A Fast Maze-Free Routing Congestion Estimator With Hybrid Unilateral Monotonic Routing

Wen-Hao Liu[12], Yih-Lang Li[1], and Cheng-Kok Koh[2]
[1]Department of Computer Science, National Chiao-Tung University, Hsin-Chu, Taiwan
[2]School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, USA
dnoldnol@gmail.com, ylli@cs.nctu.edu.tw, chengkok@purdue.edu

Abstract – **Considering routability issue in the early stages of VLSI design flow can avoid generating an unroutable design. Several recent routablity-driven placers [8-11] adopt a built-in global router to estimate routing congestion. While the routability of the placement solution improves, the performance of these placers degrades. Many of these built-in global router and state-of-the-art academic global routers use maze routing to seek a detoured path. Although very effective, maze routing is relatively slower than other routing algorithms, such as pattern routing and monotonic routing algorithms. This work presents two efficient routing algorithms, called unilateral monotonic routing and hybrid unilateral monotonic routing, to replace maze routing and to realize a highly fast maze-free global router that is suited to act as a built-in routing congestion estimator for placers. Experimental results indicate that RCE achieves similar routing quality when compared with [20], as well as an over 20-fold runtime speedup in large benchmarks.**

1. INTRODUCTION

Routability is of primary concern in nanometer-scale design. Considering the routability issue in placement stage can avoid generating an unroutable design. Two strategies are generally adopted by routability-driven placement to estimate the congested regions (hot spots). First, the probabilistic method estimates the routing congestion of a region by using the pin density and the nets' bounding box or stenier tree [1,2]. Although fast, this method typically fails to capture actual routing behavior, and therefore has low estimation accuracy. The second congestion estimation strategy performs global routing to analyze routing congestion [3]. The latter method can identify more precisely the congestion information. However, such an approach is markedly slower than the former one. Among the modern routability-driven placers, Ripple [4], NTUplace [5] and the placers in [6][7] used the former strategy, whereas simPLR [8], IPR [9], CRISP [10] and GRplace [11] adopted the latter one. Clearly, it is inevitable to trade-off routing quality for better run-time performance when these built-in global routers are concerned.

Maze routing with A* search scheme is the indispensable kernel algorithm of state-of-the-art global routers [12-21]. For hard-to-route benchmarks, these routers attempt to eliminate overflows by iteratively ripping up and rerouting overflowed nets by using maze routing. However, maze routing is slower than other routing algorithms, such as pattern routing and monotonic routing algorithms. Several works have attempted to reduce runtime by developing alternative routing algorithms in order to lower the frequency of invoking maze routing. For instance, Archer [14] developed the U-shaped pattern routing algorithm; NTUgr [16] presented the escaping routing algorithm; and FastRoute4.0 [17] developed the 3-bend routing algorithm. These routing algorithms run faster than maze routing within a quite limited solution space. Thus, in these global routers, maze routing continues to be the last-gasp approach to identify better routes. Consequently, maze routing still consumes the majority of the runtime in the entire routing flow.

This work develops an extremely fast global router without invoking maze routing to achieve competitive routing quality as an ideal built-in routing congestion estimator (RCE) for placers.

(a) This work presents two efficient routing algorithms, called unilateral monotonic routing and hybrid unilateral monotonic (HUM) routing. HUM routing can identify a better routing path than U-shaped pattern routing, 3-bend routing, and escaping routing. Moreover, the time complexity of HUM routing is the same as those of these three approaches, linear in terms of the size of the routing graph.

(b) Many routers adopt bounding boxes to limit the searching region of routing. Consequently, the bounding box size affects the routing quality and runtime. This work presents an efficient congestion-aware bounding box expansion scheme. With this scheme, the proposed router can improve runtime by 50% than without this scheme.

(c) The proposed router relies on HUM routing to eliminate overflows without invoking maze routing. Experimental results indicate that the proposed router achieves a routing quality similar to that of a conventional maze-routing-based router [20]. Moreover, the run-times of the proposed router are up to 26 times faster than those of [20] on large benchmarks.

The rest of this paper is organized as follows. Section 2 introduces the global routing problem and the research objective. Section 3 then presents the proposed unilateral monotonic routing, HUM routing algorithms and a congestion-aware bounding box expansion scheme. Section 4 displays the design flow of the proposed global router. Section 5 summarizes the experimental results. Conclusions are finally drawn in Section 6.

2. PROBLEM DESCRIPTION

Global routing is formulated as the routing problem on a grid graph $G(V, E)$, where V denotes the set of grid cells, and E refers to the set of grid edges. The layout is typically partitioned into an array of global cells (G-cells). Each grid edge is termed by the proximity of the related G-cells to its two end nodes. The capacity $c(e)$ of a grid edge e indicates the number of routing tracks that can legally cross the abutting boundary. The number of wires that pass through grid edge e is called the demand of the grid edge $d(e)$. The overflow of a grid edge e is defined as

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
IEEE/ACM International Conference on Computer-Aided Design (ICCAD) 2012, November 5-8, 2012, San Jose, California, USA
Copyright © 2012 ACM 978-1-4503-1573-9/12/11... $15.00
* This work was partially supported by the National Science Council of Taiwan by Grant Nos. NSC 101-2220-E-009 -043.

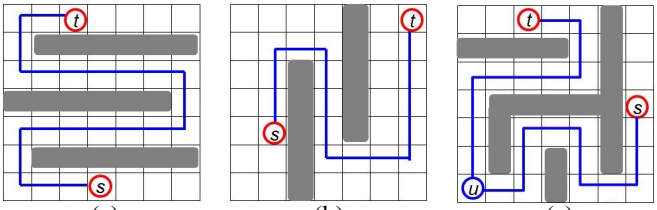

Fig. 1. (a) Vertically monotonic routing path; (b) horizontally monotonic routing path; (c) routing path combining vertically and horizontally monotonic routing.

follows. The total overflow is the sum of overflows on all grid edges of E.

$$overflow(e) = \begin{cases} (d(e) - c(e)) * (w_L + s_L), if\ d(e) > c(e) \\ 0 \qquad\qquad\qquad , otherwise \end{cases} \quad (1)$$

where w_L and s_L respectively denote the minimum wire width and wire spacing at layer L where e belongs. In modern designs, higher layers have larger wire width and wire spacing.

Conventionally, overflow minimization has a higher priority than runtime improvement for global routing that offers a global path to guide the detailed routing of each net. However, when global router plays the role as a congestion estimator, the runtime issue become more critical because the estimator have to report the congestion information to placers in a limited time budget (e.g. around 1~5 min). Accordingly, this work focuses on comply with the limited time budget for the global routing in all benchmarks.

3. UNILATERAL MONOTONIC ROUTING

Although capable of identifying a detour-free path efficiently, monotonic routing fails to replace maze routing when a detoured path is required to avoid obstacles or congested regions. A detour is viewed as a move away from the target. To approach the behavior of maze routing, this work develops an extremely fast routing algorithm, called *unilateral monotonic routing*, capable of seeking a detoured path and running in the same time complexity as that of monotonic routing. Unilateral monotonic routing identifies a least-cost routing path within a limited region using minimal horizontal or vertical distance. Two unilateral monotonic routing types are defined as follows.

Definition. *Horizontally/Vertically monotonic* (*HM/VM*) *routing* identifies the least-cost routing path from the source to the target using minimal horizontal/vertical distance.

For a HM/VM routing path, a detour occurs only in vertical/horizontal move. Figures 1(a) and 1(b) illustrate a VM routing path and a HM routing path, respectively, in which the gray rectangles represent congested regions. Although the solution space of HM or VM routing is less than that of maze routing, alternatively invoking HM and VM routings can increase the solution space significantly. Figure 1(c) depicts an example of invoking successive HM and VM routings, the path in Fig. 1(c) consists of a HM routing path from s to an internal node u and a VM routing path from t to u.

3.1. Unilateral Monotonic Routing

Without a loss of generality, the proposed unilateral monotonic routing is introduced by using an example of VM routing shown in Fig. 2. At the beginning of VM routing, the congestion map (Fig. 2(a)) is formulated into the global routing model (Fig. 2(b)), and the congestions is formulated into the

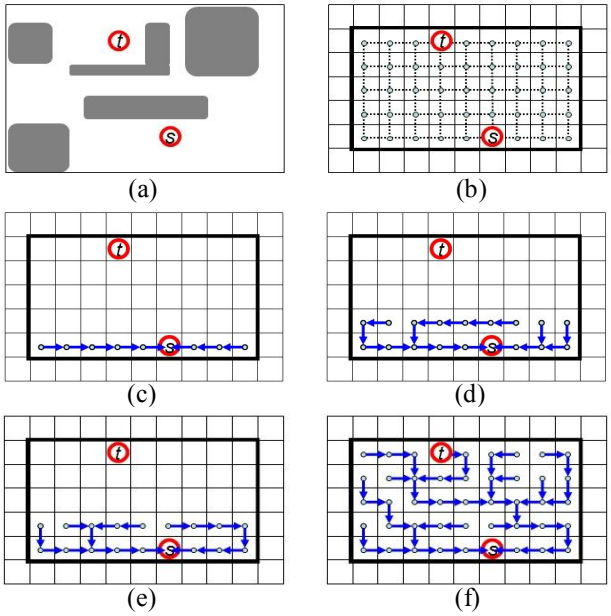

Fig. 2. Example of VM routing. (a) a congestion map; (b) the routing model of (a), the dotted lines denote the grid edges; (c) The predecessor of each node u in the row of y-coordinate y_1 after $d(u)$ is obtained, the arrow of each node denotes its predecessor; (d) the predecessor of each node u in the row of y-coordinate y_1+1 after $lc_{lb}(u)$ is obtained; (e) the predecessor of each node u in the row of y-coordinate y_1+1 after $d(u)$ is obtained; (f) the routing result of VM routing.

routing cost on each grid edge, then a window is given to enclose the source and target with the height of vertical distance between the source and target and the width of horizontal distance larger than that between the source and target (Fig. 2(b)). The window size determines the runtime and the routing quality of the unilateral monotonic routing. Section 3.3 introduces how to set the window size. Figure 3 shows the pseudo code of the VM routing algorithm, in which source s and target t are located at (x_1, y_1) and (x_2, y_2) respectively; $B.l$ and $B.r$ represent the left and right borders of windows B, respectively; $cost(v, u)$ denotes the routing cost of grid edge $e(v, u)$; $d(u)$ refers to the least cost of the VM routing path within B from s to u; and $\pi(u)$ is the predecessor of u. The algorithm in Fig. 3 consists of two stages. The first stage calculates the $d(u)$ value of each node of the bottom row, i.e. the row where the start node belongs. The second stage computes the $d(u)$ values of nodes in all rows, except for the bottom row, from the row above the bottom row to the top one. The first stage is a simple sequential examination initiating from the start node towards the left and right boundaries of B, and then the second stage processes all rows except for the bottom one sequentially and upwards. In the second stage, based on the dynamic programming algorithm, a two-phase flow is developed and the $d(u)$ value of each node is computed row by row. The first phase determines the least-cost VM path to connect every node from the start node at the left or bottom side, while the second phase determines the least-cost VM path to connect every node from the start node at the right side. By the two-phase operation, the least-cost VM path to reach every node within B from the start node is then identified.

Upon commencement of the second stage, the $d(v)$ value of each node $v \in (i, y_1)$ for $B.l \le i \le B.r$ is identified. By assuming that

Algorithm Vertically monotonic routing
Input: source $s(x_1, y_1)$, target $t(x_2, y_2)$, box B, cost array d
1. $d(s)= 0, \pi(s)= null;$
2. **for** $x= x_1$-1 **to** $B.l$
3. $u=(x, y_1)$, $v=(x+1, y_1)$;
4. $d(u)= d(v)+ cost(v, u)$, $\pi(u)= v;$
5. **end for**
6. **for** $x= x_1+1$ **to** $B.r$
7. $u=(x, y_1)$, $v=(x-1, y_1)$;
8. $d(u)= d(v)+ cost(v, u)$, $\pi(u)= v;$
9. **end for**
10. **for** $y= y_1+1$ **to** y_2
11. $u=(B.l , y)$, $v=(B.l, y-1)$
12. $lc_{lb}(u)= d(v)+ cost(v, u)$, $\pi(u)= v$
13. **for** $x= B.l +1$ **to** $B.r$
14. $u=(x, y)$, $v_1=(x-1, y)$, $v_2=(x, y-1)$;
15. **if** $lc_{lb}(v_1) + cost(v_1, u) < d(v_2) + cost(v_2, u)$
16. $lc_{lb}(u)= lc_{lb}(v_1)+ cost(v_1, u)$, $\pi(u)= v_1$
17. **else**
18. $lc_{lb}(u)= d(v_2)+ cost(v_2, u)$, $\pi(u)= v_2$
19. **end for**
20. $u=(B.r , y)$, $d(u) = lc_{lb}(u)$
21. **for** $x= B.r$ -1 **to** $B.l$
22. $u=(x, y)$, $v_3=(x+1, y)$, $d(u) = lc_{lb}(u)$
23. **if** $d(v_3) + cost(v_3, u) < d(u)$
24. $d(u) = d(v_3) + cost(v_3, u)$, $\pi(u)= v_3$
25. **end for**
26. **end for**

Fig. 3. The proposed vertically monotonic routing algorithm.

node u is located at (i, y_1+1), $lc_{lb}(u)$ is the least of all costs of the VM routing paths from s to u when the predecessor of u is at its left or bottom side, and can be obtained via the following equation in the first phase,

$$lc_{lb}(u) = \begin{cases} d(v_2) + \text{cost}(v_2, u) & \text{if } u \text{ is on the left boundary of } B \\ \min(lc_{lb}(v_1) + \text{cost}(v_1, u), d(v_2) + \text{cost}(v_2, u)), & \text{otherwise} \end{cases} \quad (2)$$

where v_1 and v_2 represent the left and bottom adjacent nodes of u, respectively. During the second phase, the least cost of VM paths to reach node u from the start node at the right side (denoted by $lc_r(u)$) and then the least-cost VM path to reach node u from the start node are determined sequentially by the following equation.

$$d(u) = \begin{cases} lc_{lb}(u) & \text{if } u \text{ is on the right boundary of } B \\ \min(lc_{lb}(u), lc_r(u) = d(v_3) + \text{cost}(v_3, u)), & \text{otherwise} \end{cases} \quad (3)$$

where v_3 represents the right adjacent node of u. If u is on the right boundary of B, the predecessor of u must be on the left side or on the bottom side of u; thus $d(u)$ equals $lc_{lb}(u)$. While each node u is examined sequentially from right to left in the second phase, the least-cost VM path to reach each node from the start node is then determined by Eq. (3).

In Fig. 3, lines 1 to 9 calculate the least cost of the VM paths from s to each node $v \in (i, y_1)$ for $B.l \le i \le B.r$ (Fig. 2(c)). Next, based on the dynamic programming method, the least-cost VM path from s to each node of each row within B is identified from the row of y-coordinate y_1+1 to the row of y-coordinate y_2, (lines 10 to 26), where lines 11 to 19 identify the values of $lc_{lb}(u)$ by Eq. (2) and lines 20 to 25 identify the values of $d(u)$ by Eq. (3). Figure 2(d) shows the predecessor of each node u in the s-to-u path of $lc_{lb}(u)$ in the row of y-coordinate y_1+1. Meanwhile, Fig. 2(e) shows the predecessor of each node u in the s-to-u path of $d(u)$ in the row of y-coordinate y_1+1. Upon completion of the VM routing, each node within B has a least-cost VM path to reach s along its predecessor (Fig. 2(f)). Therefore, the least-cost VM path from s to t is also identified. Obviously, the time complexity of unilateral monotonic routing algorithm is $O(|B|)$ where $|B|$ represents the area size of B.

Algorithm Hybrid Unilateral Monotonic Routing
Input: source node s, target node t, bounding box B
1. Initialize cost array Ary_{vs}, Ary_{vs}, Ary_{hs}, Ary_{ht}
2. //Find the paths from each node in B to s
3. Vertically_Monotonic_Routing(s, $B.bl$, B, Ary_{vs})
4. Vertically_Monotonic_Routing(s, $B.tr$, B, Ary_{vs})
5. Horizontally_ Monotonic_Routing(s, $B.bl$, B, Ary_{hs})
6. Horizontally_Monotonic_Routing(s, $B.tr$, B, Ary_{hs})
7. //Find the paths from each node in B to t
8. Vertically_Monotonic_Routing(t, $B.bl$, B, Ary_{vt})
9. Vertically_Monotonic_Routing(t, $B.tr$, B, Ary_{vt})
10. Horizontally_ Monotonic_Routing(t, $B.bl$, B, Ary_{ht})
11. Horizontally_ Monotonic_Routing(t, $B.tr$, B, Ary_{ht})
12. **foreach** node u in B
13. $mrc(u)$=min($Ary_{hs}(u)$, $Ary_{vs}(u)$)+min($Ary_{ht}(u)$, $Ary_{vt}(u)$)
14. **end foreach**
15. Select the node u in B with the least $mrc(u)$, and then trace back from this node to s and t.

Fig. 4. The pseudo code of the proposed hybrid unilateral monotonic routing algorithm.

3.2. Hybrid Unilateral Monotonic Routing

Compared to maze routing, unilateral monotonic routing still offers a limited solution space to solve overflows. This section introduces a hybrid unilateral monotonic (HUM) routing algorithm to search for larger solution space than unilateral monotonic routing offers. The HUM routing concept assumes that each node within B is an intermediate point connecting the start and target nodes. The final path consists of two paths, i.e. the path linking the start node with the intermediate point and the path linking the intermediate point with the target node. Each path can be formed by unilateral monotonic routing.

Since a path can be formed by VM or HM routing, four combinations are available to form a HUM routing path. By assuming that bounding box B encloses nodes u and v, $VMP_{B(u,v)}$ and $HMP_{B(u,v)}$ represent a VM routing path and a HM routing path connecting u with v within B, respectively. A HUM routing path connecting s with t belongs to one of the following four path types: ($VMP_{B(s,u)}$, $VMP_{B(u,t)}$), ($VMP_{B(s,u)}$, $HMP_{B(u,t)}$), ($HMP_{B(s,u)}$, $VMP_{B(u,t)}$) and ($HMP_{B(s,u)}$, $HMP_{B(u,t)}$) for each node u within B. Whereas ($VMP_{B(s,u)}$, $VMP_{B(u,t)}$) denotes a path concatenation operation that combines two unilateral monotonic paths of one or two type to form a HUM path linking start and end nodes. Figure 4 shows the proposed HUM routing algorithm. The least costs of $VMP_{B(s,u)}$, $VMP_{B(t,u)}$, $HMP_{B(s,u)}$ and $HMP_{B(t,u)}$ of each node are stored in the arrays Ary_{vs}, Ary_{vt}, Ary_{hs} and Ary_{ht}, respectively, while $B.bl$ and $B.tr$ represent the nodes at the bottom-left and top-right corners of B, respectively. Lines 3 – 6 regard s as the start node, and $B.bl$ and $B.tr$ as pseudo targets. Then, lines 3 and 4 invoking VM routing from s to the pseudo targets obtain VM routing paths from s to every node within B; lines 5 and 6 invoking HM routing from s to the pseudo targets obtain HM routing paths from s to every node within B. Similarly, lines 8 – 11 regard t as the start node and $B.bl$ and $B.tr$ as the pseudo targets, and then obtain VM routing paths and HM routing paths from t to every node within B. Accordingly, lines 2 – 11 identify the value of each element in Ary_{vs}, Ary_{vt}, Ary_{hs} and Ary_{ht}. Thereafter, the least costs of $VMP_{B(s,u)}$, $VMP_{B(t,u)}$, $HMP_{B(s,u)}$ and

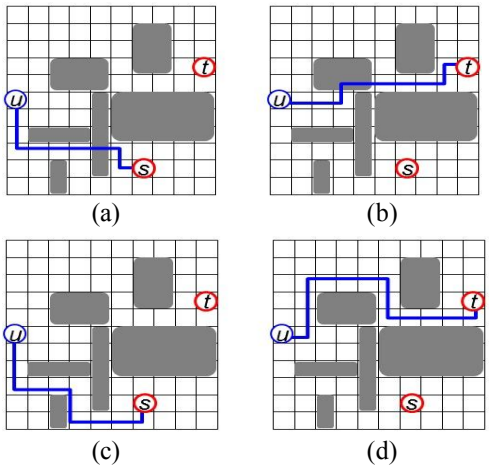

Fig. 5. Four path types within B with congested regions (gray rectangles). (a) $VMP_{B(s,u)}$, (b) $VMP_{B(t,u)}$, (c) $HMP_{B(s,u)}$, and (d) $HMP_{B(t,u)}$.

$HMP_{B(t,u)}$ for each node u within B are obtained (Fig. 5(a)-(d)). The algorithm then selects the least-cost HUM routing path among the candidates of four path types (lines 12 – 15).

The time complexities of three parts, lines 2-11, lines 12-14 and line 15 are all O(|B|). Correspondingly, the time complexity of HUM routing algorithm is still O(|B|), which is much faster than that of maze routing with A* search scheme (O(|B|log|B|)). Figure 6 compares the proposed HUM routing with 3-bend routing and escaping routing, indicating that the time complexities of 3-bend routing and escaping routing are also O(|B|). Figures 6(a) and 6(b) summarize the routing results of 3-bend routing and escaping routing with two overflows and with an overflow, respectively. In contrast, the proposed HUM routing algorithm can identify an overflow-free path (Fig. 6(c)) with the pattern ($HMP_{B(s,u)}$, $HMP_{B(u,t)}$). Notably, even if HUM routing cannot identify an overflow-free path, it can always identify a least-cost HUM path which must cost less than or equal to that of 3-bend routing and escaping routing. Because, the solution space of HUM routing totally covers and is much larger than that of 3-bend routing and escaping routing.

Assume that most of overflowed grid edges within B are aligned in a row similar to the congestion map in Fig. 1(a), the least costs of $HMP_{B(s,u)}$ and $HMP_{B(u,t)}$ are likely larger than the least costs of $VMP_{B(s,u)}$ and $VMP_{B(u,t)}$. Therefore, the operations of exploring $HMP_{B(s,u)}$ and $HMP_{B(u,t)}$ can be regarded as redundant and are thus omitted. Based on this observation, four HUM routing types are explored only once for every net at the first time when it is routed by HUM routing. If a net is rerouted by HUM routing in the later routing stage, only the HUM routing type that initially identified the least-cost path is invoked. By this scheme, experimental results indicate that similar routing quality and an approximately 23% decrease in runtime of HUM routing can be achieved.

3.3. Congestion-aware Bounding Box Expansion

Bounding box is widely adopted to limit the searching region of routing. In conventional global routers, the initial bounding box is slightly larger than the minimum rectangle enclosing the terminals of the routed net. The inability to identify an overflow-free path within the bounding box causes the bounding box to expand and, then, the overflowed net is rerouted again. The box expansion policy based on current congestion information has

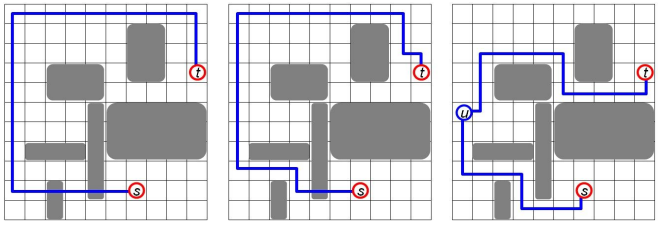

Fig. 6. (a) Routing result of 3-bend routing with two overflows; (b) routing result of escaping routing with an overflow; (c) routing result of the proposed HUM routing without overflows.

seldom been discussed in the literature. The traditional box expansion scheme tends to over-expand, subsequently increasing the runtime. For instance, Fig. 7(a) shows a routing path with a vertical overflowed edge. Traditional box expansion chooses to expand the bounding box along both x and y coordinates to resolve the overflow. However, the bounding box only needs to expand horizontally in Fig. 7(a) while the vertical expansion is unnecessary. Figure 7(b) displays the congestion map of the benchmark superblue1 after the initial routing. The red regions represent the overflowed grid edges, which normally range horizontally or vertically, implying that the situation in Fig. 7(a) occurs frequently during routing. Based on this observation, this work presents a novel congestion-aware bounding box expansion scheme to avoid over expanding.

Before rerouting a net, this work analyzes the amount of horizontal overflowed grid edges (HOEs) and vertical overflowed grid edges (VOEs) by tracing the routing path of the rerouted net. If the number of HOEs is more than that of VOEs, the bounding box expands vertically by δ units; on the contrary, the bounding box expands horizontally. If a tie occurs, the bounding box randomly chooses to expand horizontally or vertically. Single-direction expansion can restrict the sizes of bounding boxes to reduce the runtime. In our implement, the initial bounding box is set as the minimum rectangle enclosing two terminals to be routed, and δ is set to $5+30/r_i$, where r_i denotes the rip-up and rerouting times of the rerouted net. Moreover, based on the assumption that two opposite sides have different congestion states, extending the side near the congested region may be unnecessary, implying that the extension of each boundary of B should be discussed separately. The algorithm examines each boundary side of B to determine the necessity of box boundary expansion at the end of HUM routing. Without a loss of generality, the left boundary of B is used to illustrate the concept. Left boundary expansion can be regarded to have the intention to find a path $L_{s,t}$ on the left side of B; in addition, $L_{s,t}$ has a lower routing cost than that of the currently identified path $P_{s,t}$ (Fig. 7(c)). Namely, a situation in which the routing cost of $P_{s,t}$ is lower than the least cost of $L_{s,t}$ implies that the left boundary expansion is unnecessary. However, the least cost of $L_{s,t}$ is unknown because the region on the left side of B has not been explored yet. Thus, the estimated lower-bound cost of $L_{s,t}$, ec_L, is defined by the following equation to evaluate the necessity of boundary expansion. If the currently identified path $P_{s,t}$ costs less than ec_L, the left boundary remains unchanged at the next expansion of B.

$$ec_L = min_{u \in V_L, v \in V_L}(d(s,v) + d(t,u) + manh(v,u) \times \alpha) \quad (4)$$

where V_L denotes the set of grid nodes on the left boundary of B; $d(s,v)$ and $d(t,u)$ represent the least cost of the unilateral

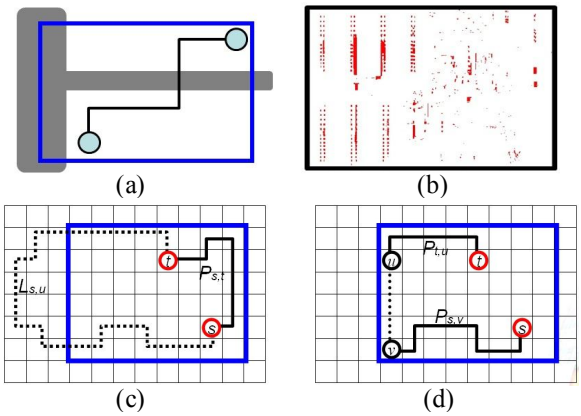

Fig. 7. (a) Routing path with an vertical overflow; (b) the overflow map of the benchmark superblue1 after the initial routing; (c) currently identified path $P_{s,t}$ and path $L_{s,t}$ that is expected to be across the left side of the bounding box; (d) the estimated lower bound cost of $L_{s,t}$ is the sum of the costs of $P_{s,v}$ and $P_{t,u}$ plus $manh(v, u)$* α.

monotonic routing paths from s to v and from t to u, respectively; $manh(v,u)$ refers to the Manhattan distance between v and u; and α is the lower-bound routing cost of a grid edge. In this work, α is set to 1. Notably, $d(s,v)$ and $d(t,u)$ are known values that have been computed by the HUM routing (Fig. 7(d)). With this, before a net n_i is rerouted, the path of n_i is first traced to obtain HOEs and VOEs. If the number of VOEs is more than that of HOEs, extending the left and right boundaries of the bounding box B of n_i is considered. If n_i is not routed by HUM routing in previous routing, the left and right boundaries of B extend immediately. Otherwise, the decision of boundary expansion is made according to the previous discussion.

4. DESIGN FLOW OF RCE

Figure 8 shows the design flow of the proposed routing congestion estimator. First, the multi-layer routing region is projected on a 2D plan and each net is decomposed into two-pin nets based on the topology of the rectilinear minimum spanning tree. An initial congestion graph is then generated by pattern routing and monotonic routing. Next, the rip-up and rerouting stage iteratively reroutes the overflowed net until an overflow-free routing result is obtained or the runtime exceeds the given time budget. In the rip-up and rerouting stage, before net n_i is rerouted, the bounding box of n_i is expanded according to the proposed congestion-aware expansion scheme. For a situation in which the width of the bounding box is equal to the x-distance between the source and the target of n_i, n_i is rerouted using HM routing. Moreover, if the height of the bounding box is equal to the y-distance between the source and the target of n_i, n_i is rerouted using VM routing. Otherwise, n_i is rerouted by HUM routing.

5. EXPERIMENTAL RESULTS

The proposed algorithms are implemented in C/C++ language on a quad-core 2.4 GHz Intel Xeon-based linux server with a 50GB memory (only a single core is used). By recently hosting a routability-driven placement contest, ISPD'11 has motivated many researchers to develop effective modern placers [4, 5, 8]. Ripple [4] and mPL11 placed first in the contest; their contest placement results are adopted here as the input benchmarks in our experiments. The proposed router is

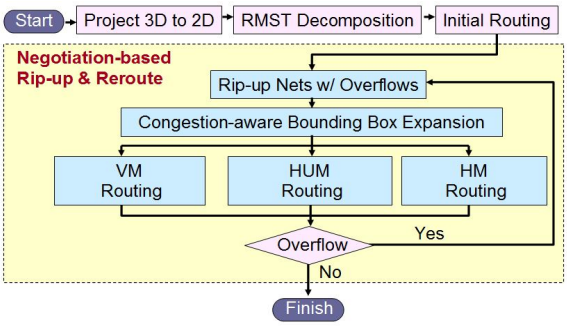

Fig. 8. Design flow of the proposed routing congestion estimator.

compared with NCTUgr2 [20] which is one of the fastest global routers. The experiments in [20] indicate that NCTUgr2 runs 1.90X, 1.77X and 18.66X faster than NTHU-Route2.0 [15], FastRoute4.1 [17], and NTUgr [16], respectively. In addition, in the old benchmarks [23], the minimum wire spacing and width are uniform and all pins locate at the lowest layer. In contrast, in new benchmarks [24] used in this work, the minimum wire spacing and width of different layers are different and pins may locate at high layers. Because most of traditional routers do not consider these new features, we cannot directly adopt them to route the new benchmarks. However, recent routers NCTUgr2 and CGRIP [3] can handle these new features, but the runtime of CGRIP is much larger than NCTUgr2. For benchmark s1_Ripple, CGRIP consuming 15 minutes still identifies an overflowed result, while NCTUgr2 identifies an overflow-free result in a minute. Owing to its robustness and efficiency, the routability-driven placement contest in DAC12 [25] adopts NCTUgr2 to be the evaluation tool. In the following experiments, NCTUgr2 and the proposed router perform on the same machine. Notably, NCTUgr2 has the parameters of via cost, wirelength optimization level, pattern routing iteration, monotonic routing iteration and post routing iteration, which are set to 1, 50, 2, 2 and 0, respectively. The setting of the rip-up and rerouting stage will be detailed in following sections. In addition, NCTUgr2 is a multi-thread parallel global router, but it only uses a single thread in the following experiments.

5.1. Comparison between NCTUgr2 and RCE

Table 1 compares NCTUgr2 and RCE in terms of the ability to eliminate overflows in a given time budget. The frameworks of NCTUgr2 and RCE are nearly same; the only difference is NCTUgr2 adopts bounded-length maze routing [20] while RCE adopts unilateral monotonic routing and HUM routing in the rip-up and rerouting stage. Each major column in Table 1 lists the total overflows of the routing results generated by NCTUgr2 and RCE within a specified time budget for the rip-up and rerouting stage. Notably, for every benchmark, the runtime excluding the rip-up and rerouting stage is always less than 40 seconds. Table 1 reveals that RCE can eliminate overflows more efficiently than NCTUgr2. Given a 30 second time constraint, RCE identifies 4 overflow-free routing results while NCTUgr2 identifies none. Given a 240 second time constraint, RCE can identify 8 routing results with overflows less than a thousand, whereas NCTUgr2 identifies only 5 routing results for less than a thousand overflows.

Table 2 compares the routing results of NCTUgr2 and RCE without a runtime limitation. Both NCTUgr2 and RCE iteratively rip-up and reroute the overflowed nets until either all overflows

TABLE 3 THE USAGE OF THE PROPOSED UNILATERAL MONOTONIC ROUTING AND HUM ROUTING ALGORITHMS

	Unila.	HUM		Unila.	HUM
s1_Ripple	15519	110	s1_mPL11	73803	3429
s2_Ripple	127654	43237	s2_mPL11	227191	286871
s4_Ripple	71128	13438	s4_mPL11	95966	86565
s5_Ripple	72995	25694	s5_mPL11	64080	178135
s10_Ripple	107671	89837	s10_mPL11	261525	104806
s12_Ripple	375051	369483	s12_mPL11	54984	1103318
s15_Ripple	79893	32905	s15_mPL11	167487	17001
s18_Ripple	62168	175323	s18_mPL11	101220	7846

are eliminated or overflows cannot be reduced by more than 3% in 5 consecutive iterations. In Table 2, MO, TOF, WL and RCPU represent the maximum overflow, total overflows, total wirelength (including wires and vias) and the runtime (seconds) of the rip-up and rerouting stage, respectively. Table 2 treats NCTUgr2 as a base line; $Ratio_{each}$ averages the ratio of each entry, while $Ratio_{sum}$ sums up the numbers at each column and then then get the ratio of the sum. To reduce the runtime, NCTUgr2 and RCE both adopt the greedy layer assignment method, explaining why the via count is not optimized. According to our experiments, the total wirelength can be further reduced by 7%~15% if the layer assignment algorithms [26] or [27] are adopted. Table 2 indicates that, although the wirelength and overflows of RCE are slightly worse than that of NCTUgr2, RCE can achieve a 2.75X to 26.83X higher speedup than that of NCTUgr2. Moreover, if NCTUgr2 obtains overflow-free results for a benchmark, RCE also can achieve an overflow-free result for the same benchmark. Therefore, we believe that RCE can determine placement results quickly regardless of whether or not it is routable, thus making is very suitable as a fast congestion estimator embedded into routability-driven placers.

5.2. Effectiveness of the Proposed Algorithms

Table 3 details the amount of usages of unilateral monotonic routing and HUM routing in Table 2. The column "Unila." represents the number of two-pin nets having been routed only by unilateral monotonic routing in the rip-up and rerouting stage. The column "HUM" represents the number of two-pin nets having been routed by HUM routing. Table 3 reveals that most overflows in each benchmark can be solved by unilateral monotonic routing. Because overflows in each benchmark normally range horizontally or vertically, the proposed unilateral monotonic routing can remove them efficiently. This table also indicates that most nets do not need complicated detours during routing, so maze routing is unnecessary for these nets.

Table 4 shows the overflow-free routing results of RCE using traditional bounding box expansion scheme. Compared to the results in Table 2, RCE using the traditional scheme causes roughly twice the runtime than that using the congestion-aware bounding box expansion scheme. This difference is owing to that the traditional scheme may over expand bounding boxes. However, RCE using traditional scheme can obtain the results with a slightly lower wirelength because large bounding boxes can reduce redundant detours, as demonstrated in [20].

6. CONCLUSIONS

This work presents two efficient routing algorithms, unilateral monotonic routing and HUM routing. Unilateral monotonic routing can horizontally or vertically detour around the congestion regions with the same time complexity of

TABLE 4 COMPARISON BETWEEN CONGESTION-AWARE BOX EXPANSION SCHEME AND TRADTIONAL SCHEME

Benchmarks	The proposed router using traditional bounding box expansion scheme			
	WL	RCPU	WL imp.%	RCPU ratio
s1_Ripple	161.62	9	0.00%	1.11
s5_Ripple	201.42	234	0.29%	1.72
s12_Ripple	277.67	1890	0.65%	2.9
s15_Ripple	208.6	173	0.10%	1.87
s1_mPL11	168.84	27	0.11%	2.92
s15_mPL11	196.40	68	0.10%	2.05
s18_mPL11	110.16	46	0.51%	1.49
Average			0.25%	2.01

monotonic routing. HUM routing can identify an overflow-free path on a challenged congestion map, thus making it significantly faster than maze routing with A* search scheme. Moreover, a congestion-aware bounding box expansion scheme is developed to avoid over expanding bounding boxes. Based on these contributions, a maze-free router is developed, capable of achieving 2.75X to 26.83X speedup than NCTUgr2. We believe that the proposed maze-free router is highly promising for use as a fast routing congestion estimator embedded into routability-driven placers.

REFERENCES

[1] J. Lou, S. Takur, S. Krishnamoorthy and H. S. Sheng, "Estimating routing congestion using probabilistic analysis", in IEEE Trans. on CAD, 21(1):32–41, 2002.

[2] J. Westra, C. Bartels and P. Groeneveld, "Probabilistic congestion prediction", In Proc. ISPD, pages 204–209, 2004.

[3] H. Shojaei, A. Davoodi and J. T. Linderoth, "Congestion analysis for global routing via integer programming", in Proc. ICCAD, pages 256–262, 2011.

[4] X. He, T. Huang, L. Xiao, H. Tian, G. Cui and F. Y. Young, "Ripple: an effective routability-driven placer by iterative cell movement", in Proc. ICCAD, pages 74–79, 2011.

[5] M.-K. Hsu, S. Chou, T.-H. Lin and Y.-W. Chang, "Routability-driven analytical placement for mixed-size circuit designs", in Proc. ICCAD, pages 80–84, 2011.

[6] K. Tsota, C.-K. Koh and V. Balakrishnan, "Guiding global placement with wire density", in Proc. ICCAD, pages 212–217, 2008.

[7] J. A. Roy, J. F. Lu, and I. L. Markov, "Seeing the forest and the trees: Steiner wirelength optimization in placement", IEEE Trans. on CAD, 26(4):632–644, April 2007.

[8] M.-C. Kim, J. Hu, D.-J. Lee and I. L. Markov, "A SimPLR method for routability-driven placement", in Proc. ICCAD, pages 80–84, 2011.

[9] M. Pan and C. Chu, "IPR: An integrated placement and routing algorithm", in Proc. of DAC, pages 67–73, 2007.

[10] J. Roy et al, "CRISP: Congestion reduction by iterated spreading during placement", in Proc. ICCAD, pages 357–362, 2009.

[11] K.-R. Dai, C.-H. Lu, and Y.-L. Li, "GRPlacer: Improving routability and wire-length of global routing with circuit replacement", in Proc. of ICCAD, pages 351–356, 2009.

[12] J. A. Roy and I. L. Markov, "High-performance routing at the nanometer scale,", *IEEE Trans. on Computer-Aided Design*, vol. 27, no. 6, pp. 1066-1077, 2008.

[13] M. D. Moffitt, "MaizeRouter: Engineering an effective global router," in *Proc. ASP-DAC,* pages 232-237, 2008.

[14] M. M. Ozdal and M. D.F. Wong, "ARCHER: A history-driven global routing algorithm," in *Proc. ICCAD.,* pages 488-495, 2007.

[15] Y.-J. Chang, Y.-T. Lee, and T.-C. Wang, "NTHU-Route 2.0: a fast and stable global router," in *Proc. ICCAD,* pages 338-343, 2008.

[16] H.-Y. Chen, C.-H. Hsu, and Y.-W. Chang. "High-performance global

routing with fast overflow reduction." in *Proc. ASP-DAC.,* pages 582–587, 2009.
[17] Y. Xu, Y. Zhang, and C. Chu, "FastRoute 4.0: Global Router with Efficient Via Minimization," in *Proc. ASP-DAC,* pages 576-581, 2009.
[18] K.-R. Dai, W.-H. Liu, and Y.-L. Li, "NCTU-GR: Efficient Simulated Evolution-Based Rerouting and Congestion-Relaxed Layer Assignment on 3-D Global Routing," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 20, issue 3, pp. 459-472, 2012.
[19] T.-H. Wu, A. Davoodi, and J. T. Linderoth, "A parallel integer programming approach to global routing", *in Proc. DAC*, pages. 194-199, 2010.
[20] W.-H. Liu, W.-C. Kao, Y.-L. Li and K.-Y. Chao, "Multi-Threaded Collision-Aware Global Routing with Bounded-Length Maze Routing" *in Proc. Des. Autom. Conf.*, pages. 200-205, June 2010.
[21] Y. Han, D. M. Ancajas, K. Chakraborty and S. Roy, "Exploring high throughput computing paradigm for global routing," in *Proc. Int. Conf. Comput.-Aided Des.,* pages 298-305, 2011.
[22] Y. Xu and C. Chu, "MGR: Multi-Level Global Router," in *Proc. Int. Conf. Comput.-Aided Des.*, 2011.
[23] http://archive.sigda.org/ispd2008/contests/ispd08rc.html
[24] N. Viswanathan et al, "The ISPD-2011 Routability-driven Placement Contest and Benchmark Suite", in Proc. ISPD, 2011.
[25] http://archive.sigda.org/dac2012/contest/dac2012_contest.html
[26] T.-H. Lee, T.-C. Wang, "Congestion-constrained layer assignment for via minimization in global routing," *IEEE Trans. On Computer-Aided Design of Integrated Circuits and Systems*, pages 1643-1656, 2008.
[27] Wen-Hao Liu and Yih-Lang Li, "Negotiation-Based Layer Assignment for Via Count and Via Overflow Minimization," in *The 16th Asia and South Pacific Design Automation Conferenc*e, Jan. 2011, P. 539-544.

TABLE 1 COMPARISON TOTAL OVERFLOWS BETWEEN NCTUGR2 AND RCE IN A GIVEN TIME BUDGET.

Benchmarks	30 seconds		60 seconds		120 seconds		240 seconds	
	NCTUgr2	RCE	NCTUgr2	RCE	NCTUgr2	RCE	NCTUgr2	RCE
s1_Ripple	14	0	0	0	0	0	0	0
s2_Ripple	724796	130866	626050	28556	472104	3798	323878	1410
s4_Ripple	66690	236	45100	222	12130	212	698	214
s5_Ripple	107896	4790	73360	404	28354	0	9030	0
s10_Ripple	597656	216364	548004	114176	491294	69646	409308	53502
s12_Ripple	315122	174162	265444	107274	180118	42604	105984	8600
s15_Ripple	92574	2664	72012	16	21048	0	5096	0
s18_Ripple	356348	175542	313146	145808	272598	130446	188594	118630
s1_mPL11	62368	0	8780	0	0	0	0	0
s2_mPL11	1152682	486966	997796	226470	857516	67202	739474	24396
s4_mPL11	112856	11010	78640	900	19334	46	4444	42
s5_mPL11	354374	71632	294312	42916	230216	27378	126398	20842
s10_mPL11	733948	271370	655464	86286	578376	29116	484624	15314
s12_mPL11	1589998	1563470	1549036	1412980	1627508	1292848	1377296	1219146
s15_mPL11	126138	0	67846	0	11410	0	0	0
s18_mPL11	66710	0	27126	0	0	0	0	0

TABLE 2 COMPARISON THE ROUTING RESULTS BETWEEN NCTUGR2 AND RCE WITHOUT TIME LIMITATION.

Benchmarks	NCTUgr2				RCE				
	MOF	TOF	WL (10^5)	RCPU (s)	MOF	TOF	WL (10^5)	RCPU (s)	speedup
s1_Ripple	0	0	160.84	31	0	0	161.63	4	7.61
s2_Ripple	10	1442	350.75	3623	8	1430	358.04	203	17.81
s4_Ripple	8	224	122.80	536	6	236	124.67	30	17.75
s5_Ripple	0	0	198.44	573	0	0	202.02	86	6.67
s10_Ripple	8	39222	321.59	26509	8	40300	336.61	1227	21.61
s12_Ripple	0	0	260.86	1333	0	0	279.49	484	2.75
s15_Ripple	0	0	205.07	651	0	0	208.81	60	10.80
s18_Ripple	20	109514	159.10	5174	22	125988	162.25	550	9.40
s1_mPL11	0	0	168.81	79	0	0	169.02	9	8.66
s2_mPL11	8	3884	362.96	51356	8	4566	363.57	1914	26.83
s4_mPL11	2	42	123.89	922	2	42	126.52	94	9.76
s5_mPL11	14	11210	219.77	6114	18	15474	221.02	739	8.27
s10_mPL11	6	9678	315.31	29158	8	9370	331.49	1159	25.16
s12_mPL11	64	1041384	272.34	11221	70	1094646	259.25	1145	9.8
s15_mPL11	0	0	193.71	192	0	0	196.6	22	8.62
s18_mPL11	0	0	108.55	95	0	0	110.72	18	5.15
$Ratio_{each}$	**1**	**1**	**1**	**1**	**1.040**	**1.089**	**1.018**	**0.114**	-
$Ratio_{sum}$	**1**	**1**	**1**	**1**	**1.071**	**1.062**	**1.019**	**0.056**	-

MOF = maximum overflow; TOF = total overflows; WL = total wirelength; RCPU = runtime of the rip-up and rerouting stage (sec)