Multi-Layer Chip-Level Global Routing Using an Efficient Graph-based Steiner Tree Heuristic

Le-Chin Eugene Liu and Carl Sechen

Department of Electrical Engineering, Box 352500
University of Washington, Seattle, WA 98195

Abstract

We present a chip-level global router based on a new, more accurate global routing model for the multi-layer macro-cell technology. The routing model uses a 3-dimensional mixed directed/undirected routing graph, which accurately models the multi-layer routing problem. However, the complexity of the routing graph challenges previous route-generating algorithms. Generating the routes is to search for the Steiner minimum trees for the nets, which is an NP-hard problem. We developed an improved Steiner tree heuristic algorithm suitable for large routing graphs and able to generate high quality Steiner tree routing. Tested on industrial circuits, our algorithm yields comparable results while having dramatically lower time and space complexities than the leading heuristics[2][14]. While minimizing the wire length, our global router can also minimize the number of vias or solve the routing resource congestion problems.

1 Introduction

The macro-cell design style allows for very compact and high performance designs. But the chip-level routing process has a much higher complexity than other methods. Traditionally, the routing process can be divided into global and detailed routing. Global routing is to find a routing path for each net, and detailed routing assigns the actual tracks and vias. In the past, with one or two layers available for global routing, the job was simply to find the routing channels needed for each net. As the VLSI technologies evolved, the multi-layer routing problem makes issues, such as via minimization and layer assignment, much more difficult. Much research has been done for the multi-layer detailed routing. On the other hand, few multi-layer global routers have been proposed. This indicates that the multi-layer routing is intended to be handled in the detailed routing stage due to the high complexity of the problem. After reviewing the research on routing problems, we found the following issues. First, many global routers[1] have been developed by using planar routing graphs. Basically, they are only for up to two routing layers. Obviously, the planar routing graphs can not be applied to multi-layer technologies since they can not model the new technologies properly. Second, handling multi-layer routing in detailed routing stage lacks a global view for optimization. Optimization in the global routing stage could greatly enhance routing results.

In general, the global routing problem is studied as a graph problem. A routing graph is defined from the layout and the nets are routed on the graph. Searching for the route is best formulated as the Steiner problem in networks, also known as the Graph Steiner Tree Problem. But finding a Steiner minimum tree is an NP-hard problem. Hence, many heuristics have been proposed for practical applications. Conventional global routers use planar routing graphs to model the 2-routing-layer problem. The size of the routing graph (in terms of nodes) is small. Nowadays, for large circuits and multi-layer technologies, it requires complicated routing graphs to accurately handle the problem. The time and space complexities of the applied Steiner tree heuristic become very important factors.

Based on the above observation, we developed a chip-level global router to work with new technologies. First, it is obvious that only 3-dimensional graphs can accurately model the multi-layer technologies. For the complicated routing graphs, we developed a low-time-and-space-complexity Steiner tree algorithm based on an existing heuristic. The improved heuristic can generate better results without sacrificing much efficiency. Our global router generates an initial minimum-cost route for each net. A routing channel may be over-congested when too many nets require routing tracks in that channel. Including congestion factors in the cost function suffers from a net-ordering dependency problem. We used a second stage to solve the congestion problems and partially avoid the net-ordering problem.

The rest of the paper is organized as follows. Section 2 is a brief review. Section 3 describes the global routing model and the methodologies used in our router. Section 4 discusses the heuristic Steiner tree algorithm. Section 5 explains how we solved the congestion problem due to channel capacity constraints. Section 6 shows experimental results and performance. Section 7 concludes the paper.

2 Review

The goal of a global router is to decide a “rough” route for each net. Basically, it assigns routing channels or regions for each net under some constraints. Usually, the nets are routed sequentially. The nets which are routed later face more constraints than those routed earlier. This is known as the net-ordering problem. To totally avoid a net-ordering problem, it is common to either use a random technique or route all nets simultaneously. An integer programming method was proposed in [8] to route all nets at

ED&TC ’97 on CD-ROM

the same time. It is good for solving channel congestions, but the routing results are not good because they are only an approximate solution.

Many global routing methods can be found in [1]. Usually graph-based methods are used for global routing problems. The routing model is based on defining a routing graph from a layout placement. Grid graphs are widely used in many routers[9][10]. Grid graphs are good for array style or sea-of-gates circuits. However, they are not suitable for general macro-cell global routing. Routing graphs directly derived from a floor plan/placement can model the macro-cell design more accurately. These kinds of graphs were used in [11] and [12]. The most accurate routing graph is the channel graph defined in [3].

The key issue for global routing is finding the optimal routes. For two-pin nets, the problem is reduced to the shortest path problem. There are efficient algorithms for solving such a problem. The difficult part is how to deal with the nets with more than two pins. In one approach, the multi-pin nets are decomposed into pairs of pins, then a shortest path method is used for each pair. This method yields poor results. The natural approach is the Steiner formulation of the problem. But all Steiner problems are *NP*-hard[6]. There are no efficient ways of finding the optimal results.

A two-phase global routing model is used by a few routers. They usually use channel graphs and generate a set of routes in the first phase. In the second phase, the program decides one route out of the set of routes of a net to meet the constraints such as channel capacities. Mickey[2] is one of the routers using this technique. Mickey is also one of the best irregular-graph based global routers. It significantly outperformed the previously best known irregular-graph based global router, Mercury[13] and TimberWolfMC[3]. It uses an *M*-route Steiner tree algorithm for finding routes on channel graphs. Mickey has been used in the industry because of its outstanding performance on global routing quality. Mickey's *M*-route method is an improvement over the well-known *KMB* Steiner tree heuristic[4], and it tries to find *M* shortest routes for a net. The first route is found by the *KMB* method. Then it searches for *M*-1 other shortest routes based on the first route. If a better route is found during the searching process, the new route is used as the new first tree. One advantage of this method is that the Steiner minimum trees on the routing graph can almost always be found during the process of searching for the other *M*-1 routes. To solve the congestion in the routing channels, a random method was used to avoid the net-ordering problem. An alternate route is randomly chosen from the *M*-1 routes for a randomly chosen net to replace the shortest route. A cost function is used to evaluate the congestion condition and decide whether the replacement is accepted or not. The two-phase method has two disadvantages. First, the channel graph can not be extended for multi-layer technologies. Second, storing a set of routes for each net requires too much memory. For large VLSI circuits, a high space complexity is not acceptable.

One of the few multi-layer global routers is introduced in [10]. The global router used a 3-dimensional grid routing graph. A hierarchical structure is used, so there are fewer nodes in the graph on higher layers. The route-searching is based on the shortest path method. The router is mainly for sea-of-gate circuits. For macro-cell circuits, grid graphs and hierarchical structures result in inaccuracy.

3 Routing architecture

Our global router is for macro-cell layout, where we assume the macro cells are rectilinear. Given a placement of macro cells, the chip area is divided into small regions by cut lines which are the extension lines of the boundaries of the macro cells. In each region, we place a node for each layer. The nodes for different layers in the same region are connected by via edges. If a layer is used for horizontal tracks, horizontally adjacent nodes of the layer are connected by edges. That means each node of the layer has horizontal edges connected to the nodes of the adjacent regions. Similarly, for layers used for vertical tracks, vertical edges are presented between the adjacent nodes. Furthermore, if a certain layer cannot go through the cells, there will be no edges connecting the nodes of the layer inside the cells. The only exception is the boundary regions of the cells. The nodes in the regions adjacent to cell boundaries still have edges connecting to regions outside the cell. That is because the pins on the cell's boundaries are mapped inside the boundary regions. So the edges across the boundaries on a cell-blocked layer are needed for the pins to exit. But those edges are directed edges. They can only be used for the pins to exit and are not used for any other routing purposes. The directed edges make route searching on the blocked layer efficient. Figure 1 shows an example of the routing graph. The dashed lines are cut lines. For this example, there are two layers. One is for horizontal and the other is for vertical tracks. The horizontal layer is blocked by the cells. Only the vertical layer is available over the top.

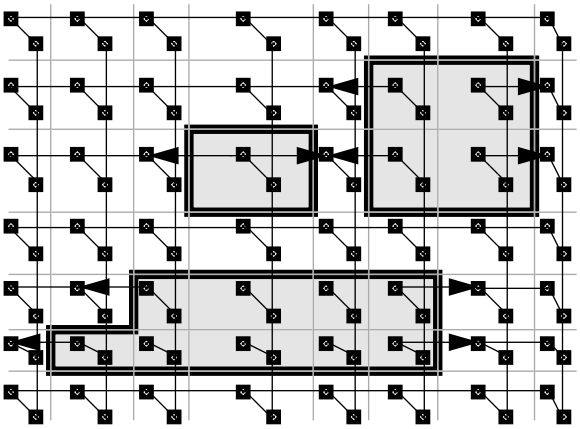

Figure 1. Cut lines and the routing graph.

Some regions can be very small due to cut lines which are close to each other. Those regions cause efficiency problems and do not need to exist. We therefore merge two cut lines when they are too close to each other. We set a threshold for merging as two times the track pitch. That means if a region can accommodate fewer than two tracks, it's merged. The capacity of the regions have to be adjusted due to the merging. The routes are searched on the routing graph according to the weights of the edges. The weights of vias can be assigned to reflect the resis-

tance of vias. Or if the number of vias is to be minimized, the via edges can be assigned a huge weight. This is an effective way of modeling the use of vias for VLSI layout. Usually, the weights of non-via edges are set to reflect the wire length. But the edge weights on different layers can also be adjusted according to the different resistance or other measures which differentiate the layers.

The advantage of the routing graph is that it closely models the actual multi-layer features and it is also very flexible. For each additional layer, it requires one more layer of nodes. Each layer can be configured individually. The routing graph is constructed due to the configuration of each layer. The via factor is also taken care of inherently in the routing graph. This routing model provides the ability to evolve with modern VLSI technology.

After the routing graph is constructed, the global routing is done net by net. For a net, all the pins are mapped to the nodes corresponding to the layer specified and the regions where the pins reside. For the pins on the boundaries of the cells, we map them inside the cells unlike many other global routers which map the pins outside the cells. Our graph yields a more accurate estimate of via usage. Finding a route for a net is to search for a Steiner minimum tree on the routing graph. The algorithm will be introduced in next section.

4 Route-generating algorithm

4.a Introduction

Searching for the minimum-cost route for a net on a routing graph is naturally the Steiner problem in networks. The definition of the Steiner problem in networks from [6] is as follows:

- GIVEN: An undirected network $G=(V,E,c)$ where
  $c: E \rightarrow R$ is an edge length function, and a non-empty set N, $N \subseteq V$, of terminals.
- FIND: A subnetwork $T_G(N)$ of G such that:
  there is a path between every pair of terminals,
  total length $\left|T_G(N)\right| = \sum_{e \in T_G(N)} c(e)$ is minimized.

$T_G(N)$ is called a Steiner minimal tree of G. Those vertices which are not terminals in the Steiner tree are called Steiner points. For the global routing problem, the pins of a net are the terminals to be connected. The Steiner minimal tree is the minimum-cost route.

Although our graph is not undirected for some cases, basically, the global routing problem is still a Steiner problem. The Steiner problem in networks is an NP-hard problem[6]. There is no efficient way of finding the Steiner minimum tree. There are many heuristics introduced in [6]. There is one parameter for evaluating the effectiveness of a heuristic. It is the worst-case bound, which is the ratio of the wire length of the worse case generated by the heuristic over the wire length of the optimal result. For most of the heuristics, the best worse-case bound is $2(1-1/n)$, where n is the number of the terminals. Zelikovsky[15] proposed a heuristic which has a worse-case bound of 11/6. But the time complexity is $O(|V||E|+n^4)$, where $|V|$ is the number of vertices, and $|E|$ is the number of edges. It is too high to be used for global routing. The well-known *KMB* algorithm[4] has a worse-case bound of $2(1-1/n)$. It has a low time complexity. Later research showed that the heuristic can be implemented with a time complexity of $O(|E|+|V|log|V|)$. For global routing purpose, the *KMB* algorithm's results need further improvement to be satisfactory, like Mickey[2] did. In [14], an iterated algorithm is proposed to improve any Steiner tree heuristic. The time complexity can be as high as $O(|N||V|t(H))$, where $t(H)$ is the time complexity of the original heuristic.

To find the Steiner minimal tree is to find the proper Steiner points in the network. When the Steiner tree heuristics are applied to the routing problem, extra improvement stages are used to generate better results. For example, Mickey[2] used the *KMB* method to find the first tree. A few more trees are searched for based on the first tree. The process explores the possible Steiner points for the Steiner minimal tree, so the optimal results can often be found when the first tree is not optimal.

We developed our algorithm based on Wang's heuristic[7] which belongs to the category of the shortest path Steiner tree heuristics. In general, the shortest path heuristics often generate better results than many other Steiner tree heuristics[6]. Our algorithm shares the same worse-case bound of $2(1-1/n)$. We modified the original heuristic in a way to explore more Steiner points for finding the optimal tree. Wang's heuristic searches one shortest path between the terminal sets at one step. We keep multiple shortest paths during the process of searching for connection. This can be done virtually without increasing the time complexity. Then a greedy improvement stage is used to generate better results. In addition, we took advantage of our sparse routing graph to reduce the time complexity. This will be shown in sub-section IV.f. The result is a new low-time-and-space-complexity Steiner tree algorithm.

4.b Generate_route

In the beginning, each terminal (pin) is made a set in a disjoint set data structure. The shortest path is found among all pairs of sets. For the two sets corresponding to this short path, we store *all* shortest paths if more than one exists. All the pin nodes and the nodes of the paths are merged into one set. This process continues until only one set remains. At this point, we have a sub-graph of the routing graph. It is not a tree because it may contain cycles due to the set of equivalent-weight paths retained. This paths graph may not consist of the shortest paths for some nodes, so an improvement stage is needed. The improved paths graph is sent to the next stage to remove the cycles. After the cycles are removed, another improvement stage is applied to further improve the tree. The procedure of *Generate_route* is as follows:

```
Generate_route(net)
1. each terminal forms a set
2. while (number of sets > 1) {
3.       find the shortest path(or paths) between any
         two sets
4.       merge the two sets into one set according to
         the paths found
5. }
6. Improve_route(path_graph)
7. Remove_cycles(path_graph)
8. Improve_route(path_tree)
```

Lines 1-5 is the original Wang's algorithm except keeping the multiple paths. After the execution down to line 5, a paths graph is obtained. Figure 2 shows an example. Figure 2(b) is the top view of Figure 2(a). The top view doesn't show the via edges, so it demonstrates how *Generate_route* works more clearly. Three pins on the cell boundaries are mapped to the nodes inside the cells. Since pin *A* is closer to pin *B* on the routing graph, the path between them is found first. During the second iteration, two shortest paths between pin *A* and pin *C* are found. All the pins are now connected, so the loop concludes. Obviously, the paths graph is not optimal. It will be improved in the next stage.

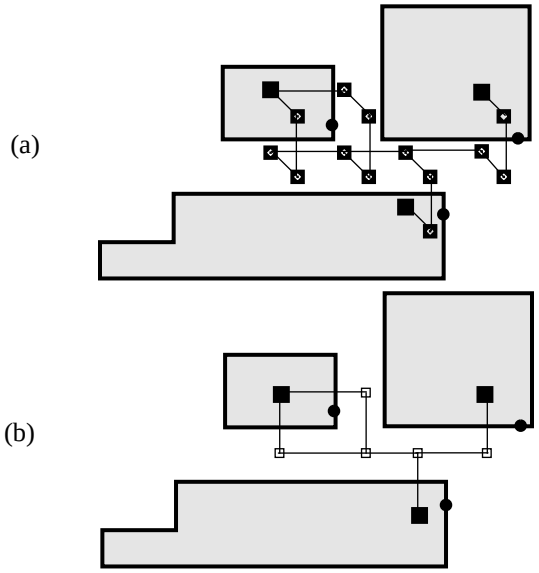

Figure 2. Paths graph of a net and its top view.

4.c Improve_route

To improve the routing, all the segments in the paths graph are examined. First, all the edges (in the paths graph) are put in an array. One starting edge is chosen from the beginning of the array. This edge, i.e. its two endpoints, is extended in both directions to form a segment. The end points of such a segment are either a required node (pin) in the original routing graph or a node with degree more than two. For example, in Figure 2(a), the edge (*e*,*f*) is selected as a starting edge. The edge is extended from both end points and forms a segment from *d* to *a*. We mark all edges in the segment, so that the edges will not be used as starting edges later on. The segment is removed from the paths graph. If the paths graph is now divided into two disconnected sets, find a shortest path between the two sets. If this new path has a lower weight than the original segment, the new path is used instead of the original segment. The process continues until all edges are marked. The procedure of *Improve_route* is as follows:

```
Improve_route(graph)
1.  for (all edges in the graph) {
2.      if (the edge is marked) continue
3.      create a segment by extending the edge to a
        required node or a node with degree > 2
4.      mark all edges in the segment
5.      if (two sets are created by removing the seg-
        ment) {
6.          find a shortest path between the two sets
7.          if (the new weight is lower) replace the
            old segment with the new path
8.      }
9.  }
```

• Example

The paths graph shown in Figure 2 is sent to *Improve_route*. There are a few segments in that graph. Only the segment between the bottom pin (A) and the right pin (B) can be improved. That segment is removed and a shorter path between the two separated sets is found. Hence the original segment is replaced by the new path. The improved paths graph is shown in Figure 3.

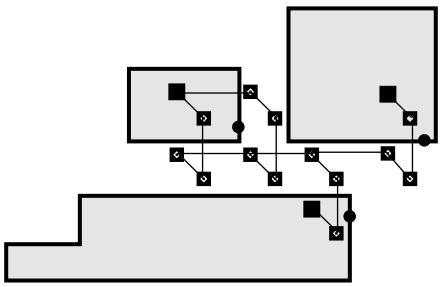

Figure 3. Improved paths graph.

4.d Remove_cycles

To remove the cycles in the paths graph, all segments in the paths graph are generated. The segments are sorted according to the weight of the segment. Starting from the largest-weight segment, the segment is removed. If this action causes the graph to be divided into two sets, restore the segment. Otherwise, remove the segment from the paths graph permanently. This is done sequentially for all segments. The procedure of *Remove_cycles* is as follows:

```
Remove_cycles(graph)
1.  sort the segments of the paths graph according to
    their weights
2.  the sorted segments are put into a queue with the
    largest-weight segment in the front
3.  for (all the segments in the sorted queue) {
4.      remove the segment
5.      if (the graph is not divided into two sets) {
6.          if (two other segments can be merged
            due to the removal of the segment) {
7.              merge the two segments and adjust
                the queue
8.          }
9.          continue
10.     }
11.     restore the segment
12. }
```

• Example

Figure 4 shows that the cycle in the paths graph has

been removed and an optimal route tree is obtained.

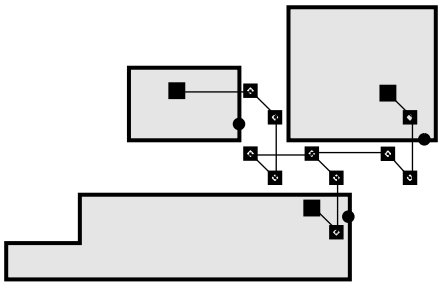

Figure 4. Final route for a net.

Figure 5 is used to show how lines 6-9 of *Remove_cycles* work. The largest-weight segment may be *B*-*C*-*F*, so it is removed first. Originally, there are two segments *A*-*B* and *B*-*E*. Because of the removal of *B*-*C*-*F*, *A*-*B* and *B*-*E* are connected to form a new segment.

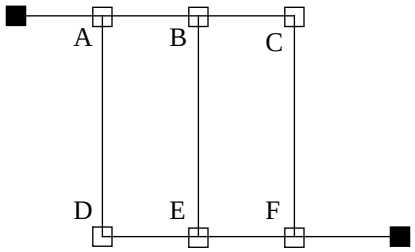

Figure 5. Example of a paths graph.

For the example in Figure 4, the second execution of *Improve_route* does nothing. For some other cases, the optimal tree might not be obtained after *Remove_cycles* has been executed. A second execution of *Improve_route* is needed for those cases. This will be shown in next subsection.

4.e Discussion

Figure 6 shows the importance of keeping multiple paths in the paths graph and the first improvement stage. It is an example of a 3-pin net. Nodes *A*, *B*, and *C* are the required nodes. The path between *B* and *C* is formed first, because it is shorter than the path between *A* and *B*. Then, paths between *A* and *B* are found. The paths graph is shown in the bold lines. If the paths graph directly goes to the cycle_removing stage, segment *A*-4-*B* may be removed. Without node 4, it is not possible to find the Steiner minimum tree. Figure 7 shows the improved paths graph. The remove_cycle procedure will remove segment *A*-5-*B* now. The minimum Steiner tree was found. If multiple paths were not kept, the Steiner minimum tree may not be found. For example, if the tree *A*-5-*B*-6-*C* was found as the sole route, then this tree cannot be improved by *Improve_route*. This example shows the advantage of retaining multiple paths and why we need the first improvement stage.

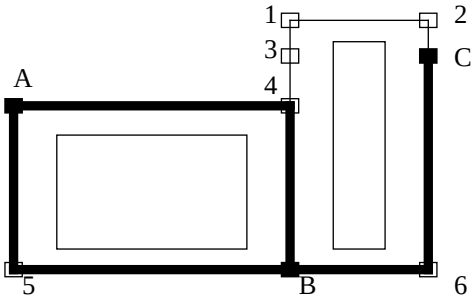

Figure 6. Example of a net before improvement.

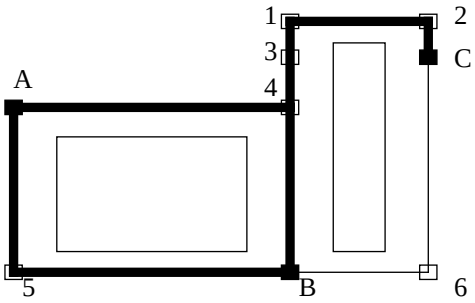

Figure 7. Example of a net after improvement.

Figure 8 shows another problem we encountered. Figure 8(a) is the paths graph for a 4-pin net. There are two equal-weight paths between each pair of pins. This is not unusual for a multi-layer layout. Segments 3-4-*D* and 7-8-*D* are the largest-weight segments. Segments *A*-1-2 and *A*-5-6 are the second largest-weight segments. During the cycle-removing stage, it is possible that segment 3-4-*D* and *A*-5-6 are removed. The tree, after the cycles have been removed, is shown in Figure 8(b). Obviously, this is not the Steiner minimum tree. With the second execution of *Improve_route*, the Steiner minimum tree is found. The result is shown in Figure 8(c).

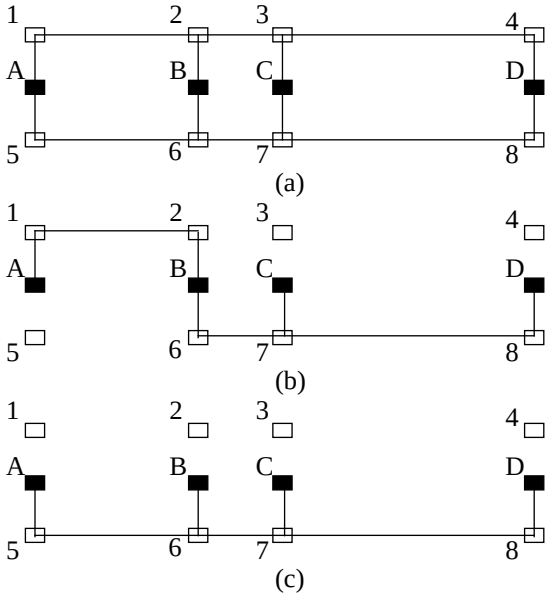

Figure 8. An example of the need for the second execution of Improve_route.

The key to our algorithm is that we incorporate more Steiner points for the later improvement stages to work on. But if there are too many equivalent-weight paths, it could cause efficiency problems and make the cycle-removing part less effective. Two methods are used to prevent the algorithm from incorporating too many paths. One is that we try to keep the outside paths only, i.e. the paths which are enclosed by other paths are discarded, because the inside paths don't provide useful Steiner points. The other is that we directly limit the number of paths which can be incorporated. In the current implementation, the limit is 20. On the other hand, for some graphs, we found that there hardly exists any equivalent paths. We therefore retain and treat the paths which are slightly higher in weight (up to 1% higher than the minimum weight path) as equivalent-weight paths.

4.f Time complexity

Given is the routing graph $G=(V,E,c)$. Because there are at most six edges connected to a vertex, the relation, $|E| < 6|V|$ holds. We route a net n_i with p pins. For *Remove_cycles*, the sorting has a time complexity of $O(|S| \log |S|)$, where $|S|$ is the number of the segments. In the loop, the worst case occurs when we must update the queue every time, so the time complexity for the loop is $O(|S|^2)$. The complexity of *Remove_cycles* is $O(|S|^2)$. The end point of a segment has a degree greater than 2 if the end point is not a terminal (pin). A theorem from [6] says that for a Steiner minimum tree the number of such vertices is less than two times the number of terminals (pins). The number of segments is proportional to the number of vertices. So $|S|$ is proportional to p. To find the shortest paths between the sets, we use an algorithm similar to Dijkstra's algorithm. The difference is that we have multiple sources instead of one. We start from multiple sources and update the weight for the vertices and put the vertices into a priority queue. Because the routing graph is sparse, according to [5], the time complexity is $O((|V|+|E|) \log |V|)$. For our case, it can be simplified to $O(|V| \log |V|)$. So the *Improve_route* subroutine has the time complexity of $O(|S||V| \log |V|)$. For *Generate_route*, the main loop has a time complexity of $O(p|V| \log |V|)$. So for the subroutine, the dominant parts are $O(p|V| \log |V|)+O(|S||V| \log |V|)+O\ |S|^2)$ for net n_i. Since $|S|$ is proportional to p and p is not greater than $|V|$, the overall time complexity of our algorithm is $O(p|V| \log |V|)$, which is also the time complexity of the original Wang's algorithm. This means our modification retains the same time complexity.

5 Resolving congestion problems

After the global routing is done for all nets, the global router can estimate the routing resources needed. If a route consists of a certain node, it means a track is required in the corresponding routing region for that net. So we can estimate the number of tracks used for each layer in each region. The number of via needed can be estimated by counting the number of via edges used by the routes. The wire length of each net can be estimated by adding the wire length of all the edges used by the net. The path for each net consists of not only the topological information but also the layer and via information. The information can facilitate the detail routing process.

If a region has the exact number of tracks which it can accommodate, the region is a full region. If a region has more tracks than it can accommodate, it is called an over-congested region. For global routing, the goal is to have no over-congested regions. Since no congestion information is included in the edge's weight function, we have to resolve the congestion problems after the initial routing is generated. We use an exhaustive rip-up and re-route method to eliminate the over-congested regions. Starting from the most over-congested region, the following algorithm is executed:

1. for (each net in the over-congested region) {
2. try to find a new route which avoids all over-congested and full regions
3. if (no such a new route exists) continue
4. put the new route in a priority queue according to the increase of the weight
5. }
6. while (the region is still over-congested)
7. replace an old route by its corresponding new route in the priority queue in order

The method seeks to re-route all the nets using that region while avoiding the creation of additional congestion problems. This is different from a simple rip-up and re-route method. Although it is more time-consuming, it can partially avoid the net-ordering problem. Since our Steiner tree heuristic is very efficient, it is practical to use such a strategy. After the most congested region is processed, it moves on to the next most congested region. It continues until all the over-congested regions have been processed.

6 Results

We compared our Steiner tree algorithm with four other graph-based algorithms capable of handling irregular graphs. One is the *KMB* method[4]. The second algorithm we compared against is the *M*-route method introduced in Mickey[2] which is an improvement over the *KMB* algorithm. The drawback is that it requires too much memory and is therefore impractical for large circuits. The third algorithm is *IKMB*[14]. It is an iterated improvement over *KMB*. In [14], it was shown that *IKMB* outperforms the Zelikovsky algorithm[15]. But *IKMB* stills bears a high time complexity. The fourth algorithm is *IZEL*, which is an iterated improvement over the Zelikovsky algorithm. *IZEL* is shown to be better than *IKMB* in [14], although it has an enormous time and space complexity. We tested the programs on some industrial circuits. Those circuits are shown in Table 1. The placements were generated by TimberWolfMC v.3.1. We ran all the programs on the same global routing graph. They were run on a DEC 3000 APX Model 400 workstation.

number of	cells	nets	pins	nodes of graph	edges of graph
hp	11	83	309	26	39
ami33	33	83	376	64	101
qpdm_b	17	121	645	37	58
xerox	10	203	696	21	30
amd	17	288	837	39	57
ami49	49	408	953	108	172
4832	20	586	1,576	64	98
intel	62	570	4,309	161	243

Table 1. Circuit information.

The wire length results for the four algorithms are shown in Table 2. The *KMB* algorithm is outperformed by the other algorithms by about two percent on average. Our Steiner tree algorithm consistently generates better results than Mickey's algorithm, and about the same as *IKMB* in general.

Table 3 shows the memory usage and run time of Mickey, *IKMB*, and our Steiner tree program. The results in Table 3 show our algorithm uses much less memory than Mickey, while having a comparable (or slightly better) run time. The memory usage was reduced by a factor of nearly nine for the largest circuit (*intel*). Table 3 also shows that for circuits with complicated routing graphs, the *IKMB* and *IZEL* algorithms really suffer from their high time complexities. The asterisks in Table 3 imply that *IZEL* could not complete the routing of these circuits after being allotted more than 200 hours of CPU time. The pound signs (#) indicate that the job could not be completed because it required more than the 384MB virtual memory available on our machine. Note that for the largest circuit, *intel*, our Steiner tree algorithm was more than 500 times faster and used 15 times less memory than *IKMB* and yielded similar results. Even for the small circuit, *xerox*, our Steiner tree algorithm was more than 40,000 times faster than *IZEL* and yet yielded equivalent results. For large VLSI circuits, it is obvious that only our algorithm is applicable.

Table 4 shows the results of our global router for two very large industrial circuits. Four routing layers were

	wire length / percentage difference w.r.t. KMB				
	KMB	Mickey *M*-route	*IKMB*	*IZEL*	New Steiner tree algorithm
hp	176,808 / 0	171,430 / -3.04%	169,927 / -3.89%	169,239 / -4.28%	170,063 / -3.81%
ami33	56,770 / 0	55,865 / -1.59%	55,815 / -1.68%	*	55,815 / -1.68%
qpdm_b	633,540 / 0	626,907 / -1.05%	626,127 / -1.17%	625,930 / -1.2%	625,930 / -1.2%
xerox	568,480 / 0	561,935 / -1.15%	561,935 / -1.15%	561,935 / -1.15%	561,935 / -1.15%
amd	261,478 / 0	259,856 / -0.62%	259,782 / -0.65%	259,740 / -0.66%	259,843 / -0.63%
ami49	371,362 / 0	361,378 / -2.69%	360,927 / -2.81%	360,278 / -2.98%	360,592 / -2.9%
4832	1,934,200 / 0	1,894,400 / -2.06%	1,893,475 / -2.11%	1,888,215 / -2.38%	1,891,390 / -2.21%
intel	6,087,362 / 0	5,942,640 / -2.38%	5,913,290 / -2.86%	#	5,925,695 / -2.66%

Table 2. Wire length comparison.

	Mickey		*IKMB*		*IZEL*		New Steiner tree alg.	
	memory (bytes)	run time (sec)	memory (bytes)	run time (sec)	memory (bytes)	run time (sec)	memory (bytes)	run time (sec)
hp	666K	3.1	2,080K	1.2	2,176K	284.1	355K	1.7
ami33	1,855K	2.3	3,112K	30.3	*	*	438K	6.3
qpdm_b	2,124K	2.2	2,448K	6.2	3,184K	30,866.1	784K	5.9
xerox	1,269K	4.2	3,592K	6.8	9,400K	107,202.8	735K	2.6
amd	4,594K	25.0	2,120K	3.3	2,176K	51.6	1,272K	4.7
ami49	6,228K	20.0	2,579K	20.8	2,918K	16,466.9	1,316K	5.4
4832	9,032K	16.7	2,528K	14.0	2,816K	2,978.5	2,142K	8.9
intel	50,642K	224.7	89,824K	105,027.8	#	#	5,707K	203.9

Table 3Memory and run time comparison.

	cells	nets	pins	memory (bytes)	wire length	vias	run time (sec)
Intel1	37	7,285	17,578	37,725K	42,299,146	49,806	1,430.6
Intel2	189	9,497	31,647	104,607K	24,290,738	58,612	63,930.4

Table 4Results on two industrial circuits.

available for both circuits. The lowest horizontal and vertical layer are blocked by the cells and the other two, one for horizontal and one for vertical, are available over the cells. Because Intel2 has so many cells, the 4-layer routing graph consists of more than 38,000 nodes. Yet, the job was completed with reasonable run time. The results demonstrate that our global router can handle modern large industrial circuits, whereas previous Steiner tree algorithms could not be successfully applied to these large circuits.

7 Conclusion

We presented a chip-level global router based on a new, more accurate global routing model for the multi-layer macro-cell technology. The routing model uses a 3-dimensional mixed directed/undirected routing graph, which provides not only the topological information but also the layer information. The irregular routing graph accurately models the multi-layer routing problem, so the global router can give a good estimate of the routing resources needed. However, the complexity of the routing graph challenges previous route-generating algorithms. Generating the routes is to search for the Steiner minimum trees for the nets. Since the Steiner problem in networks is an NP-hard problem, we developed an improved Steiner tree heuristic algorithm which is suitable for large routing graphs and able to generate high quality Steiner tree routing. Tested on industrial circuits, our algorithm yields comparable results while having dramatically lower time and space complexities than the leading heuristics. This advantage makes our global router applicable to large industrial circuits, easily handling multi-layer problems consisting of 200 macro cells and 10,000 nets. While minimizing the wire length, our global router can also minimize the number of vias, or solve the routing resource congestion problems.

8 Acknowledgments

We are grateful to Professor Gabriel Robins and Michael Alexander of the Department of Computer Science at the University of Virginia for providing their Iterated Graph Steiner Minimum Tree program. We also want to thank the Semiconductor Research Corporation, Intel Corporation, Digital Equipment Corporation, and Cadence Design Systems for their financial support.

References

[1] N. Sherwani, “Global Routing,” Chapter 6 in “Algorithms for VLSI Physical Design Automation,” Kluwer Academic Publishers, 1993.

[2] D. Chen, and C. Sechen, “Mickey: A Macro Cell Global Router,” Proceedings of the European Conference on Design Automation, pp. 248-252, Feb. 1991.

[3] C. Sechen, “VLSI Placement and Global Routing Using Simulated Annealing,” Kluwer Academic Publishers, 1988.

[4] L. Kou, G. Markowsky, and L. Berman, “A Fast Algorithm for Steiner Trees,” Acta Informatica 15, pp. 141-145, 1981.

[5] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, “Introduction to Algorithms,” McGraw-Hill book company, 1992.

[6] F. K. Hwang, D. S. Richards, and P. Winter, “The Steiner Tree Problem,” North-Holland, 1992.

[7] S. M. Wang, “A Multiple Source Algorithm for Suboptimum Steiner Trees in Graphs,” H. Noltemeier (Ed.) Proceedings of International Workshop on Graph-Theoretic Concepts in Computer Science, pp. 387-396, 1985.

[8] J. Heisterman and T. Lengauer, “The Efficient Solution of Integer Programs for Hierarchical Global Routing,” IEEE Transactions on Computer-Aided Design, Vol. 10, No. 6, pp. 748-753, June 1991.

[9] Y.-L. Lin, Y.-C. Hsu, and F.-S. Tsai, “Hybrid Routing,” IEEE Transactions on Computer-Aided Design, Vol. 9, No. 2, pp. 151-157, Feb. 1990.

[10] M. Hayashi and S. Tsukiyama, “A Hybrid Hierarchical Global Router for Multi-Layer VLSI’s,” IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol. E78-A, No. 3, pp. 337-344, Mar. 1995.

[11] J. G. Xiong, “Algorithms for Global Routing,” 23rd Design Automation Conference, pp.824-830, June 1986.

[12] W. K. Luk, D. T. Tang, and C. K. Wong, “Hierarchical Global Wiring for Custom Chip Design,” 23rd Design Automation Conference, pp.481-489, June 1986.

[13] Y. Nishizaki, M. Igusa, and A. Sangiovanni-Vincentelli, “Mercury: A New Approach to Macro-cell Global Routing,” Proceedings of VLSI 89 Conference, Munich, Germany, pp. 411-420, Aug. 1989.

[14] M. J. Alexander, and G. Robins, “New Performance-Driven FPGA Routing Algorithms,” 32nd Design Automation Conference, pp.562-567, June 1995.

[15] A. Z. Zelikovsky, “An 11/6 Approximation Algorithm for the Network Steiner Problem,” Algorithmica, 9, pp.463-470, 1993.