NTHU-Route 2.0: A Fast and Stable Global Router

Yen-Jung Chang
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan 30013
Email: jalamorm@gmail.com

Yu-Ting Lee
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan 30013
Email: leetroy@gmail.com

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

***Abstract*—We present in this paper a fast and stable global router called NTHU-Route 2.0 that improves the solution quality and runtime of a state-of-the-art router, NTHU-Route, by the following enhancements: (1) a new history based cost function, (2) new ordering methods for congested region identification and rip-up and reroute, and (3) two implementation techniques. The experimental results show that NTHU-Router 2.0 solves all ISPD98 benchmarks with very good quality. Moreover, it routes 7 of 8 ISPD07 benchmarks without any overflow. In particular, for one of the ISPD07 benchmarks which are thought to be difficult cases previously, NTHU-Route 2.0 can completely eliminate its total overflow. NTHU-Route 2.0 also successfully solves 12 of 16 ISPD08 benchmarks without causing any overflow.**

I. Introduction

As feature size in VLSI design continues to shrink, the wire resistance and the interconnect delay keeps increasing. Furthermore, the interconnect delay has replaced transistor delay as the main determinant of chip performance and makes every stage of the design cycle target for minimizing wirelength to reduce circuit delay. Therefore the routing problem, the last stage of the design cycle, becomes more critical in modern VLSI design.

Typically, the routing problem can be divided into global routing and detailed routing due to the problem complexity. During global routing, a design is often modeled as a two-dimensional (2D) coarse-grain grid graph and then a global route for each net is determined from the graph. After that, a layer assignment process is applied for a multi-layer design, and finally detailed routing is solved by taking the solution from global routing as the input. The quality of global routing influences the timing, power and density of a chip, and thus global routing is a very crucial stage in the design cycle.

One of the most commonly used techniques for global routing is rip-up and reroute. This technique starts by routing each net without considering congestion. After routing all nets, a congestion map can be obtained and nets passing through congested regions will be ripped up and rerouted for finding alternative routes with less costs. The process is a sequential one because only one net will be rerouted at a time. Therefore the order of rip-up and reroute influences the solution quality significantly. Chi Dispersion [1] and Labyrinth [2] are global routers which utilize this routing technique earlier. After them, two efficient global routers, DpRouter [3] and FastRoute [4][5], and an ILP-based global router, BoxRouter [6], are proposed and all of them can achieve high-quality solutions.

Recently, the International Symposium on Physical Design (ISPD) announced two global routing contests in 2007 [7] and 2008 [8], respectively for boosting the research and development of new global routing techniques. Contributed by the spirited competition, BoxRouter [9], Archer [10], FGR [11], MaizeRouter [12], and NTHU-Route [13] were developed, and most of them applied the negotiation-based routing technique which was introduced in PathFinder [14]. In these worldwide contests, the adopted global routing techniques can be categorized into two classes: full 3-dimensional (3D) routing and 2D routing followed by layer assignment. In full 3D routing, FGR solves the routing problem directly by using full 3D maze routing. Due to the complexity of modern designs, full 3D routing normally takes longer time than the other method; therefore, FGR also adopted the other routing method. The method of 2D routing followed by layer assignment [9][10][11][12][13] projects the routing instance onto a plane, routes the new 2D problem instance, and then maps the solution from the projected plane to the original multiple layers by a layer assignment method.

In this paper, we present a fast and stable global router called NTHU-Route 2.0 that improves the solution quality and runtime of NTHU-Route [13] by the following enhancements:

1) a new history based cost function,
2) new ordering methods for congested region identification and rip-up and reroute, and
3) two implementation techniques.

The experimental results show that NTHU-Route 2.0 solves all ISPD98 benchmarks [15] with very good quality. Moreover, it successfully routes 7 of 8 ISPD07 benchmarks[1] [7] without any overflow. In particular, for one of the ISPD07 benchmarks which are thought to be difficult cases previously, NTHU-Route 2.0 can completely eliminate its total overflow. NTHU-Route 2.0 is also capable of solving 12 of 16 ISPD08 benchmarks [8] without causing any overflow.

The rest of the paper is organized as follows. In section II we give the problem formulation of global routing and a review of NTHU-Route. In section III we describe the enhancements in detail. In section IV we provide the experimental results and we conclude the paper in section V.

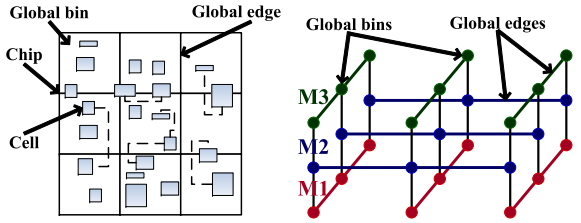
II. Preliminaries

A. Problem Formulation

The global routing problem can be modeled as follows. There is a grid graph $G(V, E)$. As illustrated by the three-layer (denoted by M1, M2, and M3) example shown in Fig. 1, each vertex in V corresponds to a global bin and each edge in E corresponds to a global edge which is a boundary between two adjacent global bins. There is also a set of nets, where each net is composed of a set of pins, and each pin corresponds to a vertex. The routing problem for a net is to find a route connecting all the pins of the net using the edges of G.

The capacity c_g of an edge g represents the number of available routing tracks. The demand d_g represents the number of nets that pass through edge g. The overflow of an edge is defined to be the amount of demand that exceeds the capacity of the edge. We also define the congestion of edge g as the ratio of the demand to the capacity, i.e., $\frac{d_g}{c_g}$. The major objective of the global routing problem is to minimize the sum of the overflows among all edges (total overflow), while the second objective is to minimize the wirelength. In multi-layer designs, wirelength calculation also involves vias.

[1]The remaining case has been proved unroutable (i.e., no overflow-free solution exists) before.



(a) (b)

Fig. 1. (a) Decomposition of global bins (b) Grid graph for routing

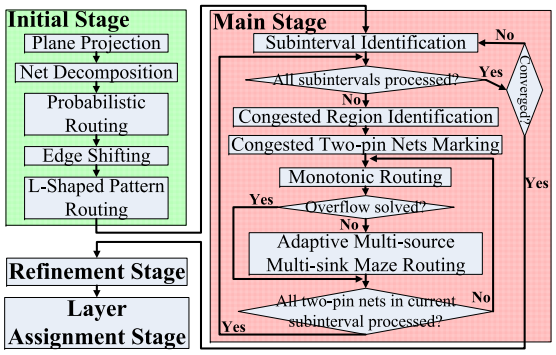


Fig. 2. Flow of NTHU-Route

B. Previous Work: NTHU-Route

In this section, we give a review of NTHU-Route [13]. The flow of NTHU-Route for multi-layer designs is illustrated in Fig. 2. There are 4 stages in NTHU-Route: initial stage, main stage, refinement stage, and layer assignment stage.

In the initial stage, an initial global routing solution is generated as follows. NTHU-Route first projects a multi-layer design on a plane, and then uses FLUTE [16] to decompose each multi-pin net into a set of two-pin nets. After that, NTHU-Route routes every two-pin net with two probabilistic L-shaped patterns, i.e., it adds the half demand (0.5) to each edge along a probabilistic L-shape route or the full demand (1) to each edge along a straight route. Next, NTHU-Route modifies the topology of every multi-pin net by the edge shifting technique [4] and then reroutes every two-pin net with the L-shaped pattern with least cost (note that in these two steps, NTHU-Route applies the cost function presented in [4] to formulate the edge cost).

In the main stage, NTHU-Route improves the initial solution by iteratively ripping up and rerouting every congested two-pin net based on congested region identification. A two-pin net is congested if it passes one or more overflowed edges along its path. The concept of congested region identification works as follows: First of all, NTHU-Route calculates the congestion of every edge. Because only edges with overflow are to be considered, it defines an interval between the maximum congestion value and 1. Then it partitions the interval into m sub-intervals $\{I_1, I_2, ..., I_m\}$. For example, when the maximum congestion is 2 and m = 10, the sub-intervals are [2, 1.9), [1.9, 1.8), ..., [1.1, 1). After that, it sequentially picks every edge g whose congestion value is within I_1 (starting from the most congested one) as the center and expands a region r_g from the edge until the average congestion of this region is smaller than the lower bound of I_1. Then it marks every congested two-pin net which has both pins located in r_g, and begins to rip up and reroute the marked two-pin nets one at a time by the non-decreasing order of bounding box size (note that a marked two-pin net needs to be rerouted only when it remains congested after the nets prior to it have been processed). Every ripped-up two-pin net is rerouted by monotonic routing [4] first; if the monotonic routing method fails to find an overflow-free path, then an adaptive multi-source multi-sink maze routing method is applied. After I_1 is processed, NTHU-Route continues to apply the same congested region identification strategy to each of the remaining subintervals and rip up and reroute marked two-pin nets. The rip-up and reroute process is kept repeating until the total overflow is no more than a pre-defined threshold or a pre-defined number of iterations is reached.

The history based cost function used in the main stage for calculating the cost of an edge g is defined as follows:

$$cost_g = b_g + h_g \times p_g + vc_g \tag{1}$$

where b_g is the base cost of using edge g and is set to 1 (which means one unit of wirelength), $h_g \times p_g$ is the congestion cost of edge g, and vc_g is the via cost when using edge g. The historical term h_g is updated in the following way during subsequent iterations:

$$h_g^{i+1} = \begin{cases} h_g^i + 1 & \text{if } g \text{ has overflow} \\ h_g^i & \text{otherwise.} \end{cases} \tag{2}$$

where i is the iteration count and $h_g^1 = 1$. The penalty term p_g is defined as follows:

$$p_g = \left(\frac{d_g + 1}{c_g} \right)^{k_1} \tag{3}$$

where k_1 is a user-defined parameter and is set to 5. The vc_g is defined in the following way:

$$vc_g = \begin{cases} 1 & \text{if passing } g \text{ makes a bend} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

The refinement stage mainly focuses on finding an overflow-free path for every congested two-pin net and is modified from the main stage as follows. First, the congested region identification is not applied and every congested two-pin net is ripped up and rerouted in the non-increasing order of the number of overflowed edges a net passes through. Second, the cost of an edge is set to 1 if the edge has overflow; otherwise it is set to 0.

For multi-layer designs, the method in [17] is applied in the layer assignment stage to map the solution from the projected plane to the original multiple layers.

III. Enhancements for NTHU-Route

In this section, we propose three enhancements for NTHU-Route to get NTHU-Route 2.0, and the details are described in the following three sub-sections.

A. New History Based Cost Function

The first enhancement is a new history based cost function to be used in the main stage. It is divided into 3 sub-cost functions: base cost function, congestion cost function, and via cost function.

1) Adaptive base cost function: We first discuss the base cost function. Take Fig. 3 for example, where the darkest area is a region which contains overflowed edges, and the second darkest area is a region which contains edges that are nearly overflowed. For the given two-pin net, A is the path found by a global router without considering the impact of wirelength, and it occupies more routing resources than the other two paths B and C, and may cause other two-pin nets to have overflow in subsequent steps. B is the path found by a global router which takes the impact of wirelength too seriously, and it may pass through a highly congested area because of shortest wirelength. C is a better solution than the other two paths for now, because it occupies less routing resources than A and passes through a less congested region than B.

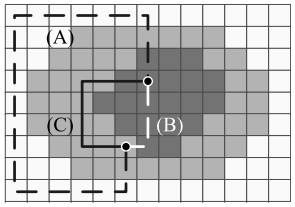


Fig. 3. Different paths found by different base costs.

Actually, all of the aforementioned paths are needed at different time during the whole routing process. In the beginning of a routing process (i.e., without any net routed), we tend to find paths like B because we do not want a two-pin net to occupy too many routing resources and cause other two-pin nets to have overflow in subsequent steps. Then we find paths like C in order to reduce total overflow. Finally, we have to make a compromise between wirelength and overflow again, and begin to find paths like A to further reduce total overflow. Since paths like A do not consider the impact of wirelength, they can pass through regions which are located in farther area from pins but with lower congestion.

Therefore, we propose an adaptive base cost function and it is defined for each edge g as follows:

$$b_g = 1 - e^{-\beta e^{-\gamma i}} \tag{5}$$

where β and γ are user-defined parameters, and i is the current iteration count. In NTHU-Route 2.0, β is set to 5 and γ is set to 0.1. As a result, b_g will be bounded between 1 and 0.

This base cost function is based on a Gompertz curve [18][19], which has the slowest growth rate at the start and the end of a time period. The curve of the base cost function we use in NTHU-Route 2.0 is shown in Fig. 4(a). As the iteration count increases, the base cost will be reduced and hence encourages NTHU-Route 2.0 to obtain paths with less overflow rather than paths with shorter wirelength.

2) Congestion cost function with overflow prediction: We next discuss the congestion cost function. From our empirical observation, the congestion cost function applied in NTHU-Route dose not perform well when the maximum overflow among all edges is close to one. Therefore, we propose a congestion cost function with overflow prediction to overcome this problem. The congestion cost function, like the original one, is still a compound of the historical term and penalty term, but we replace the penalty term with

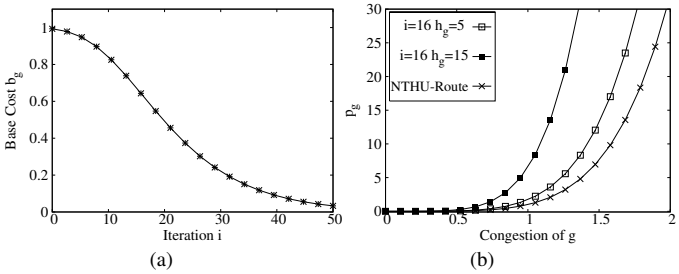
$$p_g = \left(\frac{d_g + 1}{c_g} \times f(h_g, i) \right)^{k_1} \tag{6}$$

where i is current iteration count, h_g is the historical term used in NTHU-Route, and $f(h_g, i)$ amplifies congestion.

Before describing $f(h_g, i)$, we provide a simplified version of $f(h_g, i)$, which is defined as follows:

$$f_{simple}(h_g, i) = \left(\frac{i \times k_2}{i \times k_2 - (h_g - 1)} \right) \tag{7}$$

where k_2 is a user-defined parameter and controls the maximum value of $f_{simple}(h_g, i)$. Since h_g in the first iteration is 1, we get the overflow count of edge g in the past by h_g minus 1. As the iteration count reaches i, the possible highest value of h_g among all edges g's is i, so we set k_2 to 1.5 to make the maximum value of $f_{simple}(h_g, i)$ close to 3. From our empirical observation, the value 3 achieves a good balance between the reduction rate of the total overflow and the increase rate of wirelength. For example, assume that NTHU-Route 2.0 is running the ith iteration of the main stage. From



(a) (b)

Fig. 4. (a) The curve of adaptive base cost function b_g. (b) Curves of the penalty terms utilized in NTHU-Route and NTHU-Route 2.0.

$f_{simple}(h_g, i)$, we can get $\left(\frac{i \times k_2}{i \times k_2 - (i-1)}\right) \leq \left(\frac{i \times k_2}{i \times k_2 - i}\right) = \frac{k_2}{k_2 - 1}$. Since the historical term of an edge g is always no more than the current iteration count, the value of $\frac{k_2}{k_2 - 1}$ is the upper bound of $f_{simple}(h_g, i)$.

As i is small in the first several iterations of the main stage, the penalty terms of overflowed edges are amplified by values that are close to the maximum value of $f_{simple}(h_g, i)$, and hence those edges will push ripped-up two-pin nets away from them eagerly. As a result, the ripped-up two-pin nets will consume too many routing resources for finding paths which avoid using these overflowed edges and may cause other two-pin nets to have overflow in subsequent steps. To overcome this problem, we add an adjustment term $adj(i)$ to $f_{simple}(h_g, i)$. We now have $f(h_g, i)$ defined as follows:

$$f(h_g, i) = \left(\frac{i \times (k_2 + adj(i))}{i \times (k_2 + adj(i)) - (h_g - 1)} \right) \tag{8}$$

where $adj(i)$ is defined below:

$$adj(i) = k_3 \times (1 - e^{-\beta e^{-\gamma i}}) \tag{9}$$

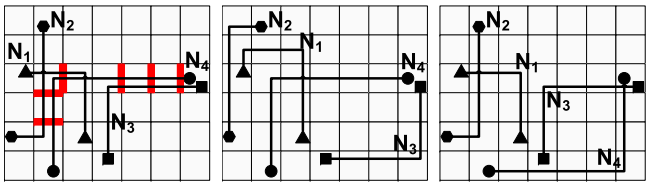
The k_3 of $adj(i)$ is a user-defined parameter and is set to 3 by default. The values of β and γ of $adj(i)$ are the same as those in (5). With this adjustment term, $f(h_g, i)$ will not over amplify the penalty term in the first several iterations.

Now NTHU-Route 2.0 not only drastically increases the penalty term of an edge when the edge has overflow, but also drastically increases the penalty term before it has overflow if the edge has high frequency to have overflow in the past iterations. This fact can be seem from Fig. 4(b) which shows the curves of the penalty terms of NTHU-Route (crossed curve) and NTHU-Route 2.0 (squared curves). Here the two squared curves are derived when $i = 16, h_g = 5$ and $i = 16, h_g = 15$, respectively.

3) Via cost function for multi-layer designs: The via cost function defined in NTHU-Route may not reflect the real situation of a design with more than 2 layers, because a bend of a wire on a plane may become one or more vias after mapping it back to a multi-layer design. As a result, we redefine the via cost function as follows:

$$vc_g = \begin{cases} v_g \times c_g \times b_g & \text{if passing } g \text{ makes a bend,} \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

where v_g is the expected amount of vias for a bend, c_g is the cost of a via (typically measured by units of wirelength), and b_g is the same as the one defined in (5). Taking a six-layer design with preferred directions for example, a bend corresponds to $\lceil \frac{1 \times 5 + 3 \times 3 + 5 \times 1}{9} \rceil = 3$ expected vias. Furthermore, if a via equals three units of wirelength, then vc_g will be set to $3 \times 3 \times b_g$.



(a) (b) (c)

Fig. 5. (a) A routing solution obtained from a previous iteration. (b) A routing solution obtained by NTHU-Route. (c) A routing solution obtained by NTHU-Route with the new ordering method for rip-up and reroute.

B. New Ordering Methods for Rip-up and Reroute and Congested Region Identification

The next enhancement is new ordering methods for rip-up and reroute, and congested region identification. These ordering methods will be used in the main stage.

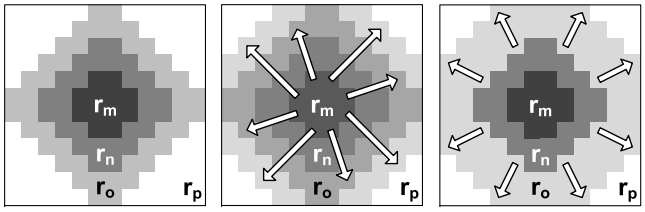
1) New ordering method for rip-up and reroute: Every time when a set of congested two-pin nets are identified and marked, NTHU-Route begins to rip up a congested two-pin net with smallest bounding box in the set, and then reroutes it immediately. After that, it processes the next two-pin net in the set which still remains congested. It will keep doing this until all congested two-pin nets are processed. Take Fig. 5 for example, where Fig. 5(a) shows a routing solution obtained from a previous iteration. The red-bolded edges are overflowed edges. All two-pin nets in this figure are marked for rerouting. NTHU-Route first rips up and reroutes N_1, and then N_2 and N_3. N_4 will not be ripped up for rerouting because it is no longer congested after N_1, N_2, and N_3 are rerouted (see Fig. 5(b)). In Fig. 5(b), we can also see that N_1 has to detour because N_4 occupies the routing resources inside the bounding box of N_1.

If we reverse the ordering of the rip-up and reroute process (i.e., starting from the net with largest bounding box), we can make N_4 avoid passing through overflowed edges without trouble because it has more routing choices than N_1, N_2, and N_3 (see Fig. 5(c)). Moreover, N_1, N_2, and N_3 can even leave untouched because they are no longer congested after N_4 is ripped up and rerouted. Although the solutions in Fig. 5(b) and Fig. 5(c) both have no overflow, the one in Fig. 5(c) has shorter wirelength.

As motivated by the example in Fig. 5, in NTHU-Route 2.0 we reverse the rip-up and reroute ordering of NTHU-Route.

2) New ordering method for congested region identification: From our empirical observations on the routing results obtained by NTHU-Route, congestions are often radially distributed (see Fig. 6(a)). In Fig. 6(a), the regions r_m, r_n, and r_o have overflowed edges and the region r_p has no overflowed edges, where a region with darker color is more congested. NTHU-Route rips up and reroutes the set of congested two-pin nets which are located in r_m, say $nets_m$, first. Since the routing resources located in r_n and r_o are not released yet, the nets in $nets_m$ may pass through r_n and r_o for finding paths with less cost. Hence the nets in $nets_m$ may obtain paths with longer wirelength, and these paths may increase the congestion of outer region r_n, r_o, or r_p (see Fig. 6(b)). After the nets in $nets_m$ are processed, NTHU-Route rips up and reroutes the set of congested two-pin nets located in r_n, say $nets_n$. Again, the nets in $nets_n$ may pass through r_o for finding paths which have less cost but have longer wirelength and may increase the congestion of outer region r_o or r_p.

In NTHU-Route 2.0, we reverse the ordering for congested region identification. For the example in Fig. 6(a), NTHU-Route 2.0 will rip up and reroute the set of congested two-pin nets which are located in the region r_o, says $nets_o$, first. The nets in $nets_o$ are likely rerouted to outer region(s) for avoiding the edges with higher costs in r_n and



(a) (b) (c)

Fig. 6. (a) Congestions are often radially distributed. (b) NTHU-Route first rips up and reroutes the congested two-pin nets located in r_m. (c) NTHU-Route 2.0 first rips up and reroutes the congested two-pin nets located in r_o and some routing resources in r_o are likely released before the congested two-pin nets located in r_m and r_n are ripped up and rerouted.

r_m; therefore some routing resources in r_o are likely released (see Fig. 6(c)). After then, the nets in $nets_n$ can use the released routing resources, and therefore they may reduce the usage of some routing resources in r_p and also reduce wirelength.

C. Two Techniques for Runtime Reduction

In NTHU-Route, there are two serious bottlenecks in runtime. One of them is the need to calculate the costs of edges when searching paths for congested two-pin nets. The other is to determine if a part of a multi-pin net already passes through an edge.

We first discuss the first bottleneck. Since NTHU-Route only stores the demand and capacity for each routing edge, it has to calculate the $cost_g$ of an edge g on the fly whenever $cost_g$ is needed. Even worse, our new history based cost function consumes more time to calculate than the one in NTHU-Route. As a result, pre-computing and storing edge costs are needed badly. Accordingly, in NTHU-Route 2.0, we also compute and store the sum of the base cost and the congestion cost for every single edge and update them when necessary. Doing this makes NTHU-Route 2.0 obtain each $cost_g$ faster.

Next we discuss the second bottleneck. In NTHU-Route, every edge has a lookup table to store which two-pin net passes through it. This lookup table is used frequently in NTHU-Route, and it is implemented by a balanced search tree. In NTHU-Route 2.0, we implement the lookup table by a hash table, which makes NTHU-Route 2.0 not only work more efficiently but also consume less memory than NTHU-Route.

IV. Experimental Results

In this section we present the experimental results of NTHU-Route 2.0. NTHU-Route 2.0 was implemented in C++ and executed on a Linux Workstation with an AMD Opteron 2.2Ghz CPU and 8GB memory. Three sets of benchmarks from ISPD98 [15], ISPD07 [7], and ISPD08 [8] were used in our experiments.

A. Results on ISPD98 Benchmarks

We used ISPD98 benchmarks to compare NTHU-Route 2.0 with several recently published global routers, including Archer [10], BoxRouter 2.0 [9], FastRoute 2.0 [5], FGR [11], and NTHU-Route [13]. TABLE I shows the performance of each router on ISPD98 benchmarks. We compare the results in terms of total overflow (TOF) and wirelength (WL). BoxRouter 2.0 was tuned for runtime or quality, and therefore two sets of results (represented as (r) and (q)) are reported.

For each benchmark, NTHU-Route 2.0, like the other routers except FastRoute 2.0, obtained the solution without any overflow. For the benchmarks without overflow, NTHU-Route 2.0 averagely achieved 0.92%, 4.37%, 0.36%, 2.81%, 0.05%, and 0.4% shorter wirelength than Archer, BoxRoute 2.0(r), BoxRoute 2.0(q), FastRoute 2.0, FGR, and NTHU-Route, respectively. Note that as we

TABLE I
THE RESULTS OF FIVE STATE-OF-THE-ART GLOBAL ROUTERS AND NTHU-ROUTE 2.0 ON ISPD98 BENCHMARKS. THE RUNTIMES FOR EACH ROUTER ARE GIVEN IN SECONDS. THE AVERAGE WIRELENGTH IMPROVEMENT RATE OVER FASTROUTE 2.0 DOSE NOT INCLUDE THE BENCHMARKS ON WHICH FASTROUTE 2.0 FAILED TO GENERAT OVERFLOW-FREE SOLUTIONS.

Benchmark	Archer			BoxRouter 2.0(r)			BoxRouter 2.0(q)			FastRoute 2.0			FGR			NTHU-Route			NTHU-Route 2.0		
	TOF	WL	cpu	TOF	WL	cpu	TOF	WL	cpu	TOF	WL	cpu	TOF	WL	cpu	TOF	WL	cpu	TOF	WL	cpu
ibm01	0	64389	11	0	66529	3.5	0	62659	32.8	31	68489	0.72	0	63332	10	0	63321	4.2	0	62498	2.4
ibm02	0	171805	25	0	180053	4.6	0	171110	35.9	0	178868	0.93	0	168918	13	0	170531	7.4	0	169881	3.3
ibm03	0	146770	10	0	151185	3.5	0	146634	17.6	0	150393	0.6	0	146412	5	0	146551	5.9	0	146458	2.5
ibm04	0	169977	24	0	176765	27.4	0	167275	115.9	64	175037	1.88	0	167101	29	0	168262	13.6	0	166452	5.9
ibm06	0	278841	23	0	288420	8.4	0	277913	47.4	0	284935	1.36	0	277608	18	0	278617	12.8	0	277696	5.5
ibm07	0	370143	25	0	377072	14.4	0	365790	85.9	0	375185	1.6	0	366180	20	0	366288	15.9	0	366133	6.4
ibm08	0	404530	42	0	418285	17.1	0	405634	90.1	0	411703	2.36	0	404714	18	0	405169	13.2	0	404976	5.9
ibm09	0	414223	37	0	431298	17.1	0	413862	273.1	3	424949	1.92	0	413053	20	0	415464	11.6	0	414738	5.7
ibm10	0	583805	45	0	610680	17.2	0	590141	352.4	0	595622	2.79	0	578795	92	0	580793	33.7	0	579870	12.3
avg. improv.		0.92%	5.07		4.37%	2.22		0.36%	18.34		2.81%	-3.57		0.05%	3.94		0.40%	2.3			

did not execute every router on our own machine, the runtime of each router is quoted from [10], [9], [5], [11], and [13], respectively. TABLE II shows the list of machines used by each router. Although all the other routers except NTHU-Route were not run on the same platform as NTHU-Route 2.0, the experimental results still show that NTHU-Route 2.0 was at least 5.07×, 2.22×, 18.34×, 3.94×, and 2.30× faster than Archer, BoxRouter(r), BoxRouter(q), FGR, and NTHU-Route, respectively, because NTHU-Route 2.0 was executed on a slowest platform. FastRoute 2.0 was faster than the other routers; however, it still left 3 cases with overflow.

B. Results on ISPD07 Benchmarks

The ISPD07 benchmarks provide multi-layer designs for 2-layer and 6-layer versions. We report the results of five state-of-the-art global routers, Archer, BoxRouter 2.0, FGR 1.1 [20], MaizeRouter [12], and NTHU-Route, and compare them with ours in TABLE III by total overflow (TOF) and wirelength (WL). The wirelength is composed of the total length of wire segments plus 3 times the amount of vias.

As can be seen from TABLE III, NTHU-Route 2.0 was able to generate a solution with the same or smaller total overflow for each benchmark, as compared with the other routers. On average, NTHU-Route 2.0 achieved 1.05%, 2.19%, 4.03%, and 1.42% shorter wirelength than Archer, BoxRouter 2.0, MaizeRouter, and NTHU-Route, respectively, on 2-layer benchmarks, and 19.38%, 2.51%, 6.46%, and 1.59%, respectively, on 6-layer benchmarks. Although FGR 1.1 obtained better solutions than NTHU-Route 2.0 in terms of wirelength for most benchmarks which have no overflow, NTHU-Route 2.0 achieved an overflow-free solution for newblue1 and a solution with lower overflow for newblue3.

TABLE III also shows the runtimes of FGR 1.1, NTHU-Route, and NTHU-Route 2.0. NTHU-Route 2.0 was 28.38× and 24.06× faster than FGR 1.1 on 2-layer and 6-layer benchmarks, respectively, even though NTHU-Route 2.0 was executed on a slower platform. Furthermore, NTHU-Route 2.0 was 6.79× and 6.44× faster than NTHU-Route on 2-layer and 6-layer benchmarks, respectively.

C. Results on ISPD08 Benchmarks

The ISPD08 benchmarks provide 6-layer and 8-layer designs (half of them are from the ISPD07 6-layer benchmarks). We report the results of NTHU-Route 2.0 and compare them with those of NTHU-Route in TABLE IV by total overflow (TOF) and wirelength (WL). The wirelength is the total length of wire segments plus the amount of vias. NTHU-Route left 5 benchmarks with overflow and failed to generate solutions for 3 benchmarks because of out of memory, whereas NTHU-Route 2.0 produced overflow-free solutions for 12 benchmarks. On average, NTHU-Route 2.0 achieved 0.94% shorter wirelength than NTHU-Route for the benchmarks which have no overflow. TABLE IV also shows the runtimes of both routers. NTHU-Route 2.0 was averagely 8.17× faster than NTHU-Route.

TABLE II
LIST OF MACHINES USED BY OTHER ROUTERS

Router	CPU
Archer	Intel Xeon 3.6Ghz
BoxRouter 2.0	Intel Pentium 4 2.8Ghz
FastRoute 2.0	Intel Pentium 4 3.0Ghz
FGR	AMD Opteron 2.4Ghz
NTHU-Route	AMD Opteron 2.2Ghz

It is worthy noting that an earlier version of NTHU-Route 2.0 won the first place of ISPD 2008 Global Routing Contest [8]. In the contest, our router generated the best solutions for 11 of 16 benchmarks. To save space, we do not report the contest results here.

V. CONCLUSION

In this paper, we have incorporated several enhancements into NTHU-Route to get NTHU-Route 2.0. Experimental results show that NTHU-Route 2.0 outperforms most state-of-the-art global routers in overflow, wirelength and runtime on ISPD98 and ISPD07 benchmarks. NTHU-Route 2.0 also solved 12 of 16 ISPD08 benchmark without causing overflow. A future work is to enhance our router to see if the remaining 3 ISPD08 benchmarks (excluding the unroutable newblue3) can be routed without any overflow.

VI. ACKNOWLEDGMENTS

This work was supported in part by National Science Council of Taiwan under Grant Numbers NSC-97-2220-E-007-015, NSC-97-2220-E-007-030, and NSC-97-2220-E-007-032.

REFERENCES

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

[2] R. Kastner, E. Bozorgzadeh, and M. Sarrafzadeh, "Pattern routing: use and theory for increasing predictability and avoiding coupling," *IEEE Transactions on Computer-Adided Design of Integrated Circuits and Systems*, vol. 21, no. 7, pp. 777–790, 2002.

[3] 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 *Proceedings of Asia and South Pacific Design Automation Conference*, Yokohama, Japan, 2007, pp. 256–261.

[4] M. Pan and C. Chu, "Fastroute: a step to integrate global routing into placement," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, San Jose, CA, 2006, pp. 464–471.

TABLE III

THE RESULTS OF FIVE STATE-OF-THE-ART GLOBAL ROUTERS AND NTHU-ROUTE 2.0 ON ISPD07 BENCHMARKS. THE NAMES OF BENCHMARKS WHICH END WITH ".2D" (".3D") ARE 2-LAYER (6-LAYER) BENCHMARKS. THE RESULTS OF FGR 1.1 WERE OBTAINED UNDER THE DEFAULT MODE. EACH AVERAGE WIRELENGTH IMPROVEMENT RATE IS CALCULATED ONLY FOR THE BENCHMARKS ON WHICH THE COMPARED ROUTER GENERATED OVERFLOW-FREE SOLUTIONS.

Benchmark	Archer		BoxRouter 2.0		MaizeRouter		FGR 1.1			NTHU-Route			NTHU-Route 2.0		
	TOF	WL(e5)	TOF	WL(e5)	TOF	WL(e5)	TOF	WL(e5)	cpu(min)	TOF	WL(e5)	cpu(min)	TOF	WL(e5)	cpu(min)
adaptec1.2d	0	57.66	0	58.37	0	62.26	0	54.73	203	0	57.11	93.0	0	55.91	9.7
adaptec2.2d	0	54.62	0	55.69	0	57.23	0	52.40	24	0	54.46	16.3	0	54.01	1.9
adaptec3.2d	0	135.18	0	137.96	0	137.75	0	131.50	94	0	137.16	64.4	0	134.39	10.3
adaptec4.2d	0	126.32	0	127.79	0	128.45	0	125.00	14	0	128.66	8.7	0	127.81	2.2
adaptec5.2d	0	162.71	0	162.11	2	176.89	0	153.21	616	0	160.33	266.5	0	157.03	20.5
newblue1.2d	494	48.74	400	51.13	1348	50.93	400	46.33	503	352	47.78	37.5	0	48.60	5.5
newblue2.2d	0	77.94	0	78.68	0	79.64	0	76.54	4	0	79.22	3.5	0	78.54	1.1
newblue3.2d	31928	109.59	38958	107.23	32588	114.63	32840	163.36	1549	31800	111.00	356.3	31454	110.82	120.9
avg. improv.		1.05%		2.19%		4.03%		-2.46%	23.38		1.42%	6.79			
adaptec1.3d	0	113.80	0	92.04	0	99.61	0	88.59	212	0	90.56	93.6	0	88.81	10.0
adaptec2.3d	0	112.56	0	94.28	0	98.12	0	90.08	28	0	92.17	16.8	0	90.88	2.1
adaptec3.3d	0	244.08	0	207.41	0	214.08	0	200.59	141	0	205.04	64.9	0	200.84	10.9
adaptec4.3d	0	221.57	0	186.42	0	194.38	0	182.99	36	0	188.43	10.1	0	185.99	2.6
adaptec5.3d	0	334.09	0	270.41	2	305.32	0	261.26	655	0	265.03	268.4	0	260.18	23.0
newblue1.3d	682	116.08	394	92.94	1348	101.74	400	90.56	511	352	90.91	38.0	0	90.96	6.2
newblue2.3d	0	166.50	0	135.64	0	139.66	0	132.54	9	0	136.01	4.3	0	134.63	1.2
newblue3.3d	33394	198.77	38958	172.44	32840	184.40	32840	163.36	1561	31800	168.40	357.7	31454	166.95	122.4
avg. improv.		19.38%		2.51%		6.46%		-0.68%	24.06		1.59%	6.44			

TABLE IV

COMPARISON BETWEEN NTHU-ROUTE AND NTHU-ROUTE 2.0 ON ISPD08 BENCHMARKS, WHERE EACH "-" INDICATES THAT THE CORRESPONDING WIRELENGTH (RUNTIME) IMPROVEMENT RATE IS NOT CALCULATED WHEN NTHU-ROUTE COULD NOT RESOLVE OVERFLOW (FAILED TO GENERATE A SOLUTION).

Benchmark	NTHU-Route			NTHU-Route 2.0			WL improv.	Runtime
	TOF	WL(e5)	cpu(min)	TOF	WL(e5)	cpu(min)	rate (%)	improv. rate
adaptec1	0	54.18	93.6	0	53.44	10.0	1.37	9.36
adaptec2	0	52.77	16.8	0	52.29	2.1	0.91	8.00
adaptec3	0	132.85	64.9	0	131.01	10.9	1.39	5.95
adaptec4	0	122.65	10.1	0	121.69	2.6	0.78	3.88
adaptec5	0	157.30	268.4	0	155.38	23.0	1.22	11.67
bigblue1	0	56.23	171.97	0	55.95	13.7	0.50	12.55
bigblue2	198	90.23	73.17	0	90.59	8.5	-	8.61
bigblue3	18	131.62	93.05	0	130.68	5.0	-	18.61
bigblue4	-	-	-	162	231.04	131.6	-	-
newblue1	352	46.17	38.0	0	46.53	6.2	-	6.13
newblue2	0	76.25	4.3	0	75.70	1.2	0.72	3.58
newblue3	31800	106.98	357.7	31454	106.53	122.4	-	2.92
newblue4	448	129.10	144.89	138	130.46	96.7	-	1.50
newblue5	-	-	-	0	231.58	19.4	-	-
newblue6	0	177.97	244.32	0	176.89	18.1	0.61	13.50
newblue7	-	-	-	62	353.35	143.9	-	-
average							0.94	8.17

[5] ——, "Fastroute 2.0: A high-quality and efficient global router," in *Proceedings of Asia and South Pacific Design Automation Conference*, Yokohama, Japan, 2007, pp. 250–255.

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

[7] ISPD 2007 Global Routing Contest. [Online]. Available: http://www.sigda.org/ispd2007/contest.html

[8] ISPD 2008 Global Routing Contest. [Online]. Available: http://www.ispd.cc/contests/ispd08rc.html

[9] M. Cho, K. Lu, K. Yuan, and D. Z. Pan, "Boxrouter 2.0: architecture and implementation of a hybrid and robust global router," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, San Jose, CA, 2007, pp. 503–508.

[10] M. M. Ozdal and M. D. F. Wong, "Archer: a history-driven global routing algorithm," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, San Jose, CA, 2007, pp. 488–495.

[11] J. A. Roy and I. L. Markov, "High-performance routing at the nanometer scale," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, San Jose, CA, 2007, pp. 496–502.

[12] M. D. Moffitt, "Maizerouter: engineering an effective global router," in *Proceedings of Asia and South Pacific Design Automation Conference*, Seoul, Korea, 2008, pp. 226–231.

[13] J.-R. Gao, P.-C. Wu, and T.-C. Wang, "A new global router for modern designs," in *Proceedings of Asia and South Pacific Design Automation Conference*, Seoul, Korea, 2008, pp. 232–237.

[14] L. McMurchie and C. Ebeling, "Pathfinder: a negotiation-based performance-driven router for FPGAs," in *Proceedings of International Symposium on Field-Programmable Gate Arrays*, Monterey, CA, 1995, pp. 111–117.

[15] Labyrinth. [Online]. Available: http://www.ece.ucsb.edu/kastner/labyrinth

[16] C. Chu and Y.-C. Wong, "Fast and accurate rectilinear steiner minimal tree algorithm for VLSI design," in *Proceedings of International Symposium on Physical Design*, San Francisco, CA, 2005, pp. 28–35.

[17] T.-H. Lee and T.-C. Wang, "Congestion-constrained layer assignment for via minimization in global routing," *IEEE Transactions on Computer-Adided Design of Integrated Circuits and Systems*, 2008 (to appear).

[18] Gompertz curve. [Online]. Available: http://en.wikipedia.org/wiki/Gompertz_curve

[19] Gompertz curve. [Online]. Available: http://mathworld.wolfram.com/GompertzCurve.html

[20] FGR. [Online]. Available: http://vlsicad.eecs.umich.edu/BK/FGR/