NCTU-GR: Efficient Simulated Evolution-Based Rerouting and Congestion-Relaxed Layer Assignment on 3-D Global Routing

Ke-Ren Dai, Wen-Hao Liu, and Yih-Lang Li, *Member, IEEE*

***Abstract*—The increasing complexity of interconnection designs has enhanced the importance of research into global routing when seeking high-routability (low overflow) results or rapid search paths that report wirelength estimations to a placer. This work presents two routing techniques, namely circular fixed-ordering monotonic routing and evolution-based rip-up and rerouting using a two-stage cost function in a high-performance congestion-driven 2-D global router. We also propose two efficient via-minimization methods, namely congestion relaxation by layer shifting and rip-up and reassignment, for a dynamic programming-based layer assignment. Experimental results demonstrate that our router achieves performance similar to the first two winning routers in ISPD 2008 Routing Contest in terms of both routability and wirelength at a 1.05× and 18.47× faster routing speed. Moreover, the proposed layer assignment achieves fewer vias and shorter wirelength than congestion-constrained layer assignment (COLA).**

***Index Terms*—Algorithms, design automation, optimization, routing.**

I. Introduction

As VLSI design complexity continues increasing and semiconductor manufacturing advances to the nanometer scale, interconnection delay begins to dominate the circuit delay. Placement and routing stages mainly determine interconnection delay, where placement determines the lower bound of total wirelength and routing realizes physical wires to determine final wirelength. In recent years, there has been an increasing amount of literature concerning routing issues for nanometer designs, including global routing for nanometer designs, variable-rule routing [1]–[3], double-via-aware routing [4], [5], lithography-friendly routing [6]–[9], and density-driven routing [10], [11], etc.

Conventional routing flow is composed of global routing and detailed routing. Global router identifies a global path for every net to constraint the path searching conducted by detailed router within the global path. Path search space is substantially reduced and path searching time is thus massively dropped. Meanwhile, global routing result can offer a good estimation of routing congestion and routed wirelength. 2-D global routing determines the global path for each net, while 3-D global routing additionally determines the used layer of every routing path. Mostly, 3-D global routing is accomplished by 2-D global routing as well as layer assignment. The ideal approach for interconnection optimization is to undertake a simultaneous placement and global routing [12] or take interconnection delay into account in early stage. The necessity for fast and efficient global routing algorithms for cooperating with placers and tackling modern designs with high congestion and complexity is increasing.

Lee's algorithm [13] has been widely employed in path searching of various routing problems and promises to identify a feasible path for a routing problem that has a feasible solution. However Lee's algorithm costs large runtime for the routing of large modern designs. Hence initial routing generally applies another fast routing algorithm, such as pattern routing [14] or monotonic routing [15]. The most commonly used patterns are L-shaped or Z-shaped patterns. Pattern routing used in multi-net routing offers extremely fast routing speed at the cost of worse routing quality than maze routing. Monotonic routing is another very fast routing technique and produces a path without detour as well. Dynamic programming algorithm can efficiently realize monotonic routing. The number of routing paths a routing algorithm searches for determines the routing speed and quality. For a 2-pin net routing in a $m \times n$ bounding box, Z-shaped pattern routing identifies the best path out of $m + n - 2$ paths, while monotonic routing identifies the best path out of $(m + n - 2)!/(m - 1)!(n - 1)!$ paths. With the aid of these two fast routing techniques, routing congestion estimation before real routing becomes an important operation in most modern leading-edge global routers.

Besides, novel history-based cost function design for rip-up and rerouting process develops the improvement of overflow elimination and runtime speedup by increasing the routing cost of congested regions [16]. Congestion history scheme forces the preceding routing to evade congested regions such that the routing resource of congested regions are reserved for subsequent routing. Ideal congestion history scheme is to remove overflow quickly with slightly increasing wirelength. Current state-of-the-art global routers apply unique history-based cost function in rip-up and rerouting stage. A routing using a short wire consumes less routing resource than using a long wire and more routing resource is thus reserved for subsequent routing. Adopting too many detours in early routing stage should be well

Manuscript received February 09, 2010; revised August 05, 2010; accepted October 25, 2010. Date of publication January 28, 2011; date of current version February 17, 2012. This work was supported in part by the National Science Council of Taiwan under Grant NSC 99-2220-E-009-036, Grant NSC 99-2220-E-009-012, and Grant NSC 98-2220-E-009-059.

The authors are with the Department of Computer Science, National Chiao Tung University, HsinChu 30010, Taiwan (e-mail: daikeren@gmail.com; is90200@cis.nctu.edu.tw; dnoldnol@gmail.com; ylli@cis.nctu.edu.tw).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2010.2102780

controlled to evade lowering the routability and overflow-free rate of subsequent routings due to overusing routing resources in early routing stage. Thus control over the increase of wirelength is very important in global routing.

In this paper, we present a novel 2-D global router based on simulated evolution rip-up and rerouting with two-stage congestion history scheme. The simulated evolution technique can encourage the router to escape from local optimum and achieve better performance. Furthermore, we also propose a congestion-relaxed layer assignment (CRLA) algorithm that can explore more solution space than congestion-constrained layer assignment (COLA) [17]. Compared with other global routers, the proposed router takes less time to have a level of routing quality to that of the first two winning routers in the ISPD 2008 global routing contest [18].

NCTU-GR is compared to the first two winners in ISPD 2008 Routing Contest, produces the shortest average wirelength and completes 12 out of 16 benchmarks at a 1.05$\times$ and 18.47$\times$ faster routing speed. Our key contributions include the following:

- circular fixed-ordering monotonic routing (CFOMR) to efficiently remove overflow before rip-up and rerouting stage;
- simulated evolution-based rip-up and rerouting technique with two-stage cost function to accelerate routing speed and decrease overflow;
- CRLA algorithm based on layer shifting and rip-up and reassignment to achieve fewer vias than congestion constrained layer assignment.

The rest of this paper is organized as follows. Section II presents background and previous works. Section III gives an overview of the proposed router. Section IV presents the proposed 2-D global routing technique. Section V describes the proposed layer assignment algorithm. Section VI summarizes the experimental results. Section VII draws conclusions and discusses future work.

II. Background and Previous Works

A. Problem Formulation

A routing region can be partitioned into an array of global cells or bins (G-cells) and modeled by a grid graph $G(V,E)$, where every node of the graph denotes a global cell and every edge (global edge) is termed the adjacency of the related global cells of its two end nodes. The capacity of an edge indicates the number of routing tracks that can be accommodated across the abutting boundary. Fig. 1(b) shows a three-layer grid graph of a circuit containing 4×4 G-cells in Fig. 1(a). An overflow occurs when the number of wire demand exceeds the number of capacity. The number of overflow in a global edge is calculated as the exceeding demand. The problem of global routing is defined as follows: given a set of nets, a grid graph, and the capacity of every grid edges, the global routing is to find the paths to connect all the pins for every net such that the number of overflows is minimized.

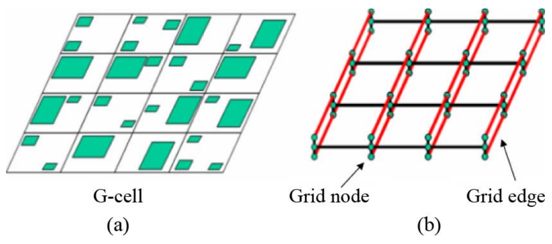


Fig. 1. G-cells and their corresponding grid graph.

B. Metrics of Global Routing

A global router largely focuses on producing a highly routable global path for every net. The metrics of routability can be measured based on the congestion of all global edges and wirelength of every net. Few overflow global edges and short wirelength imply high routability. The increasing number of overflow global edges raises the difficulty of routing; reducing the wirelength lowers the congestion of some global edges. In the ISPD 2007 Global Routing Contest [19], the quality of a global router is evaluated by the following equation:

$$WL = \sum_{n \in N}^{n} wl(n) + \text{via\_weight} * \text{via}(n) \tag{1}$$

where N is the set of nets, $wl(n)$ is the number of grid edges used by net n, $\text{via}(n)$ is the number of vias of net n, via_weight is the weighting factor of one via. In the ISPD 2007 Global Routing Contest, via_weight is set to three.

In the ISPD 2008 contest [18], run time is considered as a part of weighted total wirelength which is formulated as follows:

$$WL_{wt} = WL\left(1 + \min\left(0.1, 0.04 \times \log_2\left(\frac{\text{time}_{\text{cpu}}}{\text{med\_time}_{\text{cpu}}}\right)\right)\right) \tag{2}$$

where WL is the total wirelength cost in (1), time_{cpu} is the runtime of a router and $\text{med\_time}_{\text{cpu}}$ is the median runtime of all routers in the routing contest. In the ISPD 2008 Global Routing Contest, via_weight is set to one. According to (2), a router reduces its wirelength by 4% when it runs two times faster.

C. History-Based Routing

Fig. 2 displays an example of using L-shape pattern routing and congestion history during the rip-up and rerouting stages. Fig. 2(a) depicts four congested global cells containing nets b and c. In conventional rip-up and rerouting flow, nets b and c are ripped up and rerouted. If net b is rerouted prior to net c, then the new routing path of net b remains the same as that of the previous rip-up and rerouting iteration, since the path has the smallest routing cost. In this case, every rip-up and rerouting iteration produces the same congested regions (global cells 1, 2, 3, and 4) and thus never removes overflow. However, if the routing cost for passing global cells 1, 2, 3, and 4 is increased, then rerouting encourages net b to seek other routing regions, as shown in Fig. 2(b). Overflow can thus be removed, as displayed in Fig. 2(c). McMurchie and Ebeling [16] proposed the following history-based congestion cost function:

$$c_n^k = (b_n + h_n^k) * p_n \tag{3}$$

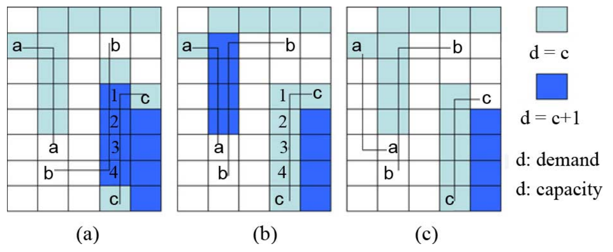


Fig. 2. Example of the use of congestion histories.

where c_n^k is the cost of node n in the kth iteration; b_n is the base cost of node n; h_n^k is the history cost of node n and p_n represents the penalty cost of node n. The history cost h_n^k increases as overflow continues in subsequent rip-up and rerouting iterations. The value of h_n^k in the kth iteration is given by

$$h_n^{k+1} = \begin{cases} h_n^k + 1, & \text{if } n \text{ is overflowed} \\ h_n^k, & \text{otherwise.} \end{cases} \quad (4)$$

D. Simulated Evolution-Based Rip-Up and Rerouting

Rip-up and rerouting technique is widely used in global and detailed routing. Given an illegal routing solution, rip-up, and rerouting technique iteratively rips up the nets with violation and reroutes them to solve the routing violation. However, only greedy rip-up and rerouting may yield local optimum. Lin *et al.* presented a simulated evolution-based rip-up and rerouting detailed router called SILK [20]. The simulated evolution scheme scores each individual in a population and then determines its survival rate in the next generation. SILK scores each net in a generation by its routing violation and path quality as follows:

$$\begin{aligned} \text{score}_i &= \alpha * (\#\text{vio}) + \beta * (\#\text{vias} - \#\text{pins}) \\ &\quad + \gamma * \left(\frac{\text{actual\_wirelength}}{\text{lower\_bound\_}wl} \right) \end{aligned} \quad (5)$$

where $\#\text{vio}$ is the violation number of net i, $\#\text{vias}$ is the number of vias in net i, $\#\text{pins}$ is the number of pins of net i, actual_wirelength is the actual routed wirelength and the $\text{lower\_bound\_}wl$ is the minimal possible wirelength of net i. SILK is a detailed router and the violation number of net i is referred to as the number of DRC and short errors on the routing of net i. The last two items in (5) reflect the routing quality of net i. A net is considered to have good quality if it comprises less vias and shorter wirelength. All scores are then normalized to 0.1–0.9. Finally, SILK generates a random number between 0.0 and 1.0 for each net. The nets with normalized scores greater than their random numbers are weeded out in next generation, i.e., the paths of these nets will be ripped up. These removed nets are then rerouted in decreasing order of their normalized scores. Thus, the nets with low cost also have chance to be ripped up and rerouted. Additionally SILK also employs rip-up and rerouting to improve routing quality.

E. Previous Works—Global Router

Kastner *et al.* [14] developed a pattern-based global routing algorithm. Hadsell and Madden [21] proposed the Chi dispersion router by using a linear cost function and amplified congestion estimation for path searching and rip-up and rerouting, respectively. In recent years, many global routers have been proposed to advance the study. Cho and Pan [22] presented BoxRouter to utilize integer linear programming (ILP) to solve global routing problem. In the beginning, BoxRouter finds the most congested region in the design by prerouting to get the initial box and then solves the routing problem by rapidly progressive ILP, adaptive maze routing and box expansion. Finally, post routing is performed to balance the optimizations in wirelength and routability. FastRoute [23] employs congestion-driven Steiner tree construction and edge shifting to find a good net topology and then extremely fast completes the design using pattern routing and maze routing. FastRoute 2.0 [15] utilizes monotonic routing and multi-source multi-sink maze routing to further develop routing quality.

ISPD 2007 Global Routing Contest stimulates researches in global routing for modern congested benchmarks from industry. Archer [24] proposed a framework to achieve a smooth tradeoff between overflow and wirelength minimization. A Lagrangian relaxation-based topology improvement algorithm is employed to alter the Steiner tree for congestion minimization. BoxRouter is upgraded with negotiation-based $A*$ search and topology-aware wire rip-up in BoxRouter 2.0 [25]. FGR [26] suggests a routing technique based on discrete Lagrange multipliers in rip-up and rerouting. NTHU-Route [27] employs a congested region identification method to determine the net order in rip-up and rerouting. MaizeRouter [28] uses different net topology representations and employs extreme edge shifting, edge retraction and garbage collection to minimize overflow. NTUgr [29] develops least-flexibility-first routing and multi-source multi-sink escaping-point routing to minimize the congestion of in-tile nets and the residual via capacity for global routing. FastRoute 3.0 [30] uses the concept of virtual capacity to complete high quality routing. FastRoute 4.0 [31] uses via-aware Steiner tree and 3-bend routing to further reduce the via count of FastRoute 3.0. NTHU-Route 2.0 [32] improves the quality of NTHU-Route by using a new history based cost function and a new ordering method for rip-up and rerouting. As for 3-D global routing problem, FGR [26] can directly solve 3-D global routing by using 3-D maze routing. GRIP [33] utilizes integer linear programming to complete 3-D global routing by column generation and branch-and-bound without adopting layer assignment. Since the complexity of 3-D global routing is very huge, GRIP partitions a routing region into a set of sub-regions and assigns the routing job of each sub-region to a computer in a clustered computer system through network to achieve scalable speedup.

F. Previous Work—Layer Assignment

Cho *et al.* proposed an integer linear programming formulation to solve via/blockage aware layer assignment problem in BoxRouter 2.0 [25]. The formulation tries to complete as many nets as possible. The unassigned nets are then completed by maze routing. Lee and Wang [17] presented an efficient dynamic programming algorithm to solve congestion-constrained layer assignment for via minimization problem (COLA) on a grid graph. They firstly determine the assignment order of each net and then adopt a dynamic programming algorithm to assign the layer of each net sequentially. The net order influences the layer assignment results, because the nets in the later assigning

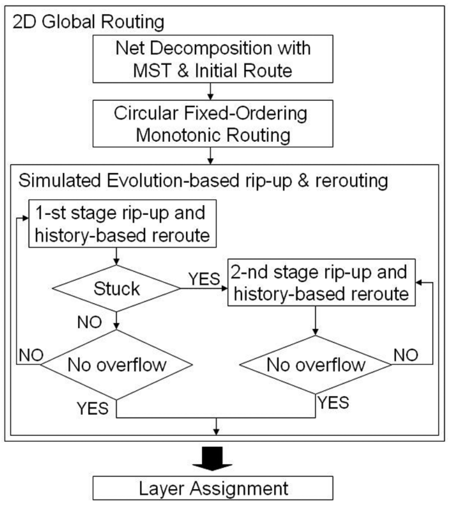

2D Global Routing
Net Decomposition with MST & Initial Route
Circular Fixed-Ordering Monotonic Routing
Simulated Evolution-based rip-up & rerouting
1-st stage rip-up and history-based reroute
Stuck
YES
NO
2-nd stage rip-up and history-based reroute
No overflow
NO
YES
No overflow
NO
YES
Layer Assignment

Fig. 3. Design flow of the proposed router.

order have less available layer resources than the nets in the early assigning order.

III. Overview of Our Router

Fig. 3 presents the design flow of the proposed router. The 3-D routing problem is transformed into a 2-D routing problem. This study then tries to minimize overflow and wirelength in 2-D global routing via simulated evolution-based rip-up and rerouting using two-stage history-based maze routing. Finally, the number of vias is minimized in the Layer Assignment for 3-D global routing.

IV. 2-D Global Routing

A. Net Decomposition and Initial Route

Many state-of-art global routers use FLUTE [34], [35] to decompose multiple-pin nets into two-pin nets. Net decomposition with a rectilinear Steiner minimal tree (RSMT) reduces wirelength more than a minimum spanning tree (MST); additionally, the RSMT has less routing flexibility than the MST as it generates more flat segments than the MST. This study decomposes each net into two-pin nets via MST. Then, FLUTE is employed to yield RSMT as the initial routing solution.

B. Circular Fixed-Ordering Monotonic Routing

Net ordering influences routing results. Precisely identifying a good net ordering is difficult. A fixed net ordering would likely hinder some nets, while adopting different net orderings could release this blocking.

Rip-up and rerouting is widely used in global routers to remove overflows. However, it is very time-consuming. To achieve better performance, it is desirable to rapidly solve as many overflows as possible before entering rip-up and rerouting stage. The proposed router employs a novel routing technique, called *circular fixed-ordering monotonic routing* (CFOMR), to improve the overflow-removing efficiency of applying fixed net-ordering routing. The CFOMR repeatedly rips up and reroutes all nets in a decreasing order of wirelength using monotonic routing k_1 times, where k_1 is a user-defined

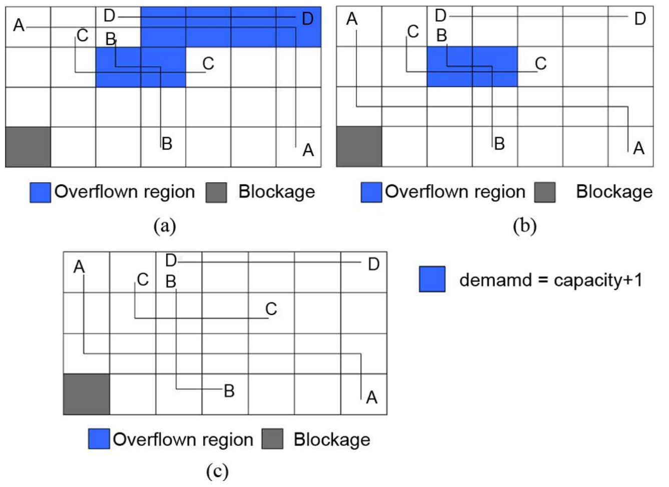


Fig. 4. Example of CFOMR with routing order $A \longrightarrow B \longrightarrow C \longrightarrow D$: (a) initial routing; (b) rerouting net A in the second round; (c) the second round routing result.

constant. Ripping up and rerouting is performed on one net at a time. For every net, the existing routing paths on the routing graph vary at different cycles when rerouting the net, which is equivalent to perform the net routing with different net orderings at different cycles. Adopting a decreasing order of wirelength in each cycle improves routing performance when processing long wires. In sum, this stage has the benefits on designed and random net ordering. Fig. 4 shows a four-net example of CFOMR. The horizontal and vertical capacity of each grid is one and via cost is ignored in this example. Fig. 4(a) shows the initial routing result with six overflows, using the routing order $A \longrightarrow B \longrightarrow C \longrightarrow D$. The second round routing rips up and reroutes each net using the same net routing order based on the routing result of the first round routing. Fig. 4(b) depicts the routing result after rerouting net A. Notably, as net A is rerouted, it looks like to route net A after routing nets B, C and D. Similarly, as net B is rerouted, it also look like to route net B after routing nets A, C and D. Six overflows are totally removed after the second round routing [see Fig. 4(c)]. For this case, the optimal routing order is $D \longrightarrow C \longrightarrow B \longrightarrow A$. In the second routing of CFOMR, rerouting net A matches the optimal routing order (route nets B, C, and D before net A) while rerouting net B fulfills another partial order (route net C before net B). Experimental results indicate that the number of overflows stabilizes when constant k_1 reaches a threshold. Fig. 5 shows that the number of overflows of adaptec5 declines as the number of monotonic routings increases. In practice, this study set k_1 to 5.

C. Simulated Evolution-Based Rip-Up and Rerouting

Fig. 6 presents the modified routing algorithm from SILK for global routing. Routing violations in global routing are regarded as overflows. Furthermore, rerouting in the proposed global routing applies two different cost functions in two stages to tackle some hard-to-route nets. In the first stage, most or all nets are complete. If the minimum number of overflows cannot be decreased further within several iterations, i.e., the router is stuck in a local optimum, rerouting enters the second stage

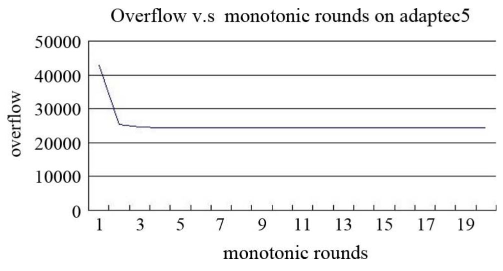


Fig. 5. Effects of different k_1 values on the benchmark adaptec5.

```
Algorithm Simulated evolution Rip-up and Reroute
Input A set of nets N
1. for (each net i) calculate score_i by equation (2);
2. for (each net i) normalize score_i into norm_score_i ranging
      between 0.1 and 0.9;
3. for (each net i in N) {
4.     if (net i intersects any overflow net) {
5.         Generate a pseudo-random number r between 0.0 and 1.0;
6.         if (normscore_i > r)    mark net i as a ripped-up net; }}
7. Rip up and reroute each marked net in the decreasing order of
      norm_score_i;
```

Fig. 6. Proposed simulated evolution-based rip-up and rerouting algorithm.

and employs another cost function for the remaining incomplete nets.

1) Simulated Evolution-Based Rip-Up and Rerouting: The proposed simulated evolution-based global router modifies the scoring function and scheme for selecting ripped up nets. Equation (5) is improved by introducing the number of rip-up and rerouting iterations as

$$\text{new\_score}_i = \begin{cases} \text{Quality\_Cost} & \text{if } \#\text{vio} = 0 \\ \frac{\omega}{\#\text{vio}*\log(k+1.0)} + \text{Quality\_Cost} & \text{otherwise} \end{cases} \quad (6)$$

where Quality_Cost is the last two terms of (5) $(\beta * (\#\text{vias} - \#\text{pins}) + \gamma * (\text{actual\_wirelength}/\text{lower\_bound\_}wl))$, ω is a user-defined constant, k is the number of rip-up and rerouting iterations, and #vio is the number of violations of a net i. In global routing, a violation is an overflowed edge in the routing path of net i.

In (5), nets with large violations are ripped up and rerouted first. However, since nets with large violations usually exist in a congested area, rerouting these nets first may generate a long detour and use additional routing resources. In (6), the nets with few violations have larger scores than nets with many violations in the first few iterations; thus, nets with few violations are ripped up and rerouted first. As the iteration process proceeds, when the number of overflows cannot be decreased, rerouting violation-free and bad-quality nets likely frees routing resources for overflowing nets. Thus, the score is divided by $\log(k+1.0)$ to reduce the influence of the number of violations. The design principle of the cost function is to increase the possibility of ripping up and rerouting violation-free bad-quality nets. Moreover, in SILK, all nets are candidates to be selected, ripped up, and rerouted. In the proposed global routing scheme, millions of nets exist. To keep the number of ripped up nets within a reasonable range, only the nets intersecting expanded bounding boxes of overflowed nets are selected as possible candidates to be ripped

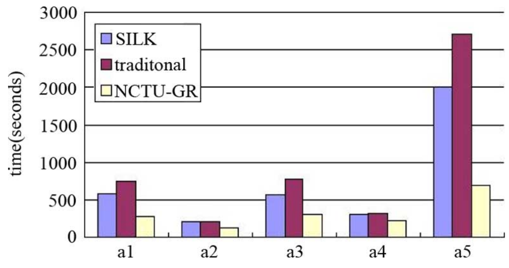


Fig. 7. Runtime comparison of five overflow-free benchmarks in ISPD'07 Routing Contest among SILK, traditional rip-up and rerouting, and the proposed scheme.

up and rerouted. The expanded bounding boxes of overflowed nets will be enlarged as the number of rip-up and rerouting iterations increases to improve routing flexibility. In SILK, all selected nets are ripped up and then rerouted. In NCTU-GR, one net is ripped up and rerouted at a time. Traditional routers rip up all overflowed nets and then route them sequentially.

Fig. 7 compares the runtime to find overflow-free solutions among SILK, traditional rip-up and rerouting scheme, and the proposed one. The experiments using five benchmarks from ISPD 2007 global routing contest show that our simulated evolution-based method can find overflow-free solutions in a much shorter time than traditional method and the original method in SILK.

2) Two-Stage Cost Function: In recent years, the concept of history-based routing is widely used in many global routers [24]–[27], [29], [31] to efficiently identify the overflow-free routing solution. Equations (3) and (4) show the McMurchie's history-based cost function. However, McMurchie's cost function has two problems. First, h_n increases as the rip-up and rerouting operation continues in order to overestimate earlier congested regions, even when these previously congested regions become un-congested. Second, the cost function of maze routing is given by $f(x) = g(x) + h(x)$, where $h(x)$ is the estimation from the current position x to the target and $g(x)$ is the cost from the source to x. If McMurchie's function is given by $g(x)$, then as rip-up and rerouting procedure continues, increasing $g(x)$ in proportion to the number rip-up and rerouting iterations dominates the value of $f(x)$. The new history-based cost function, which solves these problems, is as follows:

$$c_e^k = \left(1 + \frac{h_e^k}{k}\right) * p_e \quad (7)$$

where c_e^k represents the cost of edge e and h_e^k is the history cost of edge e and p_e is the penalty of edge e. The proposed router applies the logistic function in [23] as the penalty of edge e. The function is defined as follows:

$$p_e = 1 + \frac{\text{penalty}}{1 + e^{-\text{slope}*(\text{demand}-\text{capacity})}} \quad (8)$$

where demand is the routing demand of edge e, capacity is the capacity of edge e, penalty and slope are user defined constants. In (7), c_e^k never exceeds $2p_e$, thus c_e^k does not become a large and dominant factor in subsequent iterations. This cost function can also balance $g(x)$ and $h(x)$ during maze routing.

Theorem 1: c_e^k in (7) does not exceed $2p_e$.

Proof: Equation (4) shows that the value of h_e^k increase by 1 only when edge e overflows. For any iteration k, the value of h_e^k does not exceed k, i.e., h_e^k/k does not exceed 1

$$c_e^k = \left(1 + \frac{h_e^k}{k}\right) * p_e \leq (1+1) * p_e = 2 * p_e.$$

Thus c_e^k does not exceed $2p_e$. □

The rip-up and rerouting scheme using the cost function in (7) can rapidly remove most overflows for ISPD benchmarks. Some nets cannot be completed because the historical cost in (7) is constrained to at most 1; thus, (7) may not have sufficient intensity to make the router avoid previously overflowed regions. If the rip-up and rerouting scheme cannot reduce the minimum number of overflows within several iterations, the rip-up and rerouting scheme enters its second stage to solve the remaining incomplete nets by using the following cost function:

$$c_e^k = \left(1 + \left(\frac{h_e^k}{\log(k) + 1.0}\right)^{\alpha}\right) * p_e \tag{9}$$

where c_e^k and h_e^k are defined as in (7) and α is a user-defined constant that is set to 2. In the second stage, routability is more important than wirelength and historical cost has more impact on the cost function than in the first stage. The resulting effect removes additional overflows at the cost of increased run time and wirelength.

3) Issues on Wirelength and Via: Via and wirelength are two important minimized objectives in routing problem. Large via number would increase yield loss in the circuit, while long wires would occupy routing resources to decrease routability. In this work, three simple effective schemes are employed to reduce wirelength and the number of vias.

- During the rip-up and rerouting stage, a wirelength constraint is imposed on each net. The wirelength constraint is set to be slightly larger than the half perimeter wirelength (HPWL). As the iteration proceeds, the wirelength bound is relaxed to allow for additional detours.
- The third term in (6) defines routing quality using wirelength. The nets with long wire have an increased possibility to be ripped up.
- In the circular fixed-ordering monotonic routing stage, this study first sets a large via cost and via cost decreases as the number of iterations increases.

D. Implementation Details

This section describes some implementation details. At first, implementation issues of maze routing are presented. Then some of the tuning techniques of the proposed router will be discussed.

1) Implementation of Maze Routing: FastRoute 2.0 [15] proposed a multi-source, multi-sink maze routing algorithm by treating the whole sub-tree as a source or target point not only to avoid yielding redundant wire segments, but also to yield better routing results than that of pin-to-pin routing. However, identifying all grid points on a large sub-tree expends a considerable amount of time, with subsequent grid-point expansion tending to explore a large space. To remedy the decline in routing speed,

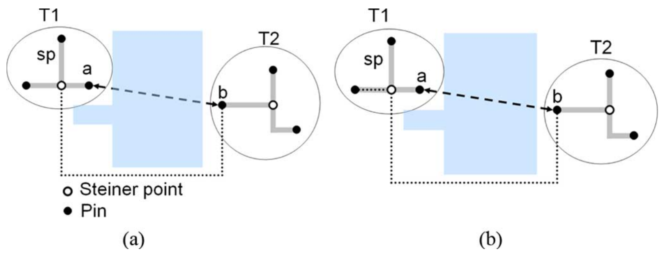


Fig. 8. Difference of multi-pin maze routing between this work and NTHU-R [27]. (a) The multi-pin routing result of a two-pin routing demand (from point a to point b) in NTHU-R [27]. (b) The routing result of this work.

the proposed routing scheme considers only the pins on the sub-trees as a source and target pins. The pins on two sub-trees to be connected can be clustered efficiently into two sets using the Disjoint set data structure [36], with one set containing the original source pin and the other set containing the original target pin. The points in two sets are starting and target points of maze routing. The proposed scheme differs from the method in [27] mainly in that the latter adopts the pins and Steiner points on the sub-trees as the source and target points while the former ignores the Steiner points. Although this work omits Steiner points from the source and target points, the identified result is still the same as that in [27] by setting the costs of edges used by two sub-trees as zero if the minimum-cost path connects to the existing Steiner point.

Lemma 1: The minimum-cost path identified this work to connect two sub-trees is the same as that in [27].

Proof: Assume that two end points of the minimum-cost path $(pa_{\min})$ found in [27] are pn_1 and pn_2 in sub-trees T_1 and T_2, respectively. If pn_1 and pn_2 are both pins, obviously, path $pa_{\min}$ can also be identified in this work. Additionally, if points pn_1 and pn_2 are both Steiner points, a zero-cost path (pa_1) on T_1 can definitely be found from any pin on T_1 to pn_1 and a zero-cost path (pa_2) on T_2 from any pin on T_2 to pn_2. The costs of paths pa_1 and pa_2 are 0, explaining why the total cost of paths $pa_{\min}$ plus pa_1 and pa_2 is the same as that of path $pa_{\min}$. Additionally, paths pa_1 and pa_2 are on sub-trees T_1 and T_2 and their edges are redundant. The routing tree obtained in this work after removing redundant edges is the same as that in [27]. As for one point of pn_1 and pn_2 being a Steiner point, the proof can be easily conducted using the previous two cases. □

Fig. 7(a) and (b) display how this work differs from the work in [27]. Dashed-dotted lines represent the original point-to-point routing problem; light blue regions are congested regions; bold gray wires imply existing routing wires and dotted lines represent the found minimum-cost rerouting path to connect two sub-trees after ripping up the edge ab. In [27], the source and target sets both contain three pins and one Steiner point [see Fig. 8(a)] while, in this work, the source and target sets only contain three pins [see Fig. 8(b)]. According to Fig. 8(b), the intersections of bold gray wires and dot lines are the redundant edges. Following removal of redundant edges in Fig. 8(b), the final routing paths in Figs. 8(a) and 8(b) are the same. The proposed scheme is characterized mainly by the relatively simple structure to record routing paths and its subsequent efficiency because only the path of each point-to-point routing is stored in the structure and storing Steiner points is unnecessary.

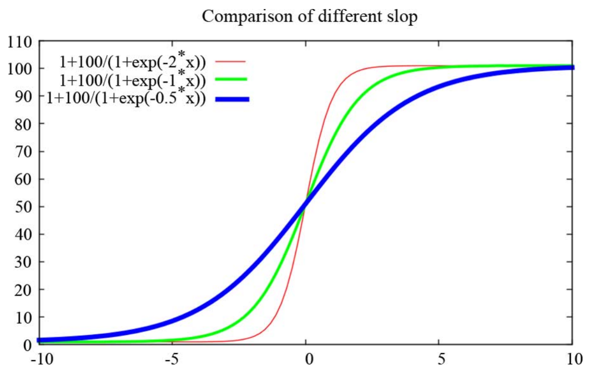


Fig. 9. Comparison of the logistic function with different slopes.

The proposed routing scheme also utilizes $A*$ searching algorithm to accelerate path searching. For current position x of a routing path, the path cost function by maze routing can be expressed as $f(x)=g(x)+h(x)$. In $A*$ searching algorithm, a priority queue is maintained. The path priority at x is determined by the function $f(x)=g(x)+h(x)$, where $g(x)$ denotes the cost from the source to point x and $h(x)$ refers to the heuristic function to estimate the cost from point x to the target. In multiple-pin maze routing, the target set may contain multiple pins rather than one. Despite the linear complexity of minimum-distance computation from a point to a set of points, the number of such computations may be extremely large during a two-pin connection, necessitating the lowering of the complexity associated with computing the target cost. Notably, if the heuristic function $h(x)$ is admissible, i.e., the cost to reach the target is never overestimated, then $A*$ algorithm seeks the optimal solution. To achieve this objective, a bounding box that encloses all target points is identified first. The heuristic function $h(x)$ is defined as the minimum Manhattan distance from position x to the bounding box.

2) Parameter Tuning: Some parameters must be tuned in the proposed router. This section discusses the experiences of tuning the parameters in our implementation. For global routing, a tradeoff always occurs between wirelength and overflow minimization. Of priority concern in generating overflow-free routing results is not only to ensure that as many edges as possible have a lower demand than their capacity in the early routing stage, but also to gradually release edge usage constraint as the routing iteration proceeds. To adhere to this principle, NCTU-GR adequately controls the edge cost.

The proposed router adopts the logistic function in (8) with two parameters. Parameter $\mathrm{penalty}$ refers to the ability to avoid an overflowed edge, while parameter slope is the function's slope. Fig. 9 compares the functions of (8) using the same $\mathrm{penalty}$ and various slopes. Additionally, x-axis denotes the difference between demand and capacity, while y-axis refers to the function value of (8). Among the slopes of 0.5, 1, and 2, the slope of 0.5 has the largest increasing rate in function value as demand is significantly lower than capacity; meanwhile, that of 2 has the least increasing rate in the same range of x-axis. As demand approaches and slightly exceeds capacity, the slope of 2 has the largest raising rate in function value; meanwhile, that of 0.5 has the least raising rate. In the beginning of the rip-up and rerouting stage, a small slope can offer an early alarm as to increasing edge usage in order to ensure that as many edges as possible have a lower demand below their capacity. With a growing number of rip-up and rerouting iterations, the value of slope also grows to free the edge usage constraint. Moreover, for congested benchmarks, the initial slope and $\mathrm{penalty}$ should be larger than those for uncongested benchmarks because uncongested benchmarks are likely routed without overflow. A small slope and $\mathrm{penalty}$ can thus yield overflow-free routing results faster than the large ones can.

```
Algorithm Congestion-Relaxed Layer Assignment
Input: 3D grid graph G, A set of nets N
1. S← Net_Decomposition(N);
2. Determine the assigning order of each subnet in S by Eq. (10);
3. for (each subnet $s_i$ in decreasing order of S) {
4.     heursitic_assignment($s_i$, G);
5.     if (net($s_i$) is completely assigned) { rip_up(net($s_i$), G);
6.         DPLALS(net($s_i$), G); }}
7. Rip-up_and_re-assigning(N)
8. end
```

Fig. 10. Flow of the proposed layer assignment.

In the proposed simulated evolution based rip-up and rerouting scheme, each routed net is scored using (6). A net with a large score is likely ripped up and rerouted. The parameter ω in (6) determines the penalty of a routing violation. A large ω raises the likelihood of ripping up and rerouting the nets with violation. In the beginning of the rip-up and rerouting stage, a small ω encourages poor quality nets to be ripped up and rerouted to release additional routing resources for subsequent routings. With a growing number of rip-up and rerouting iterations, the value of ω increases the likelihood of ripping up and rerouting the nets with violation. For congested benchmarks, the initial value of ω should be set to a larger value than that for uncongested benchmarks, since additional routed nets have violations in the congested benchmarks than those in the uncongested ones in the beginning of the rip-up and rerouting stage.

V. Layer Assignment

The purpose of layer assignment is to map a 2-D routing solution into a 3-D solution while minimizing the number of vias, without changing routing topology or increasing any overflow. This problem is called constrained via minimization (CVM), which has been proven to be NP-complete [37].

In previous layer assignment work, the net assignment order strongly influences the layer assignment results because the nets in the later assigning order have less available layer resources than those in the early assigning order. In this work, the proposed net decomposition offers more accurate assignment order; besides a layer assignment algorithm with layer shifting technique is proposed to further reduce congestion by pushing away early assigned nets to increase the flexibility of later assigned nets. This problem is named as congestion-relaxed single net layer assignment for via minimization.

Fig. 10 presents the flow of the proposed layer assignment. At first, each net is decomposed into a set of subnets S (line 1). Then the assignment order of each subnet in S is determined

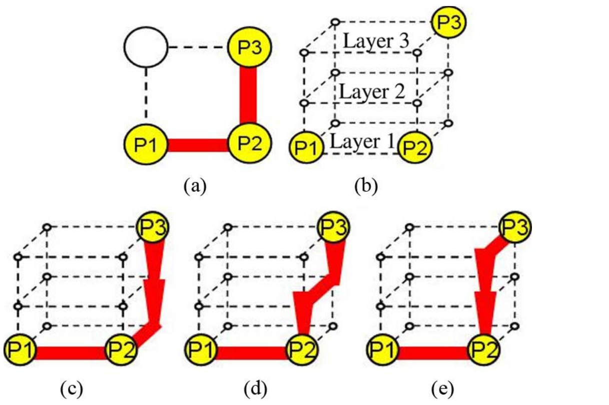


Fig. 11. Layer assignment example: (a) 2-D routing result; (b) 3-D grid graph; (c)–(e) three optimal layer assignment results.

with (10) (line 2). Next, each subnet s_i in S is assigned to corresponding layer by a heuristic assignment algorithm (line 4). Since the subnets of a net are assigned independently, the assignment result of a net tends to fall into the local optimum. To further minimize via number, we rip up and reassign a net when all subnets of the net have been assigned (line 5 $\sim$ 8). Note that $\mathrm{net}(s_i)$ is the net that subnet s_i belongs to. The proposed layer assignment algorithm with layer shifting technique is adopted to reassign nets to yield a better assignment result in global view (line 6). Finally, rip-up and reassigning stage iteratively further minimize via number until no room for improvement is available (line 7).

The proposed layer assignment has the following improvements:

1) refined net ordering by ***net decomposition***;
2) via minimization of a net by congestion-relaxed dynamic programming-based layer assignment with ***layer shifting*** and ***rip-up and reassignment***.

A. Net Decomposition and Ordering

In this work, we discover that a good layer assignment result can be obtained by decomposing each net into several subnets and then sequentially assigning all subnets. For example, Fig. 11(a) is a 2-D routing of a net and Fig. 11(b) displays the pin locations of the net in 3-D grid graph. If there is no preferable routing direction constraint, Fig. 11(c)–(e) show three optimal layer assignment results. In these three optimal assignment results, the subnet between $P1$ and $P2$ must be assigned to layer 1, but the subnet between $P2$ and $P3$ have three choices. Obviously, the optimal assignment requires the subnet between $P1$ and $P2$ to be assigned first for using layer 1 while the subnet between $P2$ and $P3$ does not need to own high assigning order like $P1 - P2$ subnet. In COLA [17], the order of a net is determined by its average congestion, wirelength and number of pins. However, if the congestions in the passed regions of a long net vary significantly, then the average congestion will not accurately reflect the state of the whole net. Net decomposition can well tackle this problem by splitting a long net into several subnets to reduce the congestion variation.

Breadth-first search (BFS) with queue is adopted to identify the break points. The root node is labeled as 0 and then pushed into the queue. The nodes that are subsequently pushed into the queue are labeled in increasing order. A break point must be

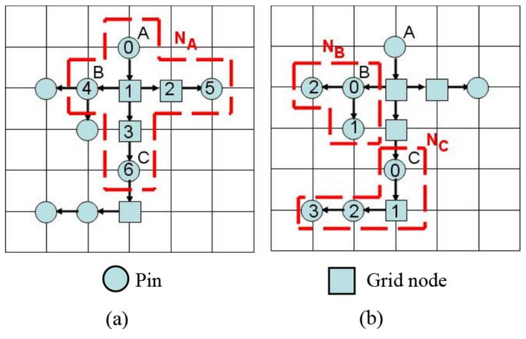


Fig. 12. Net decomposition: (a) A is the root of subnet N_A. B and C are decomposition points; (b) B and C are the roots of sub-nets N_B and N_C, respectively.

an internal node, a pin and a node with a label that exceeds a constant m which determines the subnet size and is set to 3 in this study. Increasing subnet size raises congestion variation and thus assignment quality degrades. On the other hand, if subset size is set to 2, over-fragmented subnets suffer from locally limited view and thus assignment quality also degrades. Fig. 12 shows an example of net decomposition. In Fig. 12(a), node A is chosen as the root of a BFS tree. Nodes B and C are identified as break points, but the node labeled 5 is not a break-point since it is a leaf. Break points B and C are treated as new roots to identify new subnets N_B and N_C, as presented in Fig. 12(b). The score of each subnet is given by

$$\mathrm{Score}(N) = \frac{\alpha}{\mathrm{Length}(N)} + \beta \times \mathrm{PinNum}(N) + \frac{\gamma \times \mathrm{AvgCongestion}(N)}{\mathrm{VarCongestion}(N)} \quad (10)$$

where $\mathrm{AvgCongestion}(N)$ is the average congestion of net N and $\mathrm{VarCongestion}(N)$ is the congestion variation of net N. Layer assignment is then performed on all nets sequentially in the decreasing score order.

Fig. 13 shows different net orderings using the method in COLA and the proposed method. Dark regions represent congested area. In Fig. 13(a), if we use the method in COLA, the net ordering is $N_C \longrightarrow N_B \longrightarrow N_A$. However, if there is another net N_D, as shown in Fig. 13(b), the order of nets N_C and N_D is hard to determine. The proposed approach decomposes the long net N_D into three subnets N_{D1}, N_{D2} and N_{D3} [see Fig. 13(c)] and assigns the subnets in the order of $N_C \longrightarrow N_{D2} \longrightarrow N_B \longrightarrow N_{D1} \longrightarrow N_A \longrightarrow N_{D3}$.

B. Heuristic Algorithm to Preassign a Subnet

We present a heuristic algorithm to efficiently obtain a preassigning solution for each subnet. Initially, the tree structure of a subnet is constructed and then each edge in the tree from bottom (leaf) to top (root) is sequentially assigned to the layer with least assignment cost.

The cost of assigning a tree edge t_e to a layer $-l$ grid edge e_l consists of upstream via cost, downstream via cost and congestion penalty. The fact that upstream tree edges of t_e have not yet been assigned makes it impossible to determine the required number of vias to connect t_e to its upstream tree edges. The default upstream via cost is thus set to zero. However, a pin located at the upstream terminal of t_e explains why the lower bound of upstream via cost is the required number of vias to connect the

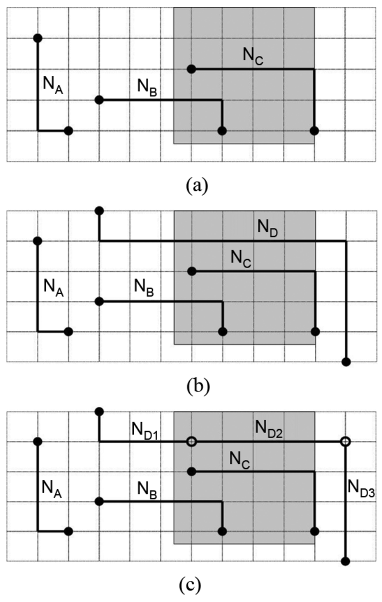


Fig. 13. Comparison between the net ordering in COLA and the proposed method: (a) simple example; (b) the net ordering in COLA; (c) the net ordering in the proposed router.

pin to the terminal on layer l. A pin is said to be located at the terminals of a grid edge if the pin and the terminal of an edge are located at the same position of a 2-D routing graph. The upstream via cost of t_e is thus formulated as follows:

$$uvc(l, t_e) = \begin{cases} |l - PL(ut(t_e))|, & \text{if a pin is located at } ut(t_e) \\ 0, & \text{otherwise} \end{cases} \quad (11)$$

where $ut(t_e)$ denotes the upstream terminal of t_e and $PL(ut(t_e))$ denotes the layer of the pin located at $ut(t_e)$. The left and right figures in Fig. 14 depict 2-D and 3-D routing graphs, respectively, and t_e is assumed to be assigned to layer 2. The yellow circles are pins. In Fig. 14, $PL(ut(t_e))$ equals 1 and, thus, the upstream via cost of t_e is $|2-1| = 1$. The downstream via cost is the required number of vias to connect t_e to its child edges. The fact that the child edges of t_e have been assigned before processing t_e makes it easy to calculate the required number of vias to connect t_e with its child edges. For a pin located at the downstream terminal $(dt(t_e))$ of t_e, the pin must also be considered to involve via number calculation. First, the maximum and minimum layers that child edges have been assigned to must be calculated

$$EL_{\max} = \max_{e \in CE(t_e)} \{EL(e)\} \quad (12)$$

$$EL_{\min} = \min_{e \in CE(t_e)} \{EL(e)\} \quad (13)$$

where $CE(t_e)$ represents the set of t_e's child edges and $EL(e)$ refers to the layer that edge e has been assigned to. The downstream via cost of t_e is thus formulated as follows:

$$dvc(l, t_e) = \begin{cases} \max(l, EL_{\max}, PL_{dt}) - \min(l, EL_{\min}, PL_{dt}), & \text{if a pin is located at } dt(t_e) \\ \max(l, EL_{\max}) - \min(l, EL_{\min}), & \text{otherwise} \end{cases} \quad (14)$$

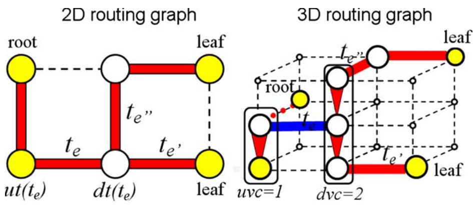


Fig. 14. Example for illustrating how to compute the via cost in heuristic subset preassinging algorithm. The cost of assigning Edge t_e to layer 2 in 3-D routing graph is computed. The upstream tree edge from $ut(t_e)$ to root has not been assigned.

where $dt(t_e)$ denotes the downstream terminal of t_e and PL_{dt} is $PL(dt(t_e))$. In Fig. 14, $CE(t_e) = \{t_{e'}, t_{e''}\}$, $EL(t_{e'}) = 1$, and $EL(t_{e''}) = 3$, $EL_{\max}$, and $EL_{\min}$ are 3 and 1, respectively, and the downstream via cost is 2.

The total cost of assigning t_e to e_l is formulated as follows:

$$ac(e_l, t_e) = \begin{cases} w * p(e_l) + \text{dvc}(l, t_e) + \text{uvc}(l, t_e), & \text{if } e_l \text{ overflows} \\ p(e_l) + w * (\text{dvc}(l, t_e) + \text{uvc}(l, t_e)), & \text{otherwise} \end{cases} \quad (15)$$

where $p(e_l)$ represents the congestion penalty of e_l calculated by (6) and w refers to a weighted factor. For e_l with an overflow, congestion penalty is amplified to avoid increasing the overflow; otherwise, via cost is amplified to minimize the number of vias. Here, weighted factor w is set to 10 000.

Because the cost of assigning each tree edge to each layer is calculated, the time complexity of heuristic algorithm for assigning a net is $O(kE)$, where k denotes the layer number of the 3-D grid graph and E represents the number of tree edges of the assigned net. The heuristic assignment algorithm is very fast to the extent that each subnet in all ISPD benchmarks can be preassigned in one second.

C. Dynamic Programming-Based Layer Assignment With Layer Shifting (DPLALS)

In single net layer assignment, the 2-D routing result is transferred into a tree structure. The solution of COLA is obtained using

$$\text{MVC}_t(v, l) = \begin{cases} \min\limits_{\substack{1 \le l_1 \le k \\ \cdots \\ 1 \le l_q \le k}} \left(\sum_{j=1}^{q} \text{MVC}_{t_j}(v_j, l_j) + vc(v, l, l_1 \cdots l_q) \right), & \text{if } \text{dem}(e_l(j)) < \text{cap}(e_l(j)) \\ \infty, & \text{otherwise} \end{cases}$$

$$vc(v, l, l_1 \cdots l_q) = \max(PL(v), l, l_1 \cdots l_q) - \min(PL(v), l, l_1 \cdots l_q) \quad (16)$$

where v is the root of tree t; v_j is v's child node and t_j is the sub-tree of t rooted at v_j. $\text{MVC}_t(v, l)$ is the minimal cost of the vias of sub-tree t when v is located on layer l under the congestion constraint and q is the number of child nodes of v; k is the highest layer; $e_l(j)$ is the grid edge on layer l that connects v and v_j; $\text{cap}(e_l(j))$ is the capacity of $e_l(j)$ and $\text{dem}(e_l(j))$ is the demand of $e_l(j)$. The total number of vias of a tree with root v on layer l is the sum of the via numbers of its sub-trees and the via cost at v on layer l. Thus $\text{MVC}_t(v, l)$ has two parts; the first is the sum of the costs of the vias on t's sub-trees and the second

```
DPLALS algorithm
Input net N
1. Construct a tree of net N;
2. Bottom-up compute the MVC value of every vertex for each
   layer;
3. Reach root r, find the via-minimum assignment;
4. Top-down layer assignment for each segment, if the destination
   layer have no resource, do layer shifting;
```

Fig. 15. Our DPLALS algorithm flow.

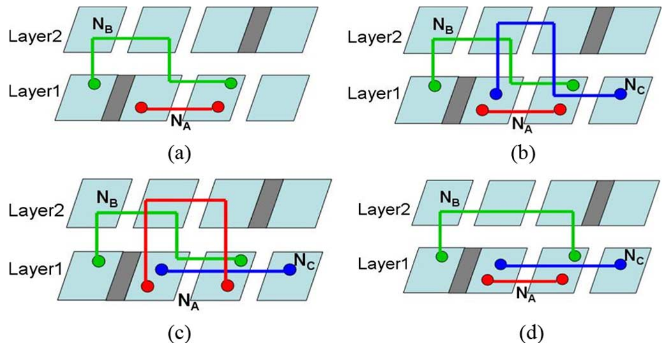


Fig. 16. Example of layer shifting: (a) N_A and N_B are already assigned nets, grey areas represent obstacles; (b) assign N_C to layer 2, $\mathtt{total\ via} = 4$; (c) push the segment of N_A to layer 2, $\mathtt{total\ via} = 4$; (d) push the segment of N_B to layer 2, $\mathtt{total\ via} = 2$.

is the via cost at vertex v. Recursively computing $\mathrm{MVC}_t(v,l)$ yields the assignment solution for tree t. Notably, the solution is found only when $\mathrm{cap}(e_l(j))$ is larger than $\mathrm{dem}(e_l(j))$, if $\mathrm{cap}(e_l(j))$ equals to $\mathrm{dem}(e_l(j))$, no segment will be assigned to $e_l(j)$ because $e_l(j)$ is short of resources.

The proposed layer-shifting scheme solves the layer resource shortage problem for the nets in the later assigning order. The concept of layer shifting is to push a previously assigned net away from its current layer to accommodate a currently assigned net. It is worthy of noting that the number of vias increases once an already assigned net segment in a grid edge is shifted to another grid edge on different layer. To realize this concept, the MVC cost function is modified as follows:

$$\begin{aligned}&\mathrm{MVC}_t(v,l)\\&= \min_{\substack{1\le l_1\le k\\ \cdots\\ 1\le l_q\le k}}\left(\sum_{j=1}^{q}\mathrm{MVLC}_{t_j}(v_j,l_j)+vc(v,l,l_1\cdots l_q)\right)\\&\mathrm{MVLC}_{t'}(v,l)\\&=\begin{cases}\mathrm{MVC}_{t_j}(v_j,l_j), & \text{if } \mathrm{dem}(e_l(j))<\mathrm{cap}(e_l(j))\\ \mathrm{MVC}_{t_j}(v_j,l_j)+LSC(e_l(j)), & \\ \quad \text{if } \mathrm{dem}(e_l(j))=\mathrm{cap}(e_l(j))\&\&\mathrm{cap}(e_l(j))>0 & \\ \infty, & \text{if } \mathrm{cap}(e_l(j))=0\end{cases}\end{aligned} \tag{17}$$

where $\mathrm{LSC}(e_l(j))$ is the increase in the number of vias due to layer shifting and $\mathrm{MVLC}(v_j,l)$ is the minimal cost of vias of layer assignment with layer shifting. In this problem, the maximum q is four.

If $\mathrm{cap}(e_l(j))$ exceeds $\mathrm{dem}(e_l(j))$, then a cost function that is similar to (16) will be adopted. If $\mathrm{cap}(e_l(j))$ equals $\mathrm{dem}(e_l(j))$ and $cap(e_l(j))$ is positive, the other net's segment will be pushed to shift its layer l_j to another layer. Two terms are associated with this situation; the first term indicates the total number of vias of sub-tree t_j at v_j on layer l_j and the second term is the layer shifting cost. To reflect the layer shifting cost, LSC is introduced. In $\mathrm{LSC}(e_l(j))$, all segments currently in $e_l(j)$ are tentatively shifted to other layers that still have resources to seek the lowest via-cost solution and then the layer with the least via-cost is recorded. If $\mathrm{cap}(e_l(j))$ is zero, then no segment will be assigned to $e_l(j)$ so MVLC is set to ∞.

Fig. 15 depicts the flow of the DPLALS algorithm. Initially, the tree structure of net N is constructed; then the MVC and MVLC values of every vertex are computed from the bottom up for each layer. A dynamic programming algorithm identifies the global assignment solution for the whole net when the root is reached. Then, top-down assignment determines the layer of every segment of net N based on the global assignment solution. If no resource exists on the destination grid edge e, the via cost will be reduced by layer shifting the segment based on the solution of $LSC(e)$. The corresponding segment on e will be shifted to another layer.

Fig. 16 presents an example of DPLALS. The rectangles represent grid cells; the gray areas represent obstacles. In Fig. 16(a), N_A and N_B are already assigned. The capacities of layers 1 and 2 are 2 and 1, respectively. Fig. 16(b) and (c) display the results of assigning N_C without layer shifting and with bad layer shifting, in which the required numbers of vias are both 4. The best result in this case is to shift the segment of N_B in layer 1 to layer 2 and the required number of vias is then reduced to 2, as shown in Fig. 16(d).

The time complexity of DPLALS is $\mathrm{O}(k^4V)$, k represents the number of layers of the 3-D grid graph, V refers to the total number of internal nodes and leaves of the assigned net. This algorithm works well in practice because k is a small constant. For ISPD'07 and ISPD'08 benchmarks, k is 6 or 8.

D. Rip-Up and Reassigning

Layer shifting is adopted to seek a local optimum for a segment of a net. Throughout the net, layer shifting may have room

TABLE I
ISPD07 BENCHMARKS

Name	Nets	Grids	v.cap	h.cap
adaptec1	219794	324x324	70	70
adaptec2	260159	424x424	80	80
adaptec3	466295	774x779	62	62
adaptec4	515304	774x779	62	62
adaptec5	867411	465x468	110	110
newblue1	331663	399x399	62	62
newblue2	463213	557x463	110	110
newblue3	551667	973x1256	80	80

TABLE II
ISPD08 BENCHMARKS

Name	Nets	Grids	v.cap	h.cap
bigblue1	282974	227x227	110	110
bigblue2	576816	468x471	52	52
bigblue3	1122340	555x557	148	148
bigblue4	2228903	403x405	204	204
newblue4	636195	455x458	84	84
newblue5	1257555	637x640	88	88
newblue6	1286452	463x464	132	132
newblue7	2635625	488x490	212	212

TABLE III
COMPARISON OF ROUTING RESULTS AMONG FGR, MAIZEROUTER, BOXROUTER, NTHU-ROUTE, ARCHER AND OUR ROUTER

	FGR best [26]		MaizeRouter [28]		BoxRouter 2.0 [25]		NTHU-Route [27]		Archer [24]		NCTU-GR	
name	overflow	wire len(e5)	overflow	wire len(e5)	overflow	wire len(e5)	overflow	wire len(e5)	overflow	wire len(e5)	overflow	wire len(e5)
adaptec1	0	88.02	0	99.7	0	92.04	0	90.56	0	113.8	0	87.45
adaptec2	0	89.96	0	99.53	0	94.28	0	92.17	0	112.56	0	88.25
adaptec3	0	200.14	0	210.19	0	207.41	0	205.04	0	244.08	0	195.73
adaptec4	0	178.9	0	198.81	0	186.42	0	188.43	0	221.57	0	182.34
adaptec5	0	260.53	0	303.34	0	270.41	0	265.03	0	334.09	0	252.88
newblue1	238	90.68	1372	100.38	494	92.94	352	90.91	682	116.08	0	88.39
newblue2	0	129.3	0	139.22	0	134.64	0	136.01	0	166.5	0	130.64
newblue3	38398	163.41	32234	181.68	38958	172.44	31800	168.4	33394	198.77	31802	162.21
ratio	-	1.01	-	1.12	-	1.052	-	1.04	-	1.271	-	1

TABLE IV
COMPARISON OF ROUTING RESULTS AMONG NTHU-ROUTE, NTU-GR, FASTROUTE 4.1 AND OUR ROUTER

	NTHU-Route 2.0 [32]			NTUgr [29]			FastRoute 4.1 [31]			NCTU-GR			untuned NCTU-GR		
name	of	CPU	ISPD08 WL	of	CPU	ISPD08 WL	of	CPU	ISPD08 WL	of	CPU	ISPD08 WL	of	CPU	ISPD08W L
adaptec1	0	314	53.49	0	240	56.01	0	199	53.73	0	230	**53.4**	0	470	53.5
adaptec2	0	89	52.30	0	65	53.59	0	57	52.17	0	87	**51.69**	0	121	51.97
adaptec3	0	378	131.11	0	258	134.14	0	220	130.82	0	239	**130.08**	0	327	130.08
adaptec4	0	119	121.72	0	97	122.93	0	72	121.24	0	132	**120.67**	0	634	120.62
adaptec5	0	755	155.55	0	717	160.51	0	407	155.81	0	541	**154.7**	0	691	155.01
newblue1	0	229	46.52	6	45694	48.53	0	753	46.33	0	213	**45.99**	0	270	46.01
newblue2	0	58	75.71	0	42	76.67	0	49	75.11	0	53	**74.86**	0	64	74.8
newblue3	31454	4121	106.49	**31106**	38072	171.49	31276	1003	108.4	31802	6521	104.28	31872	3920	106.22
newblue4	138	3173	129.89	142	49975	137.28	136	4186	130.46	**134**	2441	126.79	152	2088	127.98
newblue5	0	710	231.66	0	1322	238.96	0	582	230.94	0	899	**230.31**	0	850	231.08
newblue6	0	650	176.95	0	734	185.47	0	520	177.87	0	553	**176.87**	0	720	180.77
newblue7	62	3328	353.57	310	64267	365.07	**54**	53006	353.38	114	4294	338.63	144	4001	338.68
bigblue1	0	418	56.31	0	590	58.45	0	253	56.64	0	378	**55.89**	0	497	58.39
bigblue2	0	403	90.59	0	12315	93.18	0	713	91.98	0	523	**89.4**	0	723	90.01
bigblue3	0	244	130.75	0	315	135.35	0	121	130.04	0	261	**129.66**	0	322	130.2
bigblue4	162	3317	230.75	188	19254	239.56	**130**	5926	230.23	164	3850	223.99	180	3344	228.09
ratio	-	1.05	1.014	-	18.47	1.084	-	1.75	1.014	-	1	1	-	1.42	1.008

for improvement. In this stage, an approach similar to the rip-up and rerouting in the routing stage is proposed. A net is ripped up and reassigned at a time to seek its global (single net) optimum. Layer shifting will then be employed to check if any via can be reduced further.

VI. EXPERIMENTAL RESULTS

The proposed global router was implemented in C/C++ language on an Intel Xeon 3.0 GHz with 16 GB memory. Two sets of benchmark circuits were used in the experiments; one set was of ISPD07 benchmarks [19] and the other was of ISPD08 benchmarks [18]. Tables I and II show the statistics of ISPD07 and ISPD08 benchmarks. Notably, NCTU-GR requires at most 9.7 G memory among all benchmarks.

A. ISPD07 Benchmarks

Table III compares the routing results of the ISPD07 benchmarks for six global routers—BoxRouter 2.0 [25], FGR [26], MaizeRouter [28], Archer [24], NTHU-Route [27], and our router. The table compares the total wire lengths and total overflows. The proposed router can identify overflow-free global paths in seven of eight circuits, while other routers can find only overflow-free solutions in six out of eight circuits.

Moreover, our router yields 1% to 27.1% shorter wirelength than the other routers. Additionally, the best results of FGR are obtained in a runtime limit of 48 hours on a 2.4 GHz Opteron processor, while all of the benchmarks, except for newblue3, are completed in no more than 10 min by the proposed router.

B. ISPD08 Benchmarks

The ISPD08 global routing contest added eight 3-D benchmarks. Table IV lists the routing results of the ISPD08 benchmarks for the proposed router and winners of ISPD08 global routing contest [18], NTHU-Route 2.0 [32], NTUgr [29], and FastRoute 4.1 [31]. All routers adopted different sets of parameters to refine quality and performance. The statistics in Table IV includes total overflow, CPU time for routing in seconds, total wirelength. The untuned NCTU-GR adopted single set of parameters to route all the benchmarks. The untuned NCTU-GR completed 12 out of 16 benchmarks, ran little slower but yielded little less wirelength than that of the router NTHU-Route 2.0 (the winner of ISPD08 global routing contest). On the other hand, NCTU-GR yielded the least wirelength among all routers and ran 1.05 times faster than NTHU-Route 2.0, 18.47 times faster than NTU-GR and 1.75$\times$ faster than FastRoute 4.1. In the benchmarks with overflow, the proposed NCTU-GR performed best among four routers in newblue4 while NTU-GR performed

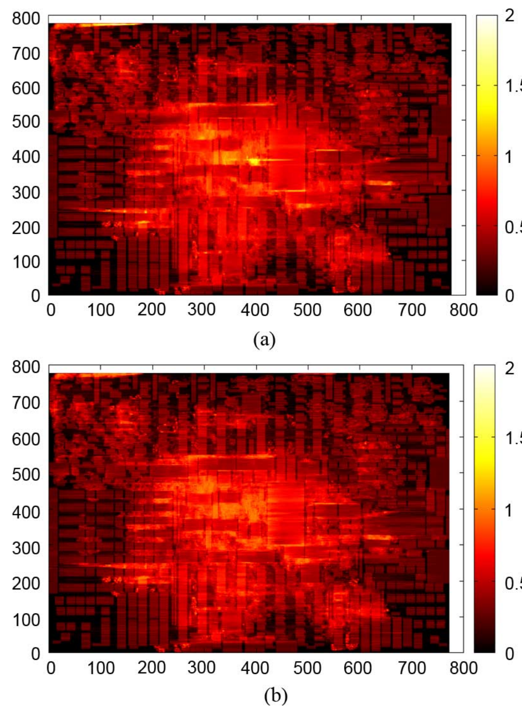


Fig. 17. Congestion map of benchmark adaptec4 in two stages: (a) stage after CFOMR; (b) final stage.

TABLE V
EFFECTIVENESS OF CFOMR

Name	OFB	OF_rate(%)	Time (s)	T_rate(%)
adaptec1	64164	12.7	15.44	3.29
adaptec2	65289	9.79	11.77	9.73
adaptec3	139864	14.31	41.28	12.62
adaptec4	38396	3.33	28.46	4.49
adaptec5	244263	13.41	39.33	5.69
newblue1	17289	5.15	8.41	3.11
newblue2	8627	2.21	14.19	22.17
newblue5	272398	7.73	57.08	6.72
newblue6	202124	11.8	30.87	4.29
bigblue1	100776	17.56	14.09	2.84
bigblue2	79219	15.29	15.88	2.20
bigblue3	69216	3.2	20.78	6.45
average	-	9.70	-	6.97

TABLE VI
ROUTING RESULT WITHOUT APPLYING CFOMR

Name	OFB	WL	Time(s)	WL_ratio	T_ratio
adaptec1	194550	54.12	307	1.011	1.096
adaptec2	125261	52.05	124	1.002	1.024
adaptec3	510639	131.28	368	1.009	1.231
adaptec4	155363	120.65	547	1.000	2.475
adaptec5	545364	158.38	1385	1.021	2.004
newblue1	50981	46.11	281	1.002	1.152
newblue2	49305	27.94	69	1.002	1.078
newblue5	547763	231.74	990	1.003	1.165
newblue6	438940	182.29	1766	1.008	2.453
bigblue1	211635	59.33	1175	1.016	2.364
bigblue2	248868	91.48	1315	1.016	1.819
bigblue3	118359	130.32	323	1.001	1.137
average	-	-	-	1.008	1.583

TABLE VII
ROUTING RESULT USING STAGE-1 COST FUNCTION

Name	OF	WL	Time (s)	WL_ratio	T_ratio
adaptec1	0	52.87	548	0.988	1.957
adaptec2	0	51.53	157	0.991	1.297
adaptec3	0	129.87	361	0.998	1.207
adaptec4	0	120.63	588	1.000	2.61
adaptec5	0	153.6	842	0.991	1.218
newblue1	58	45.98	2742	0.999	11.237
newblue2	0	74.8	61	1.000	0.953
newblue5	0	231.07	899	1.000	1.057
newblue6	0	180.78	733	1.0000	1.018
bigblue1	0	55.89	378	0.957	0.760
bigblue2	36	88.84	3309	0.987	4.576
bigblue3	0	129.5	514	0.995	1.809
average	-	-	-	0.992	2.479

TABLE VIII
ROUTING RESULT USING STAGE-2 COST FUNCTION

Name	OF	WL	Time (s)	WL_ratio	T_ratio
adaptec1	0	53.57	286	1.001	1.021
adaptec2	0	52.01	129	1.001	1.066
adaptec3	0	130.31	307	1.002	1.026
adaptec4	0	120.63	599	1.000	2.710
adaptec5	0	150.34	759	1.001	1.098
newblue1	0	46.20	309	1.004	1.266
newblue2	0	74.86	65	1.001	1.016
newblue5	0	231.21	865	1.001	1.018
newblue6	0	180.87	1365	1.001	1.896
bigblue1	0	58.48	555	1.002	1.117
bigblue2	0	90.48	732	1.005	1.012
bigblue3	0	130.23	294	1.000	1.035
average	-	-	-	1.002	1.273

best in newblue3 and FastRoute 4.1 performed best in newblue7 and bigblue4. The proposed NCTU-GR yielded a shorter wirelength than others in all of the overflow-free benchmarks.

C. Effectiveness of CFOMR

This work chooses 12 overflow-free benchmarks to demonstrate the effectiveness of CFOMR. All statistics are based on the routing results of untuned NCTU-GR. Table V lists the number of overflows before entering the rip-up and rerouting stage (OFB), the percentage of nets with overflow (OF_rate), the run time of CFOMR and the run time percentage of CFOMR (time_rate) in global routing. On average, CFOMR can use only 6.97% of the total run time to complete 90.3% of the nets. Fig. 17 shows the congestion map of horizontal edges of the benchmark adaptec4. Fig. 17(a) shows the congestion map after CFOMR, while Fig. 17(b) shows the final congestion map. According to these figures, the largest portion of the congestion map is determined by CFOMR, while the proposed simulated evolution rip-up and rerouting algorithm affects only the congestion distribution of a few regions.

Table VI summarizes the routing results of the modified routing flow by expelling CFOMR from the proposed routing flow. Statistics include the number of overflows before entering the rip-up and rerouting stage (OFB), routed wirelength (WL), routing time (Time), the ratio of the wirelength in Table VI to the wirelength of untuned NCTU-GR in Table IV (WL_ratio) and the ratio of run time to the run time of untuned NCTU-GR (T_ratio). On average, without using CFOMR, the total runtime and the routed wirelength increase 58.3% and 0.8% respectively, indicating that CFOMR can significantly reduce the total routing time.

TABLE IX
COMPARISON OF LAYER ASSIGNMENT RESULTS BETWEEN COLA AND OUR LAYER ASSIGNMENT

name	FGR [26]						MaizeRouter [28]						BoxRouter 2.0 [25]				
	COLA			Ours			COLA			Ours			COLA		Ours		
	len	via	CPU*	len	via	cpu(s)	len	via	CPU*	len	via	cpu(s)	len	via	len	via	cpu(s)
adaptec1	87.61	51.25	22.98	85.4	49.23	19.58	95.96	56.48	26.81	92.75	53.27	20.21	91.29	54.32	88.71	51.74	19.45
adaptec2	89.54	55.7	22.33	86.92	53.28	20.42	94.64	59.73	25.23	91.24	56.33	20.85	92.62	58.61	89.86	55.85	19.87
adaptec3	199.26	102.01	58.65	194.62	97.74	47.85	205.83	108.46	70.54	199.54	102.17	49.35	206.46	108.41	200.56	102.51	47.71
adaptec4	178.1	87.14	52.44	178.52	87.25	42.36	188.41	98.52	63.76	183.47	93.57	43.38	185.82	96.03	182.27	92.47	40.66
adaptec5	259.43	155.34	75.12	250.21	146.55	68.82	282.37	170.00	87.02	271.39	159.03	77.19	265.7	160.24	257.21	151.75	68.63
newblue1	90.38	65.64	19.87	87.17	62.6	19.57	93.85	68.48	21.12	90.99	65.61	20.69	92.5	67.38	90.16	65.04	18.87
newblue2	129.21	80.96	33.35	127.70	79.7	29.93	136.28	89.78	35.49	131.46	84.97	31.02	134.05	87.38	130.37	83.69	29.06
newblue3	162.25	85.4	45.52	159.73	83.11	40.78	171.64	93.09	62.92	167.58	89.02	43.54	168.65	91.92	165.11	88.37	40.71
ratio	1.022	1.035	1.133	1	1	1	1.033	1.056	1.271	1	1	1	1.027	1.046	1	1	1

* The runtime of BoxRouter 2.0 + COLA is not reported in the paper

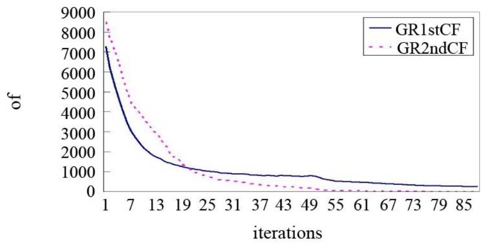
D. *Effectiveness of Two-Stage Cost Function*

To demonstrate the effectiveness of the proposed two-stage cost function, Tables VII and VIII display two sets of routing results (GR1stCF and GR2ndCF) of 12 overflow-free benchmarks using first-stage cost function and second-stage cost function respectively, including overflow (OF), routed wirelength (WL), run time (time), the ratio of the wirelength to the wirelength of untuned NCTU-GR (WL_ratio) and the ratio of run time to the run time of untuned NCTU-GR (T_ratio). Experimental results indicate that the wirelengths of GR1stCF and GR2ndCF are nearly equal to that of this work. However both routers take more runtime to complete the routing since additional routing iterations are required to remove the overflows in congested regions. Moreover, GR1stCF fails to route all overflow-free benchmarks. Although GR2ndCF can generate overflow-free solutions, the wirelength and the runtime is longer since it may over emphasize the effect of historical cost at early iterations. In some of the uncongested benchmarks, GR1stCF produces an overflow-free solution with a lower run time and shorter wirelength than GR2ndCF while GR2ndCF performs better than GR1stCF in congested benchmarks.

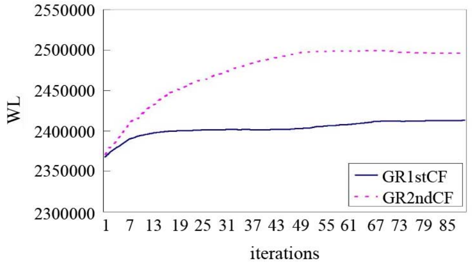
Fig. 18(a) and (b) compare GR1stCF and GR2ndCF in the congested benchmark newblue1 in terms of overflow and wirelength. The x-axis denotes the number of iterations in the rip-up and rerouting stage. In the early iterations of rip-up and rerouting, GR1stCF resolves more overflow with a lower wirelength than GR2ndCF. With a growing number of iterations, GR2ndCF removes more overflow than GR1stCF in terms of the cost of an increasing wirelength. In short, the proposed two-stage cost function scheme adopts GR1stCF in the early routing stage to quickly remove most overflows with shorter wirelength and then employs GR2ndCF to resolve the remaining overflows with additional wirelength.

E. *Comparison Between the Proposed Layer Assignment and COLA*

Table IX compares the layer assignment of the ISPD07 benchmarks between the proposed layer assignment and COLA [17]. Total wirelength, vias and CPU time are reported. Since our layer assignment runs on a different platform from COLA, the runtime of COLA is normalized according to the clock rate



(a)



(b)

Fig. 18. Comparison of GR1stCF and GR2ndCF in newblue1: (a) overflow comparison; (b) wirelength comparison.

ratio. Our layer assignment achieves 3.5% to 5.6% fewer vias and 2.2% to 3.3% shorter wirelength than COLA while the CPU time is even faster than COLA.

VII. CONCLUSION

This work proposes a high-performance congestion-driven 3-D global router. This study also presents two routing methods for 2-D global routing—circular fixed-ordering monotonic routing and simulated evolution-based rip-up and rerouting using a two-stage cost function. It also presents a congestion-relaxed dynamic programming-based layer assignment by using a layer shifting algorithm followed by layer rip-up and reassigning to further reduce the number of vias. Experimental results demonstrate that our router achieves performance similar to the first two winning routers in ISPD 2008 Routing Contest in terms of both routability and wirelength at a 1.05× and 18.47× faster routing speed. Moreover, the proposed layer assignment achieves fewer vias and shorter wirelength than COLA.

REFERENCES

[1] J. Cong, J. Fang, and K.-Y. Khoo, "DUNE: A multilayer gridless routing system," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 20, no. 5, pp. 633–646, May 2001.

[2] J. Cong, J. Fang, M. Xie, and Y. Zhang, "MARS—A multilevel full-chip gridless routing system," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 24, no. 3, pp. 382–394, Mar. 2005.

[3] Y.-L. Li, X.-Y. Chen, and Z.-D. Lin, "NEMO: A new implicit connection graph-based gridless router with multi-layer planes and pseudo-tile propagation," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 26, no. 4, pp. 705–718, Apr. 2007.

[4] G. Xu, L.-D. Huang, D. Z. Pan, and M. D. Wong, "Redundant-via enhanced maze routing for yield improvement," in *Proc. Asia South Pacific Des. Autom. Conf.*, 2005, pp. 1148–1151.

[5] H.-Y. Chen, M.-F. Chiang, Y.-W. Chang, L. Chen, and B. Han, "Full-chip gridless routing considering double-via insertion," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 27, no. 5, pp. 844–857, May 2008.

[6] Y.-R. Wu, M.-C. Tsai, and T.-C. Wang, "Maze routing with OPC consideration," in *Proc. Asia South Pacific Des. Autom. Conf.*, 2005, pp. 18–21.

[7] T.-C. Chen and Y.-W. Chang, "Multilevel full-chip gridless routing with applications to optical proximity correction," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 26, no. 6, pp. 1041–1053, Jun. 2007.

[8] T.-C. Chen, G.-W. Liao, and Y.-W. Chang, "Predictive formulae for OPC with applications to lithography-friendly routing," in *Proc. Des. Autom. Conf.*, 2008, pp. 510–515.

[9] M. Cho, K. Yuan, Y. Ban, and D. Z. Pan, "ELIAD: Efficient lithography aware detailed router with compact post-opc printability prediction," in *Proc. Des. Autom. Conf.*, 2008, pp. 504–509.

[10] M. Cho, D. Z. Pan, H. Xiang, and R. Puri, "Wire density driven global routing for CMP variation and timing," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2006, pp. 487–492.

[11] H.-Y. Chen, S.-J. Chou, S.-L. Wang, and Y.-W. Chang, "Novel wire density driven full-chip routing for CMP variation control," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 28, no. 1, pp. 193–206, Jan. 2009.

[12] M. Pan and C. Chu, "IPR: An integrated placement and routing algorithm," in *Proc. IEEE/ACM Des. Autom. Conf.*, 2007, pp. 59–62.

[13] C. Y. Lee, "An algorithm for path connections and its applications," *IRE Trans. Electron. Comput.*, no. 10, pp. 346–365, Sep. 1961.

[14] 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. 6, pp. 777–790, Jul. 2002.

[15] 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.

[16] L. McMurchie and C. Ebeling, "PathFinder: A negotiation-based performance-driven router for FPGAs," in *Proc. 3rd Int. ACM Symp. Field-Program. Gate Arrays*, 1995, pp. 111–117.

[17] 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.

[18] G. J. Nam, C. Sze, and M. Yildiz, "The ISPD global routing benchmark suite," in *Proc. Int. Symp. Phys. Des.*, 2008, pp. 156–159.

[19] G. J. Nam, M. Yildiz, Z. D. Pan, and P. H. Madden, "ISPD placement contest updates and ISPD 2007 global routing contest," in *Proc. Int. Symp. Phys. Des.*, 2007, pp. 167–1167.

[20] Y.-L. Lin, Y.-C. Hsu, and F.-S. Tsai, "Silk: A simulated evolution router," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 8, no. 10, pp. 1108–1114, Oct. 1989.

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

[22] M. Cho and D. Pan, "Boxrouter: A new global router based on box expansion and progressive ILP," in *Proc. Des. Autom. Conf.*, 2006, pp. 373–378.

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

[24] M. M. Ozdal and M. D. F. Wong, "ARCHER: A history-driven global routing algorithm," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2007, pp. 488–495.

[25] M. Cho, K. Lu, K. Yuan, and D. Z. Pan, "Boxrouter 2.0: Architecture and implementation of a hybrid and robust global router," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2007, pp. 503–508.

[26] J. A. Roy and I. L. Markov, "High-performance routing at the nanometer scale," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2007, pp. 496–502.

[27] 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.

[28] M. D. Moffitt, "Maizerouter: Engineering an effective global router," in *Proc. Asia South Pacific Des. Autom. Conf.*, 2008, pp. 232–237.

[29] C. H. Hsu, H. Y. Chen, and Y. W. Chang, "Multi-layer global routing considering via and wire capacities," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2008, pp. 350–355.

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

[31] 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.

[32] Y. J. Chang, Y. T. Lee, and T. C. Wang, "NTHU-Route 2.0: A fast and stable global router," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2008, pp. 338–343.

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

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

[35] C. Chu and Y. Wong, "Fast and accurate rectilinear steiner minimal tree algorithm for VLSI design," in *Proc. Int. Symp. Phys. Des.*, 2005, pp. 28–35.

[36] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein*, In Introduction to Algorithms*. Boston, MA: The MIT Press, 2001.

[37] N. J. Naclerio, S. Masuda, and K. Nakajima, "The via minimization problem is np-complete," *IEEE Trans. Comput.*, vol. 38, no. 11, pp. 1604–1608, Nov. 1989.

Ke-Ren Dai received the B.S. and M.S. degrees in computer and information science from National Chiao Tung University, Hsinchu, Taiwan, in 2005 and 2007, respectively, where he is currently pursuing the Ph.D. degree in computer science.

His research interests include global routing and placement.

Wen-Hao Liu received the B.S. degree in computer and information science from National Chiao Tung University, Hsinchu, Taiwan, in 2008, where he is pursuing the Ph.D. degree in computer science.

His research interests include global routing and clock tree synthesis.

Yih-Lang Li (M'04) received the B.S. degree in nuclear engineering and the M.S. and the Ph.D. degrees in computer science from the National Tsing Hua University, Hsinchu, Taiwan, in 1987, 1990, and 1996, respectively.

In February 2003, he joined the faculty of the Department of Computer Science, National Chiao Tung University (NCTU), where he is currently an Associate Professor. Prior to joining the faculty of NCTU, from 1995 to 1996 and from 1998 to 2003, he was a Software Engineer and an Associate Manager with Springsoft Corporation, Hsinchu, Taiwan, where he was heavily involved in the development of verification and synthesis tools for custom-based layout. His research interests include physical synthesis, parallel architecture, and VLSI testing.