A New Global Router for Modern Designs *

Jhih-Rong Gao
Synopsys Inc.
Taipei, Taiwan 11012
e-mail : jerrica.gao@synopsys.com

Pei-Ci Wu
Synopsys Inc.
Taipei, Taiwan 11012
e-mail : peggie.wu@synopsys.com

Ting-Chi Wang
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan 30013
e-mail : tcwang@cs.nthu.edu.tw

Abstract - In this paper, we present a new global router, NTHU-Route, for modern designs. NTHU-Route is based on iterative rip-ups and reroutes, and several techniques are proposed to enhance our global router. These techniques include (1) a history based cost function which helps to distribute overflow during iterative rip-ups and reroutes, (2) an adaptive multi-source multi-sink maze routing method to improve the wirelength of maze routing, (3) a congested region identification method to specify the order for nets to be ripped up and rerouted, and (4) a refinement process to further reduce overflow when iterative history based rip-ups and reroutes reach bottleneck. Compared with two state-of-the-art works on ISPD98 benchmarks, NTHU-Route outperforms them in both overflow and wirelength. For the much larger designs from the ISPD07 benchmark suite, our solution quality is better than or comparable to the best results reported in the ISPD07 routing contest.

I Introduction

In the recent years, feature size continues to shrink. Although the device becomes smaller and faster, the shrinkage increases the wire resistance and hence interconnect delay. Interconnect delay has replaced transistor delay as the main determinant of chip performance. Therefore the routing problem is becoming even more important in VLSI design.

Typically, the routing problem can be divided into two steps due to the problem complexity: global routing and detailed routing. During global routing, nets are connected on a coarse-grain grid graph with capacity constraints. Then detailed routing follows the solution in global routing to find the exact routing solution. The quality of global routing affects timing, power and density in the chip area, and thus global routing is a very important stage in the design cycle.

Recent global routing techniques can be roughly categorized into two classes: multicommodity flow based techniques and rip-up and reroute techniques. Multicommodity flow based techniques [1, 2] formulate global routing as a multicommodity flow problem which can be mapped to a zero-one integer linear programming (ILP) problem. Due to the large time complexity, the ILP problem is often relaxed to the fractional version and solved by an approximation method. A recent multicommodity flow based router [2] is proposed by Albrecht, which provides an approximation method to solve the LP formulation.

Rip-up and reroute approach starts by routing each net without considering congestion. After routing all nets, congested areas can be identified and the nets in those areas are ripped up and rerouted to find less congested routes. This approach is a sequential one since the net to be ripped up and rerouted has to follow a specific order. Therefore the routing order in rip-up and reroute techniques affects the solution quality a lot. Chi Dispersion [3] and Labyrinth [4] are global routers which utilize rip-up and reroute techniques. Recently two efficient algorithms DpRouter [5] and FastRoute [6, 7] are proposed and outperform Chi and Labyrinth. BoxRouter [8] expands a box which initially covers the most congested region and routes all nets within the box by ILP formulation. Although the solution quality of BoxRouter is also better than Chi and Labyrinth, the large computational time for an ILP solver makes it less scalable. In addition to the above routing approaches, an optimization method based on trunk decomposition [9] is presented, which reduces considerable overflow from an initial routing solution.

In this paper, we present a new global router, NTHU-Route, for modern designs. NTHU-Route is based on iterative rip-ups and reroutes, and several techniques are proposed to enhance our global router. These techniques include (1) a history based cost function which helps to distribute overflow during iterative rip-ups and reroutes, (2) an adaptive multi-source multi-sink maze routing method to improve the wirelength of maze routing, (3) a congested region identification method to specify the order for nets to be ripped up and rerouted, and (4) a refinement process to further reduce overflow when iterative history based rip-ups and reroutes reach bottleneck. We compare our results with two state-of-the-art works, BoxRouter and FastRoute, on ISPD98 benchmarks. Our global router solves all benchmarks without any overflow and respectively reduces the wirelength over BoxRouter and FastRoute by 1.93% and 2.59% on average. We also perform our router on ISPD07 benchmarks which contain multi-layer designs with larger size. The experiments show that our router obtains the solution with least overflow when comparing with the best results reported in the ISPD07 global routing contest.

The rest of the paper is organized as follows. Section II gives the preliminaries including the problem formulation and introduction for some routing techniques. In section III, we present our global router in detail. Section IV provides the experimental results and we conclude the paper in section V.

* This work was partially supported by Ministry of Economic Affairs under Grant Number MOEA-95-EC-17-A-01-S1-031 and National Science Council under Grant Numbers NSC-96-2220-E-007-045 and NSC-96-2220-E-007-047.

978-1-4244-1922-7/08/$25.00 ©2008 IEEE

II. Preliminaries

A. Problem formulation

The global routing problem requires a set of nets, $\boldsymbol{N} = \{n_1, n_2, \ldots, n_k\}$, to be routed over a grid graph $G(\boldsymbol{V}, \boldsymbol{E})$. A net n_i, $1 \le i \le k$, is a set of pins. Typically the layout is partitioned into rectangular tiles called global bins as shown in Fig. 1 (a), and each pin is assumed to lie at the center of the tile that contains the pin. In the grid graph G shown in Fig. 1(b), each vertex $v \in \boldsymbol{V}$ represents a global bin and each edge $e \in \boldsymbol{E}$ corresponds to a boundary between two adjacent global bins. The routing problem for a net n_i is to find an additional subset of vertices, $v_{i,Steiner} \subset \boldsymbol{V}$, and a subset of edges, $e_i \subseteq \boldsymbol{E}$, to form a Steiner tree $t_i = (v_i, e_i)$, where $v_i = n_i \cup v_{i,Steiner}$.

The supply $s(e)$ of an edge e represents the number of available routing tracks it contains. The number of wires that utilize an edge e is called the demand $d(e)$ on the edge. The overflow on an edge e is defined to be the difference between its demand and supply as shown below:

$$overflow(e) = \begin{cases} d(e) - s(e) & \text{if } d(e) > s(e) \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

And we define the congestion of e as the ratio of the demand to the supply:

$$cong(e) = \frac{d(e)}{s(e)} \quad (2)$$

The major optimization objective of global routing is to minimize the total overflow on all edges:

$$Minimize: \sum_{e_i \in \boldsymbol{E}} overflow(e_i) \quad (3)$$

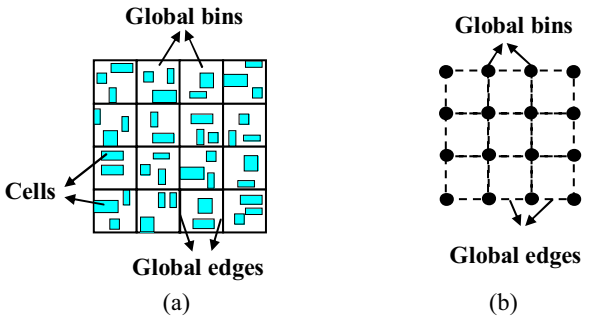

Fig. 1. (a) A routing region with 4x4 global bins. (b) The corresponding grid graph of (a).

B. Monotonic routing

Pattern routing is a method to route a two-pin net with predefined patterns. L-shaped and Z-shaped are two commonly used patterns as shown in Fig. 2(a) and Fig. 2(b), respectively. Pattern routing is much more efficient compared with maze routing which considers all edges inside a given window. However, the quality may be worse since the search space of pattern routing is more limited. In order to find the tradeoff between pattern routing and maze routing, FastRoute [7] proposed monotonic routing which provides larger solution space with the same time complexity as Z-shaped pattern routing. The idea of monotonic routing is as shown in Fig. 2(c).

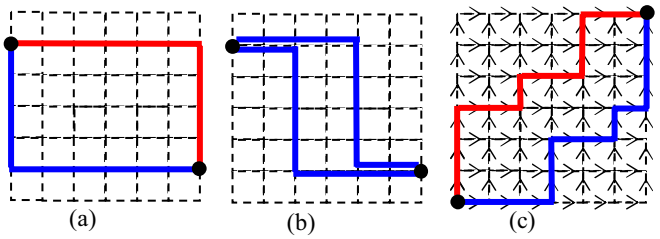

Fig. 2. (a) L-shaped patterns. (b) Z-shaped patterns. (c) Monotonic patterns.

C. Negotiated congestion routing

PathFinder [10] presented a negotiated congestion algorithm that balances the competing goals of eliminating congestion and minimizing delay of critical paths for global routing in FPGAs. The cost of using a given edge e in a route is defined as:

$$cost_e = (b_e + h_e) \cdot p_e \quad (4)$$

where b_e is the base cost of using e, h_e is related to the history of congestion on e during previous iterations, and p_e is related to the number of other nets presently passing e. In the beginning of routing, the algorithm routes all nets with their shortest-path topologies regardless of the capacity constraint. During subsequent iterations, h_e is updated in the following way:

$$h_e^{i+1} = \begin{cases} h_e^i + k_1 & \text{if } e \text{ has overflow} \\ h_e^i & \text{otherwise} \end{cases} \quad (5)$$

where k_1 is a constant. Therefore the edges which tend to be congested make their edge costs increase gradually during iterations. This helps to distribute the routing demand to other less congested edges. By setting the base cost b_e to be the wirelength of an edge and trying to minimize the total edge cost, this negotiated congestion scheme can be used to eliminate congestion and minimize wirelength for global routing in ASICs.

III. Methodology

A. Overview of our global router

In this subsection, we give an overview of our global router. First of all, we apply FLUTE [11] and the edge shifting technique [6] to generate the topology of each multi-pin net. Then we decompose each multi-pin net into a set of two-pin nets. For each two-pin net, we use L-shaped pattern routing to construct the initial congestion map. (See section III.B.) Second, we propose a history based cost function (see section III.C) which is used in iterative rip-ups and reroutes. Third we iteratively identify the congested regions on the routing area and rip-up and reroute two-pin nets within the regions by monotonic routing followed by optional adaptive multi-source multi-sink maze routing. The details of this step are in sections III.D and III.E. Finally, we apply an iterative refinement process to resolve the remaining overflow (see section III.F). Section III.G details that for multi-layer designs, an additional layer assignment step is required in order to map the solution on the projected plane to the original multiple layers.

B. Initial congestion map construction

For each multi-pin net $n_i \in \boldsymbol{N}$, we generate its rectilinear Steiner minimal tree (RSMT) by FLUTE [11]. Then we decompose each multi-pin net n_i into a set of two-pin nets as shown in Fig. 3, in which a pin can be the pin belonging to n_i or a Steiner point. A vertical or horizontal wire in the RSMT is called *flat* wire, such as wire *a-e*, *b-e*, *c-f* and *e-f* in Fig. 3(b). For each two-pin net, if it forms a flat wire, we assign the demand value 1 to the edges passed by the wire on the grid graph. Otherwise, we assign the demand value 0.5 to the edges on the bounding box of the two-pin net by assuming that two probabilistic L-shaped patterns can be used by this two-pin net. Then the edge shifting technique [6] is employed for each multi-pin net to further improve the RSMT topology by moving some edges to less congested areas. Again, we decompose each multi-pin net into a set of two-pin nets after edge shifting. In order to make the congestion map more accurate, we rip-up and reroute two-pin nets using L-shaped pattern routing. Here we adopt the cost function presented in [6] to formulate the edge cost. However, our initial congestion map construction method is different from the way proposed in [6]. We use edge shifting technique to determine the routing tree topology in initial congestion map construction, while [6] uses it to dynamically adjust the topologies during rip-ups and reroutes.

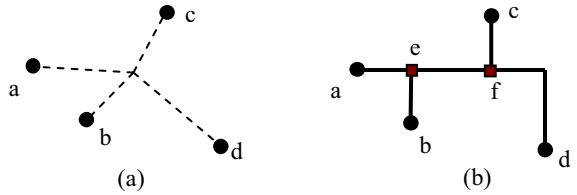

Fig. 3. (a) A multi-pin net composed of 4 pins: *a*, *b*, *c*, and *d*. (b) Decompose the net in (a) into two-pin nets by the Steiner tree topology. The black points represent pins, and the rectangles represent Steiner points.

C. History based cost function

The evaluation of edge cost in the routing graph plays an important role in the routing algorithm. As modern designs are getting larger, it is difficult to define a deterministic function that can predict the behavior of all routed nets when routing is proceeding. Therefore, we adopt the history idea of the negotiated congestion routing in our algorithm. An iterative rip-up and reroute method using the history based cost function given below is described in section III.D. For an edge e in the routing graph, the history based cost function is defined as:

$$cost_e = 1 + h_e \cdot p_e \qquad (6)$$

Basically, our cost function is a combination of wirelength and congestion. We set the first term in the function to be the value 1 which represents one unit length. In the second term, we let the historical term amplify the penalty of congestion because our main objective is to minimize the total overflow. For the historical cost h_e, it is the same as formula (5) with k_1=1. As more and more iterations complete, the edges that tend to be congested have larger h_e than those with available capacity. Therefore, h_e helps to distribute the routing demands to the less congested areas in the routing graph. The congestion penalty term p_e is defined as follows:

$$p_e = \left(\frac{d(e)+1}{s(e)}\right)^{k_2} \qquad (7)$$

where k_2 is a constant and controls the rising rate of p_e. Our idea is to drastically increase the penalty term especially when the demand exceeds the supply. For an edge which still has much available capacity, the penalty is relatively small and thus encourages routing paths to pass the edge. Fig. 4 gives the curve of formula (7) where k_2 is 5.

In addition, if an edge e is to be considered as a part of a route and is already passed by another route which belongs to the same multi-pin net as e, then the cost of e is zero rather than the definition in formula (6). Doing this encourages a two-pin route of one multi-pin net to share the same edges with other two-pin routes, and thus reduces the wirelength and routing demands.

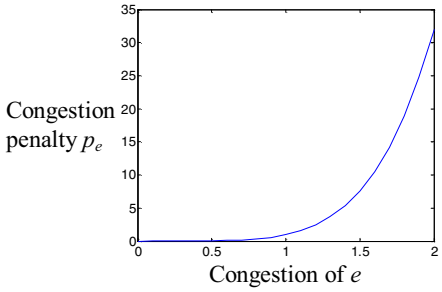

Fig. 4. The curve of the congestion penalty. When an edge has no overflow, the penalty is relatively low, but grows drastically when there is overflow.

D. Rip-up and reroute based on congested region identification

In a rip-up and reroute based method, the order of nets to be ripped up and rerouted affects the routing quality very much. Usually we prefer to route nets with smaller size of bounding box earlier because they are less flexible than those with larger size. However, this strategy does not take the current congestion on the routing area into consideration. As a result, we propose a method to identify congested regions and rip-up and reroute two-pin nets which locate in those regions.

The method starts with the routing solution generated in section III.B. First of all, we calculate the congestion for all edges. Because only edges with overflow are to be considered, we define an interval between the maximum congestion value and 1. Then we partition the interval into m sub-intervals $\{I_1, I_2, ..., I_m\}$ (we use m=10 by default). For example, when the maximum congestion is 2, the sub-intervals are {[2, 1.9), [1.9, 1.8), [1.8, 1.7),..., [1.1, 1)}. The definition that an edge e belongs to one sub-interval I_i is as follows:

$cong(e) > I_i.min\_cong$ and

$cong(e) \le I_i.max\_cong$

where $I_i.min\_cong$ represents the minimum value in I_i and $I_i.max\_cong$ represents the maximum value in I_i. By the definition, edges with overflow are assigned to their corresponding sub-intervals.

Following that, we identify the congested regions for edges in the sub-intervals. The average congestion of a region r is defined as follows:

$$avg\_cong(r) = \frac{\sum d(e_i)}{\sum s(e_i)} \quad e_i \text{ is an edge inside } r \qquad (8)$$

In the beginning, we pick the most congested edge e from I_1 as the center and expand a rectangular region r_e from it until avg_cong(r_e) is smaller than $I_1.min\_cong$. The rest of the edges in I_1 are expanded in the same way in decreasing order of congestion. Then the two-pin nets that locate within all regions expanded from edges in I_1 are ripped up and rerouted. We first use historical monotonic routing in which the edge cost is formulated by the history based cost function as described in section III.C. If there still exist two-pin nets that fail to find overflow-free paths, then detour is allowed by applying historical adaptive multi-source multi-sink maze routing (see section III.E).

Similarly, the remaining sub-intervals I_2, I_3,.., I_m are processed one by one as the same way described above. We apply this algorithm for several iterations to obtain a convergent solution. Note that after multi-pin nets are rerouted, the topology of each of them may have slight difference. As a result, we readjust the two-pin nets for each multi-pin net whenever an iteration completes. This increases the flexibility of our algorithm because the topologies of multi-pin nets are dynamically modified according to the current best routing solution.

E. Adaptive multi-source multi-sink maze routing

A general maze routing algorithm must start and end in the original two pins of a two-pin net. Take Fig. 5(a) as an example. Suppose we are rerouting two-pin net (a, b) and the shaded region between a and b represents congested area. The path found by general maze routing has to detour to avoid congested area. Pan *et al*. proposed multi-source multi-sink maze routing [7] to improve original maze routing. After ripping up the path of the two-pin net, the multi-pin net can be broken into two subtrees T_1 and T_2 as illustrated in Fig. 5(a). [7] treats all grid points on T_1 as sources and all grid points on T_2 as sinks. Therefore it can find a path $p_{(x, y)}$ that connects the multi-pin net with a shorter wirelength. However, a problem with this method is that when the number of edges utilized by a multi-pin net is very large, it consumes much time to identify all grid points on the subtrees. Different from [7], our adaptive multi-source multi-sink maze routing only considers pins and Steiner points on the multi-pin net when identifying sources and sinks. Since the number of pins and Steiner points is usually smaller than the number of grid points on the subtrees, our method is more efficient. Besides, we can show that under our cost formulation as defined in section III.C, the minimum cost found by adaptive multi-source multi-sink maze routing is the same as that found by [7].

Theorem 1 *The solution obtained by adaptive multi-source multi-sink maze routing has the same cost as the solution found by multi-source multi-sink maze routing [7].*

Proof: Due to page limitation, the proof is briefly explained. Take Fig. 5 as an example. Suppose we are rerouting the two-pin net (a, b). For simplicity we consider the case where there is only one minimum-cost solution $p_{(x, y)}$ that reconnects T_1 and T_2 and is found by multi-source multi-sink maze routing [7] as shown in Fig. 5(a).

Because x and y are neither pins nor Steiner points, adaptive multi-source multi-sink maze routing does not treat them as sources or sinks. Therefore, the searching would start form pins on T_1 and Steiner point s, and end at pins on T_2 and Steiner point t (see Fig. 5(b)). Assume the minimum-cost solution to reconnect T_1 and T_2 found by our method is $p_{(s, t)}$ which is composed of $p_{(s, x)}$, $p_{(x, y)}$, and $p_{(y, t)}$ as shown in Fig. 5(b) (note that s and t could be replaced by any other pin on T_1 and T_2, respectively). According to the cost definition in section III.C, if an edge is a part of the multi-pin net, its cost is zero. This implies that the costs of $p_{(s, x)}$ and $p_{(y, t)}$ are zero because they are parts of the multi-pin net. Therefore the total cost of path $p_{(x, y)}$ in Fig. 5(a) is equal to the total cost of path $p_{(s, t)}$ in Fig. 5(b).

For the case where more than two minimum-cost solutions exist for multi-source multi-sink maze routing (i.e., paths other than $p_{(x, y)}$ can also reconnect the subtrees with the minimum cost), adaptive multi-source multi-sink maze routing can still find a solution with the same minimum cost. □

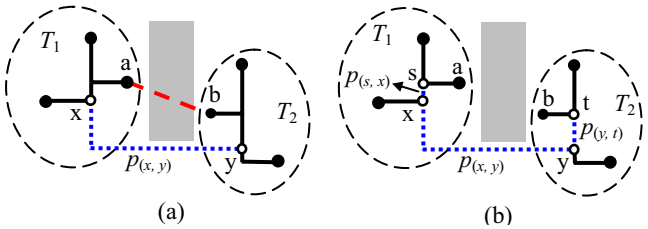

Fig. 5. (a) The path connecting a and b is ripped up and a new path $p_{(x, y)}$ connecting x and y is found. (b) The path $p_{(s, t)}$ is composed of $p_{(s, x)}$, $p_{(x, y)}$, and $p_{(y, t)}$, and it is represented as dotted lines.

F. Refinement for congested nets

During iterative rip-ups and reroutes as described in section III.D, the number of overflow decreases gradually. As the iteration goes on, the costs of edges in congested area get dominated by the history term of the cost function. An overflow-free edge may have no opportunity to be chosen when routing a path due to its high history term accumulated in previous iterations. Therefore minimizing total edge cost does not work well for minimizing total overflow in this circumstance.

We propose a refinement method to minimize total overflow when iterative historical routing gets stuck. In this method, the cost of a routing path is defined as how much overflow the path induces in all passed edges. When routing a path, if passing edge e induces overflow, e is said to be an overflow edge with $cost_e = 1$; otherwise $cost_e = 0$. In other words, the goal of rerouting a two-pin net here is trying to find a path with minimum number of overflow edges. It is apparent that if there is a path without any overflow edge, it must be picked by this strategy. For each congested two-pin net, we rip-up and reroute it with monotonic routing first and adaptive multi-source multi-sink maze routing if necessary.

G. Global routing for multi-layer designs

Traditionally, global routing is assumed to be a 2-dimensional problem in which a design is considered as a plane containing both horizontal and vertical routing tracks. However in real industrial cases, designs are usually composed of multiple metal layers and each layer prefers one routing direction. For multi-layer designs, the objective of global routing is to find a routing solution without overflow and minimize both wirelength between tiles and via usage

between layers.

We extend our algorithm to perform global routing for multi-layer designs. First of all, we project the routing resource in each layer to a common plane. Then our algorithm can be applied to solve the 2-dimensional problem. In order to deal with the objective of minimizing via usage, formula (6) is modified as follows:

$$cost_e = 1 + h_e \cdot p_e + vc_e$$

$$vc_e = \begin{cases} 1 & \text{if passing } e \text{ makes a bend,} \\ 0 & \text{otherwise.} \end{cases} \quad (9)$$

where vc_e represents whether passing the edge makes a bend.

At last, we adopt a layer assignment method [12] to assign wires to the given layers and insert vias to connect different layers. Due to limited space, we briefly explain the idea of the layer assignment method. The method routes nets one by one in the order related to the estimated routability. For each net, it uses a dynamic programming based algorithm to assign net segments to proper layers with minimum via count under the condition where no additional overflow is induced.

In our experiment, the 2-layer designs have preferred directions which allow horizontal wires in one layer and vertical wires in the other layer. We can transform our solution on the projected plane to the solution in the 2-layer designs by directly inserting a via for each bend on the routing path. Therefore we apply the layer assignment method only for the 6-layer designs.

IV. Experimental Results

We implement NTHU-Route in C++ language and execute it on the Linux operating system with an AMD 2.2GHz CPU and 8GB memory. Two benchmark suits are used in our experiment. We first compare NTHU-Route with two state-of-the-art works using the ISPD98 benchmark suite [13]. Then the ISPD07 benchmark suit [14] is used to show our performance on designs with larger size.

A. Results on ISPD98 benchmarks

Table 1 shows the statistics of ISPD98 benchmarks and the comparisons with BoxRouter [8] and FastRoute 2.0 [7]. We compare the results in terms of overflow (OF), wirelength (WL) and runtime. The results of BoxRouter and FastRoute 2.0 are quoted from [8] and [7], respectively. BoxRouter is performed on a 2.8 GHz Pentium 4 Linux machine and FastRoute 2.0 is on a Linux workstation with Intel Pentium 4 3.0 GHz CPU and 2GB memory.

For all the benchmarks, our global router obtains the solutions without any overflow, while BoxRouter and FastRoute leave four and three cases with overflow, respectively. We can observe that although FastRoute is better in reducing overflow than BoxRouter, its wirelength is worse in most of the benchmarks. However, our global router outperforms them in both wirelength and overflow. For the cases without overflow, we achieve 1.93% and 2.59% wirelength reduction rates over BoxRouter and FastRoute, respectively. The experiment shows that we achieve a good balance between solution quality and runtime, and outperform the two routers for all benchmarks.

Benchmark	Grids	# nets
adaptec1	324x324	219794
adaptec2	424x424	260159
adaptec3	774x779	466295
adaptec4	774x779	515304
adaptec5	465x468	867441
newblue1	399x399	331663
newblue2	557x463	463213
newblue3	973x1256	551667

Table 2. The statistics of ISPD07 benchmark suite.

B. Results on ISPD07 benchmarks

Table 2 shows the statistics of ISPD07 benchmarks in which the size of routing grids and the scale of the number of nets are much larger than ISPD98 benchmarks. The ISPD07 benchmarks provide multi-layer designs for 2-layer and 6-layer versions. For each benchmark, we report the results of three best global routers in the ISPD07 routing contest (i.e., FGR, MaizeRouter, and BoxRouter) and compare them with our results in Table 3 by total overflow (Total OF), maximum overflow (Max OF), and total cost (Total cost). The total cost is composed of the total length of wire segments used on the projected plane plus three times the number of vias. Our router generates overflow-free solutions in 6 of 8 benchmarks in both 2-layer and 6-layer versions. For the two remaining and difficult benchmarks newblue1 and newblue3, we achieve the least total overflows among all routers. Table 3 also shows the runtime of our router. The runtimes of the other routers are not reported because they are not available from the contest. Basically we use the same parameters in our router for all benchmarks except the number of rip-up and reroute iterations. For the difficult cases adaptec5, newblue1 and newblue3, we apply more iterations to improve the solution quality with the relative increase in runtime.

Benchmark	Grids	# nets	BoxRouter			FastRoute 2.0			NTHU-Route			WL reduction over BoxRouter	WL reduction over FastRoute 2.0
			OF	WL	Runtime(s)	OF	WL	Runtime(s)	OF	WL	Runtime(s)		
ibm01	64x64	11507	102	65588	8.3	31	68489	0.72	0	63321	4.17	-	-
ibm02	80x64	18429	33	178759	34.1	0	178868	0.93	0	170531	7.44	-	4.66%
ibm03	80x64	21621	0	151299	16.9	0	150393	0.60	0	146551	5.86	3.14%	2.55%
ibm04	96x64	26163	309	173289	23.9	64	175037	1.88	0	168262	13.61	-	-
ibm06	128x64	33354	0	282325	33.0	0	284935	1.36	0	278617	12.75	1.31%	2.22%
ibm07	192x64	44394	53	378876	50.9	0	375185	1.60	0	366288	15.89	-	2.37%
ibm08	192x64	47944	0	415025	93.2	0	411703	2.36	0	405169	13.17	2.37%	1.59%
ibm09	256x64	50393	0	418615	63.9	3	424949	1.92	0	415464	11.59	0.75%	2.23%
ibm10	256x64	64227	0	593186	95.1	0	595622	2.79	0	580793	33.72	2.09%	2.49%
Average												1.93%	2.59%

Table 1. The comparison of NTHU-Route to BoxRouter and FastRoute on the ISPD98 benchmark suite. ibm05 is not included in our experiment because it is a trivial case. The WL reduction is marked by "-" when the compared target cannot resolve overflow.

	Benchmark	FGR			MaizeRouter			BoxRouter			NTHU-Route			
		Total OF	Max OF	Total cost (e5)	Total OF	Max OF	Total cost (e5)	Total OF	Max OF	Total cost (e5)	Total OF	Max OF	Total cost (e5)	Runtime(s)
2-layer version	adaptec1	0	0	55.8	0	0	62.26	0	0	58.84	0	0	57.11	5579.98
	adaptec2	0	0	53.69	0	0	57.23	0	0	55.69	0	0	54.46	977.5
	adaptec3	0	0	133.34	0	0	137.75	0	0	140.87	0	0	137.16	3802.87
	adaptec4	0	0	126.05	0	0	128.45	0	0	128.75	0	0	128.66	522.29
	adaptec5	0	0	155.82	2	2	176.69	0	0	164.32	0	0	160.3	15990.29
	newblue1	1218	10	47.51	1348	16	50.93	400	2	51.13	352	4	47.78	2251.45
	newblue2	0	0	77.67	0	0	79.64	0	0	79.78	0	0	79.22	210.21
	newblue3	36970	1090	108.18	32588	1236	114.63	38976	1088	111.64	31800	608	111	21380.57
6-layer version	adaptec1	60	2	90.92	0	0	99.61	0	0	104.05	0	0	90.56	5613.41
	adaptec2	50	2	92.19	0	0	98.12	0	0	102.97	0	0	92.17	1010.35
	adaptec3	0	0	203.44	0	0	214.08	0	0	235.87	0	0	205.04	3892.72
	adaptec4	0	0	186.31	0	0	194.38	0	0	211.95	0	0	188.43	603.54
	adaptec5	2480	2	264.58	2	2	305.32	0	0	298.08	0	0	265.03	16104.34
	newblue1	2668	4	92.89	1348	16	101.74	400	2	101.83	352	2	90.91	2279.57
	newblue2	0	0	136.08	0	0	139.66	0	0	155.07	0	0	136.01	256.62
	newblue3	53648	636	168.42	32840	1058	184.4	38976	1088	195.5	31800	204	168.4	21464.88

Table 3. The comparison of NTHU-Route with three global routers whose results are quoted from the ISPD07 global routing contest. The overflow in the ISPD07 global routing contest is two times the overflow defined in formula (1).

Compared with MaizeRouter and BoxRouter, the total cost of our results is smaller in most of the benchmarks which have no overflow. On average, we achieve 2.78% and 1.83% less total costs over MaizeRouter and BoxRouter, respectively on 2-layer benchmarks; 5.01% and 11.83% respectively on 6-layer benchmarks. FGR performs slightly better in terms of total cost on 2-layer benchmarks. However, the solution quality of FGR is much worse than other routers on 6-layer designs in which it only solves 3 of 8 benchmarks without overflow.

C. Remarks

Recently, FGR and BoxRouter both have been improved and obtain better results in [15] and [16], respectively. In addition, a new global router, Archer, has also been proposed in [17]. All these routers can solve ISPD98 benchmarks without overflow. Compared with them on ISPD07 benchmarks, our router still achieves the least total overflow among all routers. As for the wirelength on ISPD98 benchmarks and the total cost on ISPD07 benchmarks, our solution quality is comparable to their results.

V. Conclusion

In this paper, we present a new global router, NTHU-Route, for modern designs. We provide a history based cost function to perform iterative rip-ups and reroutes. A congested region identification method is proposed to specify the order for nets to be ripped up and rerouted. We also provide a refinement method to further reduce overflow when iterative history based rip-ups and reroutes reach its bottleneck. For the multi-layer designs, our router finds the solution on the projected plane followed by a layer assignment step. Compared with two state-of-the-art works on ISPD98 benchmarks, NTHU-Route outperforms them in both overflow and wirelength. For the much larger designs on ISPD07 benchmarks, NTHU-Route obtains the solution with least overflow when comparing with the best results reported in the ISPD07 global routing contest.

References

[1] R. Carden, J. Li and C. K. Cheng, "A global router with a theoretical bound on the optimal solution," in *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, pp. 208-216, 1996.

[2] C. Albrecht, "Global routing by new approximation algorithms for multicommodity flow," in *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, pp. 622-632, 2001.

[3] R. T. Hadsell and P. H. Madden, "Improved global routing through congestion estimation," in *Proc. of Design Automation Conference*, pp. 28–31, 2003.

[4] R. Kastner, E. Bozorgzadeh, and M. Sarrafzadeh, "Pattern routing: use and theory for increasing predictability and avoiding coupling," in *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, pp. 777-790, 2002.

[5] Z. Cao, T. Jing, J. Xiong, Y. Hu, L. He, and X. Hong, "Dprouter: A fast and accurate dynamic-pattern-based global routing algorithm," in *Proc. Asia and South Pacific Design Automation Conf.*, pp. 256-261 2007.

[6] M. Pan and C. Chu, "Fastroute: A step to integrate global routing into placement," in *Proc. Int. Conf. on Computer-Aided Design*, pp. 464–471, 2006.

[7] M. Pan and C. Chu, "Fastroute 2.0: a high-quality and efficient global router," in *Proc. Asia and South Pacific Design Automation Conf.,* pp. 250-255, 2007.

[8] M. Cho and D. Z. Pan, "Boxrouter: a new global router based on box expansion and progressive ILP," in *Proc. of Design Automation Conference*, pp. 373–378, 2006.

[9] D. Jariwala and J. Lillis, "Trunk decomposition based global routing optimization," in *Proc. Int. Conf. on Computer-Aided Design*, pp. 472-479, 2006.

[10] L. McMurchie and C. Ebeling, "Pathfinder: a negotiation-based performance-driven router for FPGAs," in *Proc. of ACM Int. Symp. on FPGAs*, pp. 111–117, 1995.

[11] C. Chu and Y. Wong, "Fast and accurate rectilinear Steiner minimal tree algorithm for VLSI design," in *Proc of Int. Symp. on Physical Design*, pp. 28–35, 2005.

[12] T.-H. Lee, *Congestion-constrained Layer Assignment for Via Minimization in Global Routing*, Master Thesis, Department of Computer Science, National Tsing Hua University, 2007.

[13] http://www.ece.ucsb.edu/~kastner/labyrinth

[14] http://www.sigda.org/ispd2007/contest.html

[15] J. A. Roy and I. L. Markov, "High-performance routing at the nanometer Scale," in *Proc. Int. Conf. on Computer-Aided Design*, pp. 496–502, 2007.

[16] M. Cho, K. Lu, K. Yuan, and D. Z. Pan, "BoxRouter 2.0: architecture and implementation of a hybrid and robust global router," in *Proc. Int. Conf. on Computer-Aided Design*, pp. 503–508, 2007.

[17] M. M. Ozdal and M. D. F. Wong, "Archer: a history-driven global routing algorithm," in *Proc. Int. Conf. on Computer-Aided Design*, pp. 488–495, 2007.