</sup>This work is partially supported by DARPA/ETO under Contract DAAL01-96-K-3600 managed by the US Army Research Laboratory and NSF under grant MIP-93-57582.

<sup>†</sup>This work was performed when the author was with UCLA VLSI CAD Lab

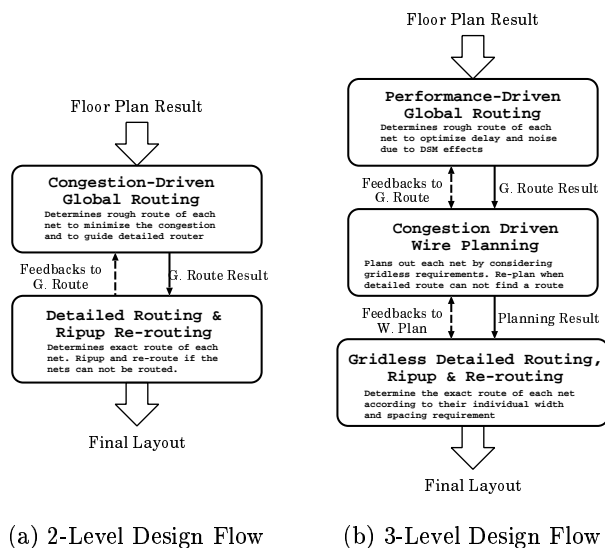


Figure 1: Routing System Design Flows

this uniform-grid approach is not efficient to handle variable width and variable spacing designs because a very fine grid may be needed, as shown in Figure 2(b). Due to the requirement of off-grid routing induced by variable width and variable spacing design rules, several gridless detailed routing algorithms have been published during the past years. In general, there are two major approaches for the gridless routing. One approach uses the tile-based algorithms [10; 11; 12]. The routing region is partitioned into tiles induced by the boundaries of obstacles and the routing problem is reduced to searching a tile-to-tile path among these tiles, represented by a corner-stitching data structure [13], as shown in Figure 2(c). The other approach uses the connection graph based algorithm [14; 15; 16]. A connection graph is built based on the obstacles in the routing region and usually the special width and spacing requirements for the net to be routed are encoded in the graph. A maze searching algorithm is applied on the graph to find the route, as shown in Figure 2(d). In general, searching a tile-to-tile path is faster due to the smaller number of tiles and the use of corner stitching data structure. However, tiles are more complex to manage, and a tile-to-tile path needs post-processing to obtain a final design rule correct route, and there are some difficulties in using the tile-based algorithm for multi-layer routing with more complex design rules.

When a net cannot be routed in detailed routing, rip-up and reroute is carried out to free out routing resource and re-do the routing. Many algorithms have been proposed on rip-up and reroute strategies [17; 18; 19]. However, one of the fundamental assumptions they have is that there exists uniform underlying grids and all net segments can be simplified as a zero width lines on these grids. This makes it easy to model the resources in the routing region and simplifies the operation to exchange the resources between nets. However, this assumption does not hold in variable width and variable spacing routing. Net segment cannot be simplified to zero width lines and rerouting a net may affect multiple nets in the region. One simple example is shown in Figure 3. In this

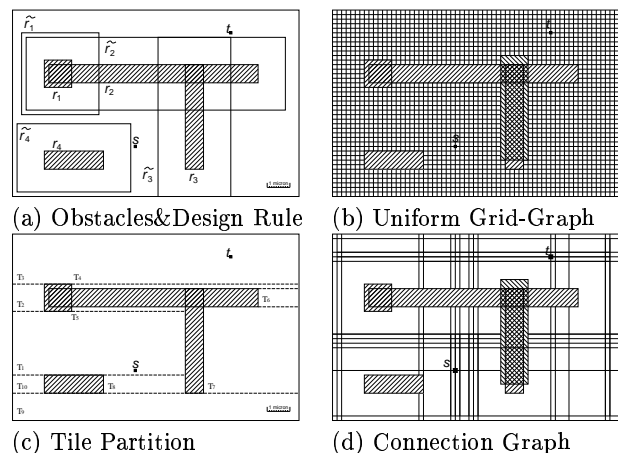


Figure 2: Different approaches for detailed routing: A routing region with obstacle is show in (a). (b) shows a uniform grid approach which uses very fine grids due to the width and spacing rules. The routing region is cut into tiles according to the obstacle boundaries for a tile based approach in (c) and a connection graph is constructed in (d) based on the design rules.

case, an accurate model of local resource and the flexibility to select the re-routes globally are both needed to find the solution.

We believe that a successful gridless routing system requires not only an efficient multi-layer detailed routing algorithm as the routing engine, it also requires a nicely crafted framework that consists of a congestion driven planning tool which can schedule out all the nets together while takes the width and spacing requirements of each net into consideration. It shall have efficient rip-up and reroute capabilities when some nets cannot be routed. However, little progress has been reported in this area in the research community. Some solutions have been attempted by the EDA vendors. One of the notable one is the IC Craftsman from Cooper and Chyan Technology (now part of Cadence). It offers a great deal of flexibility due to its multiple iterations of rerouting, but this approach may not scale well to future IC designs with billions of transistors on a chip.

In this paper, we propose a multi-layer gridless routing system with wire planning. It features an efficient connection graph based maze searching algorithm as the gridless routing engine and a *wire planning* algorithm as the global-planner for the routing engine. The whole system provides an efficient solution to variable width, variable spacing gridless detailed routing problem. Our congestion-driven wire planning algorithm not only distributes nets into routing regions prior to detailed routing, but also enables efficient rip-up and reroute by re-planning nets during the detailed routing. When a net cannot be routed, it is sent back to be re-planned by the wire planning algorithm. This integration of planning and routing algorithms helps the planning algorithm keeping up-to-date resource information in the routing region and enabling the routing algorithm to change the route globally. In Section 2, we briefly review the connection-graph based detailed routing algorithm in our routing system. In Section 3, we propose our congestion driven gridless planning algorithm which effectively plans each net to constrain

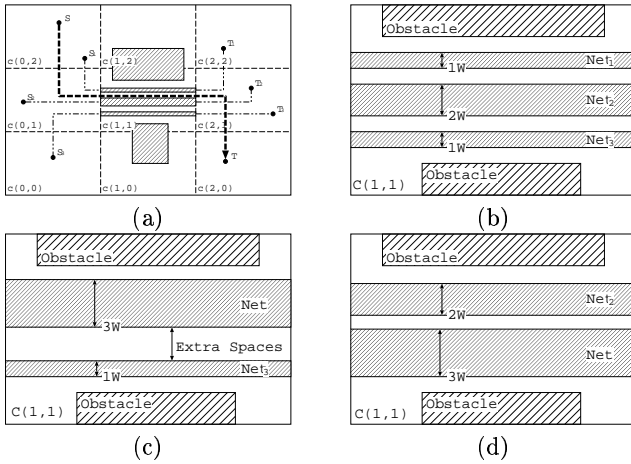


Figure 3: Difficulties of rip-up and reroute in gridless routing: A  $3W$  net is scheduled to be routed through a congested region, as shown in (a). There are three previously routed nets with  $1W$ ,  $2W$  and  $1W$  width respectively, as shown in (b). It is obvious that ripping up any net will not release enough resources for the new net. What is more, picking up different nets have different affects. For instance, if net 1 and 2 are removed, there will be extra spaces after route the new net, as shown in (c). The *best* solution here is to remove net 1 and 3 and reroute net 2 and the new net, as shown in (d).

individual net's searching space and minimize global congestion. The interaction between detailed routing algorithm and planning algorithm which provides an effective rip-up and reroute capability is discussed in Section 4. Finally, the effectiveness of our algorithm is validated with experimental results in Section 5.

## 2. POINT-TO-POINT GRIDLESS ROUTING ENGINE

The fundamental problem in detailed routing is to seek a design-rule-correct path between two given points in the routing region. This problem can be solved by a point-based approach where the routing area is conceptually populated with *feasible points*. The feasible points and their neighborhood information can be abstracted as nodes and edges in a *connection graph* so that the point-to-point connection can be found using the shortest path (Dijkstra [20]) algorithm. Since the connection graph can be very large and complex due to the high-performance designs and the gridless design rules that are being used. We use an *implicit* representation which effectively takes gridless design rules (variable width and variable spacing among different nets) and compute the connection graph on-the-fly. Auxiliary data structures that can efficiently answer maze expansion queries are also developed. Part of the results in this section was also presented in [21] as part of an efficient ECO router.

### 2.1 Simplified Connection Graph

The first problem that we need to address in realizing a gridless detailed router using a point-based approach is how to construct a connection graph that is simple yet contains the optimal route, if one exists. Many algorithms simplify

the connection graph [22; 23; 16; 14] at the expense of very costly pre-construction and representation. Therefore, their usefulness is limited for large designs.

In our gridless detailed routing algorithm, we use a connection graph called *Non-Uniform Grid Graph*  $G_S$ , based on the expansion of rectangular obstacles in the routing region according to wire/via width and spacing rules. The obstacles in the routing region can be most conveniently defined as a set of, possibly overlapping, rectangles at different layers  $R = \{r_1, r_2, \dots, r_{N_R}\}$ . The layout design rules create an obstruction zone [24] around each obstacle where the centerlines of wires and center of vias cannot be placed. We let  $\tilde{R}$  and  $\tilde{R}^v$  be the set of rectangles in  $R$  expanded according to wire width/spacing and via width/spacing rules, respectively. A *Non-Uniform Grid Graph* is defined as follows:

**Definition:** Given a multi-layer routing problem with the obstacle set  $R$ , a source  $s$  and a sink  $t$ . A *Non-Uniform Grid Graph*  $G_S$  is an orthogonal grid graph where its  $x$  grid locations are the vertical boundary locations of  $\tilde{R}$  and  $\tilde{R}^v$  plus the  $x$  locations of  $s$  and  $t$ . Similarly, we can define the  $y$  grid locations, as shown in Figure 2(d).

$G_S$  is a strong connection graph, that is, it guarantees to contain a shortest path from  $s$  to  $t$  if any such connection exists among the set of obstacles  $R$ , according to design rules [21]. The biggest advantages for non-uniform grid graph is that the gridded nature of  $G_S$  makes it very easy to come up with an implicit representation which is both highly compressed in storage and efficient in supporting maze-routing-related queries.

### 2.2 Implicit Representation of Connection Graph

Instead of pre-compute and store  $G_S$  explicitly. We use an implicit representation of the connection graph. Two sorted arrays  $X_S$  and  $Y_S$  are used to store the  $x$ -coordinates and  $y$ -coordinates of the grid-lines, respectively. In the implicit representation, we compute the graph nodes and the edges on-the-fly. The computation, a *query*, consists of two steps: first, compute the possible position of the neighbor, and second, determine the feasibility of a point. The first step is fairly easy to solve: If the  $x$ -coordinates of the current point corresponds to  $X_S[i]$ , the  $x$ -coordinates of the horizontal neighboring points is either  $X_S[i + 1]$  or  $X_S[i - 1]$ , which can be found in  $O(1)$  time. Similarly, we can easily find the vertical neighboring nodes' position using  $Y_S$ . However, the feasibility of a point is not a trivial problem. A point is feasible for placing a wire or via if it is not enclosed by the applicable expanded rectangles in  $\tilde{R}$  or  $\tilde{R}^v$ . Therefore, finding the feasibility of a point requires a point enclosure query into the set of expanded rectangles:

**Point enclosure problem:** Given a set of rectangles  $R = \{r_i \mid i = 1, 2, \dots, N_R\}$  and a point  $v$ , returns the set of rectangles that contain  $v$ .

### 2.3 Efficient Query Data Structure

The feasibility check answers a simple question whether a point falls into any expanded rectangles. However, the nature of gridless detailed routing make this problem not easy to solve. First, the data structure needs to represent a fairly large and, possibly, congested region which may contain *huge* amount of rectangular objects. Second, the queries

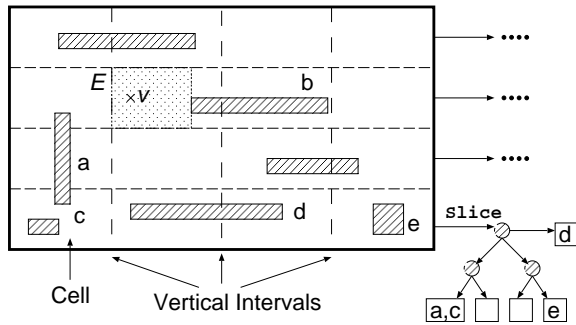


Figure 4: Slit+Interval Tree: The horizontal slices are cut into cells by vertical intervals. Rectangular obstacles are stored on cut lines or leaf cells. The empty-rectangle  $E$  is generated as a result of a query to point  $v$ .

are being made many times during the routing. So it must be answered *very fast*.

We apply a simple yet very efficient two-dimensional data structure for point-enclosure querying, as shown in Figure 4. The first level data structure is a “slit-tree” [25; 26] that recursively bisects the layer into slices along the preferred direction. Rectangles intersecting or overlapping a common slice are managed by a second level data structure, which is an “interval-tree” [27]. There are three advantage of this two-level data structure. First, the storage space is linear to the number of obstacles in the region. Second, the average query time into the data structure is constant time. Last, we store the non-expanded rectangles directly and compute the expansion on-the-fly in the query.

We further exploit the locality of the point enclosure query due to the point-to-point expansion nature in maze routing by storing latest query results in a *cache* data structure, independent of the query data structure. The basic operation of maze routing is to expand node by node. So the nature of its queries is local rather than random — the queries propagate from the source node location and each time goes to a neighboring location not far away. We can exploit this locality by recording previous query results in a *cache*: a small fixed size vector of rectangles from recent query results.

## 2.4 Advantages and Limitations

Our detailed routing engine, featuring an implicit representation of a non-uniform grid graph for gridless maze routing algorithm and a novel 2-D query data structure combined with a cache structure, is a very efficient multi-layer gridless routing algorithm. Due to these features, it has been successfully applied to the ECO type of routing where the routing problem is very big and a truly gridless router is needed [21].

However, to our experience, applying the gridless routing algorithm itself on a net-by-net approach is very inefficient to route multiple nets in the overall detailed routing problem. Without global guidance, the maze searching algorithm tends to search every possible locations. Thus, the algorithm is very slow and consumes a lot of memories. What is more, without a pre-planning, previously routed net may block later nets due to local congestion in a net-by-net approach.

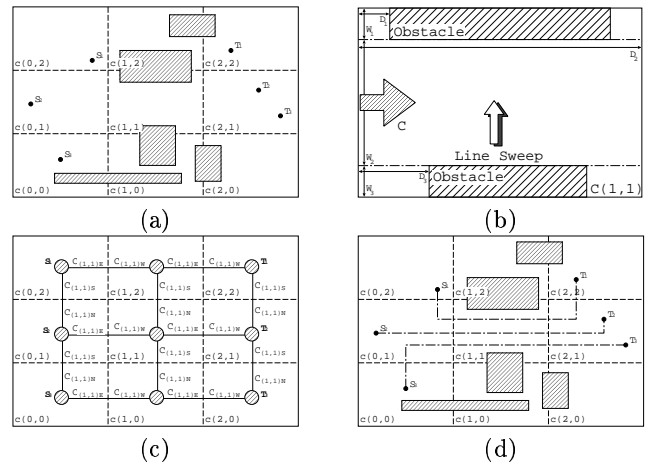


Figure 5: Planning graph construction and wire planning: the routing region is first partitioned into tiles, as shown in (a) and the weighted capacities of each cell is computed using a line-sweeping algorithm (b). Wire planning graph  $G$  is a gridded graph and each node corresponds to a tile. The capacity of the graph edge is the capacity of each tile, as shown in (c). The algorithm finds planning result for each net by searching a minimum weighted path in  $G$ , as shown in (d).

## 3. CONGESTION-DRIVEN WIRE PLANNING

To overcome the problems of a net-by-net approach using maze algorithm, we propose a planning algorithm that has the following three features. First, it spreads out nets to reduce overall congestion thus improves routability. Second, it constrains each net’s searching space into pre-assigned regions to speed up the run time. Third, it provides an accurate topology for each net in determining its final route or reroute if needed. We call this algorithm a *wire planning* algorithm. Major steps in our wire planning algorithm are highlighted in Figure 5. There are two features which make the planning algorithm very useful in a gridless detailed routing environment. First, it uses an accurate model to estimate the routing resource in a multi-layer gridless routing environment. It can accommodate different wire widths and spacings for different nets. Details of the resource modeling and planning graph construction are discussed in Section 3.1. Second, it uses an multi-iteration planning method to overcome the net ordering problem which is discussed in Section 3.2.

### 3.1 Partitioning of Routing Regions and Modeling of Routing Resources

Prior to planning the nets, the routing region is partitioned into tiles. Each tile is of fixed height and width. A three-dimensional (3-D) planning graph,  $G$ , is built based on these tiles — each graph node corresponds to a tile and the edges link neighboring tiles, as shown in Figure 5 (c). Each net is planned through these tiles by finding a tile-to-tile path on the planning graph. However, the wire planning algorithm will not be useful without an accurate estimation of the routing resources in each tile. Due to the gridless nature of our routing problem, we cannot simplify the routing resources as the number of grids or routing tracks. So we are using

the actual dimensions of the obstacles in the tile to compute the routing resources, the *capacity*  $C$ , on the tile boundary. As part of the detailed routing flow, our wire planning algorithm has the advantage of knowing the exact situations in the routing region: prerouted wires and pins. The boundary capacity of a tile is a weighted length of the cut-line in between the tiles. Using a line sweeping algorithm [28], we are able to get the accurate blockage information within each tile. The sweeping algorithm cuts the routing region into horizontal (or vertical) empty rectangles, called *slices*, and  $W_i$  and  $D_i$  are the width and depth of a slice  $S_i$ , as shown in 5(b). The tile depth is  $D$  and the boundary capacity is a *weighted* sum of empty slides' width along the tile boundary computed by the following formula:

$$C = \sum_i W_i \times D_i / D \quad (1)$$

The inter-layer capacity of a tile, which corresponds to the resources taken by vias, is computed by the sum of the area of all empty slices in the tile.

### 3.2 Planning of Nets

Before detailed routing, each net is planned one-by-one in the routing region. We use a maze searching algorithm to find a minimum cost path for each net in  $G$ . The cost of a path is the sum of the edge costs along the path. Each edge cost is determined by a weighting function based on the total consumption, including resources taken by previous planned nets and the sum of actual wire width and spacing of the planned net, and the edge capacity. Several cost functions, as presented in [4], were tested and a slope function was finally chosen based on the experimental results. After the path is found, the edge consumptions along the tiles it goes through are updated by the sum of width and spacing of the net.

Local congestion and net ordering could be a problem in our net-by-net approach. We use a negotiation-based algorithm [19] to plan the nets in multiple iterations to minimize local congestion and to avoid the net ordering problem. After one iteration of planning all the nets, the congestion of each tile is computed. An intrinsic penalty is assigned to the tile based on the congestion so that during the next iteration of planning, routes can be directed away from the potentially congested tiles which are assigned with higher penalties. Instead of finding a "good" ordering for the nets, a simple heuristic is used to determine the net ordering: those nets which go through more congested regions or have longer detours are prioritized and will be re-planned first in the next iteration. Within each iteration, each net is planned with the updated planning graph based on previous planning results. The planning iteration terminates when certain criteria are met: either the global congestion and each net's planning result (number of bends and the estimated wire length based on the tiles it is planned through) are optimized or the whole planning process has gone through the number of predetermined iterations.

By doing the planning, the routing resource is assigned to each net and local congestion is minimized by weighting the edges of the routing graph. The topology of each net is also determined by the tile-to-tile path. The searching space for the maze-based detailed router is constrained in these pre-planned tiles, called *allowed regions*. This speeds up the searching for final route in each net. However, if no design-

rule-correct connection can be found in the allowed regions, a *Rip-up and Replan* will take place between wire planning and detailed routing, as presented in Section 4.

## 4. RIP-UP AND REPLANNING

### 4.1 General Rip-up and Reroute Approaches

Traditional rip-up and reroute algorithms assume uniform underlying grids. In general, there are two classes of rip-up and reroute algorithms depending on the kind of routes that the router is allowed to create:

- always maintain the correct design rule for all routes;
- or, allows routes with temporary design-rule violations.

There are some limitations if we strictly enforce design rule correctness in every step during routing. The result will rely heavily on the ordering of nets, as previously routed nets become obstacles for later ones. The rip-up and reroute algorithm has to be *smart*, or at least *fair* in selecting proper net orders. However, there is no obvious solutions other than simple heuristics and trial-and-error methods. The problem here is that it lacks a global control over the nets.

The second type of rip-up and reroute algorithm is more flexible since by allowing design-rule incorrect routes, one can at least attempt to route all the nets and obtain a *global* picture of where the congested area and the free spaces are. In [18], the design rule violations in a gridded environment are categorized as two cases: *cross* and *touch*. However, in a gridless routing environment, the routing resources and previously routed wires cannot be simplified as grids or tracks. Thus, a *cross* or a *touch* is not well defined. A more accurate model is needed for rip-up and reroute in gridless detailed routing. Most available gridless routers allow design-rule-violations during routing and try to clean up in a multi-pass approach. The violation in this case is not necessarily overlapping wires (as in a gridded router) but simply reduced clearance. Then in each passes the design rule is tightened. However, this is a very costly approach because a complete solution is found for all the nets in each iteration. Obviously, for the current large designs, we need a faster algorithm to solve the rip-up and re-route problem.

### 4.2 Reroute with Up-To-Date Congestion Information

As we mentioned, it is very difficult to represent illegal routes in a gridless routing environment. Also, the trial-and-error method normally takes very long time to run. In our gridless detailed routing algorithm, we use wire planning to guide the rip-up and reroute. There are several advantages to let the wire planning to pick the reroutes. First, wire planning has a more global picture of the routing resources. It is easy to avoid local congested regions and pick a global alternative to route. Second, wire planning has accurate local informations. It not only knows the locally routed nets and also other planned nets in the region. It is easy for the planning algorithm to balance the current consumptions and future needs. Last, it is very fast. Searching through planning cells is much efficient than finding an actual route using the detailed routing algorithm. We call the wire planning algorithm used in rip-up and reroute a *re-planning*. The re-planning is carried out immediately after the detailed router fails to find a connection instead of being

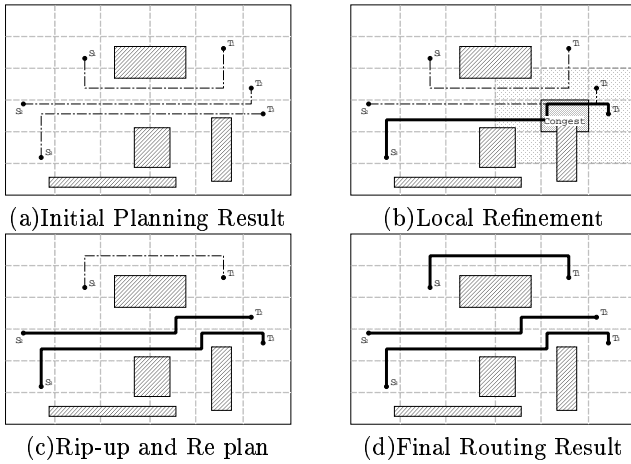


Figure 6: Re-planning strategies: An initial planning result is shown in (a). However, due to the local congestion after routing net 3, a local refinement algorithm is used to reroute net 2, as shown in (b). A rip-up and re-plan method is used to reroute net 1, as shown in (c). Final routing result is shown in (d).

postponed till all the nets are tried once. This is because the emphasis of re-planning is to plan ahead. So it is always good to execute it as early as possible instead of re-planning after most nets are routed.

The re-planning is similar to the initial wire planning that it partitions the region into tiles and build the routing graph to find an alternative route for the failed net. However, due to the updates of routing regions by detailed router, one of the key operations in re-planning is to build the up-to-date congestion information on the routing regions. Although previous planning result gives us a fairly accurate estimation of the resource consumptions in each region. An updated information after partial detailed routing is important for us to make decisions on rerouting. We use the same line sweeping algorithm in initial planning to compute updated capacities in each planning region.

During the reroute phase, we apply two methods to find the alternative route for the blocked net, based on the updated congestion information. One is *local refinement*, where the allowed region at blocked tile is expanded to allow more flexibility in the local area. The other method is to find an alternative plan for the net. We are using the same underlying planning engine to find an alternative plan on the weighted graph. Since the previous planned path fails to find a route, the regions along the previous path are given extra penalties to guide new routes away from it. The re-planned result is then given back to detailed router to search for the final connection. An example showing the rip-up and re-plan methods is shown in Figure 6.

Our re-planning strategy, although fairly simple in control flow, is very effective due to its accurate estimation of routing resources. Our approach is also unique that we are addressing the rip-up and reroute problem from a planning perspective. This avoids the difficulties of representing illegal routes in multi-layer gridless routing and can potentially speeds up the algorithm comparing with gradual tightening the design rule approach.

Table 1: Examples Used for Detailed Routing

Example	Dimension ( $\mu\text{m}$ )	Layers	Total Nets
Block	500 $\times$ 500	3	496
Mcc1-4	39000 $\times$ 45000	6	4004
Mcc1-4c	39000 $\times$ 45000	4	4004
Raytheon	2760 $\times$ 1560	3	430

Table 2: Routing Results without Wire Planning

Example	Routed Nets		Run Time (s)
	Num.	Percent (%)	
Block	489	98.6	4500.6
Mcc1-4	3939	98.4	9499.6
Mcc1-4c	3931	98.2	5621.0
Raytheon	409	95.1	518.8

## 5. EXPERIMENTAL RESULTS

We have implemented our detailed routing system featuring a wire planning guided gridless detailed routing algorithm and a rip-up, re-plan algorithm. Several multi-layer variable width, variable spacing examples are used to test our algorithm, as shown in Table 1. These examples are either chip-level designs (such as *Block*) or MCM examples (such as the *Mcc* examples and *Raytheon*). So most of the nets in these examples are fairly long and this make the detailed routing problem more difficult. The experimental results are collected on a dual 360Mhz CPU Sun Ultra-60 workstation with 1GB of memory. Since there is no state-of-art multi-layer gridless detailed routing systems available in public domain, we compare our results in Table 2 with a simple net-by-net approach using the single net routing engine presented in [21].

Table 3 shows a comparison of wire planning with no wire planning. Our wire planning algorithm first plans out every nets and, when a net cannot be routed in detailed routing phase, re-plans the failed net using updated congestion information in each cells. The experiment shows that wire planning algorithm can improve the routability while dramatically speeds up the detailed routing algorithm by 3 to 17 times.

An important parameter that determines the quality and speed of wire planning is the planning-cell size. In Table 4 we show experimental results on different cell sizes. Please note that since the examples we are using are variable width variable spacing examples. The width of *track* is the minimum wire width plus wire spacing among *all* layers and among all net. Our results shows that when a design is compacted, such as the *Mcc1*, a smaller cell size will help to get a better routability. If the design is relative sparse, we can choose a slightly larger cell size to save planning time.

## 6. CONCLUSION AND FUTURE WORKS

In this paper, we show that a variable width variable spacing detailed routing problem is difficult that a maze based gridless searching algorithm is slow and traditional rip-up and reroute algorithms cannot be efficiently applied. We propose a detailed routing framework which features a wire planning algorithm and a multi-layer gridless detailed routing engine. The wire planning algorithm is applied both prior to the

Table 3: Routing Results with Wire Planning

Example	Routed Nets		Run Time	
	Num.	Percent(%)	(s)	speedups
Block	496	100.0	270.0	16.7
Mcc1-4	3998	99.9	1365.1	7.0
Mcc1-4c	3978	99.4	1508.5	3.7
Raytheon	418	97.2	172.0	3.0

Table 4: Comparison on Different Cell Size

Example	Cell Size # tracks	Routed Nets		Run Time (s)
		Num.	Percent(%)	
Block	5	496	100.0	270.0
	10	496	100.0	215.6
	20	492	99.2	383.9
Mcc1-4	5	3998	99.9	1365.1
	10	3993	99.7	1456.2
	20	3982	99.5	2169.3
Mcc1-4c	5	3978	99.4	1508.5
	10	3971	99.2	1798.0
	20	3956	98.8	2709.2
Raytheon	5	418	97.2	172.0
	10	420	97.7	108.1
	20	419	97.4	156.5

detailed routing to plan out each net and, when a net is blocked, to rip-up and re-plan the blocked net. Our experiments show that the detailed routing system, compares to a net-by-net approach using an efficient gridless detailed router, is 3 – 17 times faster while the completion rate is also improved. These improvements are critical for applying the gridless detailed routing system in current and future VLSI designs where a true variable width and variable spacing router is needed.

Our future work includes further improving our wire planning algorithm and fine tuning of rip-up and re-routing algorithm. Comparisons with state-of-art commercial routing systems will also be made if possible.

## 7. REFERENCES

- [1] Semiconductor Industry Association, *National Technology Roadmap for Semiconductors*, 1997.
- [2] J. Cong, C.-K. Koh, and P. Madden, "Performance optimization of VLSI interconnect layout," *Integration, the VLSI Journal*, vol. 21, no. 1-2, pp. 1-94, 1996.
- [3] J. Cong and D.-Z. Pan, "Interconnect estimation and planning for deep submicron designs," in *Proc. 36th Design Automation Conference*, pp. 507-510, Jun 1999.
- [4] J. Cong and P. Madden, "Performance driven multi-layer general area routing for PCB/MCM designs," in *Proc. 35th Design Automation Conference*, pp. 356-361, Jun 1998.
- [5] C. Chang and J. Cong, "Cross talk noise control in gridless general-area routing," in *Proc. ACM/IEEE International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems (TAU)*, pp. 117-122, Mar 1999.
- [6] C. Chang and J. Cong, "Pseudo pin assignment with crosstalk noise control," in *Proc. International Symposium on Physical Design*, Apr 2000. to be appeared.
- [7] C. Lee, "An algorithm for path connections and its applications," *IRE Trans Electronic Computers*, vol. EC-10, pp. 346-365, 1961.
- [8] F. Hadlock, "A shortest path algorithm for grid graphs," *Networks*, vol. 7, no. 4, pp. 323-334, 1977.
- [9] J. Soukup, "Fast maze router," in *Proc. 15th Design Automation Conference*, pp. 100-102, 1978.
- [10] M. Sato, J. Sakanaka, and T. Ohtsuki, "A fast line-search method based on a tile plane," in *IEEE International Symposium on Circuits and Systems*, pp. 588-591, May 1987.
- [11] A. Margarino, A. Romano, A. De Gloria, F. Curatelli, and P. Antognetti, "A tile-expansion router," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 507-517, Jul 1987.
- [12] L.-C. Liu, H.-P. Tseng, and C. Sechen, "Chip-level area routing," in *Proc. International Symposium on Physical Design*, pp. 197-204, Apr 1998.
- [13] J. Ousterhout, "Corner stitching: a data-structuring technique for VLSI layout tools," *IEEE Trans. Computer-Aided Design*, vol. CAD-3, pp. 87-100, Jan 1984.
- [14] Y. Wu, P. Widmayer, M. Schlag, and C. Wong, "Rectilinear shortest paths and minimum spanning trees in the presence of rectilinear obstacles," *IEEE Trans. Computers*, vol. C-36, pp. 321-331, Mar 1987.
- [15] T. Ohtsuki, "Gridless routers — new wire routing algorithms based on computational geometry," in *Proc. International Conference of Circuits and Systems*, 1985.
- [16] S. Q. Zheng, J. S. Lim, and S. Iyengar, "Finding obstacle-avoiding shortest paths using implicit connection graphs," *IEEE Trans. Computer-Aided Design*, vol. 15, pp. 103-110, Jan 1996.
- [17] Y.-L. Lin, Y.-C. Hsu, and F.-S. Tsai, "Silk: a simulated evolution router," *IEEE Transactions on Computer-Aided Design*, vol. 8, pp. 1108-1114, Oct 1989.
- [18] K. Kawamura, T. Shindo, T. Shibuya, H. Miwatari, and Y. Ohki, "Touch and cross router," in *Proc. of IEEE Conference on Computer-Aided Design*, pp. 56-59, Nov 1990.
- [19] L. McMurchie and C. Ebeling, "Pathfinder: a negotiation-based performance-driven router for FPGAs," in *Proc. of ACM Symposium on Field-Programmable Gate Array*, pp. 111-117, Feb 1995.
- [20] E. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269-271, 1959.
- [21] J. Cong, J. Fang, and K. Khoo, "An implicit connection graph maze routing algorithm for ECO routing," in *Proc. International Conference on Computer Aided Design*, pp. 163-167, Nov 1999.
- [22] J. Cohoon and D. Richards, "Optimal two-terminal  $\alpha - \beta$  wire routing," *Integration, The VLSI Journal*, vol. 6, pp. 35-57, May 1988.
- [23] J. Jaja and S. Wu, "On routing two-terminal nets in the presence of obstacles," *IEEE Trans. Computer-Aided Design*, vol. 8, pp. 563-570, May 1989.
- [24] W. Schiele, T. Kruger, K. Just, and F. Kirsch, "A gridless router for industrial design rules," in *27th Design Automation Conference*, pp. 626-631, Jun 1990.
- [25] I. Kato, S. Ohhira, and Y. Hisatomi, "A method of pattern data management of PWB layout system," in *Proc. 35th Annual Convention IPS Japan*, pp. 2429-2430, 1987.
- [26] E. Kuh and T. Ohtsuki, "Recent advances in VLSI layout," *Proc. of the IEEE*, vol. 78, pp. 237-263, Feb 1990.
- [27] H. Edelsbrunner, "A new approach to rectangle intersections," *International Journal of Computer Mathematics*, vol. 13, no. 3-4, pp. 209-229, 1983.
- [28] J. Nievergelt and F. Preparata, "Plane-sweep algorithms for intersecting geometric figures," *Communications of the ACM*, vol. 25, Oct 1982.