Fast Multi-Layer Global Routing

ABSTRACT

Design turn-around times are decreasing rapidly, and there is a strong need for physical design tools that run quickly, and produce stable results. In traditional global routing methods, the maze routing techniques used to minimize congestion can be time consuming, and can also increase wire lengths by introducing detours. A mismatch between placement and global routing wire lengths can make timing closure difficult.

In this paper, we present a new global routing approach for use in a run time constrained physical design flow; our new tool is hundreds of times faster than *Labyrinth* and *Chi*. We focus on designs that are not space limited, which have become relatively common. Rather than minimization of congestion through the insertion of bends and detours, we expand the placement area to obtain additional routing resources. We obtain total wire lengths and congestion levels that are comparable to traditional methods, but with far lower run times. Our routes are detour free and explicitly minimize bends–making wire length estimates made during placement and gate sizing much closer to what is obtained after detail routing.

1. INTRODUCTION

As circuit sizes increase, design tool run times have become more problematic. Low turn-around times are critical to the success of any project. As most design flows include extensive iterative improvement, steps within the flow which consume a great deal of compute time can hamper optimization. Additionally, is is essential to have accurate estimates of wire length and delay–incorrect estimations can cause some portions of a circuit to be over optimized, while other portions can fail to meet performance constraints.

In physical design, placement and global routing can dominate run time. Recent advances in placement algorithms have produced substantial jumps in tool speed; the tool *FastPlace*[12], for example, can be a factor of 100 or more faster than other methods.

For global routing, traditional tools can be quite slow; this has resulted in the development of congestion estimation methods[3, 7, 11, 10, 13]. While these methods are relatively fast, the estimates do not necessarily correspond with global routing results. In particular, a global router may insert detours and bends to avoid congestion; this can invalidate optimizations done by timing-driven placement, gate sizing, and buffer insertion tools.

In this paper, we obtain a substantial speedup in global routing; the approach we present here is hundreds of times faster than the tools *Chi*[5] and *Labyrinth*[6]. Large designs which take an hour or more for *Labyrinth* to complete are finished by our tool in twelve seconds or less. While the routing congestion levels obtained by our method are higher than those of traditional methods, the congestion "hot-spots" are extremely localized.

We propose two methods to address increased congestion. First, if additional space is inserted, congestion levels drop considerably–total wire lengths produced by our method are comparable to the total wire lengths from traditional tools (which have inserted detours into their routes). Many modern designs are not space limited, making this a practical alternative.

Second, in most cases the congestion "hot-spots" are extremely small. A hybrid approach which uses our fast method for the majority of the routing surface, and more computationally intensive methods for small congested areas, is also practical. Total run times for global routing can be reduced, while still obtaining low congestion results.

Unlike prior academic tools, our routing method also explicitly supports three-dimensional routing. Modern designs can have a large number of interconnect layers, with widely varying routing capacity and performance characteristics. The increased numbers of layers does not degrade the run time of our approach, and routings found by our method minimize vias explicitly. Our tool also utilizes the industry standard LEF/DEF format, making it easy to integrate into a modern design flow.

2. PREVIOUS WORK

Global routing has been studied for many years. The problem is commonly abstracted as one of finding a multicommodity flow in a graph. The routing surface is divided into rectangular tiles or "g-cells;" each tile corresponds to a graph vertex, and each boundary between adjacent tiles corresponds to a graph edge.

Rip-up and reroute methods that utilize maze routing are extremely common; recent tools to approach the problem in this way are *Labyrinth*[6] and *Chi*[5]. An approximation algorithm for multicommodity flow was presented by Albrecht[2]; this produces excellent results, but can be extremely slow.

A common objective in global routing is to minimize *congestion*. For each edge in the routing graph, there is a capacity (in terms of the number of signal wires that can utilize it); routing solutions that exceed this capacity are "over congested." The actual capacity in a design can be difficult to measure–detail routing tools can utilize non-preferred direction routing, small open spaces on the bottom metal layers, or even in some cases polysilicon–to complete a routing problem. The effective capacity of a global routing graph greatly depends on the quality of the detail routing tool; in this paper, we assume a relatively simple capacity model.

Unlike most conventional global routers, our tool utilizes a 3-D routing graph as shown in Fig. 1. Each layer has a preferred routing direction and the routing resources are controlled by the capacity of the cell edges perpendicular to the routing direction. The three dimensional model more accurately reflects modern routing instances–there can be an abundance of metal layers, the capacity on each layer differs greatly, and the insertion of vias to change layers impacts both the routability of the design, and the delay of a wire.

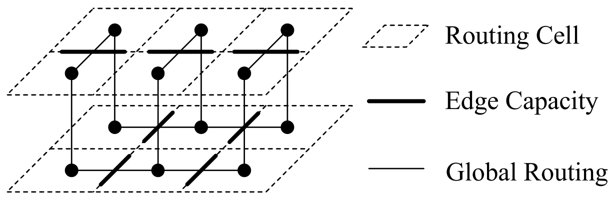


Figure 1: The 3-D routing model.

3. GLOBAL ROUTING APPROACH

With traditional global routing methods, there is an assumption that silicon area is extremely limited–and thus, the placement is as dense as possible to minimize area. To complete routing in this environment, extensive detours are sometimes required, and the routes may have many bends. Maze routing is extremely effective at congestion minimization–but can require high run times. The detours introduced to minimize congestion can increase wire length substantially; it is not uncommon to see routed wire lengths that are 30% or more above the Steiner tree lengths of the nets.

With modern designs, there can be an abundance of silicon area–and thus, there is no need for extremely dense placement. In this work, we assume that the placement area can be expanded if congestion is encountered, and focus on a method that can perform global routing extremely quickly, without introduction of detours or additional bends. While expansion of the placement region increases interconnect lengths, we find that this increase is comparable to the increase caused by detours from traditional routing tools.

We contrast our new approach to the traditional methods in Figure 2. We utilize "pattern routing" ideas implemented in *Labyrinth*, a linearly increasing cost function developed by Linsker[9] (and later implemented in *Chi*), and an efficient "lazy" update scheme. To enable our method to be extremely fast, we avoid maze routing entirely.

Our method decomposes interconnect nets into a set of two-pin tree edges; we then find paths for each edge through the global routing graph. For planar routing problems, we use the *FLUTE* Steiner tree heuristic[4]. For three-dimensional routing problems, we use a layer balancing Steiner heuristic[1]. The layer balancing heuristic determines the locations and layer assignments for all Steiner points and edges.

Tree edges in our routing solutions are restricted to have at most two bends, and are detour-free. Some of the possible routing paths for an edge are shown in Fig. 3. Depending on the amount of routing resource, the restriction on routing paths may or may not increase congestion. In some designs, only a very small portion(1.2% in [13]) of 2-pin nets have more than two bends. In the experiments we perform here, we find that this restriction can sometimes lead to

	Labyrinth	Linsker	Chi	Ours
Pattern Routing	X			X
Maze Routing	X	X	X	
Linear cost function		X	X	X
Congestion estimation			X	
Lazy update				X
3D graph model				X

Figure 2: A comparison of our approach to other recent global routing tools. By eliminating maze routing, and utilizing an efficient update scheme, we obtain routing results extremely quickly.

high congestion levels–but this is normally extremely localized.

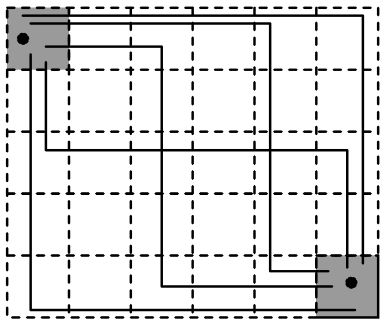


Figure 3: The L- and Z-shaped routing paths.

Our approach is as follows. For each interconnect edge in turn, we enumerate all the L- and Z-shaped paths and select the one with minimal cost. This path is added to the routing graph, and the routing demands are updated accordingly. We iteratively rip-up and reroute all edges until solution quality converges; this normally occurs within 6 passes. The pseudo-code of one iteration is shown in Fig. 4.

While the basic structure of our approach is similar to that of *Labyrinth* or *Chi*, we obtain orders of magnitude speed-up through careful calculation of path costs and updates to the routing graph.

3.1 L- and Z-shaped Routing

When a route is strictly horizontal or vertical, it is inserted into the graph, and is not modified; as there is only one possible path, there is no point to perform rip-up an reroute.

For cases where one or more bends must be introduced, we have to determine the L or Z-shaped path with minimum cost. We do this by enumerating the possible paths; as our global router works in a three-dimensional model, we may need to determine the cost of multiple segments on different interconnect layers. This is illustrated in Figure 5. The cost of a path is the sum of the costs for individual edges in the routing graph (shown with dark lines in the figure).

3.2 Routing Costs

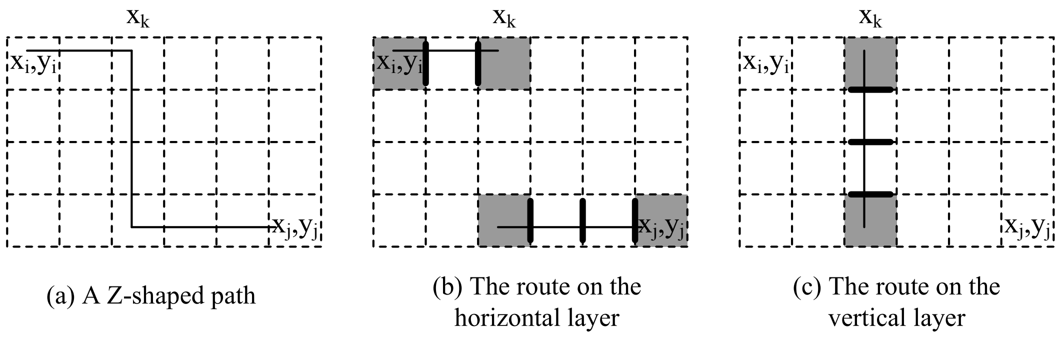


Figure 5: An example of a Z-shaped path and the computation of cost.

```
for net $n_i$ do
    rip-up net $n_i$ and release the routing demands;
    $MinCost = \infty$;
    for candidate path $p_j$ do
        compute the cost $Cost(p_j)$ of path $p_j$;
        if $Cost(p_j) < MinCost$
          then $MinCost = Cost(p_j)$;
               $choice = p_j$;
        endif
    endfor
    update the routing demands;
endfor
```

Figure 4: The Basic Algorithm.

Following the strategy presented in [5], we assign unit cost to each routing graph edge until its demand reaches 80% of its capacity. The cost of using a graph edge increases linearly until it reaches 40% above its capacity. Fig. 6 shows the function of routing cost vs. routing demand.

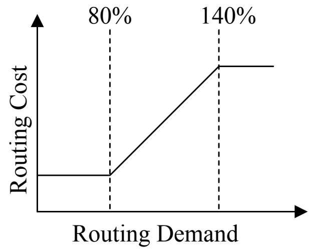


Figure 6: Routing cost is a function of the demand for graph edges.

When routing an interconnect edge, we enumerate the different possible paths, and select the path with minimum cost. After finding this path, we insert it into the routing graph, and increase the utilization on the edges that the path traverses.

3.3 Algorithmic Speedups

There are several factors that allow our approach to be significantly faster than prior tools.

First, because we restrict our routes be monotonic with at most two bends, we eliminate the need to utilize a maze routing algorithm entirely–there is no need to maintain a priority queue, and the total area searched to find a route is much smaller than would be explored by Dijkstra's algorithm.

Second, in the path-based computation above, most of the time is spent on computing the cost for traversing an edge. The cost for the same edge might be examined many times for different nets; we *precompute* a number of values to simplify cost calculations for a series of global routing graph edges. Each routing graph edge at location (x,y) in layer l is assigned a value $Sum^l(x,y)$ which indicates the cost for traversing a row(column) of edges between the left(top) boundary and the edge in question:

$$Sum^l(x,y) = \begin{cases} \sum_{i=1}^{x} Edge\_Cost^l(i,y) & \text{for horizontal layer} \\ \sum_{i=1}^{y} Edge\_Cost^l(x,i) & \text{for vertical layer} \end{cases}$$

Here $Edge\_Cost^l(i,j)$ means the cost of traversing the left edge of the routing graph(if l is a horizontal layer) or upper edge(if l is a vertical layer) to graph edge (i,j) on layer l. By using this, we can find the cost of any *series* of routing graph edges by a simple subtraction. The cost of a path can be obtained by only 3 subtractions and 2 additions. For example, for a path in Fig. 5, the cost of the path is $(Sum^h(x_k,y_i) - Sum^h(x_i,y_i)) + (Sum^h(x_j,y_j) - Sum^h(x_k,y_j)) + (Sum^v(x_k,y_j) - Sum^v(x_k,y_i))$. Only the *Sum* value of the 6 cells(the dark cells in the figure) are examined for computation.

In short – if we wish to know the cost for moving in a particular row from column i to column j, this can be computed easily if we maintain correct partial sums. The computational complexity for determing the minimum cost path is related to the *perimeter* of the bounding box of a tree edge, rather than the *area* of the bounding box.

A third element of our approach to improve tool speed is "lazy update" of the routing costs. We allow the value of $Sum^l(x,y)$ to be incorrect–and update this only when required. Once the demand of one edge is changed due to adding or ripping-up a route, all the *Sum* values succeeding it on the same row(horizontal layer) or column(vertical layer) become invalid. If we were to update the whole row or column immediately after the route is laid, we will have to spend much more time on the updating, thereby eliminating the advantage of *row/column-sum*.

By marking portions of a row or column invalid, we can delay this update, and frequently avoid it completely. For this purpose, a flag for every row(horizontal layer) or column(vertical layer) is used to indicate the boundary between obsoleted *Sum* values and the up to date ones. When we need to determine the cost of a segment, the flag is checked–and only then is the data brought up to date if needed. Further, only the required portion of the graph is brought up to date–we do not update the entire row or column.

To summarize, consider the following example. Suppose a tree edge spans *m* global routing cells horizontally and *n* cells vertically. We will have $m+n-2$ possible ways to route the edge, and for each path we have to compute the cost of a single edge for $m+n-2$ times if we were to simply follow each possible path. In total, $O(m \times n)$ computations would be required for this edge. By using the lazy-update approach, we still have $m+n-2$ routes, we need only 6 math operations to compute the value of each – and thus, the complexity is $(O(m+n))$. We must also update the *Sum* data; by sorting the edges from left to right, the amortized complexity of updating becomes close to linear in our experiments.

When the size of the bounding boxes is relatively small, the run time for our tool when we follow individual paths is comparable to the lazy update method. For problems which have large edge bounding boxes, lazy update can provide a significant advantage.

4. EXPERIMENTAL RESULTS

To evaluate our approach, we compared the performance of our approach to that of *Labyrinth* and *Chi*. We also use a mode of *Chi* that implements the approach by Linsker. These tools support only a planar routing model, so we restrict our tool in a similar manner. As our tool only routes individual tree edges, we first decomposed all nets into planar Steiner trees using *FLUTE*[4]. The Steiner decomposition takes at most a few seconds for all test cases.

For routing benchmarks, we obtained the placements of *mPL-R* from [8]. The circuits have been mapped to a commercial standard cell library, placed by *mPL-R*, and have white space allocation by a cut line shifting method. The commercial tool *Cadence WarpRoute* was able to perform global and detail routing on all designs without errors. As these circuits have been widely used for routability driven placement, and in most cases *Cadence WarpRoute* cannot find acceptable solutions, we consider these benchmarks to be difficult but not impossible.

Inspection of the *WarpRoute* result shows that in many cases, some routing capacity on the first metal layer was used–but in general, the layer is heavily congested with internal cell wiring. As routing capacity of the first metal layer is difficult to estimate, we assume that it is completely blocked, and only use the capacity of the upper layers in our global routing model. We assume that slightly exceeding routing grid capacity does not necessarily imply a routing failure.

As all three routing tools have the same capacity at their disposal, the comparisons between them are level. We cannot directly compare to *WarpRoute*, as it performs both detail and global routing, and utilizes routing capacity on the first metal layer.

An overview of the experiments performed is shown in Figure 7.

4.1 Planar Routing Comparisons

Routing results for the “easy” and “hard” versions of the benchmarks are shown in Tables 1 and 2. Our routing tool is hundreds of times faster than traditional methods, with run times of at most 12 seconds (compared to nearly 1.5 hours for *Labyrinth*). The “easy” benchmarks have more routing space–this causes an increase in run time for the traditional tools, as they are based on maze routing methods.

When given equal routing areas, our method has higher congestion; from the reduction in run time, this should not be surprising. When additional space is available, however, results can be somewhat surprising. To evaluate the impact of additional space for routing, we “stretch” the circuit horizontally by 20%, thereby obtaining an increase in vertical routing capacity.

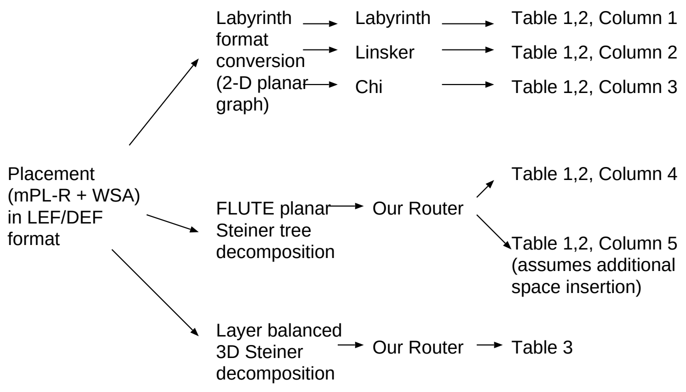

Placement (mPL-R + WSA) in LEF/DEF format
Labyrinth format conversion (2-D planar graph)
Labyrinth
Linsker
Chi
Table 1,2, Column 1
Table 1,2, Column 2
Table 1,2, Column 3
FLUTE planar Steiner tree decomposition
Our Router
Table 1,2, Column 4
Table 1,2, Column 5 (assumes additional space insertion)
Layer balanced 3D Steiner decomposition
Our Router
Table 3

Figure 7: Overview of experiments performed.

Consider, for example, the last entry from Table 1. By expanding the routing space by 20%, our fast router was able to find a solution with better total wire length and lower congestion than *Labyrinth* – and did so nearly 500 times faster.

For modern designs where space is abundant, increasing the routing resource can dramatically reduce congestion, thereby eliminating routing detours. The wire length increase caused by this “stretch” is offset by the elimination of detours. Because no detours are required, the simple pattern-based approach can find a good solution with low run times.

4.2 Three Dimensional Routing

In Table 3, we show congestion levels and run times for our tool when applied to a multi-layer routing graph. We assume that metal 1 is completely blocked, and route on metal 2 through 5.

Because we utilize pattern routing, run times do not increase; congestion and routing demand is distributed across all layers.

4.3 Congestion Distribution

In all cases, while our method produced higher congestion levels, the number of graph edges that were congested was relatively small–at most 25%, and normally much less.

Thus, our method is suitable as a fast initial routing solution; large portions of the routing surface do not require extensive rip-up and reroute with maze routing to find an acceptable solution.

We are currently developing a hybrid approach that utilizes our fast method for the majority of the routing problem. We will apply maze-based techniques to only the portions of the routing graph that exceed capacity constraints.

4.4 Congestion Maps

In Figure 8, we show congestion maps produced by our tool for one of the benchmarks. In most cases, we found that the vertical layers had higher congestion–and thus, the horizontal stretch was quite effective. We show both the “normal” and “stretched” vertical layers, with over-congested regions shown in red.

5. SUMMARY AND FUTURE WORK

In this paper, we have presented a global routing approach that is orders of magnitude faster than prior methods. Problems that take *Labyrinth* an hour or more to complete are finished by our tool in a few seconds. Our tool is fast enough to be used for congestion estimation in routability-driven placement, and is less error-prone than estimation.

In most cases, the vast majority of a design is not close to con-

Bench mark	Labyrinth			Linsker			Chi			Our Router			Our Router scaled capacity		
	wl	OC	RT	wl	OC	RT	wl	OC	RT	wl	OC	RT	wl	OC	RT
ibm01	1.11e+05	0	105	1.09e+05	0	75	1.09e+05	0	98	1.08e+05	2	0	1.27e+05	0	0
ibm02	3.30e+05	79	296	3.05e+05	26	100	3.08e+05	0	149	2.85e+05	5575	1	3.33e+05	1429	1
ibm07	9.66e+05	1510	1095	7.44e+05	103	527	7.53e+05	2	666	6.61e+05	19797	4	7.73e+05	3390	4
ibm08	7.57e+05	8	945	7.44e+05	0	375	7.46e+05	0	480	7.32e+05	1396	4	8.56e+05	929	4
ibm09	6.09e+05	0	822	5.98e+05	0	269	6.04e+05	0	372	5.93e+05	420	3	6.94e+05	168	3
ibm10	1.23e+06	43	2160	1.16e+06	9	701	1.16e+06	2	930	1.13e+06	4617	7	1.32e+06	837	7
ibm11	8.89e+05	0	1419	8.71e+05	0	476	8.72e+05	0	619	8.59e+05	679	5	1.00e+06	275	5
ibm12	2.20e+06	21035	5193	1.86e+06	7044	2760	1.92e+06	5514	3456	1.63e+06	47607	12	1.91e+06	9339	12

Table 1: Routing results on the "easy" benchmarks. For each tool we report the total wire length (wl) in terms of routing grid usage, total overcongestion (OC), and run time (RT) in seconds.

Bench mark	Labyrinth			Linsker			Chi			Our Router			Our Router scaled capacity		
	wl	OC	RT	wl	OC	RT	wl	OC	RT	wl	OC	RT	wl	OC	RT
ibm01	1.07e+05	0	138	1.05e+05	0	79	1.05e+05	0	100	1.05e+05	10	0	1.23e+05	1	0
ibm02	3.60e+05	373	245	3.09e+05	169	118	3.14e+05	28	180	2.81e+05	6725	1	3.29e+05	2207	1
ibm07	1.01e+06	4542	1412	7.54e+05	1362	650	7.74e+05	227	802	6.36e+05	35417	3	7.44e+05	10711	3
ibm08	7.88e+05	64	1043	7.36e+05	0	368	7.40e+05	0	480	7.14e+05	1604	4	8.36e+05	348	4
ibm09	6.24e+05	0	784	6.12e+05	0	312	6.15e+05	0	405	6.05e+05	589	3	7.08e+05	312	3
ibm10	1.25e+06	151	2303	1.16e+06	11	741	1.17e+06	0	1006	1.09e+06	12864	7	1.28e+06	1513	7
ibm11	9.42e+05	51	1569	8.88e+05	10	565	8.92e+05	0	734	8.42e+05	8544	5	9.85e+05	796	5
ibm12	1.95e+06	4442	3290	1.66e+06	225	1406	1.68e+06	71	1680	1.55e+06	13252	11	1.81e+06	6137	11

Table 2: Routing results on the "hard" benchmarks. For each tool we report the total wire length (in terms of routing grid usage), total overcongestion, and run time (in seconds).

gestion limits–and thus, there is little need for computationally intensive routing methods to be used across an entire design. While our routing results have higher congestion, this congestion is extremely localized. We are currently developing a hybrid method which utilizes our fast routing approach for the bulk of the design, and then extracts small portions of the routing problem to be handled by more robust methods.

Unlike prior academic work, and some industrial routing tools, our method can explicitly support a three-dimensional routing model. Different routing layers can have widely differing track pitch and performance characteristics; by using pre-specified Steiner topologies, we can route in three dimensions without increasing run times. As layer assignments are not altered, the delay of individual nets should be more predictable.

The success of our method is based integration of the pattern routing approach of *Labyrinth* with the cost functions described by Linsker. We efficiently compute the cost of individual routes, allowing extensive rip-up and reroute passes without a high run-time penalty.

Our current work involves the development of a detail routing tool. By performing detail routing on the congested regions first, potential locations for routing failure will be identified early, shortening the cycle time for physical design. We are also actively collaborating with a placement research group to develop a fully integrated placement and routing approach.

6. REFERENCES

[1] A. Agnihotri and P. H. Madden. Layer balancing for congestion reduction. In *Proc. Great Lakes Symposium on VLSI*, 2003.

[2] C. Albrecht. Global routing by new approximation algorithms for multicommodity flow. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 20(5):622–631, May 2001.

[3] C.-L. E. Cheng. RISA: Accurate and efficient placement routability modeling. In *Proc. Int. Conf. on Computer Aided Design*, pages 690–695, 1994.

[4] C. Chu. FLUTE: fast lookup table based wirelength estimation technique. In *Proc. Int. Conf. on Computer Aided Design*, pages 696–701, 2004.

[5] R. T. Hadsell and P. H. Madden. Improved global routing through congestion estimation. In *Proc. Design Automation Conf*, pages 28–31, 2003.

[6] R. Kastner, E. Bozogzadeh, and M. Sarrafzadeh. Predictable routing. *Proc. Int. Conf. on Computer Aided Design*, pages 110–113, 2000.

[7] Kusnadi and J. D. Carothers. A method of measuring nets routability for MCM's general area routing problems. In *Proc. Int. Symp. on Physical Design*, pages 186–194, 1999.

[8] C. Li, M. Xie, C.-K. Koh, J. Cong, and P. H. Madden. Routability-driven placement and white space allocation. In *Proc. Int. Conf. on Computer Aided Design*, pages 394–401, 2004.

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

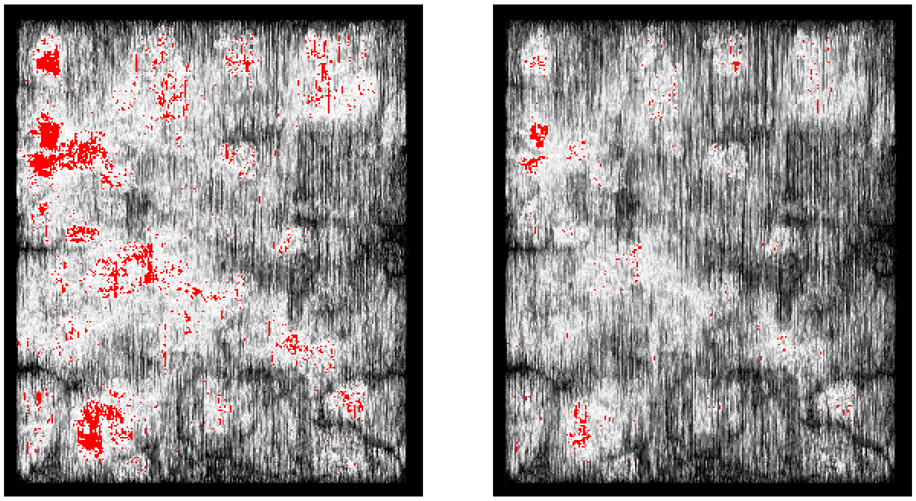
[10] Q. Liu and M. Marek-Sadowska. Pre-layout wire length and congestion estimation. In *Proc. Design Automation Conf*, pages 582–588, 2004.

[11] J. Lou, S. Krishanmoorthy, and H. S. Sheng. Estimating routing congestion using probabilistic analysis. In *Proc. Int. Symp. on Physical Design*, pages 112–117, 2001.

[12] N. Viswanathan and C. C.-N. Chu. Fastplace: Efficient analytical placement using cell shifting, iterative local refinement and a hybrid net model. In *Proc. Int. Symp. on Physical Design*, pages 26–33, 2004.

Bench mark	Easy Benchmarks Over capacity and Run Time					Hard Benchmarks Over capacity and Run Time				
	M2	M3	M4	M5	RT	M2	M3	M4	M5	RT
ibm01	24	27	256	461	0	10	28	176	300	0
ibm02	23	168	138	1	1	36	228	412	2	1
ibm07	41	2829	506	75	3	26	3715	74	37	3
ibm08	106	33	310	0	4	33	93	424	26	4
ibm09	44	25	4	3	3	46	6	90	36	3
ibm10	36	764	713	8	6	43	1662	20	1	6
ibm11	57	109	366	62	5	45	1034	96	146	5
ibm12	214	6954	467	371	11	249	2964	15686	22	10
Avg										

Table 3: Three dimensional routing results. By using layer assignments provided by the Steiner decomposition heuristic, and applying pattern routing, we find a three dimensional global routing solution quickly. Congestion results on individual layers can provide white space insertion methods information on how to adjust a placement to improve routability. The method remains as fast as planar congestion estimation.



Congestion on vertical layer for
a dense placement of IBM12 (hard)

Congestion on vertical layer for
a sparse (stretched) placement

Figure 8: Congestion maps for the vertical routing layer of IBM12 hard, in a "normal" configuration, and with space insertion to reduce congestion. After space insertion, congestion is extremely localized; maze routing methods may be applied without increasing run time excessively.

[13] J. Westra, C. Bartels, and P. Groeneveld. Probabilistic congestion prediction. In *Proc. Int. Symp. on Physical Design*, pages 204–2009, 2004.