NTHU-Route 2.0: A Robust Global Router for Modern Designs

Yen-Jung Chang, Yu-Ting Lee, Jhih-Rong Gao, Pei-Ci Wu, and Ting-Chi Wang

***Abstract*—This paper presents a robust global router called NTHU-Route 2.0 that improves the solution quality and runtime of 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. We report convincing experimental results to show the effectiveness of each individual enhancement. With all these enhancements together, NTHU-Route 2.0 solves all ISPD98 benchmarks with very good quality. Moreover, NTHU-Route 2.0 routes 7 of 8 ISPD07 benchmarks and 12 of 16 ISPD08 benchmarks without any overflow. Compared with other state-of-the-art global routers, NTHU-Route 2.0 is able to produce better solution quality and/or run more efficiently.**

***Index Terms*—Global router.**

I. Introduction

As feature size in very large scale integration (VLSI) design continues to shrink, interconnect delay keeps increasing and has replaced transistor delay as the main determinant of chip performance. It makes wirelength minimization become an important task in every stage of a design cycle. As a result, routing—the stage of a design cycle that arranges the routes of nets physically—becomes more critical in modern VLSI designs.

Due to the problem complexity, routing is usually divided into global routing and detailed routing. During global routing, the routes of nets are determined from a coarse-grained routing graph. Then the detailed routing is solved by taking the solution from global routing as a guide and determining the exact routing tracks for nets. Consequently, the quality of a global router influences the timing, power, and density of a chip profoundly.

Manuscript received December 14, 2009; revised May 6, 2010; accepted June 17, 2010. Date of current version November 19, 2010. This work was supported in part by the National Science Council of Taiwan, under Grants NSC-98-2220-E-007-011, NSC-98-2220-E-007-013, NSC-98-2220-E-007-032, and NSC-99-2221-E-007-113-MY3. This paper was recommended by Associate Editor J. Hu.

Y.-J. Chang is with the Department of Electrical and Computer Engineering, University of Texas, Austin, TX 78712 USA (e-mail: jalamorm@gmail.com).

Y.-T. Lee is with ChungHwa Telecommunication Laboratories, Yang-Mei, Taoyuan 32601, Taiwan (e-mail: leetroy@gmail.com).

J.-R. Gao is with the Department of Electrical and Computer Engineering, University of Texas, Austin, TX 78712 USA (e-mail: jhihrong-gao@gmail.com).

P.-C. Wu is with the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL 61820 USA (e-mail: peiciwu@gmail.com).

T.-C. Wang is with the Department of Computer Science, National Tsing Hua University, Hsinchu 300, Taiwan (e-mail: tcwang@cs.nthu.edu.tw).

Digital Object Identifier 10.1109/TCAD.2010.2061590

The global routing techniques developed earlier can be roughly categorized into sequential and concurrent methods. The sequential methods perform global routing one net at a time using maze routing [2], pattern routing [3], or rip-up and reroute, and so on. On the other hand, concurrent methods simultaneously route all nets. The multicommodity flow based technique [4] is classified as a concurrent method, and it formulates global routing as a multicommodity flow problem and then solves the problem by integer linear programming (ILP). Usually, the ILP problem is relaxed to a linear programming problem and then solved by an approximation method due to its large time complexity.

In order to boost the research and development of new global routing techniques for modern designs, International Symposium on Physical Design (ISPD) announced two global routing contests in 2007 [5] and 2008 [6], respectively. Contributed by the spirited competitions, Archer [7], BoxRouter 2.0 [8], FastRoute 3.0 [9], FastRoute 4.0 [10], FGR [11], GRIP [12], MaizeRouter [13], NCTU-R [14], NTHU-Route [1], and NTUgr [15] were developed. The global routing approaches adopted by those modern global routers can be categorized into two classes: full 3-D routing and 2-D routing followed by layer assignment. In full 3-D routing, FGR and GRIP apply maze routing and ILP on a full 3-D routing graph, respectively. Due to the high complexity of modern designs, full 3-D routing normally takes longer time than the other approach; therefore, FGR also adopts the other routing approach. The approach of 2-D routing followed by layer assignment projects the routing instance onto a plane, routes the new 2-D problem instance, and then maps the solution from the projected plane to the original multiple layers by a layer assignment method [1], [7]–[11], [13]–[15].

In this paper, we present the next generation of NTHU-Route [1], called NTHU-Route 2.0, that further improves the solution quality and runtime of NTHU-Route. NTHU-Route is based on iterative rip-up and reroute with a history based cost function to help distribute overflow, and with a congested region identification method to specify the order for nets to be ripped up and rerouted. According to the experimental results reported in [1], it could generate comparable or better solutions in less runtime as compared with the winners of the 2007 ISPD Global Routing Contest (i.e., FGR [11], MaizeRouter [13], and BoxRouter 2.0 [8]). However, from our empirical observations, NTHU-Route still has the following shortcomings.[1] First, the

[1]These shortcomings will be discussed in detail in Section IV.

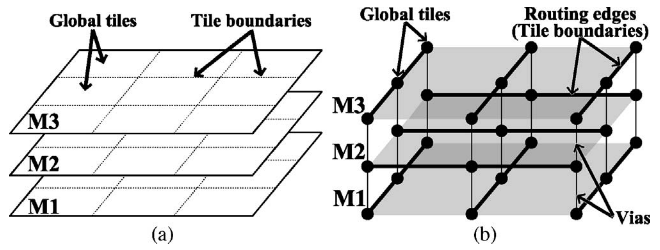


Fig. 1. (a) Multilayer design. (b) Grid graph for routing.

history based cost function adopted by NTHU-Route can reduce congestion stably in the first several rounds of rip-up and reroute, but its efficacy decreases drastically afterward. Second, NTHU-Route rips up one net and then reroutes the net immediately. Since most routing resources are not released by other nets when rerouting the ripped-up net, the rerouting order applied in NTHU-Route, i.e., rerouting nets in the non-decreasing order of their bounding box sizes, may leave room for further improvement. Finally, after checking the source code of NTHU-Route, we found that it has serious runtime bottlenecks.

In order to cope with these shortcomings, NTHU-Route 2.0 improves NTHU-Route by the following three enhancements: 1) a new history based cost function which helps NTHU-Route produce solutions with less congestion more quickly; 2) two ordering methods for congested region identification and rip-up and reroute, which help NTHU-Route generate solutions with shorter wirelength; and 3) two implementation techniques to speed up NTHU-Route. We also conduct convincing experiments to show the effectiveness of each individual enhancement.

With all these enhancements together, NTHU-Route 2.0 solves all ISPD98 benchmarks with very good quality. Moreover, NTHU-Route 2.0 routes 7 of 8 ISPD07 benchmarks and 12 of 16 ISPD08 benchmarks without any overflow. Compared with other state-of-the-art global routers, NTHU-Route 2.0 is able to produce better solution quality and/or run more efficiently.

The rest of this paper is organized as follows. In Section II we present the problem formulation and summarize previous works for global routing. In Section III we give a review of NTHU-Route. Then, in Section IV, we describe the enhancements in detail. After that, we provide the experimental results in Section V and conclude the paper in Section VI.

II. Preliminaries

A. Problem Formulation

For global routing, modern designs usually have several metal layers [as illustrated by the three-layer, denoted by M1-M3, example shown in Fig. 1(a)], and each layer can be partitioned into a set of global tiles. Meanwhile, we can model the design by a grid graph [see Fig. 1(b)]. In this graph, any two adjacent layers are linked by vias and abutting tiles are connected by routing edges. There is also a set of nets, where each net is composed of a set of pins and each pin corresponds to a tile. The routing solution for a net is to find a tree that connects all pins of the net by using vias and routing edges.

The capacity c_e of a routing edge e represents the number of available routing tracks it contains. Given a routing solution, the number of wires that utilize e is called the demand d_e on the edge. In other words, d_e represents the amount of nets that pass through e. The overflow of e is defined to be the demand that exceeds c_e as shown below

$$overflow_e = \begin{cases} d_e - c_e & \text{if } d_e > c_e \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

if $overflow_e$ is greater than zero, then we say e is an *overflowed routing edge*. Furthermore, we define the congestion of e as the ratio of the demand to the capacity

$$congestion_e = \frac{d_e}{c_e}. \tag{2}$$

The major objective of our global routing problem is to minimize the total overflow on all routing edges

$$\text{Minimize:} \sum_{e \in \boldsymbol{E}} overflow_e. \tag{3}$$

Here E is the set of routing edges. Meanwhile, the total wirelength of all routes should be as small as possible in order to minimize the circuit delay. In multilayer designs, wirelength calculation also involves vias.

B. Common Global Routing Techniques

In this subsection, we give a brief review on several global routing techniques which are commonly used by NTHU-Route and other routers.

Multi-Pin Net Decomposition is adopted by many global routers [1], [7], [8], [11], [13]. The technique decomposes a multi-pin net into several two-pin nets, and therefore routers can simplify global routing by processing each two-pin net. Two popular methods to decompose a multi-pin net are rectilinear Steiner minimal tree (RSMT) construction and minimum spanning tree (MST) construction. RSMT often provides tree topologies with shorter wirelength while MST provides greater flexibility due to more L-shaped two-pin nets produced.

Maze Routing [2] is the most classical routing method which considers all possible routes between a given source and a given sink on a routing graph. The method connects the source and sink by applying the shortest path algorithm, e.g., breadth first search, Dijkstra's algorithm, and so on, to find the route with lowest cost. Meanwhile, a bound cost can be applied by the method to improve efficiency. For example, the A*-search algorithm [16] guides a router to exclude the grid points that have estimated costs higher than the bound.

Multi-Source and Multi-Sink Maze Routing [17] evolves from maze routing and considers more potentially better routes. Since maze routing can only obtain paths that start from a given source and end at a given sink, a router utilizing maze routing may lose better solutions for multi-pin nets due to this restriction. Fig. 2 shows a routed multi-pin net which is divided into two subtrees after the route of one of its two-pin nets, i.e., $path_{AB}$, is ripped up. No matter how large the search space

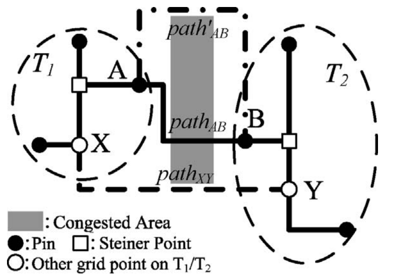


Fig. 2. Alternative paths obtained by classical maze routing ($path'_{AB}$) and multi-source and multi-sink maze routing ($path_{XY}$).

is, we can only get the paths that must start from A and end at B when finding a route for this net using maze routing. For instance, $path'_{AB}$ is an alternative path found by maze routing. However, any path that connects the two subtrees is also a possible solution to complete the routing of this multi-pin net. To overcome the drawback of maze routing, multi-source and multi-sink maze routing treats all the grid points located at one subtree as sources and those located at the other subtree as sinks. Thus, a router adopts multi-source and multi-sink maze routing is able to obtain paths like $path_{XY}$ which avoids the congested area and also has shorter wirelength (see Fig. 2).

Pattern Routing [3] routes a two-pin net with predefined patterns. Two commonly used patterns are L-shaped and Z-shaped patterns. With the predefined patterns, the method is much more efficient in comparison with maze routing. However, the solution quality may become worse because not all possible routes in the bounding box of a two-pin net are considered.

Monotonic Routing [17] finds the best monotonic routing path that can only go up or right from a grid point, assuming a given source is located below and to the left of a given sink. Since the search space for monotonic paths dose not exceed the bounding box of the two-pin net, the method only has to search $(\frac{(m+n-2)!}{(m-1)!(n-1)!})$ paths on a $m \times n$ grid graph. As a result, monotonic routing increases the search space and also preserves the same time complexity as Z-shaped pattern routing.

Rip-Up and Reroute iteratively improves the solution quality of a design. Many 2-D global routers (for instance, Chi Dispersion [18], Labyrinth [3], Fashion [19], FastRoute [17], [20], and BoxRouter [21]) utilized this technique. A typical approach adopting this technique starts by routing each net without considering congestion. After routing all nets, a congestion map can be obtained and the nets passing through congested regions will be ripped up and rerouted for finding alternative routes. If only one net will be processed at a time, this routing technique is a sequential approach and hence the order of nets to be processed influences the solution quality significantly.

Negotiated Congestion Routing was originally proposed to balance the competing goals of eliminating congestion and minimizing the performance degrading due to timing critical paths [22]. This technique was fist applied in global routing of FPGAs and introduced in PathFinder [22]. PathFinder takes a rip-up and reroute approach and defines the cost of a routing edge e as follows:

$$cost_e = (b_e + h_e) \times p_e \tag{4}$$

where b_e is the base cost of using e, p_e is related to the number of other nets presently passing e at the current iteration, and h_e is related to the history of congestion on e during previous iterations. If e has overflow frequently during previous iterations, h_e tends to discourage PathFinder to route nets through e in subsequent iterations and hence preserves the routing resource of e for nets with less routing choices. In order to achieve that, h_e is updated in the following way:

$$h_e^{i+1} = \begin{cases} h_e^i + k & \text{if } e \text{ has overflow} \\ h_e^i & \text{otherwise} \end{cases} \tag{5}$$

where i is the iteration count and k is a user-defined parameter. Equations (4) and (5) form the basis for negotiated congestion routing and their variants are derived and adopted by almost all modern global routers.

Layer Assignment is a key step for multilayer global routing because it maps a 2-D global routing result back to the original multilayer solution space. It can be done by the methods based on dynamic programming [1], [10], [23], or ILP [8]. The objective of layer assignment is usually to preserve the total overflow obtained from 2-D global routing first and to minimize the amount of vias second.

C. Modern Global Routers

Inspired by the 2007 and 2008 ISPD Global Routing Contest, many modern global routers were developed. Among them, FGR [11] (3-D routing mode) and GRIP [12] are the only two routers that apply full 3-D global routing. FGR uses minimum spanning tree construction to decompose multi-pin nets and incorporates a Lagrange multiplier scheme into its negotiated congestion routing framework. GRIP adopts ILP which is sped up by a pricing-problem solving technique and a parallel routing approach. Both FGR and GRIP obtained shorter wirelength than the other modern global routers, but their results also show that full 3-D global routing is very time consuming.

The other modern routers take another approach that first projects a multilayer design onto a 2-D plane, and then performs 2-D routing followed by layer assignment. Archer [7] adopts a Lagrangian relaxation algorithm to balance the tradeoff between total overflow and wirelength. BoxRoute 2.0 [8] uses node shifting and negotiation-based A*-search techniques to increase its routability. It also proposes progressive via/blockage-aware ILP to improve the solution quality of layer assignment. MaizeRouter [13] develops new routing techniques such as extreme edge shifting and edge retraction to improve its solution quality. NTUgr [15] uses forbidden-region and look-ahead historical cost function to improve routability. The same authors of NTUgr also develop a router that takes via capacity into consideration [24]. FastRoute [17], [20] incorporates a congestion-aware technique into FLUTE [25] and utilizes monotonic routing and multi-source and multi-sink maze routing for two-pin routing. The latest version of FastRoute [9], [10] applies a virtual capacity technique to improve routability and via-aware Steiner tree generation to

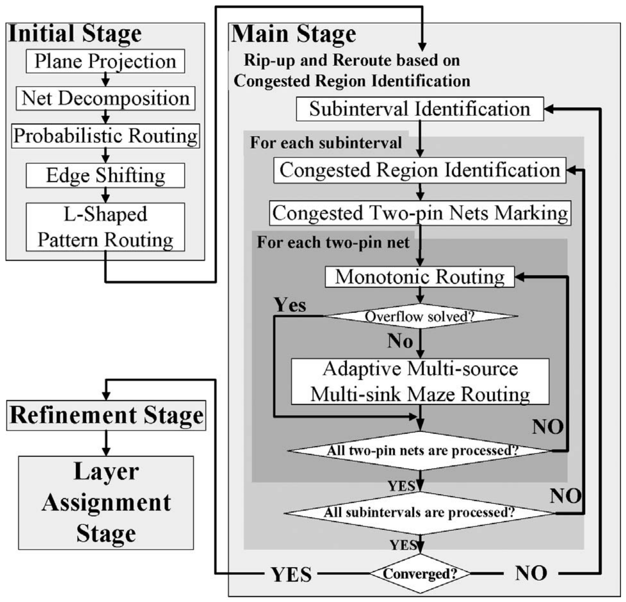


Fig. 3. Flow of NTHU-Route.

reduce the amount of vias. NCTU-R [14] adopts adaptive pseudo-random net-ordering routing and evolution-based rip-up and rerouting to accelerate runtime and reduce overflow. It also applies layer shifting and rip-up and re-assignment to minimize the amount of vias for layer assignment.

III. Previous Work: NTHU-Route

In this section, we give a detailed review of NTHU-Route [1]. The flow of NTHU-Route for multilayer designs is illustrated in Fig. 3 and it shows that there are four 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 multilayer design on a plane, and then uses FLUTE [25] 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 [20] 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 [20] to formulate the costs of routing edges).

In the main stage, NTHU-Route improves the initial solution by iteratively ripping up and rerouting every overflowed two-pin net based on congested region identification (see Section III-A). A two-pin net is *overflowed* if it passes one or more overflowed routing edges along its path. Every ripped-up two-pin net is rerouted by monotonic routing [17] 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 (see Section III-B). 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. In the main stage, a history based cost function is adopted to formulate the costs of routing edges (see Section III-C).

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

For multilayer designs, the method in [23] is applied in the layer assignment stage to map the solution from the projected plane to the original multiple layers.

A. 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 smaller nets (nets with smaller bounding boxes) earlier, because they are not as flexible as larger nets (nets with larger bounding boxes). Nets that have greater flexibility are more likely to have more overflow-free paths. Since this strategy does not consider the current congestion on the routing area, its solution quality still can be improved. For instance, assume there are a set N_s of small nets and a set N_l of large nets located in a highly congested region. According to the above-mentioned strategy, the nets in N_s will be rerouted first. After then, it is not as easy as we expect to find overflow-free paths for the nets in N_l, because most edges located nearby the region have been occupied by the nets which are rerouted earlier. As a result, NTHU-Route utilizes a method to identify congested regions, and then rip-up and reroute two-pin nets which are located in highly congested regions first.

The method starts with the solution generated in the initial stage and it works as follows. First, NTHU-Route calculates the congestion of every routing edge and defines an interval between the maximum congestion and 1. After that, it partitions the interval into m subintervals $\{I_1, I_2, ..., I_m\}$, where m is set to 10 in NTHU-Route; for example, when the maximum congestion is 2, the subintervals are [2, 1.9), [1.9, 1.8), [1.8, 1.7), ..., [1.1, 1). NTHU-Route sequentially picks every overflowed routing edge e whose congestion value falls in I_1 (starting from the most congested one), and keeps expanding a rectangular region r_e around e until the average congestion of this region is smaller than the lower bound of I_1. Then every overflowed two-pin net which has both pins located in r_e is marked[2] and will be ripped-up and rerouted later.

[2]In NTHU-Route, an overflowed two-pin net that has at most one pin located in r_e will not be marked, and therefore the net will not release routing resources in r_e and may make other ripped-up two-pin nets hard to find overflow-free paths in subsequent steps. To cope with this shortcoming, NTHU-Route 2.0 also marks overflowed two-pin nets that each have only one pin located in r_e, but for those without any pin in r_e, it still dose not mark them because of runtime consideration. By doing this, NTHU-Route 2.0 can release more routing resources than NTHU-Route in subsequent steps.

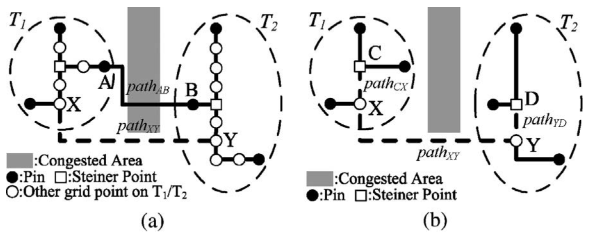


Fig. 4. (a) Paths obtained by multi-source and multi-sink maze routing. (b) Path obtained by adaptive multi-source and multi-sink maze routing.

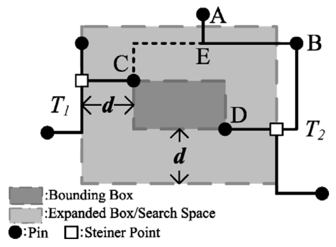


Fig. 5. Expanded box with size d limits the search space for adaptive multi-source multi-sink maze routing.

After marking all the overflowed two-pin nets in the rectangular regions which are expanded from the overflowed routing edges with respect to I_1, NTHU-Route begins to rip-up and reroute the marked overflowed two-pin nets one at a time, starting from the one with the smallest bounding box. After I_1 is processed, it iteratively applies the same region identification strategy to each of the remaining subintervals and rips up and reroutes marked two-pin nets until all subintervals are processed.

Finally, NTHU-Route rips up and reroutes the remaining overflowed two-pin nets which are never marked if there are any, before entering the next iteration of the main stage.

B. Adaptive Multi-Source and Multi-Sink Maze Routing

Multi-source and multi-sink maze routing (MMMR) proposed in [17] increases the search space by treating all grid points of one subtree as the sources and those of the other subtree as the sinks. To achieve that, it has to trace all grid points located on both subtrees. On the other hand, NTHU-Route proposed adaptive multi-source and multi-sink maze routing (AMMMR) which takes only the pins and Steiner points of a ripped up multi-pin net as the sources or the sinks. Since NTHU-Route treats the cost of every routing edge in either subtree as zero, once AMMMR reaches one grid point of the sink subtree, AMMMR will focus on finding one of the sinks along the routing edges of the sink subtree and excluding any grid point not in the sink subtree. Therefore, unlink MMMR, AMMMR dose not trace those grid points other than pins and Steiner points in both subtrees when determining the sources and sinks.

It has been proved that AMMMR preserves the same optimality as MMMR [1]. Fig. 4 demonstrates an example. In Fig. 4(a), $path_{AB}$ (solid lined) is the original path of the two-pin net net_{AB} and $path_{XY}$ (dash lined) is an alternative path found by MMMR. In Fig. 4(b), AMMMR only marks the black circles and white squares on T_1 as sources and marks those on T_2 as sinks. Since it costs 0 for AMMMR to reach X (D, respectively) from C (Y, respectively), AMMMR obtains $path_{CD}$ which has the same cost as $path_{XY}$. Note that AMMMR may alter the tree topology of a multi-pin net, and therefore retracing the tree topology is needed in order to get the new set of two-pin nets.

NTHU-Route also utilizes two methods to further improve the runtime of AMMMR. The first method is setting the path cost bound of a net to be the minimum path cost between the original path and the monotonic path which has been obtained in the previous step, i.e., monotonic routing. During the search process, it prunes any (partial) paths which have been found to have costs higher than the path cost bound.

The other method is to set an expanded box to limit the search space of AMMMR. Fig. 5 shows an expanded box with size d, where d is defined to be the distance between the boundaries of the bounding box of a ripped-up two-pin net (i.e., net_{CD} in this example) and the expanded box.[3] In NHTU-Route, the size d is set to 10 and increased by 1 and 10 at every iteration of the main stage and refinement stage, respectively.

C. History Based Cost Function

Inspired by the negotiated congestion concept of PathFinder [22], NTHU-Route applies a history based cost function to calculate the cost of a routing edge e in the main stage, and it is defined as follows:

$$cost_e = b_e + h_e \times p_e + vc_e \tag{6}$$

where b_e is the base cost of using edge e and is set to 1 (which means one unit of wirelength), $h_e \times p_e$ is the congestion cost of edge e, and vc_e is the via cost when using edge e. The historical term h_e is updated in the following way during subsequent iterations:

$$h_e^{i+1} = \begin{cases} h_e^i + 1 & \text{if } e \text{ has overflow} \\ h_e^i & \text{otherwise} \end{cases} \tag{7}$$

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

$$p_e = \left(\frac{d_e + 1}{c_e}\right)^{k_1} \tag{8}$$

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

$$vc_e = \begin{cases} 1 & \text{if passing } e \text{ makes a bend} \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

in addition, if 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, then $cost_e$ is set to zero rather than the one defined in (6). Therefore, a two-pin route of one multi-pin net is likely to share the same edges with other two-pin routes, and thus reduces the wirelength and routing demand.

[3]As shown in Fig. 5, NTHU-Route will exclude $path_{CA}$ ($path_{CB}$, respectively) because A (B, respectively) is not located in the expanded box. To consider more routing solutions than NTHU-Route, NTHU-Route 2.0 will also search any pin located outside the expended box but in either subtree.

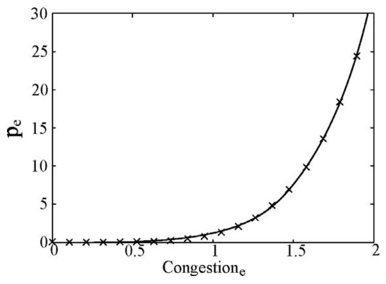


Fig. 6. Curve of the penalty term utilized in NTHU-Route.

The curve of the penalty term utilized in NTHU-Route is illustrated in Fig. 6. We can see that the penalty term of a routing edge will drastically increase after the edge becomes overflowed. As a result, the penalty term can guide NTHU-Route to try to avoid routing a net through overflowed routing edges.

IV. Enhancements for NTHU-Route

In this section, we propose three enhancements for NTHU-Route to get NTHU-Route 2.0.

The first enhancement is a new history based cost function to be adopted in the main stage. A cost function plays an important role in a global router. It should reflect the congestion of routing edges and guides a router to reduce total overflow effectively as well as to shorten total wirelength moderately. In other words, it prevents a router from routing nets through overflowed routing edges repeatedly. Meanwhile, it should discourage the router to detour around overflowed routing edges to some extent such that the wirelength and the amount of vias will not be increased too much.

As modern designs get more complicated, it is more and more difficult to define a good cost function that can predict the behavior of all routed nets during the routing process. As a result, NTHU-Route applies a history based cost function [i.e., (6)] which can reduce the congestion stably. However, the cost function dose not work well when the maximum overflow among all overflowed routing edges is close to one, i.e., every overflowed edge almost only has one routing track short. Therefore, a new history based cost function which can guide our router to avoid using these edges in this situation is needed badly and it will be presented in Section IV-A.

The next enhancement is new ordering methods for rip-up-and-reroute and congested region identification. In NTHU-Route, it rips up one overflowed two-pin net and then reroutes the net immediately. Since most routing resources are not released by other nets when rerouting the ripped-up two-pin net, the rerouting order applied by NTHU-Route, i.e., rerouting the net in the non-decreasing order of the bounding box size, does not actually work well. Therefore, we propose two ordering methods which are more suitable for rip-up-and-reroute and congested region identification, and their details are given in Section IV-B.

Finally, we develop two techniques to help reduce runtime for NTHU-Route. These two techniques are more related to the implementation aspect and will be detailed in Section IV-C.

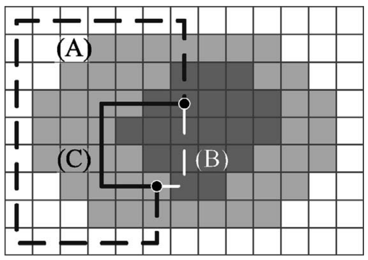


Fig. 7. Different paths found due to different base costs.

A. New History Based Cost Function

The history based cost function applied in NTHU-Route guides the router to reduce overflow significantly in the first several iterations (typically no more than ten iterations) of the main stage, but after then, it reduces overflow less effectively. Therefore, the main goal of our new history based cost function is to improve the effectiveness of the original one. The new history based cost function is divided into three sub-cost functions: base cost function, congestion cost function, and via cost function. The former two functions mainly focus on reducing overflow while the third one targets on shortening wirelength.

1) *Adaptive Base Cost Function:* In NTHU-Route, the base cost function is always set to 1 which represent one unit of wirelength. In fact, the setting makes it obtain shorter paths rather than paths with lower overflow in later iterations of the main stage. On the other hand, if we set the cost to 0, a global router may use too many routing resources without considering the impact of wirelength: the excessive usage of routing resources causes other nets more likely to pass through highly congested area in subsequent steps and hence induces overflow.

Take Fig. 7 for example, where the darkest area is a region which contains overflowed routing edges, (i.e., a region with high congestion), and the second darkest area is a region which contains edges that are nearly overflowed. For a given two-pin net, A is the path found by a global router which has no limitation on routing resource usage: it occupies excessive resources and hence leaves a potential problem. B is the path found by a global router which takes the impact of wirelength too seriously: it makes B pass through a highly congested area in order to reduce 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 lesser congested region than B.

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.

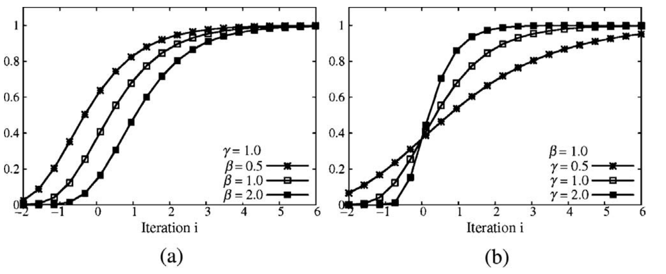


Fig. 8. (a) Gompertz curve defined as $e^{-\beta e^{-i}}$ and it moves toward right with larger β. (b) Gompertz curve defined as $e^{-e^{-\gamma i}}$ and it becomes smoother with smaller γ.

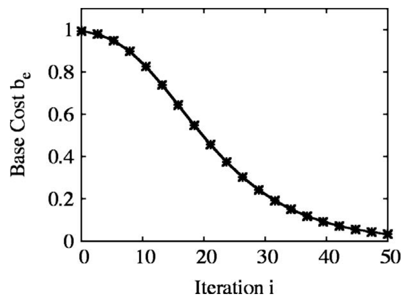


Fig. 9. Curve of adaptive base cost function b_e.

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

$$b_e = 1 - e^{-\beta e^{-\gamma i}} \tag{10}$$

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

This base cost function is based on a Gompertz curve [26] which has the slowest growth rate at the start and the end of a time period. Two sets of Gompertz curves which are respectively controlled by β and γ are shown in Fig. 8. The curve of the base cost function we use in our router is shown in Fig. 9. As the iteration count increases, the base cost will be reduced and hence encourages our router to obtain paths with less overflow rather than paths with shorter wirelength.

We have conducted experiments on all ISPD07 6-layer benchmarks [5] and five ISPD08 benchmarks [6] to observe the improvement due to the adaptive base cost function (the other enhancements were excluded) and the results are shown in Table I. For the other 11 ISPD08 benchmarks which are not shown in this table, we do not report their results because 8 of them are from the ISPD07 6-layer benchmarks (and therefore their overflow results are the same as those of ISPD07 benchmarks) and NTHU-Route failed to generate a routing solution for each of the three remaining cases due to the out-of-memory problem. The detailed characteristics of the ISPD07 and ISPD08 benchmarks will be given in Section V.

As can be seen from Table I, after NTHU-Route adopted the adaptive base cost function, it successfully generated an overflow-free solution for bigblue3 which could not be solved without inducing any overflow before. Moreover, it further reduced the total overflow (TOF) on newblue1, newblue3, newblue4, and bigblue2. For the benchmarks which could be solved by NTHU-Route without inducing any overflow

TABLE I
IMPROVEMENTS DUE TO THE ADAPTIVE BASE COST FUNCTION

Benchmark	NTHU-Route		With Adaptive Base Cost Function	
	TOF	TWL (e5)	TOF	TWL (e5)
adaptec1	0	90.56	0	93.00
adaptec2	0	92.17	0	91.74
adaptec3	0	205.04	0	204.13
adaptec4	0	188.43	0	186.58
adaptec5	0	265.03	0	268.43
newblue1	352	90.91	146	91.30
newblue2	0	136.01	0	134.78
newblue3	31 800	168.40	31 626	167.99
newblue4	448	129.10	272	129.71
newblue6	0	178.03	0	179.27
bigblue1	0	56.24	0	57.75
bigblue2	198	90.23	74	90.69
bigblue3	18	131.62	0	132.03

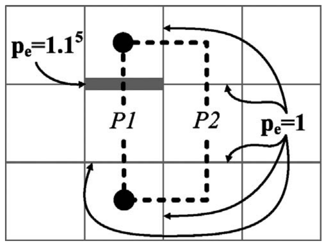


Fig. 10. Two possible paths for a ripped up two-pin net.

before, they are still solved without any overflow. As for total wirelength (TWL), the adaptive base cost function also limits its growth in the first several iterations of the main stage as the original base cost function does, and therefore the wirelength of each benchmark does not always become longer.

2) *Congestion Cost Function with Overflow Prediction:* From our empirical observation, the congestion cost function applied in NTHU-Route dose not perform well when the maximum congestion value among all overflowed routing edges is close to one. For example, assume each tile boundary e in Fig. 10 has ten routing tracks ($c_e = 10$) and has overflow nine times in previous iterations ($h_e = 10$). In the current iteration, each e is already passed by nine nets ($d_e = 9$), except the gray-bolded boundary which is passed by ten nets ($d_e = 10$). Suppose NTHU-Route is going to reroute a ripped-up two-pin net using either P_1 or P_2 as shown in the figure. Note that P_1 is an overflowed path and P_2 is an overflow-free path. According to (6), the costs of P_1 and P_2 would be $(1 \times 2 + 10 \times (1.1^5 + 1^5) + 0) = 28.1$ and $(1 \times 4 + 10 \times 1^5 \times 4 + 1 \times 2) = 46$, respectively, and therefore NTHU-Route would choose the overflowed path P_1 repeatedly for the next 11 iterations. In the 12th iteration, the cost of P_1 will finally become $(2 + (10 + 12) \times 1.1^5 + 10 \times 1 + 0) = 47.43$ and hence makes NTHU-Route choose the overflow-free path P_2 instead. The example shows that the congestion cost of an overflowed routing edge must be raised large enough in order to guide the router to converge the solution quickly. Therefore, we propose a new congestion cost function which raises the congestion cost of a routing edge drastically even before the edge is overflowed.

The new 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

$$p_e = \left(\frac{d_e + 1}{c_e} \times f(h_e, i) \right)^{k_1} \tag{11}$$

where i is the current iteration count, h_e is the historical term used in NTHU-Route, and $f(h_e, i)$ amplifies congestion. Before describing $f(h_e, i)$, we provide a simplified version of $f(h_e, i)$, which is defined as follows:

$$f_{simple}(h_e, i) = \left(\frac{i \times k_2}{i \times k_2 - (h_e - 1)} \right) \tag{12}$$

where $(h_e - 1)$ is the overflow count of routing edge e in the past, because h_e is always set to 1 in the first iteration of the main stage in NTHU-Route. And k_2 is a user-defined parameter which controls the maximum value of $f_{simple}(h_e, i)$. Assume that our router is running the ith iteration of the main stage. Since the historical term of a routing edge e is always no more than the current iteration count, we can get $f_{simple}(h_e, i) = \left(\frac{i \times k_2}{i \times k_2 - (i-1)} \right) \leq \left(\frac{i \times k_2}{i \times k_2 - i} \right) = (\frac{k_2}{k_2 - 1})$. Therefore, $(\frac{k_2}{k_2-1})$ is the upper bound of $f_{simple}(h_e, i)$. We set k_2 to 1.5 to make the upper bound close to 3 in NTHU-Route 2.0, because from our empirical observation, the value achieves a good balance between the reduction rate of the total overflow and the increase rate of wirelength.

As i is small in the first several iterations of the main stage, the penalty terms of overflowed routing edges are amplified by values that are close to the maximum value of $f_{simple}(h_e, i)$, 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 overflow-free paths 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 $f_{simple}(h_e, i)$. We now have $f(h_e, i)$ defined as follows:

$$f(h_e, i) = \left(\frac{i \times (k_2 + adj(i))}{i \times (k_2 + adj(i)) - (h_e - 1)} \right) \tag{13}$$

where $adj(i)$ is defined as

$$adj(i) = k_3 \times (1 - e^{-\beta e^{-\gamma i}}). \tag{14}$$

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 (10), i.e., 5 and 0.1, respectively. With this adjustment term, $f(h_e, i)$ will not over amplify the penalty term in the first several iterations. For instance, assume NTHU-Route is executing the second iteration of the main stage and edge e has overflow in the previous iteration. As a result, $adj(2)$ would be close to 3 and the amplifying term $f(h_e, i)$ would be close to a moderate value $\left(\frac{2 \times (1.5+3)}{2 \times (1.5+3) - (2-1)} \right) = 1.125$.

Now our router not only drastically increases the penalty term of a routing edge when the edge has overflow, but also drastically increases the penalty term before it has overflow if the edge has a high frequency to have overflow in the past iterations. This fact can be seem from Fig. 11 which shows the curves of the penalty terms of NTHU-Route (crossed curve) and our router (squared curves). Here the two squared curves are derived when $i = 16, h_e = 5$ and $i = 16, h_e = 15$, respectively.

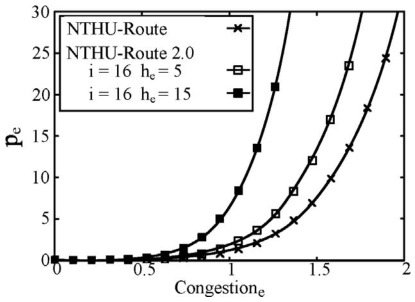


Fig. 11. Curves of the penalty terms utilized in NTHU-Route and NTHU-Route 2.0.

Again, we have conducted experiments on the ISPD07 6-layer benchmarks and ISPD08 benchmarks to observe the improvements due to the new congestion cost function (the other enhancements were excluded). Although we only show the total overflows of four benchmarks after different iterations in Fig. 12, the total overflow reductions of the other benchmarks have similar trends as the number of iterations increases.

There are two curves in each part of Fig. 12. The hollow-squared curve shows the total overflow of NTHU-Route right after each iteration, and the solid-squared curve is the one of NTHU-Route with the new congestion cost function. We can see that solid-squared curves have larger overflow right after the first iteration due to the adjustment term. After then, the new congestion cost function helped NTHU-Route reduce total overflow effectively. For each benchmark, the total overflows shown on the last points of both curves were close to each other. This indicates that NTHU-Route could achieve (almost) the same total overflow with less iterations after it adopted the new congestion cost function.

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

$$vc_e = \begin{cases} v_e \times c_e \times b_e & \text{if passing } e \text{ makes a bend} \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

where v_e is the expected amount of vias for a bend, c_e is the cost of a via (typically measured by units of wirelength), and b_e is the same as the one defined in (10). Taking a 6-layer design with preferred directions for example, a bend corresponds to $\lceil \frac{1\times5+3\times3+5\times1}{9} \rceil = 3$ expected vias. Furthermore, if a via equals three units of wirelength, then vc_e will be set to $3 \times 3 \times b_e$.

We have conducted experiments on those ISPD07 6-layer benchmarks and ISPD08 benchmarks that could be solved by NTHU-Route without inducing any overflow.[4] The results are shown in Table II. With the new via cost function (the

[4]Since overflow minimization is the first objective of NTHU-Route, wirelength improvements due to the new via cost function will become meaningful only for the benchmarks for which NTHU-Route could generate overflow-free solutions.

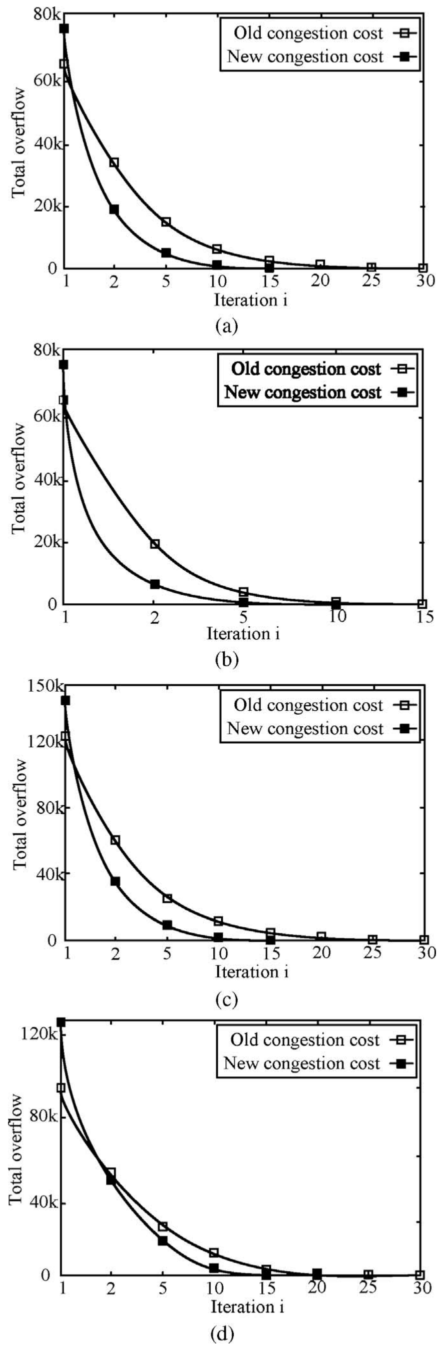


Fig. 12. (a) Total overflow on adaptec1. (b) Total overflow on adaptec3. (c) Total overflow on adaptec5. (d) Total overflow on bigblue1.

other enhancements were excluded), NTHU-Route averagely reduced the amount of vias by 1.17%. Meanwhile, the average wirelength (TWL) was also reduced by 0.43% because the total wirelength of a multilayer design is composed of the amount of vias and wire segments. Additionally, all listed cases could still be solved without inducing any overflow.

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 in order to further shorten the wirelength for a design that can be routed by NTHU-Route without inducing any overflow.

TABLE II
IMPROVEMENTS DUE TO THE NEW VIA COST FUNCTION

Benchmark	NTHU-Route		With New Via Cost Function		Improvement	
	Amount of Vias (e5)	TWL (e5)	Amount of Vias (e5)	TWL (e5)	Amount of Vias (%)	TWL (%)
adaptec1	18.19	90.56	17.85	89.59	1.87	1.07
adaptec2	19.70	92.17	19.59	91.74	0.56	0.38
adaptec3	36.10	205.04	35.44	204.13	1.83	1.23
adaptec4	32.89	188.43	32.74	186.58	0.46	0.29
adaptec5	53.87	265.03	53.17	268.43	1.30	−1.28
newblue2	29.88	136.01	29.76	134.78	0.40	0.90
newblue6	76.00	178.03	75.05	177.34	1.26	0.39
bigblue1	19.14	56.24	18.82	56.00	1.67	0.43
Average	35.72	151.44	35.30	151.07	1.17	0.43

1) *New Ordering Method for Rip-Up and Reroute:* Every time when a set of overflowed two-pin nets is identified and marked, NTHU-Route begins to rip up the one 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 overflowed. It will keep doing this until all overflowed two-pin nets are processed. Take Fig. 13 for example, where Fig. 13(a) shows a routing solution obtained from a previous iteration. Assume each edge has a unit capacity and the bold edges are overflowed edges; therefore, 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 overflowed after N_1, N_2, and N_3 are rerouted [see Fig. 13(b)]. In Fig. 13(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 largest net), 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. 13(c)]. Moreover, N_1, N_2, and N_3 can even leave untouched because they are no longer overflowed after N_4 is ripped up and rerouted. Although the solutions in Fig. 13(b) and (c) both have no overflow, the one in Fig. 13(c) has shorter wirelength.

As motivated by the example in Fig. 13, NTHU-Route 2.0 reverses the rip-up and reroute ordering of NTHU-Route. As shall be seen later in the experimental results, this ordering method helps NTHU-Route shorten wirelength for those benchmarks that can be solved without overflow. However, the efficacy of utilizing this ordering method by other routers is not clear and requires a detailed evaluation, which is beyond the scope of this paper.

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. 14(a)]. In Fig. 14(a), the regions r_m, r_n, and r_o have overflowed routing edges and the region r_p has no overflowed edges, where a region with darker color is more congested.

In NTHU-Route, the order of the overflowed two-pin nets to be ripped-up and rerouted is determined by the congestion

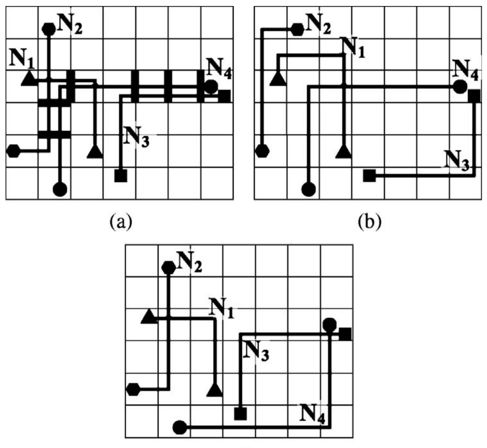


Fig. 13. (a) Routing solution obtained from a previous iteration. (b) Routing solution obtained by NTHU-Route. (c) Routing solution obtained by NTHU-Route with the new ordering method for rip-up and reroute.

value of a region where an overflowed two-pin net is located first, and the size of the bounding box of the net second. Taking Fig. 14(b) for example, NTHU-Route processes the overflowed two-pin nets located in r_m, say $nets_m$, first. When NTHU-Route processes $nets_m$, it rips up and reroutes the overflowed two-pin nets in the non-decreasing order of the sizes of the bounding boxes of the nets. After $nets_m$ are processed, NTHU-Route continuous to sequential process the overflowed two-pin nets located in r_n and r_o.

From our empirical observations, when NTHU-Route utilizes the new ordering method for rip-up and reroute (i.e., it rips up and reroutes overflowed two-pin nets in the non-increasing order of the sizes of the bounding boxes of the nets), the following situation might happen. Since the nets with larger bounding boxes in $nets_m$ have more routing choices, some of them may obtain paths with longer wirelengths in order to pass through r_n and r_o for finding routes with lower costs [see Fig. 14(b)]. Nonetheless, if NTHU-Route processes the overflowed two-pin nets located in r_o first, then the nets (especially those with larger bounding boxes) are likely rerouted to outer region(s) for avoiding the routing edges with higher costs in r_n and r_m [see Fig. 14(c)]. Consequently, the routing resources located at r_o will be released and the nets located at r_m and r_n can use them without detouring to r_p.

As motivated by the example in Fig. 14, NTHU-Route 2.0 also reverses the ordering method for congested region identification in order to further increase the effectiveness of the new ordering method for rip-up and reroute.

3) *Results:* Table III shows the experimental results obtained by applying the two new ordering methods to NTHU-Route (the other enhancements were excluded) on those ISPD07 6-layer benchmarks and ISPD08 benchmarks that could be solved by NTHU-Route without inducing overflow.[5] The new ordering method for rip-up and reroute helped NTHU-Route improve wirelength (TWL) by 0.34% on average

[5]As stated in the beginning of this subsection, the new ordering methods are mainly applied for shortening wirelength, and therefore we only show the experimental results on the benchmarks for which NTHU-Route could generate overflow-free solutions.

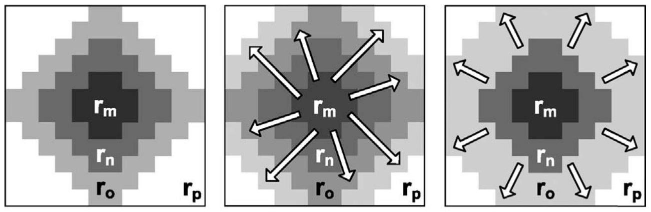


Fig. 14. (a) Congestions are often radially distributed. (b) NTHU-Route first rips up and reroutes the overflowed two-pin nets located in r_m. (c) NTHU-Route 2.0 first rips up and reroutes the overflowed two-pin nets located in r_o and some routing resources in r_o are likely released before the overflowed two-pin nets located in r_m and r_n are ripped up and rerouted.

TABLE III
IMPROVEMENTS DUE TO NEW ORDERING METHODS

Benchmark	NTHU-Route	A[1]		B[2]		C[3]	
	TWL (e5)	TWL (e5)	Improv. (%)	TWL (e5)	Improv. (%)	TWL (e5)	Improv. (%)
adaptec1	90.56	90.19	0.41	90.54	0.02	90.16	0.44
adaptec2	92.17	91.95	0.24	92.19	−0.02	91.89	0.30
adaptec3	205.04	203.87	0.57	205.11	−0.03	203.62	0.69
adaptec4	188.43	188.00	0.23	188.38	0.03	187.95	0.25
adaptec5	265.03	263.33	0.64	264.90	0.05	263.12	0.72
newblue2	136.01	135.85	0.12	136.40	−0.29	135.84	0.12
newblue6	178.03	177.44	0.33	179.06	−0.58	177.43	0.34
bigblue1	56.24	56.13	0.20	56.41	−0.30	56.12	0.21
Average	151.44	150.85	0.34	151.58	−0.11	150.77	0.38

[1]Only with the new ordering method for rip-up and reroute.
[2]Only with the new ordering method for congested region identification.
[3]With both new ordering methods.

(see column A). Although the ordering method for congested region identification increases the wirelength by 0.11% on average (see column B), the average improvement rate raises to 0.38% when combining both ordering methods (see column C), which is better than that obtained by applying either method alone. Note that the average improvement rate on wirelength looks marginal, but the wirelength of each case can be consistently improved by up to 0.72%. In addition, NTHU-Route is still able to obtain overflow-free solutions for these cases when applying the new ordering methods.

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 routing edges when searching paths for overflowed two-pin nets. The other is to determine if a part of a multi-pin net already passes through a routing 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 $cost_e$ of a routing edge e on the fly whenever $cost_e$ 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 our router, we also compute and store the sum of the base cost and the congestion cost for every routing edge and update them when necessary. Doing this makes our router obtain each $cost_e$ faster.

TABLE IV
RUNTIME IMPROVEMENTS

Benchmark	NTHU-Route	A[1]		B[2]		C[3]	
	CPU (min)	CPU (min)	Speedup (times)	CPU (min)	Speedup (times)	CPU (min)	Speedup (times)
adaptec1	79.9	54.1	1.48	35.1	2.28	32.5	2.46
adaptec2	12.6	8.8	1.43	6.3	2.00	5.6	2.25
adaptec3	47.9	35.3	1.36	23.2	2.06	21.8	2.20
adaptec4	6.9	3.7	1.86	4.8	1.44	3.6	1.92
adaptec5	215.9	154.3	1.40	82.8	2.61	69.1	3.12
newblue1	35.5	22.8	1.56	16.8	2.11	15.5	2.29
newblue2	3.0	1.6	1.87	1.8	1.67	1.4	2.14
newblue3	303.1	211.8	1.43	158.5	1.91	148.3	2.04
newblue4	108.7	64.1	1.70	59.7	1.82	50.6	2.15
newblue6	214.4	123.3	1.74	102.7	2.09	70.6	3.04
bigblue1	120.7	83.5	1.45	63.9	1.89	47.2	2.56
bigblue2	48.6	30.6	1.59	33.2	1.46	21.0	2.31
bigblue3	58.1	34.9	1.66	45.6	1.27	24.5	2.37
Average	96.56	63.75	1.58	48.80	1.89	39.36	2.37

[1]Only with the pre-computing and storing technique.
[2]Only with the hash table technique.
[3]With both techniques.

Next we discuss the second bottleneck. In NTHU-Route, every routing 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 our router, 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. Note that some modern global routers (e.g., MaizeRouter [13]) also adopt balanced search trees to implement their lookup tables, we believe their runtimes can also be reduced by replacing search trees with hash tables.

We have conducted experiments on the ISPD07 6-layer benchmarks and ISPD08 benchmarks to observe the runtime improvements due to the above-mentioned two techniques, and the results are shown in Table IV.[6] The results are collected on a Linux desktop with an Intel Core 2 Duo 2.4 GHz CPU (only one core was utilized) and 8 GB memory. We only show the runtime information in this table because both techniques do not change the routing quality of each benchmark. On average, the two techniques helped NTHU-Route run 1.58× and 1.89× faster, respectively. Furthermore, NTHU-Route could be sped up 2.37× on average after adopting both techniques.

V. EXPERIMENTAL RESULTS

In this section, we present and compare the experimental results between NTHU-Route 2.0 and other state-of-the-art global routers. The experiments were conducted on a Linux desktop with an Intel Core 2 Duo 2.4 GHz CPU (only one core was utilized) and 8 GB memory. Three sets of benchmarks from ISPD98 [27], ISPD07 [5], and ISPD08 [6] were used to measure the performance of each router. Table V shows the statistics of these benchmarks. Note that half of ISPD08 benchmarks are from 6-layer ISPD07 benchmarks.

[6]NTHU-Route were unable to generate solutions for newblue5, newblue7, and bigblue4 due to the out-of-memory problem. Therefore, the comparisons for those benchmarks are not included in Table IV.

TABLE V
STATISTICS OF ISPD98, ISPD07, AND ISPD08 BENCHMARKS

Benchmark		Grids	# Layers	# Nets
ISPD98	ibm01	64 × 64	1	11507
	ibm02	80 × 64	1	18 429
	ibm03	80 × 64	1	21 621
	ibm04	96 × 64	1	26 163
	ibm06	128 × 64	1	33 354
	ibm07	192 × 64	1	44 394
	ibm08	192 × 64	1	47 944
	ibm09	256 × 64	1	50 393
	ibm10	256 × 64	1	64 227
ISPD07 (2/6 layers)	adaptec1	324 × 324	2/6	219 794
	adaptec2	424 × 424	2/6	260 159
	adaptec3	774 × 779	2/6	466 295
	adaptec4	774 × 779	2/6	515 304
ISPD08 (6 layers)	adaptec5	465 × 468	2/6	867 441
	newblue1	399 × 399	2/6	331 663
	newblue2	557 × 463	2/6	463 213
	newblue3	973 × 1256	2/6	551 667
ISPD08	newblue4	455 × 458	6	636 195
	newblue5	637 × 640	6	1 257 555
	newblue6	463 × 463	6	1 286 452
	newblue7	488 × 490	8	2 635 625
	bigblue1	227 × 227	6	282 974
	bigblue2	468 × 471	6	576 816
	bigblue3	555 × 557	8	1 122 340
	bigblue4	403 × 405	8	2 228 903

A. Results on ISPD98 Benchmarks

In Table VI, we compare NTHU-Route 2.0 with Archer [7], BoxRouter 2.0 [8], FastRoute 3.0 [9], FGR 1.0 [11], MaizeRouter [13], and NTHU-Route [1] on ISPD98 benchmarks in terms of TOF and TWL. Both NTHU-Route and NTHU-Route 2.0 were run on our own machine, the results of the other routers are quoted from their papers.

As can be seen from Table VI, NTHU-Route 2.0, like the other routers, obtained an overflow-free solution for each benchmark. Furthermore, NTHU-Route 2.0 obtained shorter TWL than Archer, FastRoute 3.0, MaizeRouter, and NTHU-Route for each benchmark. Our router also generated shorter average TWL when compared with BoxRouter 2.0 and comparable average TWL when compared with FGR 1.0. In addition, we also report the runtimes (CPU) of NTHU-Route and NTHU-Route 2.0 in Table VI, and the results show that NTHU-Route 2.0 was 3.77× faster than NTHU-Route.

B. Results on ISPD07 Benchmarks

In this subsection, we compare NTHU-Route 2.0 with the winners of the 2007 ISPD Global Routing Contest (i.e., BoxRouter 2.0 [8], FGR 1.0 [11], and MaizeRouter [13]) and two notable routers (i.e., Archer [7] and GRIP [12]). Table VII lists the results and comparisons between NTHU-Route 2.0 and the other routers on the ISPD07 2-layer and 6-layer benchmarks in terms of TOF, TWL, and CPU. The TWL is composed of the total length of wire segments plus three times the amount of vias. When comparing the TWL results of all routers, only overflow-free benchmarks are taken into consideration because the problem objective is to reduce TOF first and TWL second. For FGR 1.0 and MaizeRouter,

TABLE VI

RESULTS OF ARCHER, BOXROUTER 2.0, FASTROUTE 3.0, FGR 1.0, MAIZEROUTER, NTHU-ROUTE, AND NTHU-ROUTE 2.0 ON ISPD98 BENCHMARKS

Benchmark	Archer		BoxRouter 2.0		FastRoute 3.0		FGR 1.0		MaizeRouter		NTHU-Route			NTHU-Route 2.0		
	TOF	TWL	TOF	TWL	TOF	TWL	TOF	TWL	TOF	TWL	TOF	TWL	CPU (s)	TOF	TWL	CPU (s)
ibm01	0	64 389	0	62 659	0	64 221	0	63 332	0	63 720	0	63 321	4.3	0	62 498	1.54
ibm02	0	171 805	0	171 110	0	172 223	0	168 918	0	170 342	0	170 531	7.8	0	169 881	3.15
ibm03	0	146 770	0	146 634	0	146 753	0	146 412	0	147 078	0	146 551	6	0	146 458	1.49
ibm04	0	169 977	0	167 275	0	170 146	0	167 101	0	170 095	0	168 262	21.7	0	166 452	3.81
ibm06	0	278 841	0	277 913	0	279 471	0	277 608	0	279 566	0	278 617	12.4	0	277 696	3.16
ibm07	0	370 143	0	365 790	0	369 023	0	366 180	0	369 340	0	366 288	16.2	0	366 133	4.07
ibm08	0	404 530	0	405 634	0	405 935	0	404 714	0	406 349	0	405 169	11	0	404 976	3.72
ibm09	0	414 223	0	413 862	0	414 913	0	413 053	0	415 852	0	415 464	12.9	0	414 738	3.95
ibm10	0	583 805	0	590 141	0	582 838	0	578 795	0	585 921	0	580 793	33.6	0	579 870	6.99
Comp.	–	1.08	–	1.03	–	1.09	–	1.00	–	1.08	–	1.04	3.77	–	1.00	1.00

TABLE VII

RESULTS OF ARCHER, BOXROUTER 2.0, FGR 1.0, GRIP, MAIZEROUTER, NTHU-ROUTE, AND NTHU-ROUTE 2.0 ON THE ISPD07 BENCHMARKS

Benchmark [1]	Archer		BoxRouter 2.0		FGR 1.0			GRIP[2]		MaizeRouter			NTHU-Route			NTHU-Route 2.0		
	TOF	TWL (e5)	TOF	TWL (e5)	TOF	TWL (e5)	CPU (min)	TOF	TWL (e5)	TOF	TWL (e5)	CPU (min)	TOF	TWL (e5)	CPU (min)	TOF	TWL (e5)	CPU (min)
adaptec1.2d	0	58.3	0	58.4	0	54.1	315.5	–	–	0	60.5	184.3	0	57.1	79.5	0	55.9	5.0
adaptec2.2d	0	54.6	0	55.7	0	52.2	33.6	–	–	0	57.4	81.3	0	54.5	12.3	0	54.0	1.0
adaptec3.2d	0	135.4	0	138.0	0	130.4	159.4	–	–	0	136.7	124.8	0	137.2	47.2	0	134.4	5.0
adaptec4.2d	0	126.3	0	127.8	0	124.9	11.6	–	–	0	128.2	15.8	0	128.7	6.3	0	127.8	1.0
adaptec5.2d	0	162.5	0	162.1	0	151.4	689.0	–	–	0	175.2	3009.5	0	160.3	215.4	0	157.0	11.9
newblue1.2d	682	49.3	400	51.1	228	47.3	1441.1	–	–	1244	51.5	870.0	352	47.8	35.2	0	48.6	3.6
newblue2.2d	0	77.9	0	78.7	0	76.4	5.1	–	–	0	79.5	9.4	0	79.2	2.5	0	78.5	0.5
newblue3.2d	33 394	109.3	38 958	107.2	40 506	108.8	1490.2	–	–	32 342	113.9	492.7	31 800	111.0	302.0	31 390	94.9	70.9
adaptec1.3d	0	113.8	0	92.0	0	87.8	324.4	0	81.0	0	99.7	104.9	0	90.6	79.9	0	88.8	5.2
adaptec2.3d	0	112.6	0	94.3	0	89.6	33.2	0	82.4	0	99.6	72.5	0	92.2	12.6	0	90.9	1.2
adaptec3.3d	0	244.1	0	207.4	0	198.8	181.6	0	185.4	0	210.2	127.5	0	205.0	47.9	0	200.8	5.3
adaptec4.3d	0	221.6	0	186.4	0	182.9	27.3	0	172.3	0	190.9	21.3	0	188.4	6.9	0	186.0	1.4
adaptec5.3d	0	334.1	0	270.4	0	258.4	708.5	0	238.9	0	302.9	3554.5	0	265.0	215.9	0	260.2	12.2
newblue1.3d	682	117.0	394	92.9	220	94.0	1440.6	0	83.9	1304	100.6	915.9	352	90.9	35.5	0	91.0	3.7
newblue2.3d	0	166.5	0	135.6	0	132.1	8.1	0	121.4	0	139.4	11.3	0	136.0	3.0	0	134.6	0.7
newblue3.3d	33 394	198.8	38 958	172.4	40 576	172.8	1467.5	52 518	156.1	32 156	181.8	541.1	31 800	168.4	303.1	31 390	167.9	72.1
comp. 2d	–	1.01	–	1.02	–	0.97	78.12	–	–	–	1.05	84.56	–	1.01	9.99	–	1.00	1.00
comp. 3d	–	1.24	–	1.03	–	0.99	77.75	–	0.92	–	1.08	85.25	–	1.02	9.42	–	1.00	1.00

The wirelength comparisons only involve the overflow-free benchmarks.
[1]The suffix ".2d" (".3d") is added to the name of each benchmark for denoting a 2-layer (6-layer) design.
[2]In GRIP's website [28], the authors mention that their results in [12] are incorrect, and therefore we quote the corrected results from [28]. Additionally, their results on the 2-layer designs are not available.

we conducted their experiments on our own machine and report the collected TOF, TWL, and CPU results. For Archer, BoxRouter 2.0, and GRIP, we quote their TOF and TWL results directly from their papers, because the source codes of Archer and GRIP are not available to us and BoxRouter 2.0 requires extra 3rd-party toolsets to execute.

As can be seen from Table VII, NTHU-Route 2.0 was able to produce overflow-free solutions for all benchmarks, except newblue3 which has been proven unroutable before. The other routers, except GRIP, failed to produce an overflow-free solution for newblue1. NTHU-Route 2.0 also generated shorter wirelengths than BoxRoute 2.0 and MaizeRouter, smaller average wirelength than Archer, and comparable wirelengths in comparison with FGR 1.0 on overflow-free benchmarks, i.e., adaptec1-adaptec5 and newblue2. Furthermore, our router is much more efficient in comparison with FGR 1.0 and Maize-Router. GRIP was able to generate the shortest wirelength among all global routers, but according to [12], it required tremendous amount of runtime (even with a heterogenous grid of CPUs) due to the large time complexity induced by full 3-D global routing and ILP. In addition, GRIP generated a much worse TOF result on newblue3 than NTHU-Route 2.0.

We also list and compare the results of NTHU-Route 2.0 and NTHU-Route in Table VII. The table shows that NTHU-Route 2.0 successfully solved newblue1 and produced lower total overflow on newblue3. Furthermore, NTHU-Route 2.0 slightly improved wirelength, and was 9.99× and 9.42× faster than NTHU-Route on the ISPD07 2-layer and 6-layer benchmarks, respectively.

C. Results on ISPD08 Benchmarks

In this subsection, we compare NTHU-Route 2.0 with the other two winners of the 2008 ISPD Global Routing Contest (i.e., FastRoute 4.0 [10] and NTUgr [15]) and two

TABLE VIII
RESULTS OF FASTROUTE 4.0, GRIP, NCTU-R, NTUGR, NTHU-ROUTE, AND NTHU-ROUTE 2.0 ON ISPD08 BENCHMARKS

Benchmark	FastRoute 4.0			GRIP[1]		NCTU-R[2]			NTUgr			NTHU-Route[3]			NTHU-Route 2.0		
	TOF	TWL (e5)	CPU (min)	TOF	TWL (e5)	TOF	TWL (e5)	CPU (min)	TOF	TWL (e5)	CPU (min)	TOF	TWL (e5)	CPU (min)	TOF	TWL (e5)	CPU (min)
adaptec1	0	54.6	3.3	–	–	0	53.5	4.1	0	56.0	4.3	0	54.2	79.9	0	53.4	5.2
adaptec2	0	52.8	0.9	–	–	0	51.7	1.4	0	53.6	1.3	0	52.8	12.6	0	52.3	1.2
adaptec3	0	132.1	3.2	–	–	0	130.4	4.9	0	134.1	4.2	0	132.9	47.9	0	130.5	5.3
adaptec4	0	122.5	0.6	–	–	0	120.7	2.2	0	123	1.8	0	122.7	6.9	0	121.7	1.4
adaptec5	0	156.7	7.7	–	–	0	154.7	9.6	0	160.1	12.3	0	157.3	215.9	0	155.4	12.2
bigblue1	0	57.8	6.1	0	53.7	0	56.6	6.7	0	58.5	10.9	0	56.2	120.7	0	56.0	7.2
bigblue2	0	93.5	8.8	0	86.0	0	89.4	13.3	0	93.2	195.5	198	90.2	48.6	0	90.6	5.7
bigblue3	0	130.7	2.1	0	126.2	0	129.7	4.2	0	134.6	5.7	18	131.6	58.1	0	130.7	3.3
bigblue4	164	251.2	48.3	186	122.0	164	224.0	–	188	239.6	315.3	–	–	–	150	230.9	60.4
newblue1	0	46.9	4.1	–	–	0	46.0	3.9	6	48.5	813.5	352	46.2	35.5	0	46.5	3.7
newblue2	0	76.4	0.4	–	–	0	74.9	0.9	0	76.7	1.5	0	76.3	3.0	0	75.7	0.7
newblue3	31 642	107.5	104.5	–	–	31 690	104.3	169.2	31 106	165.5	635.5	31 800	107.0	303.1	31 390	107.0	72.1
newblue4	160	138.1	26.9	152	124.3	142	126.9	55.5	142	138.9	863.5	448	129.1	108.7	138	130.5	60.1
newblue5	0	233.8	9.2	0	222.8	0	231.9	39.7	0	239.0	24.7	–	–	–	0	230.0	10.4
newblue6	0	180.8	12.6	0	176.9	0	176.9	12.2	0	185.5	13.5	0	178.0	214.4	0	176.9	10.4
newblue7	54	359.8	142	74	335.7	114	338.6	–	310	365.1	1109.1	–	–	–	54	355.1	102.4
comp.	–	1.01	0.87	–	0.97	–	1.00	1.53	–	1.03	19.22	–	1.01	13.79	–	1.00	1.00

The wirelength comparisons only involve the overflow-free benchmarks.
[1]In GRIP's website [28], the authors mention that their results in [12] are incorrect, and therefore we quote the corrected results from [28]. Additional, the results on the benchmarks that are original from ISPD07 6-layer benchmarks are not available.
[2]The runtimes on bigblue4 and newblue7 are not available due to out-of-memory and their TOF and TWL results are quoted from [14].
[3]The results on bigblue4, newblue5, and newblue7 are not available due to out-of-memory.

notable routers (i.e., GRIP [12] and NCTU-R [14]) on the ISPD08 benchmarks. Table VIII lists and compares the experimental results of these routers. The TWL is the total length of wire segments plus the amount of vias. Again, the TWL comparisons only involve overflow-free benchmarks. For FastRoute 4.0, NTUgr, and NCTU-R, we conducted their experiments on our own machine and report the collected TOF, TWL, and CPU results. Note that the source code of GRIP is not available, so its TOF and TWL results are quoted from [28].

As can be seen from Table VIII, NTHU-Route 2.0 generated overflow-free solutions for 12 of 16 benchmarks. It also reached the lowest TOF for three cases: bigblue4, newblue4, and newblue7. For those benchmarks that can be solved with zero overflow, NTHU-Route 2.0 achieved shorter wirelength than FastRoute 4.0 and NTUgr. Moreover, the average wirelength of our router is as good as that of NCTU-R. When compared with GRIP, NTHU-Route 2.0 was able to generate comparable average wirelength. FastRoute was the fastest router, but our router could achieve better solution quality with comparable runtime. NTHU-Route 2.0 also ran faster than NTUgr and NCTU-R.

When comparing NTHU-Route 2.0 with NTHU-Route, NTHU-Route 2.0 generated overflow-free solutions for three more benchmarks. On the other hand, NTHU-Route failed to produce solutions for bigblue4, newblue5, and newblue7 due to the out-of-memory problem. On average, NTHU-Route 2.0 achieved 0.94% shorter wirelength for the overflow-free benchmarks and was 13.76× faster than NTHU-Route.

Lower bound of the total overflow on newblue7: It is worth noting that NTHU-Route 2.0 reached the lower bound of the total overflow on newblue7. The reason is as follows: After projecting this benchmark onto a 2-D plane, we found a region—whose lower-left and upper-right corners are at (222, 246) and (228, 250), respectively—has totally 2522 nets that have to escape from it and provides only 4990 tracks totally on its boundary. According to the rule of the 2008 ISPD Global Routing Contest, one wire segment occupies two routing tracks, and therefore the total overflow for newblue7 is at least $|4990 - 2522 \times 2| = 54$, which is exactly the total overflow value achieved by NTHU-Route 2.0. Note that FastRoute 4.0 is the other router that also reaches the lower bound.

VI. CONCLUSION

In this paper, we have incorporated several enhancements into NTHU-Route to get NTHU-Route 2.0. The experimental results show that the proposed history based cost function successfully makes NTHU-Route converge solutions more quickly. They also show the proposed ordering methods for rip-up and reroute, and congested region identification are able to help NTHU-Route shorten total wirelength. Furthermore, the two runtime reduction techniques can be adopted in order to develop a high-performance global router. Finally, the experimental results show that NTHU-Route 2.0 achieves very good solution quality and is efficient and stable for ISPD98, ISPD07, and ISPD08 benchmarks.

There are two possible future works. One future work is to develop a multi-threaded global router, because the problem size of global routing keeps growing as well as the development trend in CPU has been switched from raising frequency to integrating many cores. GRIP [12] has proposed a parallel routing algorithm, but it still runs very slowly and therefore has much room for improvement. In the meantime, considering the thermal distribution issue in the global routing stage is also worthy for studying, because temperature has become an even more important problem in modern VLSI designs than before.

REFERENCES

[1] J.-R. Gao, P.-C. Wu, and T.-C. Wang, "A new global router for modern designs," in *Proc. Asia South Pacific Des. Autom. Conf.*, 2008, pp. 232–237.

[2] C. Y. Lee, "An algorithm for path connection and its application," *IRE Trans. Electron. Comput.*, vol. EC-10, no. 3, pp. 346–365, 1961.

[3] R. Kastner, E. Bozorgzadeh, and M. Sarrafzadeh, "Pattern routing: Use and theory for increasing predictability and avoiding coupling," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 21, no. 7, pp. 777–790, Jul. 2002.

[4] C. Albrecht, "Global routing by new approximation algorithms for multicommodity flow," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 20, no. 5, pp. 622–632, May 2001.

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

[6] *ISPD 2008 Global Routing Contest* [Online]. Available: http://www.sigda.org/ispd2008/contests/ispd08rc.html

[7] M. M. Ozdal and M. D. F. Wong, "Archer: A history-based global routing algorithm," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 28, no. 4, pp. 528–540, Apr. 2009.

[8] M. Cho, K. Lu, K. Yuan, and D. Z. Pan, "Boxrouter 2.0: A hybrid and robust global router with layer assignment for routability," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 14, no. 2, pp. 1–21, 2009.

[9] Y. Zhang, Y. Xu, and C. Chu, "Fastroute 3.0: A fast and high quality global router based on virtual capacity," in *Proc. Int. Conf. Comput.-Aided Design*, 2008, pp. 344–349.

[10] Y. Xu, Y. Zhang, and C. Chu, "Fastroute 4.0: Global router with efficient via minimization," in *Proc. Asia South Pacific Des. Autom. Conf.*, 2009, pp. 576–581.

[11] J. A. Roy and I. L. Markov, "High-performance routing at the nanometer scale," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 27, no. 6, pp. 1066–1077, Jun. 2008.

[12] T.-H. Wu, A. Davoodi, and J. T. Linderoth, "GRIP: Scalable 3D global routing using integer programming," in *Proc. Des. Autom. Conf.*, 2009, pp. 320–325.

[13] M. D. Moffitt, "Maizerouter: Engineering an effective global router," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 27, no. 11, pp. 2017–2026, Nov. 2008.

[14] K.-R. Dai, W.-H. Liu, and Y.-L. Li, "Efficient simulated evolution based rerouting and congestion-relaxed layer assignment on 3-D global routing," in *Proc. Asia South Pacific Des. Autom. Conf.*, 2009, pp. 570–575.

[15] H.-Y. Chen, C.-H. Hsu, and Y.-W. Chang, "High-performance global routing with fast overflow reduction," in *Proc. Asia South Pacific Des. Autom. Conf.*, 2009, pp. 582–587.

[16] G. W. Clow, "A global routing algorithm for general cells," in *Proc. Des. Autom. Conf.*, 1984, pp. 45–51.

[17] M. Pan and C. Chu, "Fastroute 2.0: A high-quality and efficient global router," in *Proc. Asia South Pacific Des. Autom. Conf.*, 2007, pp. 250–255.

[18] R. T. Hadsell and P. H. Madden, "Improved global routing through congestion estimation," in *Proc. Des. Autom. Conf.*, 2003, pp. 28–31.

[19] Z. Cao, T. Jing, J. Xiong, Y. Hu, Z. Feng, L. He, and X. Hong, "Fashion: A fast and accurate solution to global routing problem," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 27, no. 4, pp. 726–737, Apr. 2008.

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

[21] M. Cho and D. Z. Pan, "Boxrouter: A new global router based on box expansion and progressive ILP," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 26, no. 12, pp. 2130–2143, Dec. 2007.

[22] L. McMurchie and C. Ebeling, "Pathfinder: A negotiation-based performance-driven router for FPGAs," in *Proc. Int. Symp. Field-Programmable Gate Arrays*, 1995, pp. 111–117.

[23] T.-H. Lee and T.-C. Wang, "Congestion-constrained layer assignment for via minimization in global routing," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 27, no. 9, pp. 1643–1656, Sep. 2008.

[24] C.-H. Hsu, H.-Y. Chen, and Y.-W. Chang, "Multilayer global routing with via and wire capacity considerations," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 29, no. 5, pp. 685–696, May 2010.

[25] C. Chu and Y.-C. Wong, "FLUTE: Fast lookup table based rectilinear Steiner minimal tree algorithm for VLSI design," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 27, no. 1, pp. 70–83, Jan. 2008.

[26] *Gompertz Curve* [Online]. Available: http://en.wikipedia.org/wiki/Gompertz_curve

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

[28] GRIP [Online]. Available: http://wiscad.ece.wisc.edu/gr

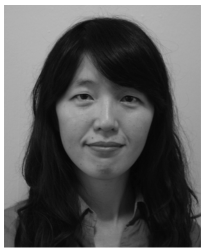
Yen-Jung Chang received the B.S. degree in computer science from National Chiao Tung University, Hsinchu, Taiwan, in 2006, and the M.S. degree in computer science from National Tsing Hua University, Hsinchu, in 2008. He is currently pursuing the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Texas, Austin.

He is a winner of the 2008 ACM ISPD Global Routing Contest (first place). He received a Master's Thesis Award from the Taiwan IC Design Society and the SpringSoft EDA Award from SpringSoft.

Yu-Ting Lee received the B.S. degree in computer science and engineering from Tatung University, Taipei, Taiwan, in 2007, and the M.S. degree in computer science from National Tsing Hua University, Hsinchu, Taiwan, in 2009.

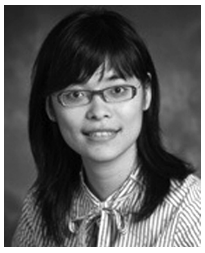
He is currently an Engineer with ChungHwa Telecommunication Laboratories, Yang-Mei, Taoyuan, Taiwan.

Mr. Lee is a winner of the 2008 ACM ISPD Global Routing Contest (first place). He received the SpringSoft EDA Award from SpringSoft.



Jhih-Rong Gao received the B.S. and M.S. degrees in computer science from National Tsing Hua University, Hsinchu, Taiwan, in 2005 and 2007, respectively. She is currently pursuing the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Texas, Austin.

She was a Research and Development Engineer with Synopsys, Inc., Taipei, Taiwan, from 2007 to 2009.



Pei-Ci Wu received the B.S. and M.S. degrees in computer science from National Tsing Hua University, Hsinchu, Taiwan, in 2005 and 2007, respectively. She is currently pursuing the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana.

From 2007 to 2009, she was a Research and Development Engineer with Synopsys, Inc., Taipei, Taiwan.

Ting-Chi Wang received the B.S. degree in computer science and information engineering from National Taiwan University, Taipei, Taiwan, and the M.S. and Ph.D. degrees in computer sciences from the University of Texas, Austin.

He is currently a Professor with the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan. His current research interests include VLSI design automation.

Dr. Wang was the recipient of the Best Paper Award from the 2006 ASP-DAC for his work on redundant via insertion. He supervised a team to win first place at the 2008 ISPD Global Routing Contest. He has served on the technical program committees of several conferences, including ASP-DAC, DATE, FPL, FPT, ICCAD, ISPD, and SASIMI.