Congestion-Constrained Layer Assignment for Via Minimization in Global Routing

Tsung-Hsien Lee and Ting-Chi Wang

Abstract—In this paper, we study the problem of layer assignment for via minimization, which arises during multilayer global routing. In addressing this problem, we take the total overflow and the maximum overflow as the congestion constraints from a given one-layer global routing solution and aim to find a layer assignment result for each net such that the via cost is minimized while the given congestion constraints are satisfied. To solve the problem, we propose a polynomial-time algorithm which first generates a net order and then performs layer assignment one net at a time according to the order using dynamic programming. Our algorithm is guaranteed to generate a layer assignment solution satisfying the given congestion constraints. We used the six-layer benchmarks released from the ISPD'07 global routing contest to test our algorithm. The experimental results show that our algorithm was able to improve the contest results of the top three winners MaizeRouter, BoxRouter, and FGR on each benchmark. As compared to BoxRouter 2.0 and FGR 1.1, which are newer versions of BoxRouter and FGR, our algorithm respectively produced smaller via costs on all benchmarks and half the benchmarks. Our algorithm can also be adapted to refine a given multilayer global routing solution in a net-by-net manner, and the experimental results show that this refinement approach improved the via costs on all benchmarks for FGR 1.1.

Index Terms—Global routing, layer assignment, physical design, via minimization.

I. INTRODUCTION

AS very large scale integration (VLSI) technology advances, multilayer routing has become a critical problem in physical design. Routing is normally conducted in the following two phases: global routing [1]–[4] and detailed routing [5], [6]. In this paper, we focus on global routing.

For multilayer global routing, there are two main approaches. One is to route all nets directly on the multilayer solution space [7], [8]. We use Fig. 1 to demonstrate this approach. Fig. 1(a) shows a three-layer global routing instance, where each layer is partitioned into tiles, and the boundary between two adjacent tiles on the same layer is associated with a capacity indicating the number of available routing tracks. Fig. 1(b) shows the corresponding grid graph where each vertex denotes a tile and each edge denotes a boundary or a via. Fig. 1(c) shows a global routing result in the three-layer grid graph for a net consisting
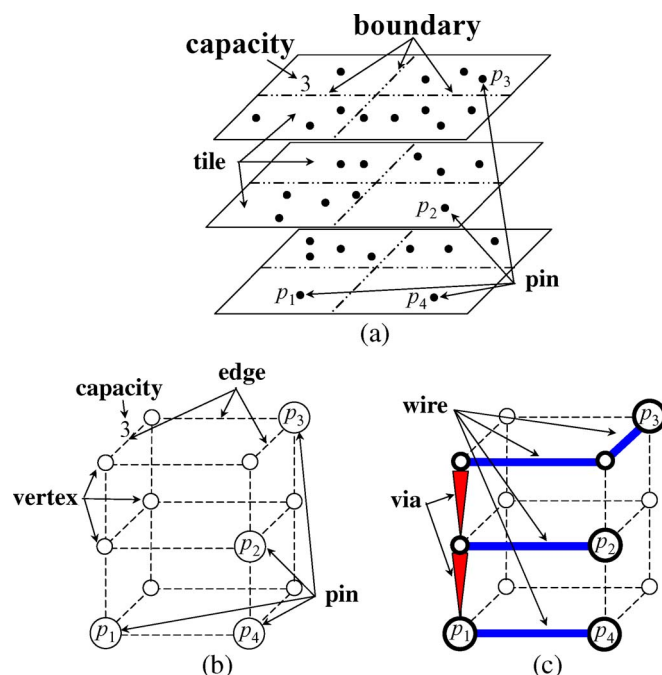
The authors are with the Department of Computer Science, National Tsing Hua University, Hsinchu 300, Taiwan, R.O.C. (e-mail: tsunghsienlee@gmail.com; tcwang@cs.nthu.edu.tw).

Fig. 1. One approach for multilayer global routing.

of four pins p_1, p_2, p_3, and p_4. Because this approach directly generates a multilayer global routing result, it can take the via cost into account during construction. However, this method may cost too much CPU time with a large problem size.

The other approach is to first compress a multilayer grid graph into a one-layer grid graph, then use a one-layer router to solve the one-layer global routing problem, and finally perform layer assignment to assign each wire in the multilayer grid graph [8]–[11]. For example, the three-layer grid graph in Fig. 2(a) is compressed into the one-layer grid graph in Fig. 2(b), where each vertex, edge, and pin in Fig. 2(a) is projected into the bottom layer. The edges corresponding to vias disappear in the one-layer grid graph, and the projected pins of a net falling in the same vertex are merged into one pin [e.g., p_2 and p_4 in Fig. 2(a) become p'_2 in Fig. 2(b)]. The capacity of each edge in the one-layer grid graph is obtained by accumulating the corresponding edge capacities in the three-layer grid graph. Fig. 2(c) shows a global routing result in the one-layer grid graph for a net consisting of three pins p'_1, p'_2, and p'_3. Fig. 2(d) shows the corresponding global routing result in the original three-layer grid graph after layer assignment. This approach can take advantage of many current full-fledged one-layer routers, e.g., [2]–[4], and use an affordable run time to generate an initial one-layer routing result; however, a layer assignment algorithm is still required to
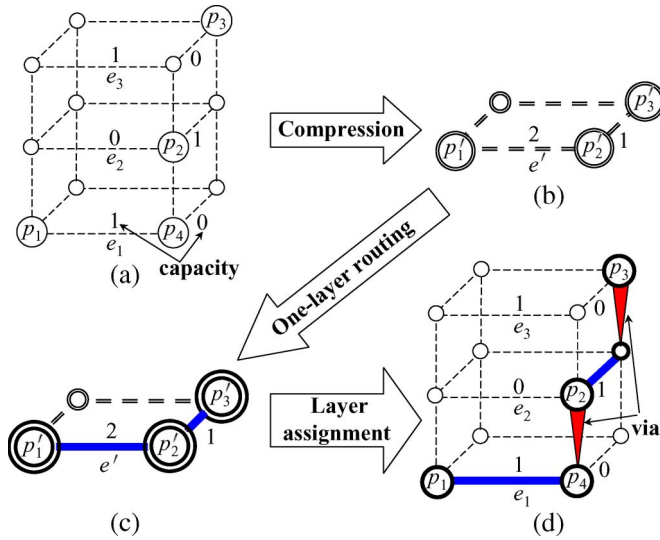
Fig. 2.   Another approach for multilayer global routing.

complete the whole process. In fact, layer assignment preserves the wirelength and the topology of the initial one-layer routing result and provides flexibility for the optimization of delay [12], crosstalk [13], [14], and vias [8], [9]. Therefore, layer assignment is one of the key steps for the success of this approach.

Via is a device for connecting wires in a net which are distributed to different layers in multilayer routing. However, vias should be used only when they are necessary, as they not only degrade the reliability and the performance of a design but also increase the manufacturing cost. As a result, it is desirable to minimize the via cost in multilayer routing. Recently, ISPD'07 held a global routing contest and released several two- and six-layer global routing benchmarks [15]. Since then, several new multilayer global routers, e.g., [8]–[11], have been reporting their results on the ISPD'07 benchmarks. Most of these global routers adopt the second approach, as shown in Fig. 2, and their layer assignment steps apply either greedy heuristics [8] or time-consuming integer linear programming methods [9] to minimize the via cost.

Motivated by the ISPD'07 global routing contest, we study, in this paper, the problem of layer assignment for via cost minimization, which arises during global routing. In addressing this problem, we take the total overflow and the maximum overflow as the congestion constraints[1] from a given one-layer global routing solution and aim to find a layer assignment result for each net such that the given congestion constraints are satisfied and the resultant via cost is minimized. To solve this problem, we propose a polynomial-time algorithm which first generates a net order based on the given one-layer routing result and then solves the layer assignment problem one net at a time according to the net order using dynamic programming [16]. Our algorithm is guaranteed to generate a layer assignment result satisfying the given congestion constraints.

The experiments on the ISPD'07 six-layer benchmarks indicate that our layer assignment algorithm was able to improve the contest results of the top three winners MaizeRouter,

BoxRouter, and FGR by reducing their via costs on each benchmark. As compared to BoxRouter 2.0 [9] and FGR 1.1 [17], which are newer versions of BoxRouter and FGR, our layer assignment algorithm respectively produced smaller via costs on all benchmarks and half the benchmarks. As for the run time, our algorithm was efficient, taking 0.5–2 min to generate the result for each benchmark. In addition, our algorithm can also be adapted to refine a given multilayer global routing solution in a net-by-net manner, and the experimental results show that this refinement approach improved the via costs on all benchmarks for FGR 1.1.

The remainder of this paper is organized as follows. Section II presents the formulation of the layer assignment problem. Section III gives the overview of our algorithm. The details of our algorithm are described in Sections IV–VI. The experimental results are provided in Section VII, and the conclusion is drawn in Section VIII.

II. PROBLEM FORMULATION

A. Preliminaries

Let $G^k = (V^k, E^k)$ denote a k-layer grid graph [see Fig. 2(a) for a three-layer grid graph], where $k \geq 2$ is a positive integer specifying the number of available routing layers. We assume that the k metal layers are numbered as $1, 2, \ldots, k$ from the bottom to the top. Each vertex v_i^k in V^k corresponds to a tile. The edge set E^k can be divided into two disjoint sets, E_b^k and E_v^k. Each edge in E^k, if it connects two adjacent tile vertices on the same layer, is called a *boundary edge* and is in E_b^k; otherwise, it is called a *via edge* and is in E_v^k. Each boundary edge e_i^k in E_b^k is denoted by $(v_{i,1}^k, v_{i,2}^k)$, indicating that the two vertices $v_{i,1}^k$ and $v_{i,2}^k$ are connected by e_i^k; in addition, e_i^k is also associated with the capacity $c(e_i^k)$ [see the numbers next to the edges in Fig. 2(a)], denoting the number of available tracks (i.e., the maximum number of nets that can pass through without causing overflow). Each edge e^k in E^k is also given a wirelength denoted by $\text{len}(e^k)$. We assume that each boundary edge has the same wirelength, and so does each via edge. However, the wirelength of a boundary edge may be different from that of a via edge.

Let N^k denote a set of unrouted nets in G^k. For each net n_i^k in N^k, n_i^k is denoted by the subset P_i^k of V^k such that, for each pin $p_{i,j}^k$ in n_i^k, $p_{i,j}^k$ is located within a tile of P_i^k. For a tile in P_i^k, if it contains more than one pin in n_i^k, we will merge the pins into one and therefore assume that the tile only contains one pin in n_i^k. This is because the intraconnection of a net within a tile need not be considered in global routing. A global routing solution for n_i^k is a connected subgraph $T_i^k = (V_i^k, E_i^k)$ of G^k such that $P_i^k \subseteq V_i^k$. The wirelength of T_i^k is defined as $\sum_{e^k \in E_i^k} \text{len}(e^k)$. A global routing solution for (G^k, N^k), denoted by S^k, is a collection of T_i^k's for all nets in N^k.

Let $G^1 = (V^1, E^1)$ denote the one-layer grid graph compressed from G^k [see Fig. 2(b) for an example]. Each edge e_i^1 in E^1, denoted by $(v_{i,1}^1, v_{i,2}^1)$, is a boundary edge (note that all via edges disappear in E^1), and $c(e_i^1)$ [see the numbers next to the edges in Fig. 2(b)] is the summation of the capacities of the boundary edges in $\text{ES}(e_i^1)$, where $\text{ES}(e_i^1)$ denotes the set of edges in E_b^k with their two connecting vertices projected to $v_{i,1}^1$

---

[1]Total overflow and maximum overflow will be defined in Section II.

TABLE I
NOTATIONS

Notation	Description
k	The number of metal layers (≥ 2)
$G^k = (V^k, E^k)$	A k-layer grid graph
$G^1 = (V^1, E^1)$	The one-layer grid graph compressed from G^k
N^k	A set of nets in G^k
N^1	The set of nets obtained by projecting N^k to G^1
S^k	A k-layer global routing solution of (G^k, N^k)
S^1	A one-layer global routing solution of (G^1, N^1)
TO(S)	The total overflow of a global routing solution S
MO(S)	The maximum overflow of a global routing solution S
$T_i^k = (V_i^k, E_i^k)$	A k-layer routed net in S^k
$T_i^1 = (V_i^1, E_i^1)$	A one-layer routed net in S^1
$c(e), d(e), o(e)$	The capacity, the demand, and the overflow of a boundary edge e
ES(e)	The set of k-layer boundary edges which are compressed into the one-layer boundary edge e
PS(p)	The set of pins which are projected to the one-layer pin p

and $v_{i,2}^1$ [e.g., in Fig. 2(b), ES(e') is $\{e_1, e_2, e_3\}$ in Fig. 2(a), and $c(e') = c(e_1) + c(e_2) + c(e_3) = 1 + 0 + 1 = 2$].

Let N^1 be the set of nets obtained by projecting N^k to the one-layer grid graph along with the layer axis. For each net n_i^1 in N^1, n_i^1 is denoted by the subset P_i^1 of V^1. For each pin $p_{i,j}^1$ in P_i^1, PS($p_{i,j}^1$) denotes the set of pins in P_i^k which is projected into the vertex containing $p_{i,j}^1$ (e.g., in Fig. 2, PS(p_1') is $\{p_1\}$, PS(p_2') is $\{p_2, p_4\}$, and PS(p_3') is $\{p_3\}$). A global routing solution for n_i^1 is denoted by the connected subgraph $T_i^1 = (V_i^1, E_i^1)$ of G^1 such that $P_i^1 \subseteq V_i^1$, and a global routing solution for (G^1, N^1), denoted by S^1, is a collection of T_i^1's for all nets in N^1.

Given an S^k, we can define the following terms. The total wirelength is the summation of the wirelengths of all T_i^k's. For each boundary edge e^k, the routing demand $d(e^k)$ is defined as the number of nets passing through e^k, and the overflow $o(e^k)$ (i.e., the excessive routing demand) is defined as $d(e^k) - c(e^k)$ if $d(e^k) > c(e^k)$, and zero otherwise. The total overflow TO(S^k) is defined as the summation of the overflows of all boundary edges, and the maximum overflow MO(S^k) is defined as the maximum overflow among all boundary edges. For an S^1, we can also define the aforementioned terms similarly. A typical multilayer (or one-layer) global router aims at minimizing the total overflow, the maximum overflow, and the total wirelength.

Some of the aforementioned notations are summarized in Table I, and they will be used throughout the rest of this paper.

B. Problem Definition

We now formally define our layer assignment problem as follows. The problem is given the following inputs: a k-layer grid graph $G^k = (V^k, E^k)$, a set N^k of unrouted nets in G^k, the one-layer grid graph $G^1 = (V^1, E^1)$ compressed from G^k, the set N^1 of nets obtained by projecting N^k to G^1, and a one-layer global routing solution S^1 of (G^1, N^1). The problem asks to transform each T_i^1 in the given one-layer global routing solution S^1 into its counterpart T_i^k in a k-layer global routing solution S^k such that, when projecting T_i^k to G^1, it results in
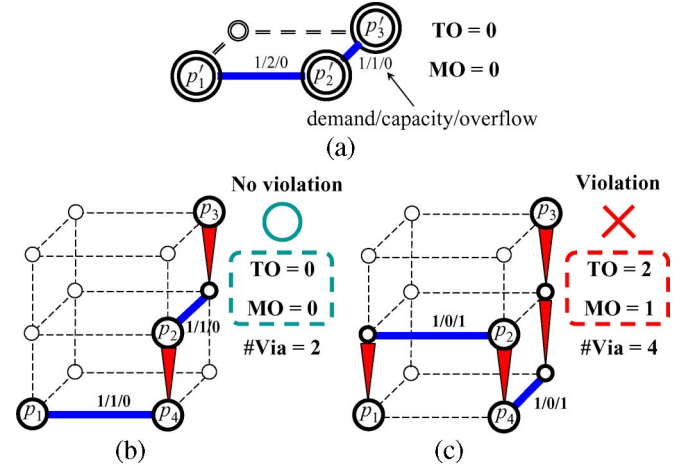

Fig. 3. Instance of the layer assignment problem and two possible solutions, where TO and MO denote total overflow and maximum overflow, respectively.

the same routing topology as T_i^1; in other words, the problem asks to assign each edge of T_i^1 to a layer j, where $1 \leq j \leq k$. Furthermore, the problem also requires S^k to satisfy the following two congestion constraints:

1) total overflow constraint

$$\text{TO}(S^k) = \text{TO}(S^1);$$

2) maximum overflow constraint

$$\text{MO}(S^k) = \lceil \text{MO}(S^1)/k \rceil.$$

The objective of the problem is to produce a feasible S^k such that its via cost (i.e., the total amount of vias) is minimized.

The two congestion constraints mentioned earlier require a layer assignment method that simultaneously maintains the same total overflow and reduces the maximum overflow to $\lceil 1/k \rceil$ times the one-layer counterpart. As shall be proved in Section V-C, TO(S^1) and $\lceil \text{MO}(S^1)/k \rceil$ are the *lower bounds* for TO(S^k) and MO(S^k) of S^k obtained from S^1, respectively, by any layer assignment method. Moreover, we shall prove in Section VI-A that our layer assignment algorithm is always able to generate an S^k achieving the lower bounds for any problem instance. Moreover, we would also like to point out that the classical layer assignment problem (e.g., [18]–[21]) is different from our layer assignment problem in the sense that it handles via minimization in the post-layout stage and does not consider the two congestion constraints.

To simplify the presentation of our algorithm, we do not make any assumption about the preferred routing direction for each layer in the layer assignment problem. Nevertheless, if each layer comes to have a preferred routing direction, we can still easily modify the problem formulation and our algorithm to account for this case; the details will be explained in Section VI-B. In addition, in our layer assignment problem, we adopt the same assumption as the ISPD'07 global routing contest [15] that the capacity of each via edge is infinite.

Fig. 3 shows an instance of the layer assignment problem and two possible layer assignment solutions. Fig. 3(a) shows the given one-layer routing solution [identical to Fig. 2(c)] for the three-layer routing instance shown in Fig. 2(a). In Fig. 3(a),

both the total and maximum overflows of the one-layer routing solution are zero, and thus, the total and maximum overflows of any feasible three-layer routing solution must also be zero. Fig. 3(b) and (c) shows two possible layer assignment solutions, and only the one in Fig. 3(b) gives a feasible solution which satisfies the congestion constraints and achieves less via cost; the other shown in Fig. 3(c) not only violates both congestion constraints but also causes more via cost.

III. ALGORITHM OVERVIEW

In this section, we give an overview of our algorithm which is called COngestion-constrained Layer Assignment (COLA) algorithm. COLA generates a net order and, following the net order, chooses the T_i^1's in S^1 one at a time. For each T_i^1, it generates the corresponding T_i^k through layer assignment. In summary, COLA mainly contains the following two parts.

1) Net order determination

    The net order has a direct influence on the utilization of routing resources, so it is one of the key parts of COLA. We propose a method to determine a proper net order which aims at maximizing the utilization of limited routing resources.

2) Single-net layer assignment considering congestion constraints

    To solve the problem of layer assignment considering congestion constraints for a given net, we propose an algorithm called SOLA+APEC which combines Single-net Optimal Layer Assignment (SOLA) and Accurate and Predictable Examination for Congestion constraints (APEC). SOLA determines an optimal layer assignment result without considering congestion constraints for a given net, whereas APEC can detect and prevent any congestion constraint violation in advance. It is worth noting that a dynamic-programming-based algorithm is proposed in [19] to handle the classical layer assignment problem which considers via minimization in the post-layout stage. SOLA also uses a similar dynamic programming technique, but because we consider a different layer assignment problem which arises during global routing, SOLA does not suffice to handle the congestion constraints alone. Instead, SOLA needs to be combined with APEC, thus making SOLA+APEC different from the one in [19].

The details of COLA are described in Sections IV–VI.

IV. NET ORDER DETERMINATION

Before executing COLA, for each k-layer boundary edge e^k, we have $d(e^k) = 0$ (and $o(e^k) = 0$, too), and for each one-layer boundary edge e^1, its $d(e^1)$ and $o(e^1)$ are given by the one-layer routing solution S^1.

Due to the limited routing resources, a net processed earlier has a higher chance to occupy more proper routing resources and get a better via minimization result. Hence, a proper net order will maximize the utilization of routing resources for all nets. For each one-layer routed net T_i^1, we use three factors to calculate $\text{Score}(T_i^1)$ for determining the net order. After all the calculations of $\text{Score}(T_i^1)$'s are produced, the net order

TABLE II
THREE FACTORS OF SCORE FUNCTION

Factor	Definition
$\text{Length}(T_i^1)$	The total wirelength of T_i^1 (i.e., $\sum_{e^1 \in E_i^1} \text{len}(e^1)$).
$\text{PinNum}(T_i^1)$	The number of pins of T_i^1 (i.e., $|P_i^1|$).
$\text{AvgDensity}(T_i^1)$	The average net density of T_i^1 (i.e., $\sum_{e^1 \in E_i^1} d(e^1) / \sum_{e^1 \in E_i^1} c(e^1)$).

is generated by sorting $\text{Score}(T_i^1)$'s in decreasing order. The definitions of these three factors are listed in Table II.

With these three factors, $\text{Score}(T_i^1)$ is defined as

$$\text{Score}\left(T_i^1\right) = \frac{\alpha}{\text{Length}\left(T_i^1\right)} + \beta \times \text{PinNum}\left(T_i^1\right)$$
$$+ \gamma \times \text{AvgDensity}\left(T_i^1\right)$$

where α, β, and γ are user-specified parameters. In our current implementation, α and β are set to be equal, whereas γ is much smaller than α and β; we have empirically observed that such a parameter setting would help to generate better layer assignment results. The γ is basically used for tie-breaking.

In the following, we explain the relationship between $\text{Score}(T_i^1)$ and the three factors. First, we explain why $\text{Score}(T_i^1)$ is inversely proportional to $\text{Length}(T_i^1)$. For a T_i^1 with larger $\text{Length}(T_i^1)$, if its optimal layer assignment result occupies too many popular routing resources, the result may deteriorate the layer assignment results of the nets processed later; on the other hand, the number of possible layer assignment results of T_i^1 is proportional to $\text{Length}(T_i^1)$, which means that a T_i^1 with larger $\text{Length}(T_i^1)$ may find its optimal layer assignment result even if T_i^1 is a net processed later. Second, we explain why $\text{Score}(T_i^1)$ is proportional to $\text{PinNum}(T_i^1)$. For a T_i^1 with larger $\text{PinNum}(T_i^1)$, its number of possible layer assignment results may be more restricted by the number of its pins because the pins are like checkpoints for routing connectivity; therefore, we prefer to process it earlier. Third, we explain why $\text{Score}(T_i^1)$ is proportional to $\text{AvgDensity}(T_i^1)$. For a T_i^1 with larger $\text{AvgDensity}(T_i^1)$, T_i^1 passes through more congested regions such that its layer assignment result may be worse if T_i^1 is a net processed later.

V. SINGLE-NET LAYER ASSIGNMENT CONSIDERING CONGESTION CONSTRAINTS

In this section, we describe the algorithm SOLA+APEC, which combines SOLA and APEC, for the problem of single-net layer assignment considering congestion constraints.

A. Preprocessing

For each one-layer routed net T_i^1, we assume that its topology is a *tree*; otherwise, a preprocessing procedure is applied to remove all cycles inherent in T_i^1 and transform the topology of T_i^1 into a tree.

There are many ways to perform cycle removal, and our algorithm arbitrarily removes a set of edges from each cycle for simplicity. We use Fig. 4 as an example to illustrate cycle removal. Fig. 4(a) shows a one-layer routed net which contains five pins (i.e., p_1, p_2, p_3, p_4, and p_5) and a cycle. Fig. 4(b) shows
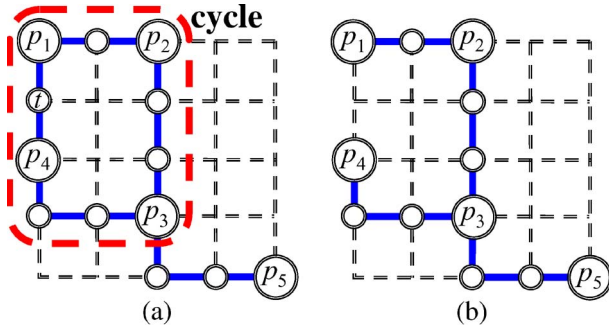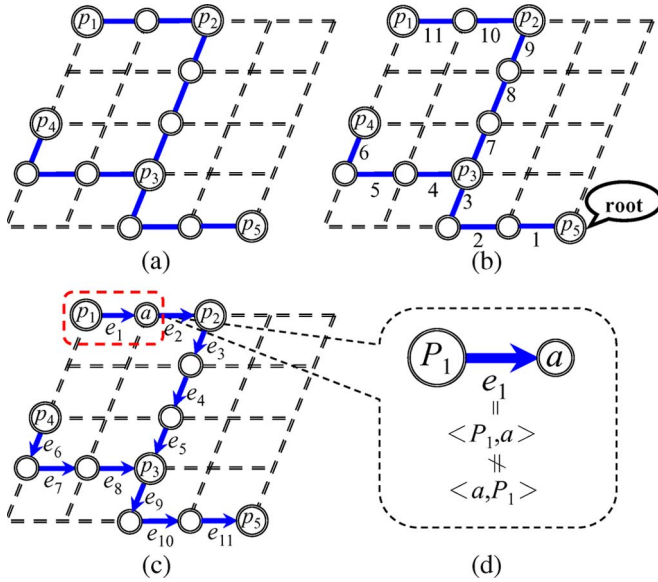
Fig. 4. Cycle removal of a net.

Fig. 5. Edge order example.

Fig. 6. Illustration of SOLA.

the resulting tree after the edge set $\{(p_1, t), (t, p_4)\}$ is removed. After the cycle removal procedure, not only the connectivity of each T_i^1 is retained but also the total or the maximum overflow of the resulting S^1 is likely reduced due to the edges which are absent after cycle removal.

B. SOLA

In the following, we describe our dynamic-programming-based algorithm SOLA for the problem of layer assignment without considering congestion constraints for a single net.[2]

For each one-layer routed net T_i^1, SOLA processes its edges one at a time. To determine an edge order, SOLA arbitrarily selects a vertex of T_i^1 as the root and uses a graph traversal algorithm (i.e., breadth-first search (BFS) or depth-first search (DFS) [16]) to get a tree traversal order as well as the reverse edge order. The determination of edge order is shown in Fig. 5.
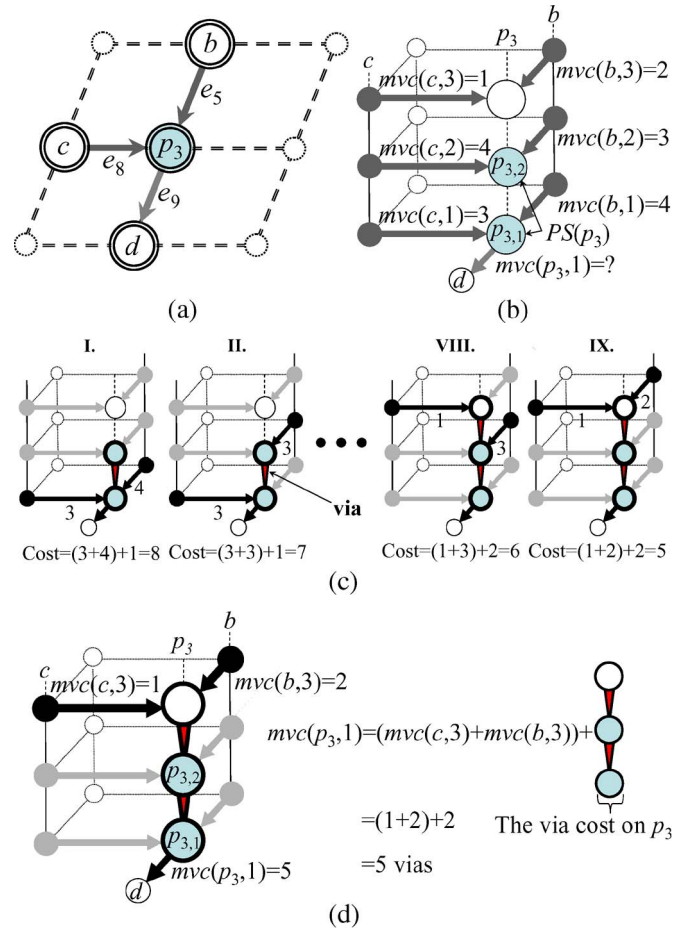
Fig. 5(a) shows a one-layer routed net which is identical to that in Fig. 4(b). In Fig. 5(b), p_5 is selected as the root, and DFS is used to traverse the tree. The number next to each edge represents its tree traversal order. In Fig. 5(c), for each edge, its edge order is assigned according to the reverse tree traversal order, and it is transformed into a directed edge. For example, Fig. 5(d) shows the edge connecting p_1 and a, whose original tree traversal order is 11 (i.e., the last one) and is renamed as e_1 (i.e., the first one); the subscript number of each edge represents its edge order. For each directed edge e_i, it is denoted by $\langle v_{i,1}, v_{i,2} \rangle$, where $v_{i,1}$ is a child of $v_{i,2}$ and $v_{i,2}$ is the parent of $v_{i,1}$, in the tree.

To facilitate the explanation of the recurrence relations adopted by SOLA, we first define the following three terms for each vertex v of T_i^1: $par(v)$, $ch(v)$, and $par\_e(v)$. $par(v)$ is the parent of v, $ch(v)$ is the set of children of v, and $par\_e(v)$ is the directed edge $\langle v, par(v) \rangle$. For example, for vertex a in Fig. 5(c), $par(a)$ is p_2, $ch(a)$ is the vertex set $\{p_1\}$, and $par\_e(a)$ is e_2. With these definitions, we then define $mvc(v, r)$ as the minimum via cost of the subtree rooted at vertex v when $par\_e(v)$ is assigned to layer r. $mvc(v, r)$ can be divided into the following two parts: the sum of $mvc(v_j, r_j)$'s for all $v_j \in ch(v)$ and the via cost on v. We use Fig. 6 as an example to show the steps of finding $mvc(v, r)$.

Fig. 6(a) shows a part of the one-layer routing tree of Fig. 5(c), which shows the subtree rooted at p_3 with two child vertices (i.e., b and c), a parent vertex (i.e., d), and three

edges (i.e., e_5, e_8, and e_9). Fig. 6(b) shows the corresponding k-layer routing graph, and we assume that $k = 3$. In addition, both $\mathrm{mvc}(b, r)$ and $\mathrm{mvc}(c, r')$ for $r = 1 \sim k$ and $r' = 1 \sim k$ have been computed, as shown in Fig. 6(b). Assuming that we want to find $\mathrm{mvc}(p_3, 1)$, Fig. 6(c) shows all combinations of $\mathrm{mvc}(b, r)$ and $\mathrm{mvc}(c, r')$ for $r = 1 \sim 3$ and $r' = 1 \sim 3$ (a total of $3 \times 3 = 9$ combinations). For each combination, we place vias on p_3 for routing connectivity; the vias on p_3 are used for connecting the following three types of objects:

1) the edges $\langle b, p_3 \rangle$ on layer r and $\langle c, p_3 \rangle$ on layer r';
2) the edge $\langle p_3, d \rangle$ on layer 1;
3) the pins on p_3 [i.e., $p_{3,1}$ and $p_{3,2}$ in Fig. 6(b)]; these pins are obtained from those in $\mathrm{PS}(p_3)$.

For routing connectivity, the vias on p_3 are placed between the lowest layer and the highest layer among these three types of objects, and the via cost on p_3 is calculated by the difference from the highest layer to the lowest one. For each combination, after placing vias on p_3, we can calculate its total via cost by the sum of $\sum_{v_j \in \mathrm{ch}(p_3)} \mathrm{mvc}(v_j, r_j)$ and the associated via cost on p_3 [see Fig. 6(c)]. $\mathrm{mvc}(p_3, 1)$, as shown in Fig. 6(d), is thus calculated by the minimum among all the results in Fig. 6(c). Although Fig. 6 only shows the steps to find $\mathrm{mvc}(p_3, 1)$, the steps to find $\mathrm{mvc}(p_3, r)$ for $r = 2 \sim k$ are similar; only the assigned layer of $\mathrm{par\_}e(p_3)$, which is determined by r, is different.

Consequently, we can derive the following recurrence relation for $\mathrm{mvc}(v, r)$:

$$\mathrm{mvc}(v, r) = \min_{\substack{1 \leq r_1 \leq k \\ \cdots \\ 1 \leq r_q \leq k}} \left(\sum_{j=1}^{q} \mathrm{mvc}(v_j, r_j) + \mathrm{vc}(v) \right)$$

where

$$
\begin{aligned}
q &= |\mathrm{ch}(v)| \\
\mathrm{ch}(v) &= \{v_1, \ldots, v_q\} \\
\mathrm{vc}(v) &= \max(\Delta) - \min(\Delta), \\
\Delta &= \{\mathrm{max\_}l(v), \mathrm{min\_}l(v), r, r_1, \ldots, r_q\}. \quad (1)
\end{aligned}
$$

$\mathrm{vc}(v)$ is the via cost on v, which is the difference from the highest layer to the lowest one in Δ. In Δ, $\mathrm{max\_}l(v)(\mathrm{min\_}l(v))$ is the highest (lowest) layer of all the pins in $\mathrm{PS}(v)$, r is the assigned layer of $\mathrm{par\_}e(v)$, and r_j is the assigned layer of $\mathrm{par\_}e(v_j)$ for each $v_j \in \mathrm{ch}(v)$.

Note that the aforementioned recurrence relation is valid only for the case where v is an internal node but not the root. For the other cases, such as a leaf or the root, the recurrence relation needs some modifications. First, if v is a leaf [e.g., p_1 and p_4 in Fig. 5(c)], because v does not have a child, the recurrence relation is modified as follows:

$$\mathrm{mvc}(v, r) = \mathrm{vc}(v)$$

where

$$
\begin{aligned}
\mathrm{vc}(v) &= \max(\Delta) - \min(\Delta), \\
\Delta &= \{\mathrm{max\_}l(v), \mathrm{min\_}l(v), r\}. \quad (2)
\end{aligned}
$$

Second, if v is the root [e.g., p_5 in Fig. 5(c)], because v does not have a parent, $\mathrm{vc}(v)$ does not need to consider the via cost

coming from r. Therefore, the recurrence relation for the root v is modified as follows:

$$\mathrm{mvc}(v, r) = \min_{\substack{1 \leq r_1 \leq k \\ \cdots \\ 1 \leq r_q \leq k}} \left(\sum_{j=1}^{q} \mathrm{mvc}(v_j, r_j) + \mathrm{vc}(v) \right)$$

where

$$
\begin{aligned}
q &= |\mathrm{ch}(v)| \\
\mathrm{ch}(v) &= \{v_1, \ldots, v_q\} \\
\mathrm{vc}(v) &= \max(\Delta) - \min(\Delta), \\
\Delta &= \{\mathrm{max\_}l(v), \mathrm{min\_}l(v), r_1, \ldots, r_q\}. \quad (3)
\end{aligned}
$$

Note that recurrence relation (3) is different from recurrence relation (1) by excluding r in Δ. Furthermore, because $\mathrm{mvc}(v, r)$ does not depend on the value of r when v is the root, we have $\mathrm{mvc}(v, 1) = \mathrm{mvc}(v, 2) = \cdots = \mathrm{mvc}(v, k)$. As a result, when v is the root, only $\mathrm{mvc}(v, 1)$ needs to be calculated, and it also represents the optimal via cost of a k-layer routed net.

We now state in the following lemma the correctness of each recurrence relation mentioned previously.

Lemma 1: By using recurrence relations (1)–(3), $\mathrm{mvc}(v, r)$ is computed correctly.

Proof: It is clear that $\mathrm{vc}(v)$ is computed correctly in recurrence relations (1)–(3), and we will use this fact in the rest of this proof. For a leaf v, because $\mathrm{mvc}(v, r)$ only involves $\mathrm{vc}(v)$, it is trivial to see that recurrence relation (2) is correct. For an internal node v, because the layer assignments of $\mathrm{par\_}e(v)$ (if v is not the root) and $\mathrm{par\_}e(v_j)$ for each $v_j \in \mathrm{ch}(v)$ are independent, and we enumerate all combinations of these layer assignments, recurrence relations (1) and (3) are both correct. ∎

SOLA will base on the edge order, as shown in Fig. 5, and on recurrence relations (1)–(3) to compute $\mathrm{mvc}(v, r)$ for each $v \in T_i^1$ and $r = 1 \sim k$. To construct T_i^k from T_i^1, we also need to know the assigned layers of all edges. To this end, for each $v_j \in \mathrm{ch}(v)$, we use $\mathrm{al}(v_j, r)$ to memorize the assigned layer of $\mathrm{par\_}e(v_j)$, which induces $\mathrm{mvc}(v, r)$. Therefore, the construction of T_i^k entails the repetition of the following two steps from the root to the leaves, where v is the node currently under consideration and $\mathrm{par\_}e(v)$ is assigned to layer r.

1) For each $v_j \in \mathrm{ch}(v)$, assign $\mathrm{par\_}e(v_j)$ to layer $\mathrm{al}(v_j, r)$.
2) Place vias on v to connect $\mathrm{par\_}e(v)$ (on layer r), $\mathrm{par\_}e(v_j)$ [on layer $\mathrm{al}(v_j, r)$] for each $v_j \in \mathrm{ch}(v)$, and each pin in $\mathrm{PS}(v)$.

Note that, in step 2), $\mathrm{par\_}e(v)$ does not exist when v is the root. We have the following lemma regarding the optimality of SOLA.

Lemma 2: Without considering the congestion constraints, SOLA optimally solves the single-net layer assignment problem.

Proof: We first use induction proof to show that, for each vertex v in a given one-layer routed tree, SOLA computes $\mathrm{mvc}(v, r)$ correctly for $r = 1 \sim k$. For the leaves in the tree, according to Lemma 1, it is easy to see that their via costs, as computed by recurrence relation (2), are minimum. For the induction step, SOLA computes $\mathrm{mvc}(v, r)$ by recurrence relation (1) or (3), depending on whether v is the root or

not, which requires to know $\text{mvc}(v_j, r_j)$ for each $v_j \in \text{ch}(v)$ and $r_j = 1 \sim k$, and the associated $\text{vc}(v)$. Because both recurrence relations (1) and (3) are correct by Lemma 1, plus each $\text{mvc}(v_j, r_j)$ is optimally computed by the induction hypothesis, we can derive that $\text{mvc}(v, r)$ is also optimally computed by SOLA. In addition, because SOLA constructs the layer assignment result by the standard top-down manner, the result has the minimum via cost. ∎

The time complexity of SOLA is analyzed as follows. Given a one-layer routed net $T_i^1 = (V_i^1, E_i^1)$, SOLA first uses a graph traversal algorithm (e.g., DFS or BFS) to obtain an edge order; this step requires $O(|V_i^1| + |E_i^1|) = O(|V_i^1|)$-time computation due to $|E_i^1| = O(|V_i^1|)$ in our problem. Second, for each vertex v of T_i^1, the steps of finding $\text{mvc}(v, r)$ for a given r require all the combinations of $\text{mvc}(v_j, r_j)$'s for each $v_j \in \text{ch}(v)$ and $r_j = 1 \sim k$. Because $|ch(v)|$ is at most four, computing an $\text{mvc}(v, r)$ can be done in $O(k^4)$ time. To compute all $\text{mvc}(v, r)$'s, it requires $O(k^4 \times k \times |V_i^1|) = O(k^5 |V_i^1|)$ time. Finally, the time spent on constructing T_i^k is $O(k|V_i^1|)$. Therefore, the overall time complexity of SOLA is $O(|V_i^1| + k^5 |V_i^1| + k|V_i^1|) = O(k^5 |V_i^1|)$.

C. APEC

When processing an edge during the layer assignment for a net, our algorithm uses an accurate and predictable examination of congestion constraints (APEC) to determine which layer that this edge can be assigned to such that no congestion constraint violation can happen due to this edge and any subsequent edge which has not been processed yet. In the following, we introduce APEC in relation to the following two aspects: maximum overflow constraint and total overflow constraint. We first have the following lemma regarding the lower bound on the maximum overflow of any S^k obtained from S^1 by layer assignment.

Lemma 3: For any S^k transformed form S^1 by layer assignment, we have $\lceil \text{MO}(S^1)/k \rceil \le \text{MO}(S^k)$.

Proof: The lemma is trivially true when $\text{MO}(S^1) = 0$, and therefore, we only focus on the case where $\text{MO}(S^1) > 0$. According to the definition of maximum overflow, there must exist a one-layer edge e^1 with $o(e^1) = \text{MO}(S^1)$. Furthermore, according to the definition of overflow, the number of routed nets passing through e^1 in S^1 is $d(e^1) = c(e^1) + o(e^1) = c(e^1) + \text{MO}(S^1)$. For any S^k transformed form S^1 by layer assignment, we know that $c(e^1) = \sum_{e^k \in \text{ES}(e^1)} c(e^k)$ and $d(e^1) = \sum_{e^k \in \text{ES}(e^1)} d(e^k)$. Because, for each $e^k \in \text{ES}(e^1)$, $d(e^k) \le c(e^k) + o(e^k)$ according to the overflow definition, we can derive $d(e^1) = \sum_{e^k \in \text{ES}(e^1)} d(e^k) \le \sum_{e^k \in \text{ES}(e^1)} (c(e^k) + o(e^k)) = \sum_{e^k \in \text{ES}(e^1)} c(e^k) + \sum_{e^k \in \text{ES}(e^1)} o(e^k)$. Therefore, $\text{MO}(S^1) = o(e^1) = d(e^1) - c(e^1) \le (\sum_{e^k \in \text{ES}(e^1)} c(e^k) + \sum_{e^k \in \text{ES}(e^1)} o(e^k)) - \sum_{e^k \in \text{ES}(e^1)} c(e^k) = \sum_{e^k \in \text{ES}(e^1)} o(e^k)$. Because $|\text{ES}(e^1)| = k$ and $o(e^k) \le \text{MO}(S^k)$ for each $e^k \in \text{ES}(e^1)$, we have $\text{MO}(S^1) \le \sum_{e^k \in \text{ES}(e^1)} o(e^k) \le k \times \text{MO}(S^k)$, implying that $(\text{MO}(S^1)/k) \le \text{MO}(S^k)$. Because $\text{MO}(S^k)$ is a nonnegative integer, we have $\lceil \text{MO}(S^1)/k \rceil \le \text{MO}(S^k)$. ∎

According to the maximum overflow constraint, the maximum overflow of any feasible k-layer global routing solution S^k can be calculated in advance from the maximum over-

flow of the given one-layer global routing solution S^1 (i.e., $\lceil \text{MO}(S^1)/k \rceil$). Consequently, to prevent the maximum overflow constraint violation, APEC will determine whether a one-layer edge e^1 can be assigned to a k-layer edge e^k, where e^k is in $\text{ES}(e^1)$, by checking whether $o'(e^k)$ is less than or equal to $\lceil \text{MO}(S^1)/k \rceil$, where $o'(e^k)$ is the overflow value of e^k right after this layer assignment.

We next prove the following lemma regarding the lower bound on the total overflow of any S^k obtained from S^1 by layer assignment.

Lemma 4: Given any S^k transformed form S^1 by layer assignment, for each one-layer edge e^1, we have $o(e^1) \le \sum_{e^k \in \text{ES}(e^1)} o(e^k)$; moreover, we have $\text{TO}(S^1) \le \text{TO}(S^k)$.

Proof: For each one-layer edge e^1, the following can be the two possible values for $o(e^1)$: $o(e^1) = 0$ or $o(e^1) > 0$. In the following, we prove that $o(e^1) \le \sum_{e^k \in \text{ES}(e^1)} o(e^k)$ holds in these two cases.

1) $o(e^1) = 0$: According to the definition of overflow, the minimum value of $o(e^k)$ is zero for each e^k in $\text{ES}(e^1)$, and thus, the minimum value of $\sum_{e^k \in \text{ES}(e^1)} o(e^k)$ is zero. Because $o(e^1) = 0$, we have $o(e^1) \le \sum_{e^k \in \text{ES}(e^1)} o(e^k)$.

2) $o(e^1) > 0$: We use proof by contradiction. First, we assume that there is an edge e^1 such that $o(e^1) > \sum_{e^k \in \text{ES}(e^1)} o(e^k)$. Then, according to the definition of overflow, we have $d(e^1) = c(e^1) + o(e^1)$, and according to the problem definition, we have $\sum_{e^k \in \text{ES}(e^1)} c(e^k) = c(e^1)$ and $\sum_{e^k \in \text{ES}(e^1)} d(e^k) = d(e^1)$. Therefore, $\sum_{e^k \in \text{ES}(e^1)} d(e^k) - \sum_{e^k \in \text{ES}(e^1)} c(e^k) = d(e^1) - c(e^1) = o(e^1)$. Because $\sum_{e^k \in \text{ES}(e^1)} d(e^k) - \sum_{e^k \in \text{ES}(e^1)} c(e^k) = \sum_{e^k \in \text{ES}(e^1)} (d(e^k) - c(e^k)) \le \sum_{e^k \in \text{ES}(e^1)} o(e^k)$ (according to the definition of overflow), it implies that $o(e^1) \le \sum_{e^k \in \text{ES}(e^1)} o(e^k)$. This contradicts the assumption.

To prove that $\text{TO}(S^1) \le \text{TO}(S^k)$, it suffices to see that $\text{TO}(S^1) = \sum_{e^1 \in G^1} o(e^1) \le \sum_{e^1 \in G1} \sum_{e^k \in \text{ES}(e^1)} o(e^k) = \sum_{e^k \in G^k} o(e^k) = \text{TO}(S^k)$. ∎

In general, the total overflow constraint is examined after S^k is completed. However, we found that the problem of meeting the total overflow constraint can be decomposed into an independent subproblem for each one-layer edge; we give the following lemma to demonstrate it.

Lemma 5: Given any S^k transformed form S^1 by layer assignment, we have $o(e^1) = \sum_{e^k \in \text{ES}(e^1)} o(e^k)$ for each one-layer edge e^1 if and only if S^k meets the total overflow constraint.

Proof: ($\Rightarrow$) Suppose that $o(e^1) = \sum_{e^k \in \text{ES}(e^1)} o(e^k)$ for each one-layer edge e^1. Then, according to the overflow definition and the problem definition, we have $\text{TO}(S^1) = \sum_{e^1 \in G^1} o(e^1) = \sum_{e^1 \in G^1} \sum_{e^k \in \text{ES}(e^1)} o(e^k) = \sum_{e^k \in G^k} o(e^k) = \text{TO}(S^k)$, implying that S^k meets the total overflow constraint.

($\Leftarrow$) Suppose that S^k meets the total overflow constraint [i.e., $\text{TO}(S^1) = \text{TO}(S^k)$]. Then, according to the overflow definition and the problem definition, we have $\text{TO}(S^1) = \sum_{e^1 \in G^1} o(e^1)$ and $\text{TO}(S^k) = \sum_{e^k \in G^k} o(e^k) = \sum_{e^1 \in G^1} \sum_{e^k \in \text{ES}(e^1)} o(e^k)$. Because $\text{TO}(S^1) = \text{TO}(S^k)$, we have $\sum_{e^1 \in G^1} o(e^1) = \sum_{e^1 \in G^1} \sum_{e^k \in \text{ES}(e^1)} o(e^k)$. Together with Lemma 4, we must have $o(e^1) = \sum_{e^k \in \text{ES}(e^1)} o(e^k)$ for each edge e^1. ∎

Therefore, to prevent the total overflow constraint violation, APEC will determine whether a one-layer edge e^1 can be assigned to a k-layer edge e^k, where e^k is in $\text{ES}(e^1)$, by checking whether $o'(e^k) + \sum_{e \in (\text{ES}(e^1) - \{e^k\})} o'(e)$ is less than or equal to $o(e^1)$, where $o'(e^k)$ and $o'(e)$ are the overflow values of e^k and e right after this layer assignment, respectively.[3]

To simultaneously prevent the total overflow constraint violation and the maximum overflow constraint violation, APEC will determine whether a one-layer edge e^1 can be assigned to a k-layer edge e^k, where e^k is in $\text{ES}(e^1)$, by checking if the following two conditions are true.

1) $o'(e^k) \le \lceil \text{MO}(S^1)/k \rceil$.
2) $o'(e^k) + \sum_{e \in (\text{ES}(e^1) - \{e^k\})} o'(e) \le o(e^1)$.

$o'(e^k)$ and $o'(e)$ are the overflow values of e^k and e right after this layer assignment, respectively.

The aforesaid two conditions are referred to as *prevention conditions*. If both prevention conditions are met, this layer assignment is *legal*; otherwise, it is *illegal*. As shall be seen in Theorem 1 (in Section VI-A), satisfying both prevention conditions can accurately avoid any congestion constraint violation in advance.

We now analyze the time complexity of APEC. APEC, examining both prevention conditions for the assignment of a one-layer edge e^1 to a k-layer e^k, where e^k is in $\text{ES}(e^1)$, requires $O(k)$-time computation because $|\text{ES}(e^1)| = k$.

D. SOLA+APEC: The Combination of SOLA and APEC

To solve the original problem of single-net layer assignment considering congestion constraints, APEC is incorporated into SOLA to prevent any congestion constraint violation. To correctly combine SOLA and APEC to get SOLA+APEC, the original recurrence relations (1) and (2) adopted by SOLA need a modification; if assigning par_$e(v)$ to layer r is illegal, the recurrence relation becomes $\text{mvc}(v, r) = \infty$. Moreover, every time SOLA+APEC completes a layer assignment for a net, it also needs to update the demand and overflow (if necessary) values for those k-layer edges involved in this layer assignment. Because the time complexity of APEC is $O(k)$, the time complexity of SOLA+APEC is still $O(k^5|V_i^1|)$ for a given one-layer routed net T_i^1.

VI. ANALYSIS AND EXTENSION OF COLA

The pseudocode of our overall layer assignment algorithm COLA is shown in Fig. 7. COLA is a heuristic algorithm for solving the whole layer assignment problem, but as shall be seen in Section VII, it reports very promising experimental results.

A. Analysis of COLA

We give the following theorem to show that COLA is guaranteed to produce a feasible layer assignment solution for any problem instance.

Algorithm: COLA

/* Part 1: Generate net order */

1: **for** $i \leftarrow 1$ **to** $|N^1|$
2:         $Score(T_i^1) \leftarrow$ Generate_net_score(T_i^1);
3: **end for**
4: Sort_nets($Score(T_i^1)$'s);   /* sort nets according to Score */

/* Part 2: Use SOLA+APEC to find the layer assignment for each net */

5: **for** $i \leftarrow 1$ **to** $|N^1|$ /* following the net order */
6:         $T_i^1 \leftarrow$ Pre_processing(T_i^1);
7:         $T_i^k \leftarrow$ SOLA+APEC(T_i^1);
8: **end for**

Fig. 7. Pseudocode of COLA.

Theorem 1: Given a one-layer global routing solution S^1, COLA always produces a k-layer global solution S^k satisfying the congestion constraints.

Proof: We first prove that SOLA+APEC always finds a layer assignment result satisfying both prevention conditions for each net. Given any net order (including the one adopted by COLA), suppose that the ith ($1 \le i \le |N^1|$) net T_i^1 in the net order is the first net which causes SOLA+APEC to fail to find a layer assignment result satisfying both prevention conditions. Then, there must exist an edge e^1 in T_i^1 such that none of the edges in $\text{ES}(e^1)$ satisfies both prevention conditions. If none of the edges in $\text{ES}(e^1)$ satisfies the first prevention condition, the first $(i - 1)$ nets must have caused $o_{i-1}(e^k) = \lceil \text{MO}(S^1)/k \rceil$ for each $e^k \in \text{ES}(e^1)$, where $o_{i-1}(e^k)$ is the overflow value of e^k right before SOLA+APEC processes T_i^1. Therefore, among the first i nets, the number of nets passing through e^1 in S^1 is at least $1 + k \times \lceil \text{MO}(S^1)/k \rceil + c(e^1) \ge 1 + \text{MO}(S^1) + c(e^1)$. However, for any given S^1, the number of nets passing through e^1 is at most $\text{MO}(S^1) + c(e^1)$, which is smaller than $1 + \text{MO}(S^1) + c(e^1)$. Thus, we have a contradiction here, and consequently, there must exist an $e^k \in \text{ES}(e^1)$ satisfying the first prevention condition. Now, suppose that, among those e^k's satisfying the first prevention condition, none of them satisfies the second prevention condition. Then, the first $(i - 1)$ nets must have caused $o(e^1) = \sum_{e^k \in \text{ES}(e^1)} o_{i-1}(e^k)$ and $d_{i-1}(e^k) \ge c(e^k)$ for each e^k in $\text{ES}(e^1)$, where $d_{i-1}(e^k)$ is the demand value of e^k right before SOLA+APEC processes T_i^1. Therefore, among the first i nets, the number of nets passing through e^1 is $1 + \sum_{e^k \in \text{ES}(e^k)} o_{i-1}(e^k) + \sum_{e^k \in \text{ES}(e^1)} c(e^k) = 1 + o(e^1) + c(e^1)$. However, for any given S^1, the number of nets passing through e^1 is $o(e^1) + c(e^1)$, which is smaller than $1 + o(e^1) + c(e^1)$. Again, we get a contradiction here; thus, among those e^k's satisfying the first prevention condition, at least one of them must also satisfy the second prevention condition. Consequently, for each edge e^1 in T_i^1, there exists an $e^k \in \text{ES}(e^1)$ such that e^1 can be assigned to e^k without violating any prevention condition. It implies that SOLA+APEC can produce a layer assignment result for T_i^1 and can satisfy both prevention conditions. Hence, SOLA+APEC always produces a layer assignment result satisfying both prevention conditions for each net.

We next prove that the overall k-layer global routing solution S^k produced by COLA satisfies the congestion constraints.

---

[3]Note that e^k is the only edge whose overflow value may get changed by this layer assignment.

Because the layer assignment result of each net generated by SOLA+APEC satisfies both prevention conditions, we have $o(e^k) \leq \lceil \text{MO}(S^1)/k \rceil$ for each k-layer edge e^k, and $\sum_{e^k \in \text{ES}(e^1)} o(e^k) \leq o(e^1)$ for each one-layer edge e^1. Therefore, $\text{MO}(S^k) = \max_{e^k \in G^k} o(e^k) \leq \lceil \text{MO}(S^1)/k \rceil$, and $\text{TO}(S^k) = \sum_{e^k \in G^k} o(e^k) = \sum_{e^1 \in G^1} \sum_{e^k \in \text{ES}(e^1)} o(e^k) \leq \sum_{e^1 \in G^1} o(e^1) = \text{TO}(S^1)$; as a result, $\text{MO}(S^k) \leq \lceil \text{MO}(S^1)/k \rceil$, and $\text{TO}(S^k) \leq \text{TO}(S^1)$. However, Lemmas 3 and 4 say that $\text{MO}(S^k) \geq \lceil \text{MO}(S^1)/k \rceil$ and $\text{TO}(S^k) \geq \text{TO}(S^1)$. Thus, we must have $\text{MO}(S^k) = \lceil \text{MO}(S^1)/k \rceil$ and $\text{TO}(S^k) = \text{TO}(S^1)$, which implies that S^k satisfies both congestion constraints. This concludes the proof. ∎

We next use the pseudocode shown in Fig. 7 to analyze the time complexity of COLA. Lines 1–3 calculate the scores for all nets, and the time complexity is $O(|N^1||V^1|)$. In line 4, COLA sorts nets according to their scores in decreasing order, and the time complexity is $O(|N^1| \log |N^1|)$. From line 5 to line 8, COLA follows the net order and solves the problem of layer assignment for each net. In line 6, the preprocessing step is based on a graph traversal algorithm which requires $O(|V_i^1| + |E_i^1|) = O(|V_i^1|)$. In line 7, the time complexity of SOLA+APEC is $O(k^5|V_i^1|)$. Hence, from line 5 to line 8, the time complexity is $O(k^5|N^1||V^1|)$ due to $|V^1| \geq |V_i^1|$. Therefore, the overall time complexity of COLA is $O(|N^1|(k^5|V^1| + \log |N^1|))$.

B. Extension to Preferred Routing Directions

In this section, we describe how to modify the problem formulation and COLA for the case where the metal layers have preferred routing directions. Without loss of generality, we assume that half of the layers are horizontal ones while the rest are vertical ones. Therefore, the maximum overflow constraint is modified as follows:

$$\text{MO}(S^k) = \lceil \text{MO}(S^1) \times (2/k) \rceil.$$

Furthermore, SOLA+APEC is modified in such a way that the assigned layer of an edge must follow its preferred routing direction or it is an illegal layer assignment. Consequently, with these modifications, the layer assignment problem with preferred routing directions can still be solved by COLA.

VII. EXPERIMENTAL RESULTS

COLA was implemented with standard ANSI C++ and tested on a Linux workstation with AMD Dual Core Opteron Processor 2.2-GHz CPU and 8-GB memory. In our experiments, we used the ISPD'07 global routing contest benchmarks [15] to test COLA because they are the first published multilayer global routing benchmarks and the sizes of these benchmarks are large enough as compared to real industry cases. Each benchmark has a two-layer and a six-layer version. The capacity of a boundary edge in a two-layer version is simply the sum of the capacities of the corresponding boundary edges in the six-layer version. The other characteristics (e.g., number of tiles, number of nets, number of pins, etc.) are the same. The detailed information is listed in Table III. Because the layer assignment problem is more meaningful and has larger solution space for instances with more than two layers, we only used

TABLE III
DETAILED INFORMATION OF THE ISPD'07 BENCHMARKS [15]

Name	#nets	#tiles	#pins
adaptec1	219794	324×324	942705
adaptec2	260159	424×424	1063632
adaptec3	466295	774×779	1874576
adaptec4	515304	774×779	1911773
adaptec5	867441	465×468	3492790
newblue1	331663	399×399	1237104
newblue2	463213	557×463	1771849
newblue3	551667	973×1256	1929360

the six-layer benchmarks as the test cases in our experiments. Furthermore, all of the results, which will be presented in this section, have been verified by the ISPD'07 official evaluation script [15].

A. Improvements on ISPD'07 Contest Results

We first tested COLA to see if it could help improve the *contest results* of the top three winners, namely, MaizeRouter, BoxRouter, and FGR, in the ISPD'07 global routing contest [15]. We also implemented a straightforward greedy algorithm for comparison. Basically, this greedy algorithm takes the same net order as COLA, but for the layer assignment of each edge of a net, it chooses one layer which induces the least overflow. Because the greedy algorithm also applies APEC to consider the congestion constraints during the layer assignment of each edge of a net, it will produce identical congestion quality to that of COLA.

The six-layer results of MaizeRouter, BoxRouter, and FGR were compressed so as to provide the one-layer global routing solutions for both COLA and the greedy algorithm [refer to Fig. 2, where the compression transforms the multilayer global routing solution in Fig. 2(d) back to the one-layer global routing solution in Fig. 2(c)]. Although the details about the approaches taken by MaizeRouter,[4] BoxRouter, and FGR were not available in the ISPD'07 contest, the way we set up the input data allows us to compare COLA and the greedy algorithm with the layer assignment method of a global router if the router adopts the approach shown in Fig. 2. In addition, this kind of input setting also tests COLA and the greedy algorithm to see whether they are able to improve a given six-layer global routing solution through layer assignment.

In our experiments, we followed the ISPD'07 contest rules to calculate wirelength and overflow. Wirelength is measured as follows.
1) Tile-to-tile connection (i.e., a wire) is one unit of wirelength.
2) Layer-to-layer connection (i.e., a via) is three units of wirelength.

Overflow is two times the one defined in Section II. Furthermore, in each benchmark, the layers *implicitly* give preferred

---

[4]The contest results of MaizeRouter also appeared later in [11]; according to [11], MaizeRouter took the approach shown in Fig. 2 to generate six-layer results.

TABLE IV
COMPARISON BETWEEN MAIZEROUTER AND TWO LAYER ASSIGNMENT METHODS
ON THE ISPD'07 BENCHMARKS WITHOUT PREFERRED ROUTING DIRECTIONS

| Name | MaizeRouter | | | | MaizeRouter+greedy | | | | | MaizeRouter+COLA | | | | |
| | wirelength(e5) | | overflow | | wirelength(e5) | | overflow | | | wirelength(e5) | | overflow | | |
	total	via	total	max	total(imp. %)	via(imp. %)	total	max	cpu(s)	total(imp. %)	via(imp. %)	total	max	cpu(s)
adaptec1	99.61	60.13	0	0(0)	116.48(-16.94)	77.01(-28.07)	0	0	35.53	95.96(3.66)	56.48(6.07)	0	0	36.56
adaptec2	98.12	63.20	0	0(0)	109.93(-12.04)	75.02(-18.70)	0	0	34.66	94.64(3.55)	59.73(5.49)	0	0	34.41
adaptec3	214.08	116.71	0	0(0)	248.96(-16.29)	151.59(-29.89)	0	0	94.67	205.83(3.85)	108.46(7.07)	0	0	96.19
adaptec4	194.38	104.49	0	0(0)	225.99(-16.26)	136.10(-30.25)	0	0	84.23	188.41(3.07)	98.52(5.71)	0	0	86.94
adaptec5	305.32	192.91	2	2(2)	327.89(-7.39)	215.52(-11.72)	2	2	129.26	282.37(7.52)	170.00(11.88)	2	2	118.66
newblue1	101.74	76.37	1348	16(16)	107.01(-5.18)	81.64(-6.90)	1348	4	28.43	93.85(7.76)	68.48(10.33)	1348	4	28.80
newblue2	139.66	93.16	0	0(0)	169.49(-21.36)	122.99(-32.02)	0	0	49.41	136.28(2.42)	89.78(3.63)	0	0	48.39
newblue3	184.40	105.80	32840	1058(1244)	224.41(-21.70)	145.85(-37.85)	32564	208	86.35	171.64(6.92)	93.09(12.01)	32564	208	85.80

TABLE V
COMPARISON BETWEEN BOXROUTER AND TWO LAYER ASSIGNMENT METHODS
ON THE ISPD'07 BENCHMARKS WITH PREFERRED ROUTING DIRECTIONS

| Name | BoxRouter | | | | BoxRouter+greedy | | | | | BoxRouter+COLA | | | | |
| | wirelength(e5) | | overflow | | wirelength(e5) | | overflow | | | wirelength(e5) | | overflow | | |
	total	via	total	max	total(imp. %)	via(imp. %)	total	max	cpu(s)	total(imp. %)	via(imp. %)	total	max	cpu(s)
adaptec1	104.05	66.77	0	0(0)	111.08(-6.76)	73.81(-10.54)	0	0	32.14	92.91(10.71)	55.63(16.68)	0	0	33.01
adaptec2	102.97	68.63	0	0(0)	107.44(-4.34)	73.11(-6.53)	0	0	31.05	93.63(9.07)	59.30(13.59)	0	0	33.26
adaptec3	235.87	136.57	0	0(0)	251.76(-6.74)	152.47(-11.64)	0	0	92.67	210.27(10.85)	110.99(18.73)	0	0	93.14
adaptec4	211.95	121.84	0	0(0)	225.69(-6.48)	135.60(-11.29)	0	0	78.87	189.16(10.75)	99.07(18.69)	0	0	82.51
adaptec5	297.89	191.73	0	0(0)	310.85(-4.35)	204.72(-6.78)	0	0	98.66	269.57(9.51)	163.44(14.76)	0	0	103.03
newblue1	101.83	76.72	400	2(2)	104.55(-2.67)	79.46(-3.57)	400	2	26.04	92.99(8.68)	67.90(11.50)	400	2	26.82
newblue2	155.07	108.19	0	0(0)	168.27(-8.51)	121.44(-12.25)	0	0	46.65	136.35(12.07)	89.50(17.28)	0	0	46.32
newblue3	195.49	118.75	38958	1088(1088)	218.56(-11.80)	141.83(-19.44)	38958	364	81.83	169.16(13.47)	92.43(22.16)	38958	364	80.04

routing directions because on odd layers, only vertical boundary edges have nonzero capacities, whereas on even layers, only horizontal boundary edges have nonzero capacities.[5] However, the ISPD'07 contest did not *explicitly* require routers to follow preferred routing directions. According to our observation on the contest results, we found that MaizeRouter did not consider preferred routing directions, whereas BoxRouter and FGR did. Therefore, to have a fair comparison with them, both COLA and the greedy algorithm were applied to the compressed one-layer results of MaizeRouter without considering preferred routing directions, but when they were applied to the compressed one-layer results of BoxRouter and FGR, preferred routing directions were considered.

The original and the new six-layer results are listed in Tables IV–VI, where the column "MaizeRouter" ("BoxRouter" and "FGR," respectively) reports the original global routing results obtained by MaizeRouter (BoxRouter and FGR, respectively), the column "MaizeRouter+greedy" ("BoxRouter+

greedy" and "FGR+greedy," respectively) reports the results obtained by the greedy algorithm using the compressed one-layer results of MaizeRouter (BoxRouter and FGR, respectively) as the inputs, and the column "MaizeRouter+COLA" ("BoxRouter+COLA" and "FGR+COLA," respectively) reports the results obtained by COLA using the compressed one-layer results of MaizeRouter (BoxRouter and FGR, respectively) as the inputs. Note that the numbers before the parentheses in column "max" refer to the maximum overflows of the original six-layer results from MaizeRouter, BoxRouter, or FGR, and the numbers in the parentheses in column "max" refer to the maximum overflows of the compressed one-layer results derived from MaizeRouter, BoxRouter, or FGR.

The experimental results in Tables IV–VI, first of all, clearly show that, although the greedy algorithm could generate a feasible layer assignment result for each benchmark, both the total wirelength and the via wirelength results were much worse than the original results; such results were not surprising because the greedy algorithm is just a very simple heuristic. On the other hand, for each benchmark, COLA always reduced the total wirelengths and the via wirelengths, and the via wirelength (total wirelength) improvement

---

[5]With the preferred routing directions, it is not possible to make any improvement on a two-layer result through layer assignment. This is another reason why we did not test COLA on the two-layer benchmarks.

TABLE VI
COMPARISON BETWEEN FGR AND TWO LAYER ASSIGNMENT METHODS ON THE ISPD'07 BENCHMARKS WITH PREFERRED ROUTING DIRECTIONS

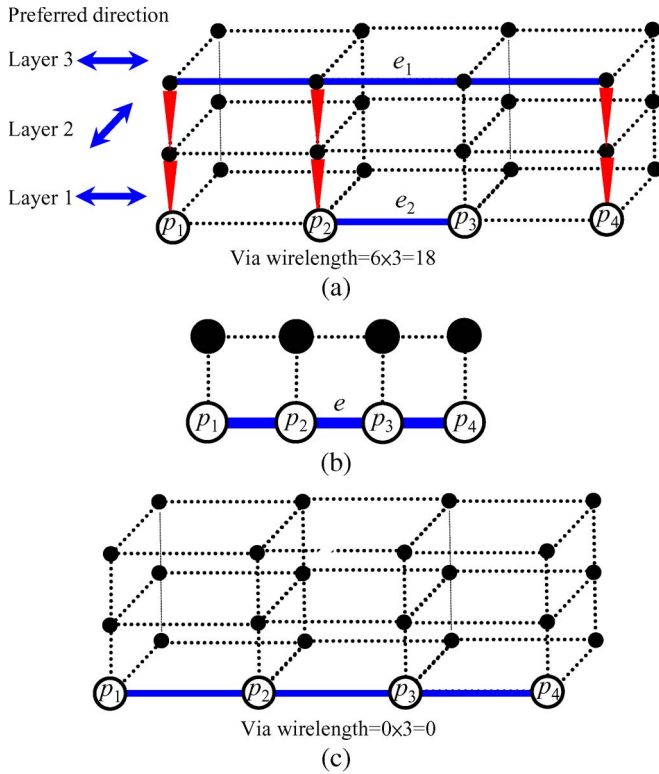| Name	FGR				FGR+greedy					FGR+COLA				
	wirelength(e5)		overflow		wirelength(e5)		overflow			wirelength(e5)		overflow		
	total	via	total	max	total(imp. %)	via(imp. %)	total	max	cpu(s)	total(imp. %)	via(imp. %)	total	max	cpu(s)
adaptec1	90.92	54.69	60	2(2)	108.89(-19.76)	72.69(-32.91)	54	2	32.47	90.20(0.79)	54.01(1.24)	54	2	32.95
adaptec2	92.19	58.83	50	2(2)	105.70(-14.65)	72.36(-23.00)	36	2	32.93	91.49(0.76)	58.15(1.16)	36	2	32.43
adaptec3	203.44	107.01	0	0(0)	246.10(-20.97)	149.70(-39.89)	0	0	92.02	202.74(0.34)	106.35(0.62)	0	0	95.88
adaptec4	186.31	97.13	0	0(0)	223.94(-20.20)	134.76(-38.74)	0	0	83.41	185.76(0.30)	96.59(0.56)	0	0	87.40
adaptec5	264.58	161.46	2480	2(6)	304.89(-15.24)	201.85(-25.02)	2156	2	102.36	262.10(0.94)	159.05(1.49)	2156	2	110.03
newblue1	92.89	68.83	2668	4(10)	102.14(-9.96)	78.10(-13.47)	2296	4	26.66	90.29(2.80)	66.25(3.75)	2296	4	26.96
newblue2	136.08	89.83	0	0(0)	166.44(-22.31)	120.19(-33.80)	0	0	48.03	134.19(1.39)	87.84(2.22)	0	0	47.54
newblue3	168.42	93.65	53648	636(1094)	213.17(-26.57)	138.43(-47.82)	50720	366	81.25	163.20(3.10)	88.45(5.55)	50720	366	79.63

Fig. 8. Net for which COLA generated a better result than FGR 1.1.

TABLE VII
COMPARISON BETWEEN BOXROUTER 2.0 AND BOXROUTER 2.0 + COLA
ON THE ISPD'07 BENCHMARKS WITH PREFERRED ROUTING DIRECTIONS

Name	BoxRouter 2.0				BoxRouter 2.0 + COLA			
	wirelength(e5)		overflow		wirelength(e5)		overflow	
	total	via	total	max	total(imp. %)	via(imp. %)	total	max
adaptec1	92.04	55.06	0	0(0)	91.29(0.81)	54.32(1.34)	0	0
adaptec2	94.28	60.26	0	0(0)	92.62(1.76)	58.61(2.74)	0	0
adaptec3	207.41	109.34	0	0(0)	206.46(0.46)	108.41(0.85)	0	0
adaptec4	186.42	96.60	0	0(0)	185.82(0.32)	96.03(0.59)	0	0
adaptec5	270.41	162.51	0	0(0)	265.70(1.74)	160.24(1.40)	0	0
newblue1	92.94	67.81	394	2(4)	92.50(0.47)	67.38(0.63)	394	2
newblue2	134.64	87.94	0	0(0)	134.05(0.44)	87.38(0.64)	0	0
newblue3	172.44	95.70	38958	364(1088)	168.65(2.20)	91.92(3.95)	38958	364

Fig. 8(a) shows a routed net which uses the bottom three layers to connect four pins p_1, p_2, p_3, and p_4 and contains two parallel edges e_1 and e_2; Fig. 8(b) shows the one-layer version of the routed net with both e_1 and e_2 projected to the one-layer edge e. Because redundant edges were removed from cycles and parallel edges were projected to the same one-layer edges, COLA and the greedy algorithm were able to reduce the total overflows for those benchmarks. Moreover, we would also like to point out the fact that, although MaizeRouter, MaizeRouter+COLA, and MaizeRouter+greedy did not *explicitly* follow the preferred routing directions, for the benchmarks where the total overflow values are zero, their respective routing results *actually* obey the preferred routing directions.

According to the experimental results in Tables IV–VI, COLA took no more than 2 min for each benchmark, which is as efficient as the greedy algorithm. Note that each CPU time result also includes the time spent on all file I/O's. It is interesting to see that the greedy algorithm has larger CPU times for some benchmarks; it is mainly because some worse results generated by the greedy algorithm have much larger output file sizes.

To see the impact of the net order determination method adopted in COLA, we also used SOLA+APEC to perform layer assignment for each net with four different net orders. The first one called "Random" is obtained from the net order specified in each input; the other three called "Length," "PinNum,"

rates were 3.63%–12.01% (2.42%–7.76%), 11.5%–22.16% (8.68%–13.47%), and 0.56%–5.55% (0.3%–3.10%) for Maize-Router, BoxRouter, and FGR, respectively. Additionally, the resulting total and maximum overflows of COLA satisfied the given congestion constraints in each benchmark. It is worth mentioning that, for one benchmark of MaizeRouter and five benchmarks of FGR, the total overflows generated by COLA and the greedy algorithm were smaller than the original ones (see the result of newblue3 in Table IV and the results of adaptec1, adaptec2, adaptec5, newblue1, and newblue3 in Table VI). For those benchmarks, their original six-layer results contain "parallel edges" (for FGR), or their compressed one-layer results contain cycles (for FGR and MaizeRouter). Note that a subset of edges in a routed net are said to be *parallel* if they are projected to the same one-layer edge. For example,

TABLE VIII
COMPARISON BETWEEN FGR 1.1 AND FGR 1.1 + COLA ON THE ISPD'07 BENCHMARKS WITH PREFERRED ROUTING DIRECTIONS

Name	FGR 1.1				FGR 1.1 + COLA					
	wirelength(e5)		overflow		wirelength(e5)		overflow		via wirelength imp.(e5)	
	total	via	total	max	total(imp. %)	via(imp. %)	total	max	with PEs	w/o PEs
adaptec1	88.02	51.58	0	0(0)	87.90(0.14)	51.73(-0.29)	0	0	-1.91	1.76
adaptec2	89.96	56.05	0	0(0)	89.54(0.47)	55.90(0.27)	0	0	-2.21	2.37
adaptec3	200.14	102.73	0	0(0)	199.80(017)	102.93(-0.19)	0	0	-4.51	4.31
adaptec4	178.90	87.20	0	0(0)	182.40(-1.96)	91.13(-4.51)	0	0	-5.19	1.27
adaptec5	260.53	156.28	0	0(0)	258.23(0.88)	154.57(1.09)	0	0	-5.39	7.11
newblue1	90.68	65.90	238	2(2)	89.38(1.43)	64.81(1.65)	236	2	-2.38	3.47
newblue2	129.3	81.03	0	0(0)	131.61(-1.79)	83.72(-3.32)	0	0	-4.85	2.16
newblue3	163.41	86.46	38398	400(1196)	162.65(0.47)	86.03(0.50)	38372	400	-3.21	3.64

and "AvgDensity" are obtained by the net orders which are determined only by the factors $\text{Length}(T_i^1)$, $\text{PinNum}(T_i^1)$, and $\text{AvgDensity}(T_i^1)$ of each one-layer routed net T_i^1, respectively. Our experimental results indicate that, as compared to the net order determination method of COLA, these four different net order policies averagely generated 3.33%–23.42% (3.42%–24.12% and 3.65%–24.8%, respectively) worse via wirelength than MaizeRouter+COLA (BoxRouter+COLA and FGR+COLA, respectively). For the greedy algorithm, we also performed the same experiments for these four different net order policies. Our results show that, as compared to our net order determination method, the other four net order policies averagely generated 3.81%–21.67% (4.34%–22.65% and 4.4%–22.76%, respectively) worse via wirelength than Maize-Router+greedy (BoxRouter+greedy and FGR+greedy, respectively). Because these results show similar tendencies, to save space, their details are omitted here. According to these results, we can infer that, in order to make our net order determination method work, all three factors must be considered in the score calculation.

By looking into the results described in the last paragraph and the results shown in Tables IV–VI, we can also infer that the net order determination method and the single-net layer assignment method are both equally important to the success of COLA. Replacing either of them by an improper method will significantly degrade the solution quality.

B. Comparison With BoxRouter 2.0 and FGR 1.1

Both BoxRouter and FGR have been improved since the ISPD'07 contest was finished; significantly better results have been recently reported by their newer versions BoxRouter 2.0 [9] and FGR 1.1 [17], respectively. To see how much improvement COLA could achieve, we tested COLA on the compressed one-layer results of BoxRouter 2.0[6] and FGR 1.1. The original and the new results are shown in Tables VII and VIII. As can be seen from Table VII, COLA was still able to improve the via wirelength and the total wirelength on each benchmark
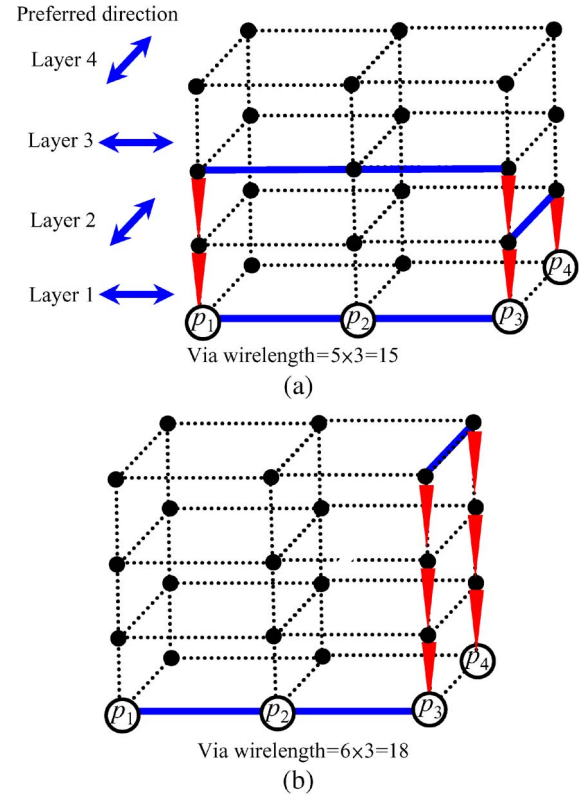

Fig. 9. Net for which COLA generated a worse result than FGR 1.1.

for BoxRouter 2.0; the reduction rates are 0.59%–3.95% and 0.32%–2.2%, respectively.

Table VIII shows that the via wirelength and total wirelength results of FGR 1.1 could be improved by COLA on half of the benchmarks and three quarters of the benchmarks, respectively. To see why COLA could not produce better via wirelength results for half of the benchmarks, we looked into the results of FGR 1.1 and found that the result of each benchmark again contains parallel edges. The results of FGR 1.1 were produced by the approach which first generates a one-layer routing result, then performs layer assignment, and finally applies a single round of rip-up-and-reroute operation for each net to further reduce the via wirelength [22]. Therefore, parallel edges were generated after layer assignment. Because each via is three

---

[6]According to [9], BoxRouter 2.0 took the approach shown in Fig. 2 to generate six-layer results.

TABLE IX
RESULTS OF FGR 1.1 + COLA_R ON THE ISPD'07 BENCHMARKS WITH PREFERRED ROUTING DIRECTIONS

Name	FGR 1.1 + COLA_R										
	1st iteration					overall					
	wirelength(e5)		overflow			wirelength(e5)		overflow			
	total(imp. %)	via(imp. %)	total	max	cpu(s)	total(imp. %)	via(imp. %)	total	max	#iterations	cpu(s)
adaptec1	87.61(0.47)	51.25(0.64)	0	0	31.34	87.41(0.69)	51.06(1.01)	0	0	5	167.77
adaptec2	89.54(0.47)	55.70(0.62)	0	0	30.45	89.31(0.72)	55.48(1.02)	0	0	5	153.90
adaptec3	199.26(0.44)	102.01(0.70)	0	0	79.98	198.80(0.67)	101.58(1.12)	0	0	5	404.79
adaptec4	178.81(0.05)	87.14(0.07)	0	0	71.51	178.78(0.07)	87.11(0.10)	0	0	4	312.91
adaptec5	259.43(0.42)	155.34(0.60)	0	0	102.43	258.81(0.66)	154.76(0.97)	0	0	7	769.09
newblue1	90.38(0.33)	65.64(0.39)	236	2	27.09	90.20(0.53)	65.46(0.67)	236	2	4	118.73
newblue2	129.21(0.07)	80.96(0.09)	0	0	45.48	129.16(0.11)	80.92(0.14)	0	0	3	154.28
newblue3	162.25(0.71)	85.40(1.23)	38376	400	62.07	161.85(0.95)	85.01(1.68)	38376	400	4	287.77

units of wirelength, it is worthy for FGR 1.1 to use parallel edges if doing so helps to reduce the via wirelength. For COLA, however, it will not produce any parallel edge due to the problem nature. After carrying out a net-by-net comparison from the results of FGR 1.1 and COLA, we observed that, for nets with parallel edges, COLA could produce better or worse layer assignment results for them in terms of the via wirelength. Fig. 8 shows a net for which COLA generated a better result [see Fig. 8(c)] than the one produced by FGR 1.1 [see Fig. 8(a)]. On the other hand, Fig. 9 shows a net for which COLA generated a worse result [see Fig. 9(b)] than the one produced by FGR 1.1 [see Fig. 9(a)]. For each benchmark, we further divided its set of nets into two categories according to the original result produced by FGR 1.1. The first category contains the nets which have parallel edges, whereas the second category contains the ones without any parallel edge. For each category, we calculated the amount of via wirelength decrease (or increase) achieved by COLA; these numbers are given in the last two columns of Table VIII, where the first (second) column "with PEs" ("w/o PEs") is for the first (second) category and each positive (negative) number denotes the amount of via wirelength decrease (increase). For each benchmark, COLA increased the via wirelength for the category of nets with parallel edges; this is because the via wirelength increase due to the case shown in Fig. 9 outnumbered the via wirelength decrease due to the case shown in Fig. 8. On the other hand, for each benchmark, COLA decreased the via wirelength for the category of nets without parallel edges, indicating that COLA performed really well for layer assignment which disallows parallel edges. Whether the overall via wirelength of each benchmark got improved or not depended on which category dominated.

Furthermore, the presence of cycles and parallel edges not only helped COLA to improve the total wirelength results for adaptec1 and adaptec3 (although the via wirelength values got increased) but also gave COLA chances to reduce the total overflow results for newblue1 and newblue3.

Although COLA is designed for performing layer assignment on a one-layer global routing solution, it can also be adapted to iteratively refine a multilayer global routing solution in a net-by-net manner. The refinement approach is referred to as COLA_R and is guaranteed to produce a result at least as good as the original one. First, COLA_R generates a net order using the net order determination method of COLA. Then, in each iteration, COLA_R tries to refine each net according to the net order. Unlike COLA which first compresses all nets and then performs layer assignment one net at a time, COLA_R performs the layer assignment for a net right after compressing the net to its one-layer counterpart. As soon as the new result of a net is produced, it is compared with the old result of the net, and if it has shorter total wirelength, it replaces the old one. The refinement process iterates until no further improvement can be made. We tested COLA_R to see whether it could refine the results of FGR 1.1 but without increasing the total overflow and the max overflow values.[7] Table IX shows the results produced after the first iteration and after the whole process of COLA_R. For each benchmark, the via wirelength and the total wirelength of FGR 1.1 were both improved by COLA_R; the via wirelength and the total wirelength improvement rates are 0.1%–1.68% and 0.07%–0.95%, respectively. In addition, COLA_R was able to reduce the total overflows for newblue1 and newblue3. Table IX also shows that COLA_R achieved more than half of the improvement at the first iteration and took three to seven iterations to terminate. On average, each iteration of COLA_R ran as efficiently as COLA.

C. Remarks

There are two other global routers, namely, NTHU-Route [23] and Archer [10], which have been proposed recently. COLA has been successfully integrated into NTHU-Route and helped to produce solutions comparable with those of the other recent global routers for the ISPD'07 six-layer benchmarks. More details can be found in [23]. However, because the results of Archer are not allowed to release due to intellectual property issues, we cannot test COLA on them.

---

[7]It is not hard to see that COLA_R cannot provide any improvement on the results produced by COLA.

VIII. CONCLUSION

In this paper, we have formulated a congestion-constrained layer assignment problem for multilayer global routing and presented an efficient and effective algorithm for the problem. Our algorithm is guaranteed to generate a feasible layer assignment result for each problem instance. Our algorithm can also be adapted to refine a multilayer global routing solution in a net-by-net manner. Extensive experiments have been conducted, and promising results have been reported to support our algorithm.

The layer assignment problem that we have considered in this paper, in fact, originates from the ISPD'07 global routing contest and ignores the "via capacity constraint". Because the amount of vias to be placed in a region should be bounded by a capacity, a more practical layer assignment problem is to take the via capacity constraint into account as well in order to bring a routable result to the detailed router. We are currently studying how to extend our layer assignment algorithm to handle this more practical problem.

REFERENCES

[1] R. Kastner, E. Bozogzadeh, and M. Sarrafzadeh, "Predictable routing," in *Proc. Int. Conf. Comput.-Aided Des.*, 2000, pp. 110–113.

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

[3] 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.

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

[5] C. Sechen and A. Sangiovanni-Vincentelli, "The TimberWolf placement and routing package," *IEEE J. Solid-State Circuits*, vol. SSC-20, no. 2, pp. 510–522, Apr. 1985.

[6] H. Shin and A. Sangiovanni-Vincentelli, "Mighty: A rip-up and reroute detailed router," in *Proc. Int. Conf. Comput.-Aided Des.*, 1986, pp. 115–122.

[7] L. E. Liu and C. Sechen, "Multilayer chip-level global routing using an efficient graph-based Steiner tree heuristic," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 18, no. 10, pp. 1442–1451, Oct. 1997.

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

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

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

[11] M. D. Moffitt, "MaizeRouter: Engineering an effective global router," in *Proc. Asia South Pacific Des. Autom. Conf.*, 2008, pp. 226–231.

[12] M. J. Ciesielski, "Layer assignment for VLSI interconnect delay minimization," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 8, no. 6, pp. 702–707, Jun. 1989.

[13] J. D. Cho, S. Raje, M. Sarrafzadeh, M. Sriram, and S. M. Kang, "Crosstalk-minimum layer assignment," in *Proc. Custom Integr. Circuits Conf.*, 1993, pp. 29.7.1–29.7.4.

[14] D. Wu, J. Hu, R. Mahapatra, and M. Zhao, "Layer assignment for crosstalk risk minimization," in *Proc. Asia South Pacific Des. Autom. Conf.*, 2004, pp. 159–162.

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

[16] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. New York: McGraw-Hill, 2001.

[17] FGR 1.1. [Online]. Available: http://vlsicad.eecs.umich.edu/BK/FGR

[18] K. Ahn and S. Sahni, "Constrained via minimization," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 12, no. 2, pp. 273–282, Feb. 1993.

[19] C. C. Chang and J. Cong, "An efficient approach to multilayer layer assignment with an application to via minimization," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 18, no. 5, pp. 608–620, May 1999.

[20] 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.

[21] K. C. Chang and H. C. Du, "Layer assignment problem for three-layer routing," *IEEE Trans. Comput.*, vol. 37, no. 5, pp. 625–632, May 1988.

[22] J. A. Roy, 2008. private communication.

[23] 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.

Tsung-Hsien Lee received the B.S. and M.S. degrees in computer science from National Tsing Hua University, Hsinchu, Taiwan, R.O.C., in 2005 and 2007, respectively.

He is currently with the Department of Computer Science, National Tsing Hua University. His current research interests include algorithm design, software engineering, and VLSI routing.

Mr. Lee was the recipient of the 2004 Programming Contest Award (first place) from National Tsing Hua University, the 2004 Two-Strait College Programming Contest Award (third place) from National Tsing Hua University and Tsinghua University, and the 2008 ISPD Global Routing Contest Award (first place).

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

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

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