Performance Driven Multi-Layer General Area Routing for PCB/MCM Designs *

Jason Cong and Patrick H. Madden
UCLA Computer Science Department
Los Angeles, California, 90095

March 27, 1998

Abstract

In this paper we present a new global router appropriate for Multichip Module (MCM) and dense Printed Circuit Board (PCB) design, which utilizes a hybrid of the classical rip-up and reroute approach, and the more recent iterative deletion[9] method. The global router addresses performance issues by utilizing recent results in high performance interconnect design, while still effectively minimizing global congestion.

With experiments on the maze-routing component of our global router, we show that the choice of routing cost functions can have a significant impact on final solution quality. The results of a number of previously proposed routers may be improved dramatically by adopting the cost functions we suggest here. We also find little evidence of the "net ordering problem" when our cost functions and routing model are applied. The iterative deletion method is shown to improve global solution quality, particularly when high performance interconnect is required. We evaluate the performance of our global router by comparing the congestion of routes produced by our global router to those of a well known MCM router, V4R [14].

Our global router, MINOTAUR, supports arbitrary numbers of routing layers, differing capacities for each layer, pre-existing congestion and obstacles, and high performance interconnect structures (including those which require variable width interconnect).

1 Introduction

Advances in VLSI fabrication technology have resulted in an increasing interest in interconnection and packaging technologies. For many systems, the delay due to circuit interconnections can dominate the total system delay [2, 6]. Interconnect optimization must be considered to achieve high performance at the system level.

One approach that provides many benefits to packaging level interconnection is the Multichip Module (MCM) technology. This technology increases practical packing densities, and removes a level of interconnection. In MCM designs, a number of bare chips are placed directly onto a substrate, with the substrate itself providing the circuit interconnections. Routing problems for MCM designs have many similarities to routing for dense Printed Circuit Boards (PCBs).

Routing within MCM substrates or PCBs, however, can be an extremely difficult problem with a potentially large number of interconnect layers. A Fujitsu supercomputer design, for example, has over 50 interconnect layers, each with a large routing capacity [10]. The MCM/PCB routing problem in general may be considered to be an extremely large general area routing problem, and cannot be decomposed into small channel or switchbox routing problems easily. Additionally, high performance designs must consider signal crosstalk (due to coupling capacitance between long parallel lines), and variable width wiring (to address impedance mismatches and wiring termination requirements).

The problem of MCM/PCB routing has been considered by a number of authors. One set of approaches perform global and detail routing in a single step.

An early MCM router was presented in [10], and performs a combination of 2D and 3D routing obtain a solution. Computation times for this method were relatively high.

*This work is partially supported by DARPA/ETO under Contract DAAL01-96-K-3600 managed by the US Army Research Laboratory.

In [13] the SLICE router was presented; it constructs detailed routes through a mixture of planar routing and two-layer maze routing. While this router was superior to a 3D maze router, it requires relatively high computation times. The V4R router [14] was subsequently presented, and efficiently routes layer-pairs one at a time using routes with no more than four vias each.

The M^2R[5] router addresses performance concerns by routing critical nets on a single layer. The remaining nets are routed using $x-y$ layer pair routing.

In [17], a linear programming approach was presented. Randomized rounding is used to obtain integer solutions. A small number of possible topologies are considered, and routes are restricted to a channel intersection graph.

The MCG [3] router considers a small number of possible routes for each net, and attempts to maximize the routing density on any layer-pair by selecting a set of compatible routes from among the unrouted nets. The results of this router are quite good, requiring fewer routing layers than many of the other MCM routers. Performance concerns are addressed by removing nets which exceed signal crosstalk thresholds, and routing them on other layers.

In [20], the MLR router was presented; it first performs layer assignment of nets, followed by Steiner optimal area routing using a hierarchical (α, β) approach (to find low-cost paths between pins).

The SEGRA [4] MCM global router has been described, and also approaches the routing problem by completing pairs of layers in each pass. A simple greedy heuristic is used to select which nets are to be extended or completed during each pass.

While the methods mentioned above consider global and detail routing in a single step, there have also been approaches which consider the problem in two steps.

In [21], the MCM substrate is decomposed into a set of "towers," rectangular regions containing portions of the routing surface through all layers. They first distribute routing density across a three dimensional surface, using hierarchical decomposition, then determine locations of nets on the boundaries of the towers, and finally obtain a routing for each tower using a detail router.

We approach the problem in a manner similar to [21], and consider the global routing problem in this paper. In contrast to these previous approachs, our MINOTAUR global router offers the following features.

- We utilize current high-performance variable-width interconnect optimization results, obtaining structures which address delay and signal integrity requirements [7]. We can in fact consider a *set* of high performance interconnect structures for each signal net, and perform congestion optimization from a global perspective.

- We optimize global congestion by not restricting the layers a route may use; some approaches route a single layer-pair at a time, and cannot obtain routing solutions which span multiple or non-paired layers.

- We combine the freedom and flexibility of maze routing solutions, with the global optimization abilities of the iterative deletion method.

While we consider MCM and PCB routing in this paper, the basic techniques used in our router can be easily applied to multilayer VLSI IC routing problems as well.

The remainder of this paper is organized as follows. In Section 2, we present our problem formulation. We also describe our approach to congestion estimation in this section. In Section 3, we describe the MINOTAUR global router. We detail our methods for applying rip-up and reroute in Section 3.1, the iterative deletion approach in Section 3.2, our support for high-performance interconnect structures in Section 3.3, and the integration of algorithms in Section 3.4. Experimental results are presented in Section 4, and we conclude our paper with Section 5.

2 Problem Formulation

Our routing model is similar to that of [21]; we divide the 3-dimensional routing surface into a set of "tiles," with each tile consisting of a number of layers. We use "tile-layer" to indicate a single layer within a tile. The routing substrate is modeled as a graph, with tile-layers as nodes, and both vias and the "borders" between tile-layers as edges. Preferred-direction routing is obtained by including (or excluding) border edges. Figure 1 shows an example of our routing model.

The routing problem consists of determining the interconnect structure for a set of nets N. In addition to basic feasibility concerns (based on physical routing capacity constraints), there are a number of performance issues as well; many of these were detailed in [5]. In this paper, we are interested in meeting delay, signal

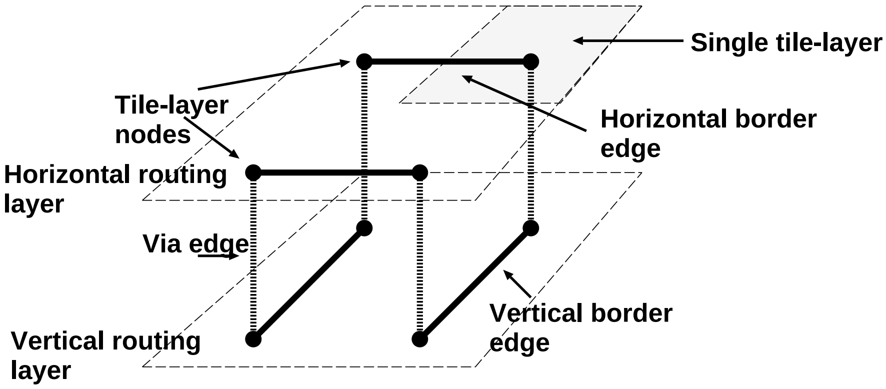


Figure 1: A two-layer four-tile routing problem.

integrity, crosstalk, and routing capacity constraints, while minimizing total wire length, maximum congestion, and routing resource costs. Our optimization concerns are as follows.

1. Delay and signal integrity constraint compliance. In high performance designs, there may be specific delay and signal integrity constraints for each net, and high performance interconnect structures (involving optimized topologies and wire sizes) may be required.

2. Crosstalk compliance. Capacitive and inductive coupling between parallel wires (in the same layer, or in adjacent layers) has become a serious concern for high performance design; crosstalk can degrade signal integrity. We are interested in obtaining a set of routes which avoid long parallel wire segments (and their resulting performance problems). Methods for efficiently estimating an modeling signal crosstalk are currently being investigated.

3. Low maximum congestion. We are interested in a *set* of routes where the maximum number of interconnections to each tile-layer is minimized. Note that minimizing congestion allows for better spacing between wires, which in turn reduces coupling capacitance (and signal crosstalk).

4. Low total wire length. Lowering wire length reduces the capacitive load, and results in lower power consumption, and lower delay (in general).

5. Minimization of routing layers. In MCM/PCB design, we may have a large number of routing layers available. Each layer, however, incurs a manufacturing expense, and we wish to minimize this expense.

Evaluation of routing results is a complex problem. The relative capacities and costs of horizontal, vertical, and via layers are dependent on specific technology parameters. Given two completed routings, it may be difficult to determine which is the "better" global solution. One may have low wire length, while the other may have a low via count; which is superior is highly dependent on fabrication technology, and the performance constraints on the design. For this reason, we evaluate the *algorithmic* aspects of our approach using a relatively simple cost function, and a single layer routing model. When comparing with practical MCM global routers, we use the same simple cost function, but route using the preferred direction multi-layer routing model.

For some routing experiments, we attempt to minimize the maximum *congestion* of the routing solution. For each tile-layer, we define the congestion to be equal to the number of routing edges which utilize the tile-layer. If variable width routing is used, we model the congestion as the sum of the wire widths. Border and via congestion levels are calculated similarly, and correspond to the "density" metric used in channel-based routing. Minimization of congestion at the global level allows for better spacing of wires within any region, and thereby can reduce coupling capacitance significantly. When comparing directly with other routers, we model congestion as the number of distinct nets utilizing a tile-layer.

We assume that a detail router will be used to complete the routings of the individual tiles, and determine solution feasibility through the maximum *congestion* of any tile-layer node, border edge, or via edge. We estimate solution congestion as the number of topology edges using a particular routing resource. Note that the border congestion and tile-layer congestion metrics are in general not comparable.

Our routing objective is to obtain a solution which meets routing capacity constraints, provides high performance interconnect structures for critical nets, and also allows for low average routing congestion and low total wire length.

To obtain interconnect structures with low delay and high signal integrity, we utilize the *Required Arrival Time Steiner Tree (RATS-tree)* algorithm of [7]. In contrast to typical performance-driven interconnect structure algorithms, the *RATS-tree* algorithm produces a set of variable-width interconnect structures that satisfy the arrival time and signal integrity (in terms of overshot and settling time) constraints at each sink for a given net. Details are given in Section 3.3. For each performance-critical net, the RATS-tree algorithm generates a *set* of candidate interconnect structures, all of which are high quality in terms of delay and signal integrity.

3 The MINOTAUR Global Router

Our approach is to first decompose each non-critical signal net into a set of edges. In most of our experiments, we use a simple minimum spanning tree algorithm, although we can also utilize Steiner tree or high performance interconnect algorithms. Steiner topologies had little impact on solution quality in our experiments. Nets which require high-performance interconnect structures are not decomposed into 2-pin edges, and are handled as described in Section 3.3.

For each edge of the non-critical nets, we wish to find a *path* through the modeled MCM substrate; we model a path as a set of tile-layer nodes, border edges, and via edges.

During optimization by iterative deletion, high performance interconnect structures are also modeled as sets of tile-layers, so our optimization approach can consider either single edges, or entire trees, interchangeably. Details of this are given in Section 3.2.

Our global router contains two basic algorithmic approaches: a traditional rip-up and reroute scheme, and a method based on iterative deletion. We first discuss the details of each, and then show how they are integrated.

3.1 Rip-Up and Reroute

Rip-up and reroute is one of the oldest approaches to the global routing problem; there are a number of criticisms for this approach. It usually requires long computation times, and there are few guarantees as to the quality of the resulting solution, or that the process will converge to a feasible solution. Despite these drawbacks, rip-up and reroute continues to be used in many routing applications. The rip-up and reroute portion of our global router is similar to that of [19], in that we iteratively change our congestion bounds to encourage convergence to a low-congestion solution. In general, this approach can yield high quality results when given sufficient time and memory.

Given the complexity of MCM and PCB designs, we believe that rip-up and reroute may be necessary for aggressive design. We implement rip-up and reroute in our router as follows. Each edge is first routed using a maze router based on Lee's classic algorithm [15]. We then iteratively rip-up and reroute each edge, continuing for a user-specified number of iterations, or until solution quality stabilizes. Details of our maze routing cost functions, and edge orderings used, are given below.

If high-performance interconnect structures are required, we select a single high-performance interconnect structure with the minimum area for the performance critical net, and do not modify it. The rip-up and reroute process operates only on the edges of non-critical nets, and attempts to find routes "around" the congestion of the critical nets. Congestion optimization of the high-performance nets by the iterative deletion method is more robust, and is detailed below.

3.1.1 Maze Routing Cost Functions

As is done in a number of other routers, we assign a cost to each tile-layer node, border edge, and via edge. Our objective is to assign costs to each resource such that we find a relatively short path (in terms of wiring distance), while avoiding heavily congested areas. We are therefore interested in knowing what cost functions result in the best performance, a question that has received relatively little attention in the literature.

In [19], the cost of routing in the most congested region is infinite, while the remaining regions have low cost. In [12], the cost of a routing path includes terms for the number of other routes that a path "touches" or "crosses," as well as two terms to capture the route length in preferred and non-preferred routing directions. In [18], steps in the preferred routing direction were assigned a cost of 1, while non-preferred steps were assigned a cost of 30. In [16], costs were piecewise-linear functions depending on a measure of the utilization and the capacity of each edge. Clearly, there are many possible ways to calculate the cost of using a particular resource.

In general, we define the *cost* c_{ijk} of using a tile-layer as follows. The subscript ijk is used to indicate the values for tile-layer t_{ijk}.

$$c_{ijk} = F(usage_{ijk}, obstacles_{ijk}, low_{ijk}, high_{ijk}, threshold_{ijk}, cap_{ijk})$$

We will use $congestion_{ijk} = usage_{ijk} + obstacles_{ijk}$. low and $high$ provide minimum and maximum values for tile-layer cost, while $threshold$ and cap are used to control the shape of our cost functions, described below. Cost functions for vias and borders are handled similarly.

Note that the values for low and $high$ influence the amount that a path may detour from its shortest-path length. In our experiments, $low = 1$ for all maze routes, and we investigate different values for $high$. In general, the larger the value of $high$, the greater distance a path may detour to avoid a congested area.

In general, we expect resources which have high usage or large numbers of obstacles to have relatively high cost. We have explored three cost functions in depth, and describe them here.

We first consider a *step* function, similar to those of [19], [18], and [12], which provides a low cost if congestion falls below a specific threshold value. A high cost results if congestion is at or above the threshold value. Second, we considered a piece-wise linear *slope* function, similar to that of [16], which provides low cost when congestion is below a threshold, high cost if congestion is above *cap*, and a linearly interpolated cost when congestion is between the threshold and *cap* levels. For each iteration of rip-up and reroute, we determine the maximum and minimum congestion for any resource. We modify the *step* and *slope* functions to reflect current congestion levels by adjusting the *threshold* and *cap* parameters. We refer to this as *Adaptive Congestion Estimation (ACE)*, and perform experiments using *ACE-step* and *ACE-slope* cost functions. A third function, which we refer to as a "two-tier" function, is similar to the *slope* function when routing congestion is below a fixed physical capacity. If congestion is above physical capacity, routing costs are increased substantially. These cost functions are illustrated in Figure 2.

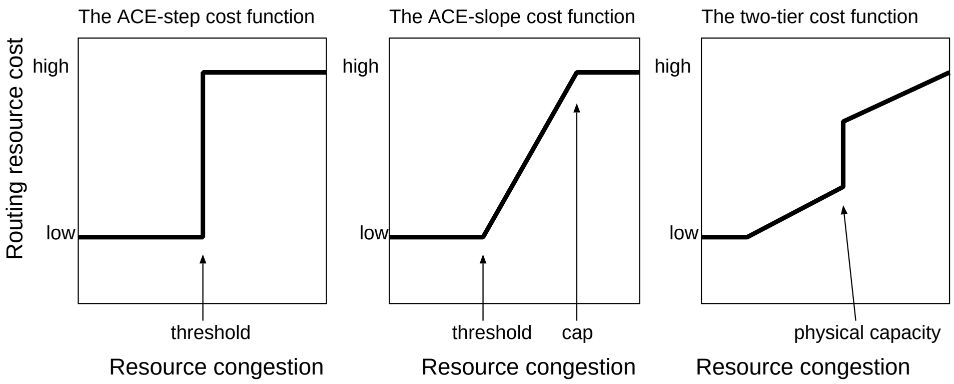


Figure 2: *ACE-step, ACE-slope*, and "two-tier" cost functions. These functions are used to determine resource cost during maze routing.

We find that the *ACE-slope* cost function is consistently superior to the *ACE-step* function. Detailed results are given in Section 4, Table 2. Many routing applications which utilize step-like functions ([19], [12],[18], for example) may obtain substantial improvements in results by adopting a slope-like cost function.

Our global router supports *separate* cost functions for each type of resource (tile-layer nodes, border edges, and via edges), and can optimize these independently.

3.1.2 The Net Ordering Problem

A perennial question in global routing problems has been *which net do we route (or reroute) first?* [9, 3]. To evaluate the importance of net ordering in our implementation, we compared five different edge orderings within the rip-up and reroute framework. Our orderings are as follows.

1. No Ordering. Edges were routed and rerouted using the order that the nets were obtained from our layout database.

2. Shortest Edges First. Edges were routed by increasing (Manhattan) length.
3. Longest Edges First. Edges were routed by decreasing (Manhattan) length.
4. Lowest Cost Edges First. Edges were routed by increasing (Manhattan) distance in the first pass (before congestion information was available), and then by the order of increasing path cost (the sum of tile costs along the path).
5. Highest Cost Edges First. Edges were routed by decreasing (Manhattan) distance in the first pass (before congestion information was available), and then by the order of decreasing path cost (the sum of tile costs along the path).

Surprisingly, there was negligible difference in the results by the various orderings. While examples can be constructed in which certain orderings are clearly superior to others, we found little evidence that this occurs in our experiments. In our model, we have no hard upper limit on routing capacity, and perform rip-up and reroute until solution quality converges. Our study concludes that solution quality measured in our routing model is far more sensitive to the choice of routing cost functions than to the edge ordering.

3.2 The Iterative Deletion Method

Rip-up and reroute is computationally expensive. Through *iterative deletion*, we can obtain high quality results while using rip-up and reroute sparingly. In this section, we describe iterative deletion; Section 3.4 describes integration with rip-up and reroute. The use of iterative deletion was first proposed in [9], and was recently improved in [8].

In many iterative improvement methods, we repeatedly modify a solution, changing a small portion at each step. Rather than representing the problem with a single instance, and changing one aspect of it at each step, the iterative deletion method represents a *set* of possible states. Consider the example in Figure 3, which is a relatively simple planar routing problem with three nets.

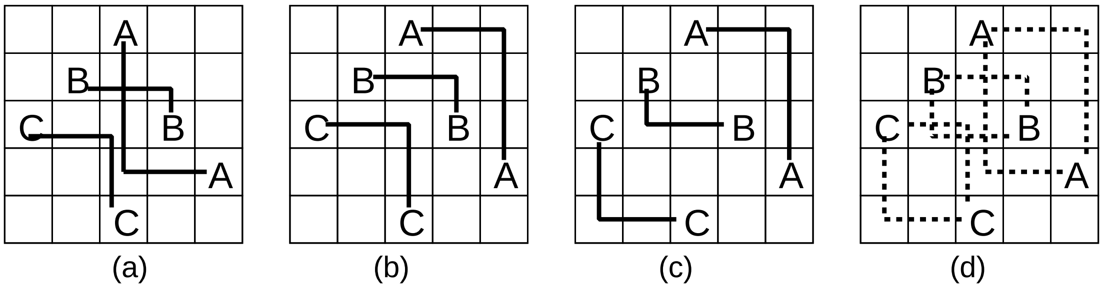


Figure 3: A simple planar routing problem (showing solutions with high congestion, high crosstalk, low crosstalk, and a set of redundant candidate routes).

Figure 3a has high congestion, while Figure 3b has high coupling capacitance (due to parallel wire segments). Figure 3c has both low congestion and low coupling capacitance, and is a *subset* of Figure 3d. The iterative deletion process begins with a redundant set of paths (such as Figure 3d), and iteratively removes a single high-cost redundant path while attempting to minimize congestion and crosstalk among the remaining connections. Interactions between paths (such as coupling capacitance) can also be modeled within this framework.

The set of possible connections for each edge is called the *path set*. In Figure 3d, the path sets for each edge consist of the two single-bend routes. By having several possible routes for each edge available at the start of the iterative deletion process, we can easily determine which paths are high quality. In the example, the lower-left routing for net C does not interfere with any other path, allowing the alternative to be removed without degrading solution quality. This approach has a number of significant features. First, we obtain a *global* estimate of congestion, and can determine not only where congestion is likely, but also where it is unavoidable. Second, crosstalk between paths can be modeled and considered during optimization. If net A has candidate paths a_1 and a_2, and net B has candidate paths b_1 and b_2, we can calculate the potential crosstalk between the various candidates. This information can then be used to influence which candidates are selected, resulting in a better global solution. By selecting a set of routes that are compatible at a *global* level, we can obtain high quality solutions quickly.

Iterative deletion was shown to be an effective method for selecting a set of channel segments in standard cell IC routing problems[8], and motivates our use of the method here. Our congestion estimation here is similar to the channel density estimation of [8]. We recalculate congestion levels after the removal of each redundant path.

As mentioned earlier, each net is decomposed into a set of edges; each edge then has a path set, which contains a number of candidate paths. Let *path set* P have n candidate paths $\{p_1, p_2, ..., p_n\}$. Each path utilizes a set of tiles; if path p_m of width w utilizes tile t_{ijk}, then it contributes w to the *current* congestion of the tile, w/n to the *probable* congestion of the tile, w to the *required* congestion of the tile if n is 1, and 0 otherwise.

In Figure 3(d), the current, probable, and required congestion measures of the upper right corner are 1, 0.5, and 0, respectively. The tile in the second row and third column has current, probable, and required congestion measures of 2, 1, and 0 respectively.

At the beginning of iterative deletion, we have many path sets with multiple paths. The *probable* congestion, however, allows us to determine which regions are likely to be congested, and which are not. After the deletion of each path, the costs of each affected tile-layer path are recalculated; with appropriate data structures, this can be done efficiently.

This method also handles multiple routing trees directly, allowing easy integration of the *RATS-tree* algorithm [7]. We use iterative deletion to select a single tree from the set of possible trees for each net, using the same methods as are used for edges.

3.2.1 Path Cost Calculation

In order for iterative deletion select the highest cost path for removal, we must determine what the cost of an individual path is. During our research, we explored a wide range of candidate cost functions to determine which worked well, and which did not. Our cost functions are detailed below.

First, we model the *cost* of any given tile-layer as a function of either the *current*, *probable*, or *required* congestion. Our router supports the above mentioned cost functions, using these metrics.

For any path p_i, we have the path sum cost $sum(p_i)$ as the sum of all tile-layers encountered by the path. The maximum cost tile-layer along the path is $max(p_i)$, and the length of the path (in terms of the number of tile-layers) is $|p_i|$. The cost of a path is given below.

$$cost(p_i) = \alpha sum(p_i) + \beta max(p_i) + \gamma |p_i| max(p_i)$$

The first term captures the congestion measure along the entire path, while the second term measures the most congested region which the path passes through. The third term allows for long paths which encounter a congested region to have higher cost than short nets which pass through similarly congested regions.

In our experiments, we evaluate a number of different values for the parameters α, β, and γ. We have found that $\alpha = 1$, $\beta = 0$, $\gamma = 0$ works well, but have not completed an exhaustive set of experiments. We are considering methods to adjust the parameters during iterative deletion.

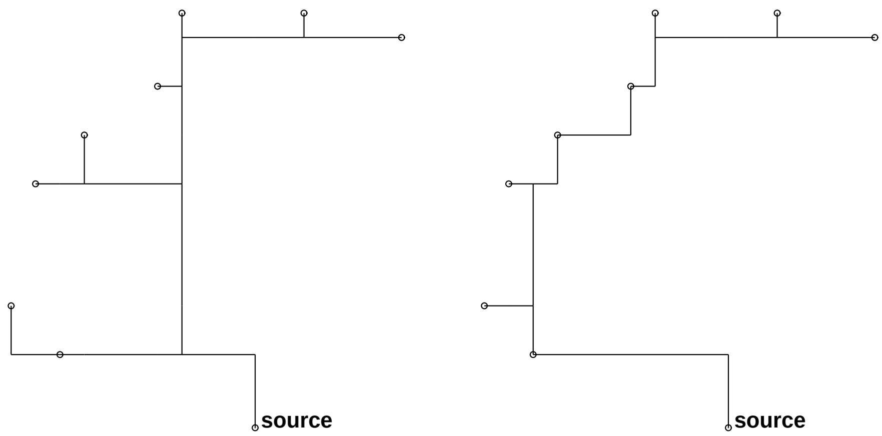
3.2.2 Path Set Cost Calculation

In [8], it was observed that comparing the two most expensive edges belonging to a net allowed for an improvement in solution quality. During iterative deletion, removal of one edge may make another required; thus, we prefer the removal of edges which do not force the acceptance of another edge that has high cost.

We define the cost of a path-set PS (which contains more than one path), with path of highest cost p_i, and second highest cost p_j, as follows.

$$cost(PS) = \theta c(p_i) - \rho c(p_j)$$

In our experiments, we have also evaluated a number of different values for the parameters θ and ρ. We have found $\theta = 1$, $\rho = 1$ to work well, but plan further experiments for these parameters. We use the sum of the routing costs of elements of the path as the total path cost for the experiments presented here. We can also consider the maximum routing element cost and total path length, and can compare the costs of a pair of paths in the path sets.



(a) *Tree 1* (b) *Tree 2*

Figure 4: An example of the topologies generated by the *RATS-tree* algorithm for a 9-pin net.

3.3 Performance Optimization

Many approaches for interconnect optimization in high performance design have been proposed. [11] and [6] provide an overview of current interconnect optimization approaches. Recently, *Required-Arrival-Time Steiner trees (RATS-trees)* have been proposed [7] to provide a set of solutions allowing trade-off among signal delay, wave-form, and routing area. This algorithm uses a bottom-up dynamic programming approach, and can synthesize a *set* of high performance structures; Figure 4 shows two possible trees for a 9-pin net.

While the our basic iterative deletion approach considers alternate routings for each topology edge, we consider alternate *trees* for each critical net. Rather than removing a single path between a pair of pins during deletion, we instead remove one of the candidate interconnect trees generated by the *RATS-tree* algorithm. The *RATS-tree* algorithm and our global router support variable-width routing.

Most of the trees generated by the RATS-tree algorithm use variable width wiring to improve signal integrity. We model these wire width changes in our formulation, and optimize congestion globally with consideration of the wire widths. For our experiments with performance optimization, however, we restrict the trees to minimum-width solutions in order to obtain a wider variety of topologies (which allows for better congestion optimization by iterative deletion).

3.4 The Hybrid Approach

We have three approaches that form a hybrid between the rip-up and reroute approach, and the iterative deletion method. The first applies iterated rip-up and reroute, and then adds the single bend paths to the path set for each edge. We then perform iterative deletion on the path sets. For single-layer routing experiments, there are two single bend paths for each edge. For multi-layer routing experiments, with h horizontal layers and v vertical layers, there are $h \times v$ single bend paths.

Our second approach begins by generating single-bend paths for each edge, and then applies iterative deletion to these path sets. After iterative deletion, we use the maze router to find a new candidate path for each edge, and add it to the path set for the edge. Multiple iterations of this process can produce high quality results quickly. If we restore all deleted paths to the path set for each edge, we call this *Cumulative Iterative Deletion*, and have an increasing number of candidate paths for each edge for each pass. If we do not restore the deleted paths, and consider for each dge only the path from the previous pass, and the newly generated path, we call this *Non-Cumulative Iterative Deletion*, and have only two candidate paths for each edge.

A third approach applies iterative deletion as above, followed by rip-up and reroute for non-critical nets.

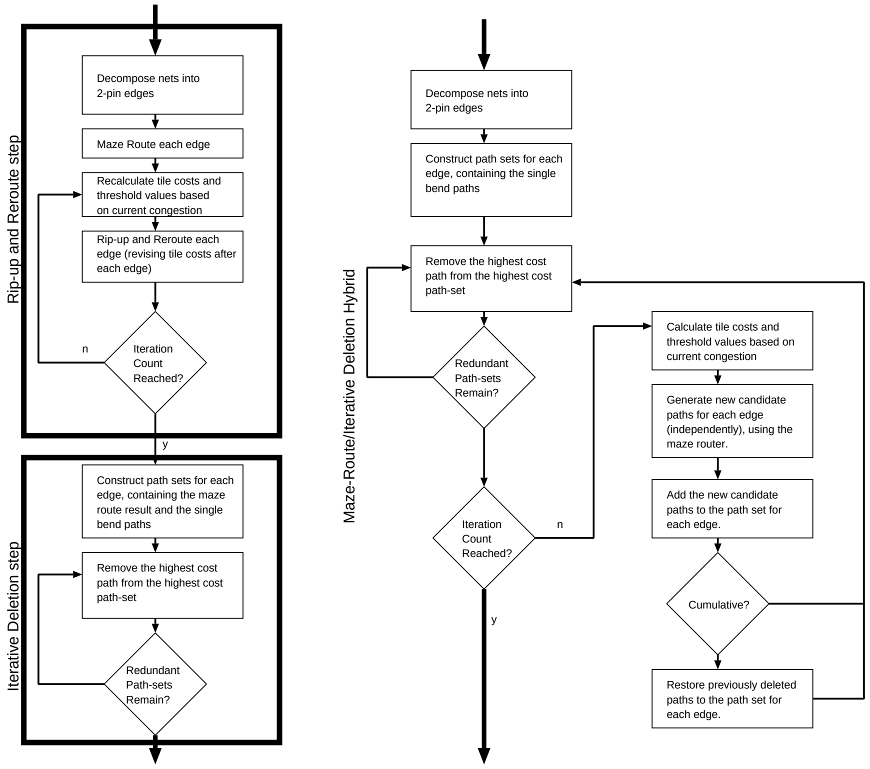


Figure 5: Outline of two basic approaches of the MINOTAUR global router. In the first, we apply a traditional rip-up and reroute approach, and may then apply iterative deletion. In the second, we begin with an iterative deletion step, and can then utilize a maze router to identify additional candidate routes.

4 Experimental Results

Experiments were performed with the MCC multichip module benchmark circuits *mcc1* and *mcc2*, and also on the three test cases from [14], summarized in Table 1. We report maximum and average congestion levels (C_{max} and C_{avg}) for either the tile-layer metric or the border metric (these two are not in general comparable). Average congestion should correlate reasonably well with wire lengths available after completion of detail routing.

Maze routing (which is unoptimized) dominates the run time. MINOTAUR routes *mcc2* in one hour on a Sun Ultra-1, using a single layer 16 by 16 routing graph. Increasing graph resolution increases run times.

Benchmark	No. of Chips	No. of Nets	No. of Pins	Net cardinality 2	3	4	5+
test1	4	500	1000	500	0	0	0
test2	9	956	1912	956	0	0	0
test3	9	1254	2508	1254	0	0	0
mcc1	6	802	2495	608	87	50	57
mcc2	37	7118	14659	6699	415	4	0

Table 1: Benchmark circuit parameters. *test1*, *test2*, and *test3* are constructed examples, containing only two-pin nets.

	Best RR		Best RR + ID		Worst RR		Worst RR + ID		Typical RR		Typical RR + ID	
Benchmark	C_{max}	C_{avg}	C_{max}	C_{avg}	C_{max}	C_{avg}	C_{max}	C_{avg}	C_{max}	C_{avg}	C_{max}	C_{avg}
test1 *slope*	27	23.7	27	23.7	28	23.7	28	23.7	27	24.3	28	24.0
step	33	24.7	30	24.1	58	23.7	33	23.7	33	24.8	30	23.1
test2 *slope*	49	46.3	51	46.1	51	46.5	51	46.2	50	47.0	52	46.5
step	57	49.1	54	47.7	112	45.9	59	45.9	61	48.4	53	47.3
test3 *slope*	62	59.0	64	59.0	63	59.9	65	59.5	63	60.3	65	59.7
step	67	60.9	67	60.2	111	59.0	73	59.0	76	60.6	67	59.9
mcc1 *slope*	59	49.3	60	49.1	67	48.5	63	48.3	60	50.1	61	49.6
step	83	51.3	69	49.6	123	48.5	81	48.5	89	51.5	68	49.9
mcc2 *slope*	295	216.0	291	211.7	363	186.3	344	186.0	303	215.6	293	210.9
step	350	209.1	312	203.8	524	192.1	376	188.0	416	212.3	356	205.2

Table 2: Experiments on MCM benchmarks considering a variety of cost functions. We report tile-layer maximum and average congestion C_{max} and C_{avg}, and show the best and worst performance using a classic rip-up and reroute approach, and the results of applying a single pass of iterative deletion to the rip-up and reroute result combined with the two single-bend candidate paths. We also show results using a "typical" cost function configuration.

	Single Layer		Multi Layer		
Benchmark	Minotaur	V4R	L	Minotaur	V4R
test1	27	52	4	16	26
test2	50	78	4	29	34
test3	63	102	4	33	45
mcc1	60	94	4	48	74
mcc2	303	496	6	196	228

Table 3: Comparison of maximum tile-layer congestion of the MINOTAUR global routing result with that of V4R. For the single-layer comparison, the V4R layers were merged into a single layer.

4.1 Comparison of Cost Functions

To compare cost function performance, we route each example as a single layer problem, using a 16 by 16 routing grid. We use a low routing cost of 1, and tested combinations with high routing costs of 2, 4, 8, and 16, *ACE* cost function threshold points at 0.2, 0.4, 0.6, and 0.8 of the range between the minimum and maximum congestion level, and also a threshold point at the average congestion level. A summary of these results is shown in Table 2, in which we show the best, worst, and "typical" performance of the rip-up and reroute method for either the step or slope cost functions, and the result of a single pass of iterative deletion considering the rip-up and reroute paths and the single-bend candidate paths. In the "typical" cost function configuration, we use a high routing cost of 16, and the cost function threshold point at the average congestion level.

While the congestion values during the rip-up and reroute process converged for all experiments, they converged to vastly different values depending on the cost function, and the *high*, *low*, and *threshold* values used to configure the cost function; maximum congestion levels differed by as much as a factor of 2.29. We find that the results using the slope cost function are superior to the step based cost function. When the *high*, *low*, and *threshold* values for tile costs are not well suited to the routing problem, a single pass of iterative deletion can improve the results considerably.

4.2 Comparison with V4R

In Table 3, we compare maximum tile-layer congestion values of our approach with the results of the MCM global router V4R [14]. Congestion estimation favors the V4R router, as the MINOTAUR result considers distinct *tree edges*, while the V4R result reported considers distinct nets. For the single layer comparisons, we merge layer assignments of the V4R routing into a single layer. Our global router obtains congestion reductions ranging from 35% to 48% for the single layer model, and reductions of 14% to 38% for the multi-layer model.

mcc1				
	C_{max}	C_{avg}	Mono.	Non-Mono.
ACE-slope	17	11.4	1597	97
two-tier (35)	27	11.2	1678	16
two-tier (24)	21	11.2	1652	42
two-tier (20)	17	11.3	1622	72
two-tier (16)	16	11.7	1515	179
two-tier (10)	22	11.6	1535	159
post-process (35)	26	11.2	1668	26
post-process (24)	21	11.2	1645	49
post-process (20)	17	11.3	1630	64
post-process (16)	16	11.3	1625	69
post-process (10)	17	11.4	1597	97

mcc2				
	C_{max}	C_{avg}	Mono.	Non-Mono.
ACE-slope	84	48.7	6874	667
two-tier (125)	103	43.8	7412	129
two-tier (100)	84	44.5	6957	584
two-tier (80)	81	48.8	6191	1350
two-tier (60)	98	54.6	5406	2135
post-process (125)	103	43.7	7474	127
post-process (100)	83	44.5	7251	290
post-process (80)	80	48.8	6759	782
post-process (60)	84	48.7	6874	667

Table 4: Four-layer routing experiments on *mcc1* and *mcc2* with the border congestion metric, using physical routing capacity constraints (shown in parenthesis). We use either the *ACE-slope* cost function, a "two-tier" function, or the *ACE-slope* cost function followed by improvement with a "two-tier" function as a post-processing step. The number of monotonic and non-monotonic paths are also shown.

4.3 Experiments with Physical Capacity Constraints

Table 4 shows experimental results and routings using a "two-tier" cost function and differing physical routing capacity constraints. Assuming a wire pitch of 75μ, we have a physical capacity per border of 36 for *mcc1* and 125 for *mcc2*. We also consider cases with increased wire pitch, resulting in physical capacities of 24, 20, 16, and 10 for *mcc1*, and 100, 80, and 60 for *mcc2*, respectively. We apply either an *ACE-slope* cost function, the "two-tier" function, or an *ACE-slope* approach followed by a post-processing step using the "two-tier" cost function. When physical constraints are relatively loose, using a two-tier cost function or *ACE-slope* followed by post-processing can further improve the averge congestion (and expected wire length), as shown in the case with a routing capacity of 125 for *mcc2*. When an unrealistically tight capacity constrnt is given (as is the case for the capacity of 60 for *mcc2*) the two-tier cost function may give a solution that is inferior to the one obtained using *ACE-slope*.

We also performed a set of experiments to evaluate the impact of extremely high costs on tiles when they exceed a set capacity. In these experiments, we set the threshold value to be either slightly lower, or slightly higher, than observed performance. Our cost function for these experiments was a step function, with unit cost below the threshold, and a cost of 1000 for above the threshold; this results in the router attempting to minimize the number of tile-layers which exceed capacity.

Interestingly, while the router was able to meet the higher capacity constraints easily (and in relatively few iterations), the router failed to meet the lower constraints in all cases, and had maximum congestion greater than the best observed result. For three of the benchmarks, use of the low capacity threshold produced resulted in inferior to the high capacity threshold. This indicates that if we do not know *a priori* the congestion level that can be obtained, setting a threshold too low may result in performance inferior to a case where no threshold value is used. If we set the threshold too high, we may obtain a result with higher than necessary congestion.

Benchmark	Low Threshold	Max	Average	High Threshold	Max	Average
test1	25	32	23.7	31	31	25.8
test2	45	56	45.9	58	58	51.6
test3	58	68	59.0	69	69	63.2
mcc1	63	77	48.6	67	67	52.6
mcc2	280	366	223.0	321	321	221.0

Table 5: Results of experiments in which a large penalty, step-based cost function, and a fixed congestion threshold were used.

Benchmark	No Order	Shortest	Longest	Low Cost	High Cost
test1	27	27	27	28	27
test2	51	51	51	51	51
test3	63	64	64	64	63
mcc1	60	59	59	59	60
mcc2	303	303	303	304	303

Table 6: Results of experiments in which a number of edge orderings were used during the rip-up and reroute process. Orders considered were an arbitrary order, shortest or longest edges by Manhattan length first, and lowest cost or highest cost path first. We observed no significant difference in final solution quality (or the number of iterations required) for any of the tested orderings.

4.4 Impact of Edge Ordering

For each type of cost function, and for a variety of physical capacities, we tested the rip-up and reroute portion of our global router, using different edge ordering strategies. Tested strategies used a preference for either edge length or cost, in increasing or decreasing order, and the "no particular order" determined by the storage of nets in our CAD database.

While we tested five different orderings, we found that final solution quality (in terms of maximum and average congestion) were nearly identical. Estimated total wirelengths were within 1% of eachother. No ordering strategy consistently outperformed the others.

In our global router, we do not drop nets when regions become over-congested, and apply a number of iterations of rip-up and reroute, and allow multiple nets to utilize a single region. If we were to adopt differing policies on these issues, net ordering may become more significant.

We next compare the results of various sorting metrics in Table 6. While the edge orderings used for each during rip-up and reroute were quite different, the final congestion levels in each case were remarkably similar. Edge ordering may have a more significant impact on routing completion rates in problems which have hard capacity limits.

4.5 Impact of Iterative Deletion

Next, we show results using the approach which begins with iterative deletion on path sets containing the single bend routes. In Table 7, we show the maximum and average congestion values for a number of passes of cumulative iterative deletion. The cost function used in maze routing was the slope function, with a high cost of 16, and a threshold value of 0.8. Note that in the first pass, no maze routing has been performed. Results of the first pass are quite good, with congestion converging to values close to those of rip-up and re-route, but with only 4 or 5 iterations (while rip-up and reroute usually required 6 or 7 iterations).

The best performing cost function used during iterative deletion appears to be the summation of linear congestion values ($\alpha = 1$, $\beta = 0$, $\gamma = 0$, $\theta = 1$, $\rho = 1$). In many cases, cumulative iterative deletion was inferior to non-cumulative iterative deletion. We suspect that an abundance of choices degrades performance in the early stages of an iterative deletion pass; if we retain candidate paths only for those connections which have high cost, the performance of cumulative iterative deletion may improve. We plan a more through set of experiments to determine how sensitive the algorithm is to these parameters, and which parameters result in

Benchmark	1st pass		2nd pass		3rd pass		4th pass		5th pass		6th pass	
test1	32	23.7	30	24.2	31	24.3	32	24.6	30	24.6	29	24.8
test2	57	45.8	55	46.5	58	47.4	57	47.6	55	47.8	54	47.7
test3	73	59.0	68	60.0	72	60.6	70	60.9	69	61.5	69	61.5
mcc1	72	48.5	67	50.1	68	50.5	66	51.1	64	51.4	64	51.7
mcc2	357	185.3	328	198.2	315	205.3	306	214.3	298	218.9	298	218.9

Table 7: Performance of cumulative iterative deletion over a number of passes. In the first pass, only single bend connections are considered for each edge. In the second pass, the single bend connections and a path determined by the maze router are considered for each edge. Subsequent passes add a single path determined by the maze router to the set considered for each edge.

Bench mark	T	Total Nets	Min/max/avg. *RATS-trees* $X = 5\%$	$X = 10\%$	$X = 15\%$
mcc1	5%	38	4/12/8.16	4/12/8.13	4/12/8.13
mcc1	10%	51	3/17/8.71	3/14/8.67	3/14/8.55
mcc2	5%	614	1/6/1.02	1/4/1.02	1/4/1.02
mcc2	10%	1278	1/6/1.01	1/4/1.01	1/4/1.01

Table 8: Number of nets with performance constraints, and the minimum, maximum, and average number of *RATS-trees* for each net.

the best performance.

4.6 Performance Optimization

To evaluate the impact of performance optimization on global solution quality, we applied the *RATS-tree* algorithm to a number of nets from the benchmarks *mcc1* and *mcc2*. The number of nets optimized, minimum and maximum number of *RATS-trees*, and average number of *RATS-trees* per net are shown in Table 8. For these experiments, we constructed Steiner trees for each net, and evaluated the delay for each using *SPICE*. $T\%$ of the pins with highest delay were selected, and we attempt to obtain an $X\%$ reduction in signal delay for these pins (no delay constraints are placed on the other nets or pins). Router performance is shown in Table 9.

For some nets, multiple routing topologies which meet the delay constraints are possible. In general, the topologies produced by the *RATS-tree* algorithm for a tight constraint are a subset of the those produced for a loose constraint.

We consider two routing approaches for this problem. The first approach selects a single minimum-area candidate *RATS-tree* for each constrained net, and then applies rip-up and reroute the the remaining nets. The second approach is a hybrid, and applies iterative deletion to select a single *RATS-tree* for each constrained net, and then applies rip-up and reroute. When a number of candidate topologies are available, the hybrid approach obtains a 19.5% reduction in maximum congestion. In this set of experiments, the minimum-area *RATS-trees* were quite similar under the different performance constraints, resulting in similar performance for the rip-up and reroute approach.

5 Conclusions

We have presented an MCM/PCB global router which integrates the classic rip-up and reroute approach, with the modern iterative deletion method. We show that the results of iterative deletion are comparable with the results of rip-up and reroute when the maze route cost function parameters are favorable, and that iterative deletion outperforms rip-up and reroute when the maze route cost function parameters are not favorable.

We draw a number of conclusions from our experiments. We find that slope-based cost functions are superior to step-based cost functions, rip-up and reroute performance is heavily dependent on cost function parameters, and net ordering has little impact for our routing model. Iterative deletion is effective for congestion

Benchmark	T	Rip-up and reroute						Iterative deletion with RR postprocessing					
		$X = 5\%$		$X = 10\%$		$X = 15\%$		$X = 5\%$		$X = 10\%$		$X = 15\%$	
		C_{max}	C_{avg}	C_{max}	C_{avg}	C_{max}	C_{avg}	C_{max}	C_{avg}	C_{max}	C_{avg}	C_{max}	C_{avg}
mcc1	5%	34	22.6	34	22.6	34	22.6	32	22.6	33	22.7	33	22.7
mcc1	10%	41	22.3	41	22.3	41	22.3	33	22.8	33	22.8	33	22.8
mcc2	5%	184	93.6	184	93.6	184	93.6	184	93.6	184	93.6	185	93.6
mcc2	10%	223	93.3	223	93.3	223	93.3	222	93.3	222	93.3	222	93.3

Table 9: Impact of performance optimization on for mcc1 and mcc2, using rip-up and reroute and a single candidate high performance interconnect structure. In these experiments, we select $T\%$ of the high-delay pins, and attempt to obtain at least an $X\%$ delay reduction for each. If no performance optimization is required, rip-up and reroute obtains maximum and average congestion levels for mcc1 are 33 and 23.0 respectively. The maximum and average congestion levels for mcc2 are 168 and 93.4, respectively.

minimization when we have signal delay and integrity performance constraints, or when rip-up and reroute cost function parameters are not well chosen.

We feel that this result may enhance the performance of a number of previous global routers, and be of general interest to the research community. We also observed that the performance of rip-up and reroute is quite sensitive to cost function parameters; if these are not selected correctly, congestion levels may be relatively high. Note that it is unclear how appropriate parameters can be determined *a priori*. Second, we find that net ordering apparently has little impact on the final global routing results in our approach.

In our experiments, iterative deletion required fewer passes to obtain a good solution, and was able to obtain high quality solutions with a *single* pass where only single-bend routes were considered. In many cases, iterative deletion was able to improve the result of the rip-up and reroute approach.

Our formulation also provides a simple mechanism to perform global performance optimization, by selecting from among a set of high-performance interconnect structures for critical nets. Iterative deletion can determine a compatible set of paths, and restriction of critical nets to specific layers is not required. We plan to add support for crosstalk optimization, by considering the interactions of the different candidate interconnect structures during the iterative deletion process.

Unlike many previous approaches, our routing solutions are not restricted to any specific layer or pair of layers. Optimization of congestion takes place globally, allowing for all layers to be used effectively.

We are currently integrating our MCM global router with a performance driven variable width multi-layer detail router, and will report results of these experiments when they are complete. We are also considering the use of a small set of candidate routes such as those suggested in [3], allowing us to eliminate maze routing entirely. Our methods may also extend to handle general area routing for performance driven multi-layer VLSI IC design problems.

The increasing number of routing layers available for VLSI IC design [1] may allow a departure from the traditional channel-based routing approach, with the routing model described here becoming practical.

6 Acknowledgements

The authors would like to thank Mr. Cheng-Kok Koh for his assistance and for supplying performance constraints and the *RATS-tree* algorithm.

References

[1] Semiconductor Industry Association. *National Technology Roadmap for Semiconductors.* 1994.

[2] H. B. Bakoglu. *Circuits, Interconnections, and Packaging for VLSI.* Addison-Wesley, 1990.

[3] J. D. Carothers, D. Li, and T. Hameenanttila. MCG: A correct-by-design multichip module router with crosstalk avoidance. In *Int. Workshop on Rapid System Prototyping*, pages 183–188, 1996.

[4] Y.-J. Cha, C. S. Rim, and K. Nakajima. A simple and effective greedy multilayer router for MCMs. In *Proc. Int. Symp. on Physical Design*, pages 67–72, 1997.

[5] J. D. Cho, K.-F. Liao, S. Raje, and M. Sarrafzadeh. M^2R: Multilayer routing algorithm for high-performance MCMs. *IEEE Trans. on Circuits and Systems*, 41(4):253–265, April 1994.

[6] J. Cong, L. He, C.-K. Koh, and P. H. Madden. Performance optimization of VLSI interconnect layout. *Integration, the VLSI Journal*, 21:1–94, 1996.

[7] J. Cong and C.-K. Koh. Interconnect layout optimization under higher-order RLC model. In *Proc. Int. Conf. on Computer Aided Design*, 1997.

[8] J. Cong and P. H. Madden. Performance driven routing for standard cell design. In *Proc. Int. Symp. on Physical Design*, pages 73–80, 1997.

[9] J. Cong and B. Preas. A new algorithm for standard cell global routing. In *Proc. Int. Conf. on Computer Aided Design*, pages 176–179, November 1988.

[10] A. Hanafusa, Y. Yamashita, and M. Yasuda. Three-dimensional routing for multilayer ceramic printed circuit boards. In *Proc. Int. Conf. on Computer Aided Design*, pages 386–389, 1990.

[11] A. B. Kahng and G. Robins. *On Optimal Interconnections for VLSI.* Kluwer Academic Publishers, 1994.

[12] K. Kawamura, T. Shindo, T. Shibuya, H. Miwatari, and Y. Ohki. Touch and cross router. In *Proc. Int. Conf. on Computer Aided Design*, pages 56–59, 1990.

[13] K.-Y. Khoo and J. Cong. A fast multilayer general area router for MCM designs. *IEEE Trans. on Circuits and Systems*, pages 841–851, 1992.

[14] K.-Y. Khoo and J. Cong. An efficient multilayer MCM router based on four-via routing. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, pages 1277–1290, 1994.

[15] C. Y. Lee. An algorithm for path connection and its applications. *IRE Trans. on Electronic Computers*, EC-10(3):346–365, 1961.

[16] R. Linsker. An iterative-improvement penalty-function-driven wire routing system. *IBM J. Research and Development*, 28(5):613–624, September 1984.

[17] S. Mehrotra, P. Franzon, and M. Steer. Performance driven global routing and wiring rule generation for high speed PCBs and MCMs. In *Proc. Design Automation Conf*, pages 381–387, 1995.

[18] H. Shin and A. Sangiovanni-Vincentelli. A detailed router based on incremental routing modifications: Mighty. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, CAD-6(6):942–955, November 1987.

[19] E. Shragowitz and S. Keel. A global router based on a multicommodity flow model. *Integration, the VLSI Journal*, 5:3–16, 1987.

[20] D. Wang and E. S. Kuh. A new timing-driven multilayer MCM/IC routing algorithm. In *Proc. IEEE Multi-Chip Module Conf.*, pages 89–94, 1997.

[21] Q. Yu, S. Badida, and N. Sherwani. Algorithmic aspects of three dimensional MCM routing. In *Proc. Design Automation Conf*, pages 397–401, 1994.

A Detailed Experimental Results

In Tables 10, 11, 12, 13, and 14, we split the results into cases in which the slope cost function was used (above), and cases in which the step cost function was used (below). The columns indicate the threshold value used for the cost function: either the average of all tile-layers, or at values uniformly spaced across the range from the value of the lowest congested tile-layer, to the highest.

The rows are arranged in pairs, with the first of each pair showing the maximum congestion value of any tile-layer, and the average across all tile-layers, using the simple rip-up and reroute strategy with the *high* tile-layer cost as indicated. The second row of a pair shows the result of applying a single pass of iterative deletion to path sets containing the final path from the rip-up and reroute, and the single bend paths. The *low* cost used during maze routing was 1 in all cases.

Slope-based cost function										
	Threshold value									
Method	Average		0.2		0.4		0.6		0.8	
RR, $high = 2$	27	23.7656	28	23.7188	28	23.7188	28	23.7422	27	23.7734
RR + ID	28	23.7344	28	23.7188	27	23.7188	28	23.7188	28	23.7344
RR, $high = 4$	27	23.8438	28	23.7500	28	23.7578	27	23.7891	27	23.9062
RR + ID	28	23.7734	28	23.7266	28	23.7344	27	23.7578	28	23.8047
RR, $high = 8$	27	24.0000	27	23.7734	27	23.7969	27	23.8984	27	24.1172
RR + ID	28	23.8047	28	23.7344	28	23.7578	27	23.8438	28	23.9062
RR, $high = 16$	27	24.3125	27	23.7656	27	23.7891	27	23.9375	28	24.5703
RR + ID	28	24.0312	27	23.7266	28	23.7578	27	23.8516	28	24.0781

Step-based cost function										
	Threshold value									
Method	Average		0.2		0.4		0.6		0.8	
RR, $high = 2$	35	23.7969	55	23.7188	39	23.7188	36	23.7812	31	23.7891
RR + ID	29	23.7578	33	23.7188	31	23.7188	29	23.7422	29	23.7422
RR, $high = 4$	34	24.4062	56	23.7656	43	23.7969	34	24.1719	33	24.2109
RR + ID	29	24.0234	33	23.7344	30	23.7812	29	23.9453	29	23.9375
RR, $high = 8$	33	24.6484	54	23.7891	51	23.8203	34	24.3828	32	24.7578
RR + ID	29	24.1328	33	23.75	32	23.7734	30	24.0703	29	24.2656
RR, $high = 16$	33	24.7812	58	23.7344	48	23.7969	35	24.2344	32	24.7891
RR + ID	30	24.1016	33	23.7344	32	23.7734	30	23.9375	29	24.2266

Table 10: Experiments on the MCM benchmark *test1*. We performed a number of passes of iterative rip-up and reroute, using the maze routing cost function indicated (low cost was 1 in all cases, and high cost is indicated in the table). After rip-up and reroute results stablized, we performed a single pass of iterative deletion considering the final maze routing path, and two single-bend paths.

Slope-based cost function										
	Threshold value									
Method	Average		0.2		0.4		0.6		0.8	
RR, $high = 2$	50	45.9336	50	45.8945	50	45.8945	50	45.9102	50	45.9492
RR + ID	51	45.9102	51	45.8945	51	45.8945	51	45.9102	51	45.9258
RR, $high = 4$	50	46.1523	50	45.9023	50	46.0117	49	46.1133	51	46.4258
RR + ID	52	45.9961	51	45.8945	50	45.957	51	46.0039	51	46.2383
RR, $high = 8$	51	46.5195	50	45.9336	50	46.1523	50	46.4414	50	46.8242
RR + ID	51	46.207	51	45.9336	51	45.9961	51	46.1445	51	46.457
RR, $high = 16$	50	47.043	50	45.9805	49	46.2617	50	46.5977	50	47.3945
RR + ID	52	46.4961	51	45.9258	51	46.0586	51	46.1602	51	46.5664

Step-based cost function										
	Threshold value									
Method	Average		0.2		0.4		0.6		0.8	
RR, $high = 2$	71	46.1758	112	45.8945	99	45.8945	62	45.9805	62	46.1836
RR + ID	54	46.043	59	45.8945	56	45.8945	56	45.9414	53	46.0117
RR, $high = 4$	62	47.3242	111	46.082	94	46.2539	61	46.3398	64	47.0664
RR + ID	53	46.6133	58	45.9648	56	46.1445	56	46.1523	53	46.5195
RR, $high = 8$	64	48.2539	110	46.1523	99	46.1992	58	46.8398	65	47.9961
RR + ID	54	47.1211	58	46.0352	56	46.0898	56	46.4102	54	47.0664
RR, $high = 16$	61	48.3711	112	46.168	94	46.2305	62	47.3633	57	49.1445
RR + ID	53	47.3086	58	46.0586	56	46.082	57	46.4102	54	47.7539

Table 11: Experiments on the MCM benchmark *test2*. We performed a number of passes of iterative rip-up and reroute, using the maze routing cost function indicated (low cost was 1 in all cases, and high cost is indicated in the table). After rip-up and reroute results stablized, we performed a single pass of iterative deletion considering the final maze routing path, and two single-bend paths.

Slope-based cost function										
	Threshold value									
Method	Average		0.2		0.4		0.6		0.8	
RR, $high = 2$	63	59.0703	63	59.0000	62	59.0000	63	59.0391	64	59.0781
RR + ID	64	59.0312	64	59.0000	64	59.0000	64	59.0234	64	59.0547
RR, $high = 4$	63	59.3438	62	59.0156	62	59.0859	62	59.2344	62	59.4453
RR + ID	64	59.1562	64	59.0000	64	59.0547	64	59.1562	64	59.2422
RR, $high = 8$	63	59.9922	63	59.0469	62	59.1719	63	59.4844	63	59.9219
RR + ID	65	59.4844	64	59.0234	64	59.1172	64	59.3047	64	59.4141
RR, $high = 16$	63	60.2812	63	59.0859	63	59.2188	62	59.625	63	60.2031
RR + ID	65	59.7188	64	59.0391	64	59.1562	64	59.3125	64	59.6875
Step-based cost function										
	Threshold value									
Method	Average		0.2		0.4		0.6		0.8	
RR, $high = 2$	79	59.2812	111	59.0156	95	59.0000	73	59.1953	70	59.1172
RR + ID	67	59.1953	73	59.0156	69	59.0000	66	59.1016	66	59.0781
RR, $high = 4$	74	60.1484	110	59.125	95	59.2109	80	59.8438	71	60.1719
RR + ID	67	59.6562	73	59.0703	70	59.1094	67	59.4609	66	59.6953
RR, $high = 8$	74	60.625	106	59.2422	95	59.2578	79	59.9609	67	60.9531
RR + ID	67	60.0547	71	59.1562	70	59.1406	67	59.5703	67	60.2266
RR, $high = 16$	76	60.5781	109	59.1484	95	59.3516	68	61.2109	83	60.25
RR + ID	67	59.9141	72	59.0703	70	59.2422	66	60.0703	67	59.7656

Table 12: Experiments on the MCM benchmark *test3*. We performed a number of passes of iterative rip-up and reroute, using the maze routing cost function indicated (low cost was 1 in all cases, and high cost is indicated in the table). After rip-up and reroute results stablized, we performed a single pass of iterative deletion considering the final maze routing path, and two single-bend paths.

Slope-based cost function										
	Threshold value									
Method	Average		0.2		0.4		0.6		0.8	
RR, $high = 2$	64	48.5312	67	48.5156	67	48.5156	67	48.5156	65	48.5391
RR + ID	62	48.5312	63	48.5156	63	48.5156	63	48.5156	61	48.5391
RR, $high = 4$	60	48.7109	66	48.7031	66	48.7891	64	48.8594	60	48.8828
RR + ID	61	48.6562	63	48.6094	62	48.6328	61	48.7812	60	48.7734
RR, $high = 8$	59	49.2891	67	48.8281	66	48.9297	63	49.0625	61	49.6562
RR + ID	60	49.0859	63	48.6484	62	48.6484	61	48.8672	60	49.2578
RR, $high = 16$	60	50.1484	67	48.8828	65	49.0547	62	49.3047	60	50.2734
RR + ID	61	49.5703	63	48.6719	62	48.7969	62	49.0703	61	49.625

Step-based cost function										
	Threshold value									
Method	Average		0.2		0.4		0.6		0.8	
RR, $high = 2$	89	48.7422	123	48.5469	105	48.6797	85	48.6328	83	48.6172
RR + ID	72	48.5625	81	48.5312	75	48.5547	68	48.6094	68	48.5625
RR, $high = 4$	84	50.3281	120	48.8281	115	48.9375	88	50.2422	81	50.2266
RR + ID	68	49.3594	82	48.5859	77	48.6797	68	49.3203	65	49.2812
RR, $high = 8$	88	51.1875	121	48.8047	116	49.1172	83	51.3359	70	49.7812
RR + ID	69	49.6172	81	48.5781	77	48.8281	69	49.6406	64	49.1719
RR, $high = 16$	89	51.5312	121	48.8047	117	49.3281	99	50.3438	76	52.3594
RR + ID	68	49.9297	81	48.5781	77	48.8906	74	49.4844	65	50.2266

Table 13: Experiments on the MCM benchmark *mcc1*. We performed a number of passes of iterative rip-up and reroute, using the maze routing cost function indicated (low cost was 1 in all cases, and high cost is indicated in the table). After rip-up and reroute results stablized, we performed a single pass of iterative deletion considering the final maze routing path, and two single-bend paths.

Slope-based cost function										
	Threshold value									
Method	Average		0.2		0.4		0.6		0.8	
RR, $high = 2$	364	186.297	364	186.141	363	186.344	360	186.039	352	185.477
RR + ID	343	186.031	344	185.906	344	186.023	341	185.938	339	185.477
RR, $high = 4$	344	192.086	361	189.305	354	191.156	340	192.188	330	189.367
RR + ID	326	190.984	340	188.242	334	189.898	324	191.086	323	189.055
RR, $high = 8$	314	204.672	355	193.078	339	198.266	319	203.609	306	201.711
RR + ID	305	201.898	335	191.57	324	195.844	311	200.852	300	199.922
RR, $high = 16$	303	215.57	352	194.984	331	202.273	314	212.266	295	215.961
RR + ID	293	210.867	333	192.906	320	199.188	302	207.508	291	211.703

Step-based cost function										
	Threshold value									
Method	Average		0.2		0.4		0.6		0.8	
RR, $high = 2$	493	187.703	518	186.328	470	187.773	434	186.859	396	185.82
RR + ID	370	186.32	375	186.039	369	186.641	365	186.359	350	185.781
RR, $high = 4$	473	197.805	512	189.383	473	198.906	411	198.438	394	191.773
RR + ID	372	192.398	376	187.156	371	193.195	350	194.328	339	189.602
RR, $high = 8$	430	205.727	522	191.859	468	204.398	395	211.531	350	209.172
RR + ID	361	199.281	375	187.914	372	197.078	326	205.172	312	203.797
RR, $high = 16$	416	212.32	524	192.094	474	207.047	384	215.805	360	221.828
RR + ID	356	205.281	376	187.922	359	199.508	327	208.227	310	213.773

Table 14: Experiments on the MCM benchmark *mcc2*. We performed a number of passes of iterative rip-up and reroute, using the maze routing cost function indicated (low cost was 1 in all cases, and high cost is indicated in the table). After rip-up and reroute results stablized, we performed a single pass of iterative deletion considering the final maze routing path, and two single-bend paths.